СТАНДАРТ ST.90

РЕКОМЕНДАЦІЇ ЩОДО ОБРОБКИ ТА ОБМІНУ ДАНИМИ
СТОСОВНО ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ З ВИКОРИСТАННЯМ
ВЕБІНТЕРФЕЙСІВ API (ІНТЕРФЕЙСІВ ПРИКЛАДНОГО ПРОГРАМУВАННЯ)

Версія 1.1

Редакція затверджена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

ЗМІСТ

СТАНДАРТ ST.90

РЕКОМЕНДАЦІЇ ЩОДО ОБРОБКИ ТА ОБМІНУ ДАНИМИ СТОСОВНО ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ З ВИКОРИСТАННЯМ ВЕБІНТЕРФЕЙСІВ API (ІНТЕРФЕЙСІВ ПРИКЛАДНОГО ПРОГРАМУВАННЯ)

Версія 1.1

Редакція затверджена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

ВСТУП

1. Цей стандарт містить рекомендації щодо інтерфейсів прикладного програмування (API) для полегшення обробки та обміну даними стосовно інтелектуальної власності (ІВ) в гармонізованому порядку через мережу Інтернет.

2. Метою цього стандарту є:

- забезпечення узгодженості шляхом встановлення єдиних принципів проєктування вебсервісів;
- вдосконалення взаємодії даних між партнерами з вебсервісів;
- заохочення можливості повторного використання завдяки уніфікованому дизайну;
- сприяння гнучкості іменування даних у бізнес-підрозділах за допомогою чітко визначеної політики простору імен у пов'язаних XML-ресурсах;
- сприяння безпечному обміну інформацією;
- пропонування відповідних внутрішніх бізнес-процесів як послуги з доданою вартістю, які можуть бути використані іншими організаціями; і
- інтегрування внутрішніх бізнес-процесів і динамічне пов'язування їх із бізнес-партнерами.

ВИЗНАЧЕННЯ ТА ТЕРМІНОЛОГІЯ

3. У цьому стандарті вираз:

- «протокол передачі гіпертексту (HTTP)» (Hyper Text Transfer Protocol) – означає прикладний протокол для розподілених, спільних і гіпермедійних інформаційних систем. HTTP є основою передачі даних для Всесвітньої павутини (World Wide Web). HTTP функціонує як протокол «запит-відповідь» у сервісно-орієнтованій моделі обчислень;
- «інтерфейси прикладного програмування» (API) («Application Programming Interfaces» (API)) - означають програмні компоненти, які забезпечують інтерфейс багаторазового використання між різними додатками, здатними легко взаємодіяти для обміну даними;
- «передача репрезентативного стану (REST)» («Representational State Transfer (REST)» - описує набір архітектурних принципів, за допомогою яких дані можуть передаватися через стандартизований інтерфейс, тобто HTTP. REST не містить додаткового рівня обміну повідомленнями і фокусується на правилах проєктування для створення сервісів без статичних даних;
- - «простий протокол доступу до об'єктів (SOAP)» («Simple Object Access Protocol (SOAP)») - означає протокол для надсилання та отримання повідомлень між

додатками, без зіткнення із проблемами сумісності. SOAP визначає стандартний протокол зв'язку (набір правил) специфікації для обміну повідомленнями на основі XML. SOAP використовує різні транспортні протоколи, такі як HTTP і SMTP. Стандартний протокол HTTP полегшує тунелювання моделі SOAP через брандмауери та проксі-сервери без будь-яких модифікацій у протоколі SOAP;

- «вебсервіс» («Web Service») - означає метод зв'язку між двома додатками або електронними машинами через Всесвітню павутину (WWW), а вебсервіси бувають двох видів: REST і SOAP;
- - «RESTful Web API» - означає набір вебсервісів, що базуються на архітектурній парадигмі REST і зазвичай використовують формат JSON або XML для передачі даних;
- - «SOAP Web API» - означає набір вебсервісів SOAP, що базуються на SOAP і передбачають використання XML в якості формату корисних даних;
- - «мова опису вебслужб (Web Services Description Language - WSDL)» - означає стандарт Консорціуму Всесвітньої павутини (W3C), який використовується з протоколом SOAP для надання опису вебслужби. До нього входять методи, які використовує вебслужба, параметри, які вона приймає, засоби пошуку вебслужб тощо;
- Restful API Modeling Language (RAML) - відноситься до мови, яка дозволяє розробникам надавати специфікацію свого API;
- Open API Specification (OAS) - відноситься до мови, яка дозволяє розробникам надавати специфікацію свого API;
- «Сервісний контракт (Service Contract)» (або контакт вебсервісу) - означає документ, який показує, як сервіс розкриває іншим програмам свої можливості у вигляді функцій і ресурсів, що пропонуються у якості опублікованого API сервісу; термін «документація REST API» взаємозамінно використовується для позначення сервісного контракту для RESTful Web API;
- «постачальник послуг» («Service Provider») - означає програмне забезпечення вебсервісу, що відкриває вебсервіс;
- «споживач послуг» («Service Consumer») - означає роль під час виконання, яку бере на себе програма, коли звертається до служби і викликає її. А саме, коли програма надсилає повідомлення сервісу, зазначеному в договорі про надання послуг. Отримавши запит, сервіс починає обробку і може повернути або не повернути споживачеві послуги відповідне повідомлення-відповідь;
- - нижній регістр «Camelcase» - це або (lowerCamelCase) (наприклад, applicantName), або (UpperCamelCase) верхній регістр (наприклад, ApplicantName) угода про іменування;
- «шашличний регістр» (Kebab-case) - це одна з конвенцій іменування, де всі слова пишуться в нижньому регістрі з дефісом «-», що розділяє слова, наприклад, a-b-c;
- - «відкриті стандарти» («Open Standards») – означають загальнодоступні стандарти, які розробляються (або затверджуються) і підтримуються в процесі співпраці та досягненні консенсусу. «Відкриті стандарти» полегшують взаємодію та обмін даними між різними сервісними продуктами і призначені для широкого впровадження;
- «уніфікований ідентифікатор ресурсів» (URI) (Uniform Resource Identifier (URI)) - ідентифікує ресурс, а універсальний покажчик ресурсів (URL) -це підмножина URI, що включає мережеве розташування;
- «мітка сутності (ETag)» («Entity Tag (ETag)») - означає непрозорий ідентифікатор, присвоєний вебсервером конкретній версії ресурсу, знайденого за URL-адресою.

Якщо представлення ресурсу за цією URL-адресою коли-небудь змінюється, призначається новий, відмінний від попереднього, ETag. ETags можна швидко порівняти, щоб визначити, чи є два представлення ресурсу однаковими;

- «реєстр послуг» («Service Registry») - означає мережевий каталог, що містить доступні послуги;
- «RMM» - означає модель зрілості Річардсона (Richardson Maturity Model) - покажчик зрілості REST API за шкалою від 0 до 3; та
- «семантичне версіонування» («Semantic Versioning») - означає схему версіонування, в якій версія ідентифікується номером версії MAJOR. MINOR. PATCH, де:

  - версія MAJOR - коли в API відбуваються несумісні зміни,
  - версія MINOR - коли додається функціональність у зворотньо-сумісний спосіб і
  - версія PATCH- коли виконуються зворотньо-сумісні виправлення помилок.

4. З точки зору відповідності, у правилах оформлення, такі ключові слова потрібно тлумачити так само, як це визначено в п. 8 стандарту ВОІВ ST. 96[1], а саме:

- ПОВИНЕН (MUST): є еквівалентом «НЕОБХІДНО» або «СЛІД», означає, що визначення є абсолютною вимогою специфікації;

- НЕ ПОВИНЕН: є еквівалентом «НЕ СЛІД», означає, що визначення є абсолютно забороненим специфікацією;

- МАЄ: є еквівалентом «РЕКОМЕНДОВАНО» означає, що можуть існувати вагомі причини для ігнорування цього пункту, але наслідки цього мають бути повністю враховані;

- НЕ МАЄ: є еквівалентом «НЕ РЕКОМЕНДОВАНО», означає, що можуть існувати вагомі причини, за яких така поведінка може бути прийнятною або навіть корисною, але наслідки такої поведінки мають бути ретельно розглянуті; і

- МОЖЕ: є еквівалентом «НЕОБОВ'ЯЗКОВО», означає, що цей пункт є дійсно необов'язковим і надається лише як одна, вибрана з багатьох, опцій, .

ПОЗНАЧЕННЯ

Загальні позначення

5. У цьому документі використовуються такі позначення:
   - < > : Вказує на умовне позначення описового терміна, який під час впровадження буде замінено конкретним значенням;
   - { }: Вказує, що елементи є необов'язковими для застосування; і
   - «»: Вказує на те, що текст, включений у лапки, має бути використаний дослівно в імплементації
   - Шрифт `Courier New`: Вказує на ключові слова або вихідний код.

6. URL-адреси, наведені в цьому Стандарті, призначені тільки для прикладу і не є

[1] Див. розділ Список використаних джерел

реальними.

Ідентифікатори правил

7. Усі правила проєктування є нормативними. Правила проєктування позначаються за допомогою префікса [XX-nn] або [XXY-nn].

(a) Значення " XX « є префіксом для класифікації типу правила наступним чином:

- WS для правил проєктування SOAP Web API;
- RS для правил проєктування RESTful Web API; і
- CS для правил проєктування як SOAP, так і RESTful WEB API.

(b) Значення «Y» використовується лише для правил проєктування RESTful і забезпечує додаткову деталізацію типу відповіді, до якої належить правило:

- «G» означає, що це загальне правило для відповідей у форматі JSON та XML;
- «J» вказує на те, що це відповідь у форматі JSON; і
- «X» означає, що це відповідь у форматі XML.

(c) Значення «nn» вказує на наступний доступний номер у послідовності конкретного типу правила. Номер не відображає позицію правила, зокрема, для нового правила. Нове правило буде розміщене у відповідному контексті. Наприклад, ідентифікатор правила [WS-4] ідентифікує четверте правило проєктування SOAP Web API. Правило [WS-4] може бути розміщене між правилами [WS-10] і [WS-11], а не після [WS-3], якщо це найбільш прийнятне розташування для даного правила.

(d) У разі видалення правила його ідентифікатор буде збережено, а текст правила замінено на «Видалено».

СФЕРА ЗАСТОСУВАННЯ

8. Цей стандарт призначений для Відомств інтелектуальної власності (BIB) та інших організацій, яким необхідно управляти даними IB, зберігати їх, обробляти, обмінюватися і поширювати з використанням Web-API. Передбачається, що використання цього Стандарту дозволить спростити та прискорити розроблення вебінтерфейсів API узгодженим чином і поліпшити сумісність між Web -API.

9. Цей стандарт призначений для взаємодії між BIB і заявниками або користувачами даних, а також між BIB через взаємодії виду пристрій – пристрій та пристрій – програмні застосунки.

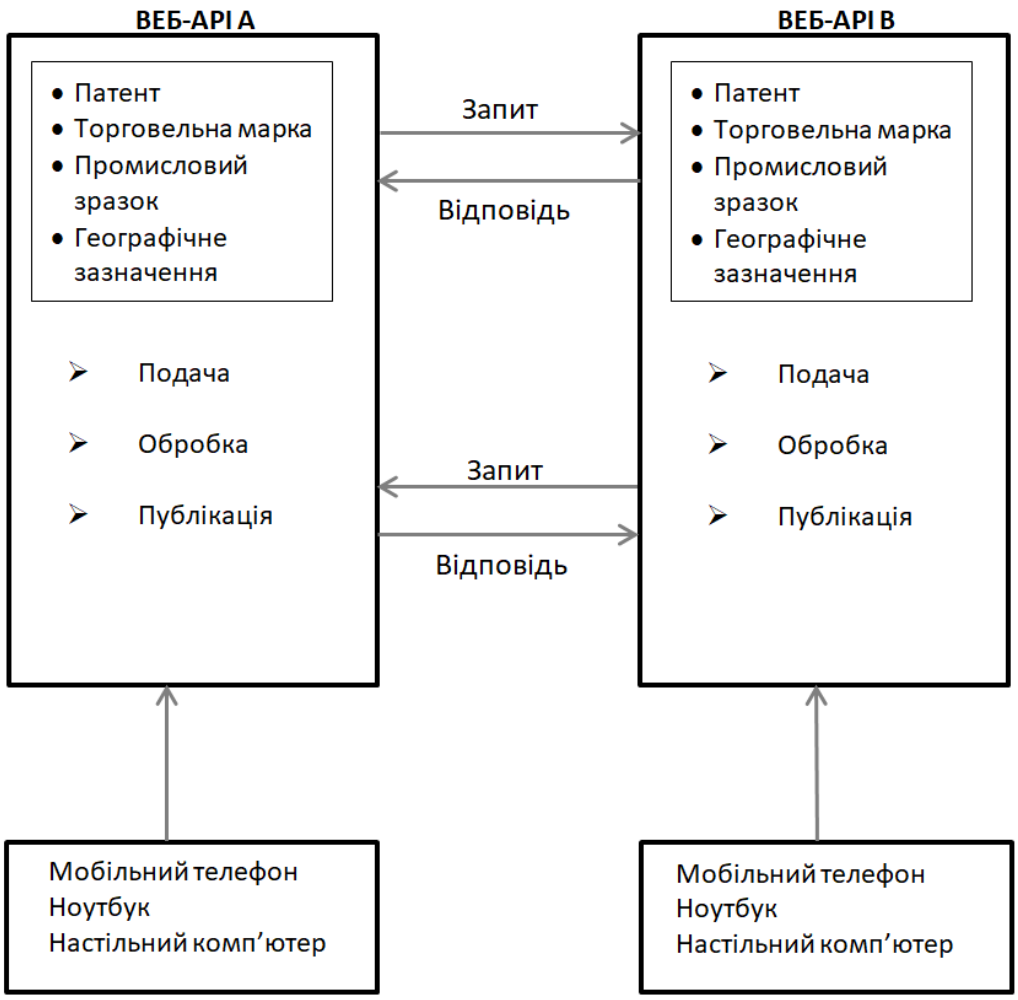


Рис. 1 Мета стандарту

10. Цей стандарт має на меті забезпечити набір правил та угод щодо проєктування RESTful та SOAP Web API; перелік ресурсів даних ІВ, якими будуть обмінюватися або які будуть розкриті; та типову документацію API або контракт про надання послуг, який може бути використаний для налаштування, з описом формату повідомлень, структури даних і словника даних у форматі JSON на основі стандарту BOIB ST.97 та/або форматі XML на основі стандарту BOIB ST.96.

11. Цей стандарт надає типові сервісні контракти для SOAP Web API, що використовують WSDL, і для RESTful Web API, що використовують REST API Modeling Language (RAML) та Open API Specification (OAS). Сервісний контракт також визначає або посилається на типи даних для інтерфейсів (див. розділ «Конвенція про типи даних» нижче). Цей стандарт рекомендує три типи інтерфейсів: REST-XML (XSD), REST-JSON та SOAP-XML (XSD).

12. Цей Стандарт виключає наступне:

(a) Прив'язку до конкретних технологічних стеків реалізації та комерційних готових продуктів (COTS);

(b) Прив'язку до конкретних архітектурних проєктів (наприклад, сервіс-орієнтована архітектура (SOA) або мікросервіс-орієнтована архітектура (MOA));

(c) Прив'язку до певних алгоритмів, таких як алгоритм обчислення ETag, тобто обчислення унікального ідентифікатора для конкретної версії ресурсу (наприклад, використовуваного для кешування).

ПРИНЦИПИ ПРОЄКТУВАННЯ WEB API

13. Як RESTful Web API, так і SOAP Web API довели свою здатність задовольняти вимоги великих організацій, а також обслуговувати невеликі вбудовані додатки у виробництві. При виборі між RESTful і SOAP можна розглянути наступні аспекти:

- безпека, наприклад, SOAP має WS-Security, тоді як REST не визначає жодних моделей безпеки;
- транзакція ACID, наприклад, SOAP має специфікацію WS-AT, тоді як REST не має відповідної специфікації;
- архітектурний стиль, наприклад, мікросервіси та безсерверна архітектура використовують REST, тоді як SOA використовує вебсервіси SOAP;
- гнучкість;
- обмеження пропускної здатності; і
- гарантована доставка, наприклад, SOAP пропонує WS-RM, тоді як REST не має відповідної специфікації.

14. При розробці Web API слід дотримуватися наступних принципів сервіс-орієнтованого проєктування:

(a) Стандартизований сервісний контракт: Стандартизація сервісних контрактів є найважливішим принципом проєктування, оскільки контракти забезпечують управління та послідовне проєктування сервісу. Сервісний контракт повинен бути простим для реалізації та розуміння. Контракт складається з метаданих, які описують, як будуть взаємодіяти постачальник і споживач. Метадані також описують умови їх взаємодії. Рекомендується, щоб сервісні контракти включали:

- Функціональні вимоги: яку функціональність забезпечує сервіс і які дані він буде повертати, або, як правило, поєднання цих двох вимог;
- Нефункціональні вимоги: інформація про відповідальність постачальників за надання своїх функціональних можливостей та/або даних, а також про очікувані обов'язки споживачів цієї інформації та про те, що вони повинні будуть надати натомість. Наприклад, доступність, безпека та інші аспекти якості обслуговування.

(b) Послуга втрачає зв'язок: Клієнти та сервіси мають розвиватися незалежно один від одного. Застосування цього принципу проєктування вимагає:
- Версії сервісу - споживачі, прив'язані до версії Web API, не повинні ризикувати несподіваними перебоями в роботі через несумісні зміни API; і
- Сервісний контракт має бути незалежним від деталей технології.

(c) Абстракція сервісу - Деталі реалізації сервісу повинні бути приховані. Дизайн API повинен бути незалежним від стратегій, що підтримуються сервером. Наприклад, для вебсервісу REST модель ресурсів API має бути відокремлена від моделі сутностей на рівні збереження даних;

(d) Нестаціонарність сервісу – Сервіси повинні бути масштабованими;

(e) Можливість повторного використання сервісу - Добре спроєктований API повинен надавати послуги багаторазового використання за допомогою типових контрактів. У зв'язку з цим, цей Стандарт надає типовий договір про надання послуг;

(f) Автономність сервісу - функціональні межі служби мають бути чітко визначені;

(g) Доступність послуг - послуги повинні бути ефективно відкриті та інтерпретовані;

(h) Компонованість сервісу - сервіси можуть використовуватися для створення інших сервісів;

(i) Використання стандартів як основи - API повинен відповідати галузевим стандартам (таким як IETF, ISO та OASIS), де це можливо, надаючи їм перевагу перед локально оптимізованими рішеннями; і

(j) Принцип вибору - не обов'язково реалізовувати всі правила проєктування API. Правила проєктування мають бути обрані на основі реалізації кожного конкретного випадку.

15. Крім того, слід дотримуватися наступних принципів, особливо щодо Web API RESTful:

(a) Кешованість: відповіді явно вказують на можливість їх кешування;

(b) Ідентифікація ресурсів у запитах: окремі ресурси ідентифікуються в запитах; наприклад, за допомогою URI у вебсистемах REST. Самі ресурси концептуально відокремлені від їхніх представлень, які повертаються клієнту;

(c) Гіпермедіа як механізм стану додатка (HATEOAS) - отримавши доступ до початкового URI для додатка REST- аналогічно до того, як людина отримує доступ до домашньої сторінки вебсайту - клієнт REST повинен мати змогу динамічно використовувати надані сервером посилання для виявлення всіх доступних дій та ресурсів, які йому потрібні;

(d) Управління ресурсами через їхні представлення - якщо клієнт має представлення ресурсу, включно з будь-якими прикріпленими метаданими, він має достатню інформацію для зміни або видалення ресурсу;

(e) Повідомлення, які не потребують додаткового опису - кожне повідомлення містить достатньо метаданих, щоб описати, як обробляти вміст повідомлення;

(f) Web API має відповідати семантиці HTTP, такій як методи, помилки тощо;

(g) Доступність для громадськості - розробка з метою, щоб API з часом був доступний з публічного інтернету, навіть якщо наразі цього не планується робити;

(h) Загальна автентифікація - використання загальної схеми автентифікації та авторизації, переважно на основі наявних компонентів безпеки, щоб уникнути створення індивідуального рішення для кожного API;

(i) Найменша привілейованість – права доступу та авторизація повинні призначатися споживачам API з міркувань мінімального обсягу прав, необхідного для виконання необхідних функцій;

(j) Максимізація ентропії - випадковість облікових даних має бути максимізована шляхом використання ключів API, а не імені користувача та паролів для авторизації API, оскільки ключі API забезпечують більш складну для потенційних зловмисників поверхню атаки; і

(k) Продуктивність проти безпеки – баланс між продуктивністю та безпекою з урахуванням часу життя ключів та накладних витрат на шифрування / дешифрування.

RESTFUL WEB API

16. RESTful Web API дає змогу системам, що запитують, отримувати доступ до текстових представлень вебресурсів і управляти ними, використовуючи однаковий і заздалегідь визначений набір операцій без стану.

Компоненти URI

17. RESTful Web API використовують URI для адресації ресурсів. Відповідно до RFC 3986, синтаксис URI повинен бути визначений наступним чином:

```
URI = <scheme> "://" <authority> "/" <path> {"?" query}

authority = {userinfo@}host{: port}
```

Наприклад, https://wipo.int/api/v1/patents?sort=id&offset=10

```
  ___________/_____/_________/__________________/
     |          |       |          |
  scheme  authority  path       query   параметри
```

18. Символ скісної риски «/» використовується у шляху URI для позначення ієрархічних зв'язків між ресурсами, але шлях не повинен закінчуватися скісною рискою, оскільки вона не несе жодного смислового навантаження і може призвести до плутанини.

[RSG-01] Символ скісної риски «/» ПОВИНЕН використовуватися у шляху URI, для зазначення ієрархічних відносин між ресурсами, але шлях НЕ ПОВИНЕН закінчуватися скісною рискою.

19. URI чутливі до регістру, за винятком частин схеми та хосту. Наприклад, хоча `https://wipo.int/api/my-resources/uniqueId` та `https://wipo.INT/api/my-resources/UniqueId` – це одне й теж саме, `https://wipo.int/api/my-resources/uniqueid` - ні. Угоди «kebab-case» і «lowerCamelCase» забезпечують для імен ресурсів хорошу читабельність і зіставляють імена ресурсів із сутностями в мовах програмування за допомогою простого перетворення. Для параметрів запиту слід використовувати нижній регістр (lowerCamelCase). Наприклад, `https://wipo.int/api/v1/inventors?firstName=John`. Імена ресурсів і параметри запиту чутливі до регістру. Варто зазначити, що імена ресурсів і параметрів запиту можуть бути скорочені:

20. RESTful Web API може мати аргументи:

- У параметрі запиту; наприклад, /inventors?id=1;
- У параметрі сегмента шляху URI, наприклад, /inventors/1; і
- У корисних даних запиту, наприклад, як частина тіла у форматі JSON.

21. За винятком вищезазначених типів аргументів, які є частиною URI, аргумент також може бути частиною корисного навантаження запиту.

[RSG-02] Назви ресурсів ПОВИННІ бути узгодженими з правилами іменування.

[RSG-03] Назви ресурсів у запиті ПОВИННІ використовувати угоди про іменування «kebab-case», і вони МОЖУТЬ бути скорочені.

[RSG-04] Параметри запиту ПОВИННІ бути послідовними у своєму іменуванні.

[RSG-05] Параметри запиту ПОВИННІ використовувати угоду про нижній регістр (lowerCamelCase), і вони МОЖУТЬ бути скороченими.

22. Кінцева точка Web API повинна відповідати запиту про надання коментарів Інженерної ради Інтернету `IETF RFC 3986` і має уникати потенційних колізій з URL-адресами сторінок вебсайту, розміщеного на кореневому домені. Web API повинен мати одну точку входу для консолідації всіх запитів. Загалом, існує два способи визначення кінцевих точок:

- як перший сегмент шляху до URI, наприклад: `https://wipo.int/api/v1/`; і
- у якості піддомену, наприклад: `https://api.wipo.int/v1/`

[RSG-06] Шаблон URL-адреси для Web API ПОВИНЕН містити слово «*api*» в URI.

23. Матричні параметри є ознакою того, що API є складним з декількома рівнями ресурсів і підресурсів. Це суперечить принципам сервісно-орієнтованого проєктування, визначеним раніше. Крім того, матричні параметри не є стандартними, оскільки вони застосовуються до конкретного елемента шляху, тоді як параметри запиту застосовуються до запиту в цілому. Прикладом матричних параметрів є такий:
`https://api.wipo.int/v1/path;param1=value1;param2=value2`.

[RSG-07] Матричні параметри НЕ ПОВИННІ використовуватись**.**

Коди стану

24. Web API повинен послідовно застосовувати коди стану HTTP, як описано в запитах про надання коментарів (RFC) Інженерної ради Інтернету (IETF). Коди стану HTTP мають використовуватися з-поміж тих, які перераховані в стандартних кодах стану HTTP (RFC 7807), наведених в Додатку V.

[RSG-08] Web API ПОВИНЕН систематично застосовувати коди стану HTTP, як описано в IETF RFCs.

[RSG-09] Для класифікації помилки у Web API МАЮТЬ використовуватися рекомендовані коди відповідно до Додатку V.

Принцип вибору

25. Сервісний контракт повинен бути толерантним до неочікуваних параметрів (в запиті, за допомогою параметрів запиту), але видавати помилку в разі неправильних значень очікуваних параметрів.

> [RSG-10] Якщо API виявляє неприпустимі значення введення, він ПОВИНЕН повернути код стану HTTP «`400 Bad Request`». Дані помилки ПОВИННІ вказувати на помилкове значення.
>
> [RSG-11] Якщо API виявляє синтаксично правильні імена аргументів (у параметрах запиту або запиті), які не очікуються, він МАЄ ігнорувати їх.
>
> [RSG-12] Якщо API виявляє допустимі значення, які вимагають нереалізованих функцій, він ПОВИНЕН повернути код стану HTTP «`501 Not Implemented`». Дані помилки ПОВИННІ вказувати на необроблене значення.

Ресурсна модель

26. Модель даних ІВ має бути розділена на обмежені контексти відповідно до підходу проєктування, орієнтованого на предметну область. Кожен обмежений контекст має бути зіставлений з ресурсом. Відповідно до принципів проєктування модель ресурсів Web API має бути відокремлена від моделі даних. Web API має бути змодельований як ієрархія ресурсів, для використання ієрархічної природи URI для позначення структури (асоціації, композиції або агрегування), де кожен вузол є або простим (одиничним) ресурсом, або колекцією ресурсів.

27. У цій ієрархічній моделі ресурсів вузли в корені називаються «вузлами верхнього рівня», а всі вкладені ресурси - «підресурсами». Підресурси слід використовувати тільки для позначення композицій, тобто ресурсів, які не можуть бути ресурсами верхнього рівня, інакше існувало б декілька способів отримання одних і тих самих сутностей. Такі підресурси, що передбачають об'єднання, називаються підколекціями. Від інших ієрархічних структур, тобто асоціації та агрегування, слід утримуватися, щоб уникнути складних API і дублювання функціональності.

28. Кінцева точка завжди визначає тип відповіді. Наприклад, кінцева точка `https://wipo.int/api/v1/patents` завжди повертає відповіді, що стосуються патентних даних. Кінцева точка `https://wipo.int/api/v1/patents/1/inventor` завжди повертає відповіді, що стосуються даних винахідника. Однак кінцева точка `https://wipo.int/api/v1/inventors` не допустима, оскільки ресурс винахідника не може бути автономним.

29. Слід використовувати тільки ресурси верхнього рівня, тобто не більше одного рівня, інакше ці API будуть дуже складними для реалізації. Наприклад, замість `https://wipo.int/api/v1/inventors/12345/patents` має використовуватись `https://wipo.int/api/v1/patents?inventorId=12345`.

> [RSG-13] Web API МАЄ використовувати тільки ресурси верхнього рівня. Якщо є підресурси, вони повинні бути колекціями і передбачати взаємозв'язок. Об'єкт має бути доступним як ресурс верхнього рівня, або як підресурс, але не можна використовувати обидва способи.

[RSG-14] Якщо ресурс може бути самостійним (автономним), він ПОВИНЕН бути ресурсом верхнього рівня, в іншому разі - підресурсом.

[RSG-15] Параметри запиту ПОВИННІ використовуватися замість шляхів URL для отримання вкладених ресурсів.

30. Існує декілька типів[2] Web API: CRUD (Create, Read, Update, and Delete) Web API та Intent (Намір) Web API. CRUD API моделюють зміни в ресурсі, тобто операції створення/читання/оновлення/видалення. Intent Web API, навпаки, моделюють бізнес-операції, наприклад, поновлення/реєстрація/публікація. CRUD-операції мають використовувати іменники, а Intent Web API - дієслова для назв ресурсів. CRUD Web API є найпоширенішими, але обидва типи можна комбінувати, наприклад, споживач послуг може використовувати Intent Web API, що моделює бізнес-операцію, яка організовує виконання однієї або декількох операцій CRUD Web API сервісу. Використовуючи CRUD Web API, користувач сервісу повинен організувати бізнес-логіку, але з Intent Web API бізнес-логіку організовує постачальник сервісу. CRUD Web API не є атомарними порівняно з Intent Web APIs [3].

- Наприклад, власник торговельних марок хоче продовжити термін дії тих, які закінчуються найближчим часом (наприклад, `yyyy-mm-dd`). Це комбінація наступних бізнес-операцій:
  - Отримання торговельних марок, термін дії яких закінчується `yyyy-mm-dd`; і
  - Поновлення чинності знайдених торговельних марок з їхнім міжнародним реєстраційним номером.

Використовуючи CRUD Web API, попередні бізнес-операції були б змодульовані за допомогою неатомарного процесу, що вимагає двох дій, таких як:

Крок 1: Отримати всі торговельні марки у форматі XML[4], які належать власнику з ім'ям Джон Сміт і термін дії яких закінчується, наприклад, 2018-12-31:

```
GET /api/v1/trademarks?holderFullName=John%20Smith&expiryDate=2018-12-31.HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Наведений далі приклад повертає HTTP-відповіді:

[2] Як варіант, можна класифікувати API відповідно до їхнього архетипу. Див. наприклад: «Збірник правил проєктування REST API: Проєктування узгоджених інтерфейсів RESTful вебсервісів»

[3] Intent API також дозволяє застосовувати шаблон розподілу відповідальності за командні запити (Command Query Responsibility Segregation - CQRS). CQRS - це патерн, в якому для оновлення інформації можна використовувати модель, відмінну від тієї, яка використовується для зчитування інформації. Це пояснюється тим, що для багатьох проблем, особливо в більш складних доменах, наявність однакової концептуальної моделі для команд і запитів призводить до більш складної моделі, яка не є корисною.

[4] Приклад JSON пропущено, оскільки в даному випадку він не додає значущості

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<tmk:TrademarkBag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
TrademarkBag.xsd">
      <tmk:Trademark xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
com:operationCategory="Delete"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
Trademark.xsd">
            ...
            <com:RegistrationNumber>
                  <com:IPOfficeCode>IT</com:IPOfficeCode>

      <com:ST13ApplicationNumber>000000000000001</com:ST13ApplicationNumber>
            </com:RegistrationNumber>
            ...
            <com:ExpiryDate>2018-12-31</com:ExpiryDate>
            ...
      </tmk:Trademark>
      ...
</tmk:TrademarkBag>
```

Крок 2: Подати запит на продовження строку дії кожної торговельної марки, отриманої на попередньому кроці (тут показано лише перший запит на продовження строку дії)

```
POST /api/v1/trademarks/renewalRequests HTTP/1.1

Host: wipo.int

Accept: application/xml

Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>

<tmk:MadridRenewal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
MadridRenewal.xsd">

       ...

       <com:InternationalRegistrationNumber>000000000000001</com:International
RegistrationNumber>

       ...

</tmk:MadridRenewal>
```

- Попередній приклад також можна змоделювати за допомогою атомарного виклику служби з використанням Intent Web API, наприклад[5]:

[5] Елемент `InternationalRegistrationNumber` видалено з корисного навантаження для позначення всіх IRN. Стандарт BOIB ST.96 не слід використовувати або послаблювати, оскільки наведений тут приклад розширює випадки використання, дозволені у стандарті BOIB ST.96

```
POST
/api/v1/trademarks/findAndRenew?holderFullName=john%20smith&expiryDate=2018-12-
31

Host: wipo.int
```

31. Тип Web API має накладати обмеження на те, як іменуються ресурси, щоб дати уявлення про те, який з них використовується. Варто зазначити, що імена ресурсів, локалізовані у зв'язку з вимогами бізнесу, можуть бути іншими мовами.

[RSG-16] Імена ресурсів МАЮТЬ бути іменниками для CRUD Web APIs і дієсловами для Intent Web APIs.

[RSG-17] Якщо ім'я ресурсу є іменником, то він МАЄ завжди використовуватися у формі множини. Неправильні форми іменників НЕ МАЮТЬ використовуватись. Наприклад, замість /people слід використовувати /persons.

[RSG-18] Імена ресурсів, сегментів і параметри запитів ПОВИННІ складатися з англійських слів і відповідати вимогам до написання, зазначеним в Оксфордському словнику англійської мови (Oxford English Dictionary). Імена ресурсів, які локалізовані у зв'язку з вимогами бізнесу, МОЖУТЬ бути іншими мовами.

Підтримка декількох форматів

32. Різні споживачі послуг можуть мати різні вимоги до формату даних для відповідей сервісу. Тип носія даних повинен бути відокремлений від самих даних, що дозволить сервісу підтримувати різні типи носіїв. Отже, Web API повинен підтримувати узгодження типу вмісту за допомогою HTTP- заголовка запиту `Accept` і HTTP-заголовка відповіді `Content-Type`, як того вимагає IETF RFC 7231. Наприклад, для запиту даних у форматі JSON заголовок *Accept* має бути `Accept: application/json`, а для даних у форматі XML - `Accept: application/xml`. Так само для заголовка `Content-Type`. Крім того, Web API може підтримувати інші способи узгодження типу вмісту, такі як параметр запиту (наприклад `?format`) або суфікс URL (наприклад, *`.json`*).

[RSG-19] Web API МАЄ використовувати для узгодження типу вмісту HTTP-заголовок запиту `Accept` та HTTP- заголовок відповіді `Content-Type`.

33. API повинні підтримувати запити та відповіді у форматі XML та JSON. Для XML відповіді повинні відповідати стандарту ВОІВ щодо використання XML, такому як ST.96, а для JSON відповіді повинні відповідати стандарту ВОІВ ST.97. Слід використовувати узгоджене відображення між цими двома форматами.

[RSG-20] Web API ПОВИНЕН підтримувати узгодження типів вмісту відповідно до IETF RFC 7231.

[RSG-21] Формат JSON ПОВИНЕН передбачатися, якщо не запитується певний тип вмісту.

[RSG-22] Web API МАЄ повертати код стану «`406 Not Acceptable`», якщо запитаний формат не підтримується.

[RSG-23] Web API МАЄ відхиляти запити, що містять несподівані або відсутні заголовки типу вмісту, видаючи кодом стану HTTP «`406 Not Acceptable`» або

«`415 Unsupported Media Type`».

[RSG-24] Запити та відповіді (угода про іменування, формат повідомлення, структура даних та словник даних) МАЮТЬ посилатись на стандарт BOIB ST.96 для формату XML або стандарт BOIB ST.97 для формату JSON.

[RSJ-25] Імена властивостей об'єктів JSON МАЮТЬ надаватися в нижньому регістрі (lowerCamelCase*)*, наприклад, applicantName*.*

[RSX-26] Імена компонентів XML МАЮТЬ надаватися в регістрі UpperCamelCase*.*

[RSG-27] Web API ПОВИНЕН підтримувати принаймні формат XML або JSON.

Методи HTTP

34. Методи HTTP- (або дієслова HTTP) - це тип функції, що надається єдиним контрактом для обробки ідентифікаторів ресурсів та даних. Методи HTTP слід використовувати за призначенням відповідно до стандартної семантики, як зазначено в IETF RFC 7231 та 5789, а саме:

- `GET` - одержання даних
- `HEAD` – як `GET`, але без корисного вмісту відповіді
- `POST` – відправка нових даних
- `PUT` – оновлення
- `PATCH` - часткове оновлення
- `DELETE` - видалення даних
- `TRACE` – повторення
- `OPTIONS` - дієслова запиту, які сервер підтримує для даної URL-адреси

35. Єдиний (уніфікований) контракт встановлює набір методів, які повинні використовуватися службами в межах даної колекції або інвентаризації. Тунелювання HTTP-методів може бути корисним, коли HTTP- заголовки відхиляються деякими брандмауерами.

36. Методи HTTP можуть слідувати принципу «ретельного вибору» («*pick-and-choose»*), який свідчить про те, що має бути реалізована тільки та функціональність, яка необхідна для цільового сценарію використання. Деякі проксі-сервери підтримують тільки методи *POST* і *GET*. Щоб подолати ці обмеження, Web API може використовувати метод *POST* з призначеним для користувача аголовком HTTP, який «тунелює» справжній метод HTTP.

[RSG-28] Методи HTTP ПОВИННІ бути обмежені стандартними методами HTTP `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` і `HEAD`, як зазначено в IETF RFC 7231 та 5789.

[RSG-29] Методи HTTP МОЖУТЬ дотримуватися принципу вибору *(«pick-and-choose»)*, який свідчить про те, що має бути реалізована тільки та функціональність, яка необхідна для цільового сценарію використання.

[RSG-30] Деякі проксі-сервери підтримують тільки методи `POST` і `GET`. Щоб подолати ці обмеження, Web API МОЖЕ використовувати метод `POST` з призначеним для користувача заголовком HTTP, що «тунелює» справжній метод HTTP. МАЄ використовуватися користувацький HTTP-заголовок `X- HTTP- Method`.

[RSG-31] Якщо метод HTTP не підтримується, МАЄ повернутися код стану HTTP «`405 Method Not Allowed`».

37. У деяких випадках мають підтримуватися декілька операцій одночасно.

[RSG-32] Web API МАЄ підтримувати пакетні операції (вони ж масові операції) замість декількох окремих запитів для зниження затримки. Така ж семантика має використовуватися для методів HTTP і кодів стану HTTP. У змістовній частині відповіді МАЄ містити інформацію про всі пакетні операції. Якщо виникає кілька помилок, змістовна частина помилки МАЄ містити інформацію про всі помилки (в атрибуті *details*). Усі пакетні операції МАЮТЬ виконуватися атомарно.

GET

38. Згідно з IETF RFC 2616, протокол HTTP не встановлює жодних попередніх обмежень на довжину URI. З іншого боку, сервери мають бути обережними, якщо довжина URI перевищує 255 байт, оскільки деякі старі реалізації клієнтів або проксі-серверів можуть не підтримувати таку довжину. У разі перевищення цього обмеження рекомендується використовувати іменовані запити. Як альтернативу необхідно вказати набір правил, що визначають, як перетворювати GET на POST. Згідно з IETF RFC 2616, запит GET має бути ідемпотентним, тобто відповідь буде однаковою незалежно від того, скільки разів виконується запит.

[RSG-33] Для кінцевої точки, яка витягує один ресурс, якщо ресурс не знайдено, метод `GET` ПОВИНЕН повертати код стану «`404 Not Found`». Кінцеві точки, які повертають списки ресурсів, просто повертають порожній список.

[RSG-34] Якщо ресурс отримано успішно, `метод GET` ПОВИНЕН повернути значення `200 OK`.

[RSG-35] Запит GET ПОВИНЕН бути ідемпотентним (незалежним від кількості виконань запиту).

[RSG-36] Якщо довжина URI перевищує 255 байт МАЄ використовуватися метод `POST` замість `GET` через обмеження `GET`, або якщо можливо, створювати іменовані запити.

HEAD

39. Коли клієнту необхідно дізнатися інформацію про операцію, він може використовувати *HEAD. HEAD* отримує HTTP- заголовок, який ви отримали б, якби зробили *GET*- запит, але без тіла. Це дозволяє клієнту визначити інформацію про кешування, який тип вмісту буде повернуто, який код стану буде повернуто. Запит HEAD ПОВИНЕН бути ідемпотентним відповідно до IETF RFC 2616.

[RSG-37] Запит `HEAD` ПОВИНЕН бути ідемпотентним.

[RSG-38] Деякі проксі-сервери підтримують тільки методи `POST` і `GET`. Web API МАЄ підтримувати користувацький заголовок запиту HTTP для перевизначення методу HTTP, щоб подолати ці обмеження.

POST

40. Коли клієнту необхідно створити ресурс, він може використовувати *POST*. Наприклад,

наступний HTTP-запит надсилає запит на патентну заявку.

- Наприклад, заявку на патент подають у такий спосіб.

  Приклад з корисним навантаженням в форматі XML на основі стандару BOIB ST.96

  Клієнт надсилає заявку на патент у форматі XML:

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/xml
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd">
      ...
</pat:ApplicationBody>
```

Повертається наступна HTTP-відповідь, що вказує на успішне подання заявки на патент:

```
HTTP/1.1 201 Created
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd" applicationBodyStatus="pending">
      ...
</pat:ApplicationBody>
```

Приклад з корисним навантаженням в форматі JSON на основі стандарту BOIB ST.97

Клієнт подає заявку на отримання патенту у форматі JSON:

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/json
Content-Type: application/json
{
      " applicationBody ": {
            ...
      }
}
```

Повертається наступна HTTP-відповідь, щоб повідомити про успішне подання заявки на патент:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      " applicationBody ": {
            "applicationBodyStatus" : "pending",
            ...
      }
}
```

[RSG-39] Запит `POST`- НЕ ПОВИНЕН бути ідемпотентним згідно з IETF RFC 2616.

[RSG-40] Якщо створення ресурсу пройшло успішно, HTTP-заголовок `Location` МАЄ містити URI (абсолютний або відносний), що вказує на створений ресурс.

[RSG-41] Якщо створення ресурсу пройшло успішно, відповідь МАЄ містити код стану «`201 Created`».

[RSG-42] Якщо створення ресурсу пройшло успішно, корисне навантаження відповіді МАЄ за замовчуванням містити тіло створеного ресурсу, щоб дозволити клієнту використовувати його без додаткового HTTP-виклику.

PUT

41. Коли клієнту необхідно повністю замінити існуючий ресурс, він може використовувати PUT. При цьому слід враховувати ідемпотентні характеристики PUT. Запит PUT має семантику оновлення (як зазначено в IETF RFC 7231) та семантику вставки.

[RSG-43] `PUT`-запит ПОВИНЕН бути ідемпотентним.

[RSG-44] Якщо ресурс не знайдено, `PUT` ПОВИНЕН повертати код стану «`404 Not Found`».

[RSG-45] Якщо ресурс оновлено успішно, PUT ПОВИНЕН повернути код стану «`200 OK`», якщо оновлений ресурс буде повернуто, або «`204 No Content`», якщо його не буде повернуто.

PATCH

42. Коли клієнту потрібне часткове оновлення, він може використовувати `PATCH`. Необхідно враховувати ідемпотентні характеристики `PATCH`.

- Наприклад, наступний запит оновлює лише мову патенту із зазначенням його номера:

```
PATCH /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
If-Match:456
Content-Type: application/merge-patch+json
{ "languageCode": "en" }
```

43. PATCH не повинен бути ідемпотентним відповідно до IETF RFC 2616. Щоб зробити його ідемпотентним, API може слідувати пропозиції IETF RFC 5789 щодо використання оптимістичного блокування (optimistic locking).

[RSG-46] Запит на `PATCH` НЕ ПОВИНЕН бути ідемпотентним.

[RSG-47] Якщо Web API реалізує часткові оновлення, то МАЮТЬ бути взяті до уваги ідемпотентні характеристики `PATCH`. Для того щоб зробити його ідемпотентним, API МОЖЕ слідувати рекомендаціям IETF RFC 5789 щодо використання оптимістичного блокування.

[RSG-48] Якщо ресурс не знайдено, `PATCH` ПОВИНЕН повернути код стану «`404 Not Found`».

[RSJ-49] Якщо Web API реалізує часткові оновлення за допомогою `PATCH`, він ПОВИНЕН використовувати формат JSON Merge Patch для опису часткового набору змін, як описано в IETF RFC 7386, використовуючи тип вмісту `application/merge-patch+json`.

DELETE

44. Коли клієнту потрібно видалити ресурс, він може використовувати *DELETE*. Запит *DELETE* не повинен бути ідемпотентним відповідно до IETF RFC 2616

[RSG-50] Запит `DELETE` НЕ ПОВИНЕН бути ідемпотентним.

[RSG-51] Якщо ресурс не знайдено, `DELETE` ПОВИНЕН повернути код стану «*`404 Not Found`*».

[RSG-52] Якщо ресурс видалено успішно, `DELETE` ПОВИНЕН повернути статус «`200 OK`», якщо видалений ресурс повернуто, або «`204 No Content`», якщо його не повернуто.

TRACE

45. Метод `TRACE` не несе семантики API і використовується для тестування та одержання діагностичної інформації відповідно до IETF RFC 2616, наприклад, для тестування ланцюжка проксі-серверів. `TRACE` дозволяє клієнту бачити, що одержано на іншому кінці ланцюжка запитів, і використовувати ці дані. Запит `TRACE` НЕ ПОВИНЕН бути ідемпотентним згідно з IETF RFC 2616.

[RSG-53] Кінцевим одержувачем є або вихідний сервер, або перший проксі-сервер чи шлюз, який отримав у запиті значення `Max-Forwards`, що дорівнює нулю. Запит `TRACE` НЕ ПОВИНЕН включати тіло (запиту).

[RSG-54] Запит `TRACE` НЕ ПОВИНЕН бути ідемпотентним.

[RSG-55] Значення поля заголовка `Via` HTTP ПОВИННО слугувати для відстеження ланцюжка запитів.

[RSG-56] Поле заголовка HTTP `Max-Forwards` ПОВИННО використовуватися для того, щоб клієнт міг обмежити довжину ланцюжка запитів.

[RSG-57] Якщо запит коректний, відповідь МАЄ містити повне повідомлення запиту в тілі відповіді, із зазначенням `Content-Type` «`message/http`».

[RSG-58] Відповіді на `TRACE` НЕ ПОВИННІ кешуватися.

[RSG-59] Код стану «`200 OK`» МАЄ повернутися в `TRACE`.

OPTIONS

46. Коли клієнту необхідно отримати інформацію про Web API, він може використовувати `OPTIONS`. `OPTIONS` не несе семантики API. Запит `OPTIONS` ПОВИНЕН бути ідемпотентним згідно з IETF RFC 2616, Custom HTTP Headers.

[RSG-60] Запит `OPTIONS` ПОВИНЕН бути ідемпотентним.

47. Поширеною практикою для Web API, що використовує власні HTTP-заголовки, є використання префікса «`X-`» як загального префікса, який RFC 6648 не підтримує і не рекомендує використовувати.

[RSG-61] НЕ МАЮТЬ використовуватися користувацькі HTTP-заголовки, що починаються з префікса «`x-`»..

[RSG-62] Користувацькі HTTP-заголовки НЕ МАЄ використовувати для зміни поведінки HTTP-методів, за винятком випадків, коли це необхідно для усунення існуючих технічних обмежень (наприклад, див. [RSG-39]).

[RSG-63] Угода про іменування користувацьких HTTP-заголовків має вигляд -`<organization>-<header name>`, де полям `<organization>` та -`<header >` МАЄ дотримуватися угоди про стиль іменування *kebab-case*.

48. Згідно з принципами сервіс-орієнтованого проєктування, клієнти та сервіси мають розвиватися незалежно один від одного. Версіонування сервісів дозволяє це зробити. Загальними реалізаціями версіонування сервісів є: версіонування заголовка (за допомогою користувацького заголовка), версіонування рядка запиту (наприклад, `?v=v1`), версіонування типу носія (наприклад, `Accept: application/vnd.v1+json`) і версіонування URI (наприклад, `/api/v1/inventors`).

[RSG-64] Web API МАЄ підтримувати єдиний метод версіонування сервісу з використанням версіонування URI, наприклад `/api/v1/inventors`, або версіонування заголовка, наприклад `Accept-version: v1`, або версіонування типу даних, наприклад `Accept: application/vnd.v1+json`. НЕ МАЄ використовуватися версіонування рядків запитів.

49. Згідно з принципами сервіс-орієнтованого проєктування, постачальники та споживачі послуг також повинні змінюватися незалежно один від одного. На споживача послуг не повинні впливати незначні (зворотно сумісні) зміни постачальника послуг. Тому під час версіонування сервісів слід використовувати тільки основні версії. Для внутрішніх неопублікованих API (наприклад, для розроблення та тестування) можуть також використовуватися мінорні версії, наприклад, *Semantic Versioning*.

[RSG-65] Схему нумерації версій СЛІД застосовувати з урахуванням тільки основного номера версії (наприклад, `/v1`).

50. Ідентифікатори кінцевих точок надання послуг містять інформацію, яка з часом може змінюватися. Замінити всі посилання на застарілу кінцеву точку може бути неможливо,

оскільки може призвести до того, що споживач послуги не зможе надалі взаємодіяти з цією кінцевою точкою. Тому постачальник послуг може повернути відповідь про перенаправлення. Перенаправлення може бути тимчасовим або постійним. Доступні такі коди стану HTTP:

	Постійний	**Тимчасовий**
Дозволяє змінити метод запиту з POST на GET	301	302
Не дає змоги змінити метод запиту з POST на GET	308	307

Оскільки `301` і `302` є більш загальними, їм надається перевага для підвищення гнучкості та подолання будь-якої непотрібної складності.

[RSG-66] Сервісні контракти API МОЖУТЬ включати функцію перенаправлення кінцевих точок. Коли споживач послуг намагається викликати послугу, може бути повернуто відповідь про перенаправлення, щоб повідомити споживача послуг про необхідність повторно надіслати запит на нову кінцеву точку. Перенаправлення МОЖУТЬ бути тимчасовими або постійними:

- Тимчасове перенаправлення - з використанням заголовка HTTP відповіді `Location` та коду стану HTTP «`302 Found`» відповідно до IETF RFC 7231; або
- Постійне перенаправлення - з використанням заголовка HTTP відповіді `Location` та коду стану HTTP «`301 Moved Permanently`» згідно з IETF RFC 7238.

51. У міру розвитку API проходить ряд основних етапів: планування і проєктування, розробка, тестування, розгортання і виведення з експлуатації. Замість того, щоб надавати рекомендації щодо періодів часу, протягом яких API бажано залишатися на певному етапі, бажано, щоб Організація або Постачальники послуг опублікували свою стратегію життєвого циклу API. Шаблон, який містить основні компоненти, що визначають стратегію життєвого циклу в Додатку VII.

[RSG-67] Розробникам СЛІД публікувати стратегії життєвого циклу API, щоб допомогти користувачам зрозуміти, як довго буде підтримуватися та чи інша версія.

Шаблони запитів до даних

Параметри пагінації

52. Пагінація - це механізм, за допомогою якого клієнт може отримувати дані по сторінках. Використовуючи пагінацію, ми запобігаємо перевантаженню постачальника послуг вимогливими до ресурсів запитами відповідно до принципів дизайну. Сервер повинен застосовувати розмір сторінки за замовчуванням, якщо споживач послуги не вказав його. Пагінаційні запити не можуть бути ідемпотентними, тобто сторінковий запит не створює знімок даних.

[RSG-68] Web API МАЄ підтримувати пагінацію (розбивку на сторінки).

[RSG-69] Пагінаційні запити НЕ МОЖУТЬ бути ідемпотентними.

[RSG-70] Web API ПОВИНЕН використовувати параметри запиту для реалізації пагінації (розбивки на сторінки).

[RSG-71] Web API НЕ ПОВИНЕН використовувати HTTP-заголовки для реалізації пагінації.

[RSG-72] СЛІД використовувати параметри запиту `limit=<кількість елементів для доставки>` і `offset=<кількість елементів для пропуску>(limit=<number of items to deliver> and offset=<number of items to skip>)`, *де limit* - кількість повернених елементів (розмір сторінки), а *пропуск (skip)* - кількість елементів, які мають бути пропущені (зміщення). Якщо обмеження на розмір сторінки не вказано, НЕОБХІДНО визначити значення за замовчуванням - глобальне або для кожної колекції; зміщення за замовчуванням ПОВИННО дорівнювати нулю «0»:

- Наприклад, наступний URL є припустимим:

```
https://wipo.int/api/v1/patents?limit=10&offset=20
```

[RSG-73] Значення параметрів `limit` і `offset` МАЄ бути включене у відповідь.

Сортування

53. Для отриманих даних може знадобитися їх сортування за зростанням або спаданням. Також може використовуватися багатоключовий критерій сортування. Сортування визначається за допомогою параметра рядка запиту `sort`. Значення цього параметра являє собою розділений комами список ключів сортування і напрямків сортування, які за бажанням можуть бути додані до кожного ключа сортування і розділені символом двокрапки ':'. Підтримуються такі напрямки сортування*:* `'asc'` - за зростанням або `'desc'` - за спаданням. Клієнт може вказати напрямок сортування для кожного ключа. Якщо напрямок сортування для ключа не вказано, то напрямок за замовчуванням задається сервером.

Наприклад:

(a) Вказані тільки ключі сортування:

```
sort=key1, key2
```

‘`key1`'- перший ключ, '`key2`'- другий ключ, а напрямки сортування задаються сервером за замовчуванням.

(b) Вказані деякі напрямки:

```
sort=key1:asc key2,
```

де *'key1'* - перший ключ (порядок зростання), а *'key2'* - другий ключ (напрямок за замовчуванням задається сервером, тобто будь-який ключ сортування без відповідного

напрямку задається за замовчуванням).

(c) кожен ключ із заданим напрямком:

```
sort=key1:asc,key2:desc
```

де `'key1'` - перший ключ (у порядку зростання), а `'key2'` - другий ключ (у порядку спадання).

54. Щоб задати сортування за багатоатрибутними критеріями, значенням параметра запиту може бути список ключів сортування і напрямів сортування, розділених комами, із символами `'asc'` для висхідного або `'desc'` для низхідного сортування, які можуть бути додані до кожного ключа сортування, розділені символом двокрапки ':'.

[RSG-74] Web API МАЄ підтримувати сортування.

[RSG-75] Для зазначення критерію сортування за декількома атрибутами ПОВИНЕН використовуватися параметр запиту. Значення цього параметра являє собою список ключів сортування, розділених комами, а напрямки сортування або `'asc'` для висхідного, або `'desc'` для низхідного МОЖУТЬ бути додані до кожного ключа сортування, розділені символом двокрапки ':'. Напрямок за замовчуванням ПОВИНЕН бути визначений сервером у разі, якщо для ключа не вказано напрямок сортування.

[RSG-76] Web API МАЄ повертати критерії сортування у відповіді.

Розширення

55. Споживач послуг може контролювати обсяг даних, які він отримує, розширюючи одне поле до більших об'єктів. Зазвичай це поєднується з підтримкою гіпермедіа. Замість того, щоб просто попросити включити пов'язаний ідентифікатор сутності, користувач сервісу може запросити повне представлення сутності в результатах. Виклики сервісів можуть використовувати розширення, щоб отримати всі необхідні дані в одному запиті API:

- Наприклад, якщо підтримується гіпермедіа, то наступний HTTP-запит витягує патент і розширює інформацію про його заявника.

<u>Приклад з корисним навантаженням у форматі JSON на основі стандарту BOIB ST.97</u>

Отримання патенту за його номером [6] :

```
GET /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
Accept: application/json
```

Відповідь HTTP має такий вигляд

[6] Patent/PatentNumber.xsd

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{

      "patentPublication":{
            "languageCode": "en",
            ...
            "bibliographicData": {
                  "st96Version": "V5_0",
                  "applicationIdentification": {
                        "ipOfficeCode": "XX",
                        "applicationNumber": {
                              "applicationNumberText": "13797521"
                        },
                        "inventionSubjectMatterCategory": "Utility",
                        "filingDate": "2013-03-12"
                  },
                  patentGrantIdentification": {
                        "ipOfficeCode": "XX",
                        "patentNumber": "100000000000001"
                  },
                  ...
                  "partyBag": {
                        "applicantBag": {
                              "applicant": {
                                    "href":
"https://wipo.int/api/v1/link/to/applicants"
                              },
                        ...
                        }
                  }
            },
            ...
      }
      }
```

Замість попереднього запиту, використовуючи наступний HTTP запит, ви отримаєте повну інформацію про заявника патенту з номером 100000000000001:

```
GET /api/v1/patents/publications?id=100000000000001&expand=applicant HTTP/1.1
Host: wipo.int
Accept: application/json
```

Відповідь HTTP виглядає наступним чином:

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{
      "patentPublication":{
      "languageCode": "en",
      ...
      "bibliographicData": {
            "st96Version": "V5_0",
            "applicationIdentification": {
                  "ipOfficeCode": "XX",
                  "applicationNumber": {
                        "applicationNumberText": "13797521"
                  },
                  "inventionSubjectMatterCategory": "Utility",
                  "filingDate": "2013-03-12"
            },
            patentGrantIdentification": {
                  "ipOfficeCode": "XX",
                  "patentNumber": "100000000000001"
            },
            ...
            "partyBag": {
                  "applicantBag": {
                        "applicant": [
                                    {
                                          "sequenceNumber": "001",
                                          "publicationContact": [
                                                {
                                                      "name": {
                                                            "personName": …,
                                          "applicantCategory": "Applicant",
                                    },
                                    {
                                          "sequenceNumber": "002",
                                          "publicationContact": [
                                                {
                                                      "name": {
                                                            "personName": …
                                                      }
                                                }
                                          ],
                                          "applicantCategory": "Applicant",
                                    },
                                    {
                                          "sequenceNumber": "003",
                                          "publicationContact": [
                                                {
                                                      "name": {
                                                            "personName": …
                                                      }
                                                }
                                          ],
                                          "applicantCategory": "Applicant",
                                    },
                              ...
                              }
                  }
            },
                  ...    }}
```

56. Web API може підтримувати розширення тіла вмісту, що повертається.

[RSG-77] Web API МОЖЕ підтримувати розширення тіла вмісту, що повертається. МАЄ використовуватись параметр запиту `expand=<розділений комами`

перелік імен атрибутів>.

Проєкція

57. Web API має підтримувати проєкцію поля, що контролює, яка частина даних об`єкта повертається у відповідь на запит до API. Проєкція поля може зменшити час відгуку та розмір корисного навантаження. Якщо потрібні тільки певні атрибути з отриманих даних, замість URL-адрес слід використовувати параметр запиту проєкції. Параметр запиту має бути сформований таким чином: "fields="<comma-separated list of attributes names (розділений комами список імен атрибутів)>. Параметр запиту проєкції простіший у реалізації та дає змогу отримати декілька атрибутів. Якщо проєкція підтримується, схема XSD/JSON не має застосовуватися у відповіді, оскільки відповідь не буде відповідати оригінальній схемі XSD/JSON.

- Наприклад, наступний запит повертає тільки повне ім'я винахідника запитуваного патенту:

У випадку коли змістова частина у форматі XML на основі стандарту BOIB ST.96

Отримати повне ім'я винахідника патенту з id, що дорівнює id12345:

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/xml
```

Наведено приклад для повідомлення HTTP відповіді:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:Inventor xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:sequenceNumber="String" com:id="ID1"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
      <Contact>
            <Name>
                  <PersonName>
                        <PersonFullName>John Smith</PersonFullName>
                  </PersonName>
            </Name>
      </Contact>
</pat:Inventor>
```

У випадку коли змістова частина у форматі JSON на основі стандарту BOIB ST.97

Отримати повне ім'я винахідника патенту з id[7], що дорівнює id12345:

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/json
```

[7] Common/id.xsd

Наведено приклад для повідомлення HTTP відповіді:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "inventor": {
            "sequenceNumber": "001",
            "Contact": [
                  {
                        "name": {
                              "personName": [
                                    {
                                          "personFullName": "John Smith"
                                    }
                              ]
                        }
                  }
            ]
}}
```

[RSG-78] Параметр запиту МАЄ використовуватися замість шляхів URL у разі, якщо вебінтерфейс підтримує проєкцію у форматі: `"fields="<розділений комами список імен атрибутів>`.

Кількість елементів

58. У деяких випадках користувача API може зацікавити кількість елементів у колекції. Це дуже поширене явище у поєднанні з пагінацією для розуміння загальної кількості елементів у колекції.

- Наприклад, наступний HTTP-запит витягує максимум 3 патентні публікації, пропускаючи перші 4 результати, і повинен також містити у відповіді загальну кількість доступних результатів:

Приклад коли змістова частина XML на основі стандарту BOIB ST.96

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Повертається наступний приклад HTTP-відповіді:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:PatentPublication
xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="de" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
   ...
</pat:PatentPublication>
<pat:PatentPublication>
   ...
</pat:PatentPublication>
   ...
<pat:PatentPublication>
   ...
</pat:PatentPublication>
<count>100</count>
```

Приклад коли змістова частина у форматі JSON на основі стандарту ВОІВ ST.97

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/json
```

Повертається наступний приклад HTTP-відповіді:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "patentPublication": [
            {
            ...
            },
            {
            ...
            },
            {
            ...
            }
      ],
      "count": 100
}
```

59. Як один з варіантів, Web API може підтримувати повернення кількості елементів у колекції в рядку, тобто як частину відповіді, що містить саму колекцію. Крім того, вона може бути частиною конверта метаданих, поза основним тілом відповіді.

[RSG-79] Web API ПОВИНЕН забезпечувати повернення кількості елементів у колекції.

[RSG-80] Параметр запиту ПОВИНЕН використовуватися для забезпечення повернення кількості елементів у колекції.

[RSG-81] МАЄ використовуватися параметр запиту `count` для повернення кількості елементів у колекції.

[RSG-82] Web API МОЖЕ підтримувати повернення кількості елементів у колекції всередині, тобто як частину відповіді, яка містить саму колекцію. При цьому ПОВИНЕН використовуватися параметр запиту.

[RSG-83] МАЄ використовуватися параметр запиту `count=true`. Якщо не вказано, то за замовчуванням для `count` слід встановити значення `false`.

[RSG-84] Якщо Web API підтримує пагінацію, то МАЄ забезпечуватися повернення всередині відповіді розміру колекції (тобто загальної кількості елементів колекції).

Складні пошукові вирази

60. Для отримання даних за кількома критеріями пошуку параметри запиту є достатніми. Якщо існує випадок використання, коли необхідно шукати дані за допомогою складних пошукових виразів (з декількома критеріями, булевими виразами і пошуковими операторами), то API повинен бути розроблений з використанням більш складної мови запитів. Мова запитів повинна підтримуватися граматикою пошуку.

61. Мова контекстних запитів (*CQL - Contextual Query Language*) - це формальна мова для подання запитів до інформаційно-пошукових систем, таких як пошукові системи, бібліографічні каталоги та інформація про музейні колекції. Заснована на семантиці стандарту Z39.50[8] мета його проєктування полягає в тому, що запити повинні бути доступними для читання і запису, а також в тому, щоб мова була інтуїтивно зрозумілою і підтримувала вираження більш складних мов запитів. Це лише один з варіантів, рекомендованих для використання, оскільки він широко використовується в промисловості.

[RSG-85] Якщо Web API підтримує складні пошукові вирази, МАЄ вказуватися мова запитів, наприклад, *CQL.*

[RSG-86] У договорі про надання послуг ПОВИННА бути вказана граматика, що підтримується (наприклад, поля, функції, ключові слова та оператори).

[RSG-87] ПОВИНЕН використовуватися параметр запиту «`q`».

Обробка помилок

62. У відповідях на помилки завжди слід використовувати відповідний код стану HTTP, обраний зі стандартного списку кодів стану HTTP (RFC 7807), наведеного в Додатку V. Якщо сторона, що запитує, очікує JSON, необхідно повернути інформацію про помилку в загальній структурі даних. Якщо проєкт не вимагає іншого, немає необхідності визначати специфічні для програми коди помилок. Трасування стека та інша інформація, пов'язана з налагодженням, не повинна бути присутня у тілі відповіді на помилку у виробничих середовищах.

Змістова частина помилки

63. Обробка помилок здійснюється на двох рівнях: на рівні протоколу (HTTP) і на рівні програми (повертається змістова частина). На рівні протоколу Web API повертає відповідний код стану HTTP, а на рівні додатка Web API повертає корисне навантаження, що повідомляє про помилку з відповідною деталізацією (обов'язкові та необов'язкові атрибути).

[8] Див. розділ «Список використаних джерел»

64. Щодо обов'язкових та необов'язкових атрибутів для обробки помилок на рівні програми,

(a) наступні атрибути `code` і `message` є обов'язковими, і хоча `message` може змінитися в майбутньому, `code` не зміниться; він фіксований і завжди стосуватиметься цієї конкретної проблеми:

- `code` (ціле число) - технічний код ситуації помилки, який буде використовуватися для цілей підтримки; і
- `message` (рядок) - повідомлення для користувача (локалізоване), що описує помилковий запит відповідно до вимог HTTP-заголовка `AcceptLanguage` (див. RSG-114).

(b) Наступні атрибути є умовно обов'язковими:

- `details` - якщо обробка помилок потребує вкладення відповідей на помилки, вона повинна використовувати поле *details* для цієї мети. Поле `details` повинне містити масив об'єктів у форматі JSON, у якому відображаються властивості коду та повідомлення з такою ж семантикою, як описано вище.

(c) Наступні атрибути є необов'язковими:

- `target` - структура повідомлення про помилку може містити атрибут *target,* що описує елемент даних (наприклад, шлях до ресурсу);
- `status` - дублікат коду стану HTTP для поширення його по ланцюжку викликів або для запису в журнал підтримки без необхідності кожного разу явно додавати код стану HTTP;
- `moreInfo` - масив посилань, що містять додаткову інформацію про ситуацію з помилкою, наприклад, давати підказки кінцевому користувачеві; і
- `internalMessage` – технічне повідомлення, наприклад, для ведення журналу.

65. Обробка помилок повинна відповідати стандартам HTTP (RFC 2616). Рекомендується мінімальна змістова частина помилок:

- Наприклад, такі відповіді HTTP повертаються, коли торговельну марку не знайдено для зазначеного номера міжнародної реєстрації:

  Приклад коли змістова частина в XML на основі стандарту BOIB ST.96

```
GET /api/v1/trademarks?irn=000000000000001John%20Smith&expiryDate=2018-12-31.
HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Повертається наступний приклад відповіді HTTP-:

```
HTTP/1.1 404
Content-Type: application/xml
 <?xml version="1.0" encoding="UTF-8"?>
<com:TransactionError xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Common
TransactionError.xsd">
       <com:TransactionErrorCode>TRADEMARK_NOT_FOUND</com:TransactionErrorCode>
       <com:TransactionErrorText>The trademark with the provided International
Registration Number was not found</com:TransactionErrorCode>
</com:TransactionError>
```

Приклад зі змістовою частиною у вигляді JSON на основі стандарту BOIB ST.97

```
HTTP/1.1 404
Content-Type: application/json
{
       "transactionError": [
              {
                     "transactionErrorCode": " TRADEMARK_NOT_FOUND",
              },
              {
                     "transactionErrorText": " The trademark with the provided
International Registration Number was not found"
              },
      ]
 }
```

[RSG-88] На рівні протоколу Web API ПОВИНЕН повертати відповідний код стану HTTP, обраний зі списку стандартних кодів стану HTTP.

[RSJ-89] На рівні додатка Web API ПОВИНЕН повертати частину, що повідомляє про помилку з достатнім ступенем деталізації. Атрибути `code` і `message` є обов'язковими, атрибут `details` є умовно обов'язковим, атрибути `target`, `status`, `moreInfo` і `internalMessage` є необов'язковими.

[RSG-90] Помилки НЕ ПОВИННІ розкривати критично важливі для безпеки дані або внутрішні технічні деталі, такі як стеки викликів, у повідомленнях про помилки.

[RSG-91] Заголовок HTTP *`Header: Reason-Phrase`* (описаний у RFC 2616) НЕ ПОВИНЕН використовуватися для передачі повідомлень про помилки.

Ідентифікатор кореляції

66. Зазвичай споживання послуги призводить до каскадного запуску декількох інших послуг. Повинен існувати механізм для кореляції всіх активацій послуг в одному контексті виконання. Наприклад, включення ідентифікатора кореляції в повідомлення журналу, оскільки це однозначно ідентифікує зареєстровану помилку. Слід використовувати ім'я заголовка. Наприклад, зазвичай використовуються *Request-ID* або *Correlation-ID*, тому що врахування цього фактора на етапі розроблення API буде сприяти сумісності між різними API і новими реалізаціями.

[RSG-92] Кожна зареєстрована помилка МАЄ отримати унікальний ідентифікатор кореляції. МАЄ використовуватися користувацький HTTP заголовок, який МАЄ отримати назву *Correlation-ID*.

Сервісний договір

67. REST - це не протокол і не архітектура, а архітектурний стиль з архітектурними властивостями та архітектурними обмеженнями. Офіційних стандартів для договорів REST API не існує. У цьому стандарті документація API називається Договором послуг REST. Сервісний договір ґрунтується на таких трьох фундаментальних елементах:

(a) Синтаксис ідентифікатора ресурсу - як ми можемо описати, куди або звідки передаються дані?

(b) Методи - які протокольні механізми використовуються для передачі даних?

(c) Типи носіїв - який тип даних передається? Окремі сервіси REST використовують ці елементи в різних комбінаціях для розкриття своїх можливостей. Визначення основного набору цих елементів для використання колекцією (або переліком) сервісів робить цей тип сервісного договору «уніфікованим».

[RSG-93] У формат сервісного договору НЕОБХІДНО включати таке:

- Версія API;
- Інформація про семантику елементів API;
- Ресурси;
- Атрибути ресурсу;
- Параметри запиту;
- Методи;
- Типи носіїв;
- Граматика пошуку (якщо підтримується);
- Коди стану HTTP;
- Методи HTTP;
- Обмеження та відмінні особливості; і
- Безпека (наприклад, приватні схеми).

[RSG-94] У формат сервісного договору МАЮТЬ бути включені запити і відповіді в схемі XML- або JSON та приклади використання API в підтримуваних форматах, тобто XML або JSON.

[RSG-95] REST API ПОВИНЕН надавати документацію API у вигляді сервісного договору.

[RSG-96] Реалізація Web API, що відхиляється від цього стандарту, ПОВИННА бути чітко задокументована в договорі про надання послуг. Якщо в договорі про надання послуг не вказано правило відхилення, ПОВИННО вважатися, що дотримується цей Стандарт.

[RSG-97] Сервісний договір ПОВИНЕН дозволяти генерувати скелетний код клієнта API.

[RSG-98] Сервісний договір МАЄ дозволяти генерацію скелетного коду сервера.

68. Документація Web API може бути написана, наприклад, мовою моделювання RESTful API Modeling Language (RAML), Open API Specification (OAS) та WSDL. Оскільки лише RAML повністю підтримує валідацію запитів/відповідей у форматах XML та JSON (за допомогою

схем XSD та JSON), цей Стандарт рекомендує використовувати RAML[9].

[RSG-99] Документація Web API МАЄ бути написана на RAML або OAS. Користувацькі формати документації НЕ МАЄ використовуватися.

Час очікування

69. Відповідно до принципів сервіс-орієнтованого проєктування, використання сервера має бути обмежене.

[RSG-100] Споживач Web API МАЄ мати можливість вказати тайм-аут сервера для кожного запиту; МАЄ використовуватися користувацький HTTP-заголовок. Максимальний час очікування сервера МАЄ також застосовуватися для захисту ресурсів сервера від надмірного використання.

Управління станом

70. Якщо розробка здійснюється відповідно до принципів REST, управління станом повинно здійснюватися на стороні клієнта, а не на сервері, оскільки REST API не має стану. Наприклад, якщо декілька серверів реалізують сесію, реплікація не повинна бути рекомендована.

Версіонування відповідей

71. Багаторазове отримання одного і того ж набору даних може призвести до споживання пропускної здатності, якщо набір даних не був змінений між запитами. Умовно витягувати дані слід лише в тому випадку, якщо вони не були змінені. Це можна зробити за допомогою перевірки ресурсів на основі вмісту або перевірки ресурсів на основі часу. Якщо використовується версіонування відповідей, споживач послуг може реалізувати оптимістичне блокування.

[RSG-101] Web API МАЄ підтримувати умовне отримання даних, щоб гарантувати, що будуть отримані тільки ті дані, які були змінені. МАЄ застосовуватися перевірка ресурсів на основі вмісту, оскільки вона точніша.

[RSG-102] Для реалізації Content-based Resource Validation (перевірка коректності ресурсу за вмістом) HTTP-заголовок `ETag` МАЄ використовуватися у відповіді для кодування стану даних. Потім це значення МАЄ використовуватися в наступних запитах в умовних HTTP-заголовках (таких як *If-Match* або *If-None-Match* – Якщо відповідає або Якщо не відповідає). Якщо дані не були змінені відтоді, як запит повернув `ETag`, сервер МАЄ повернути код стану «`304 Not Modified`» (якщо дані не змінені). Цей механізм описаний в IETF RFC 7231 і 7232.

[RSG-103] Для реалізації перевірки ресурсів за часом МАЄ використовуватися HTTP-заголовок `Last-Modified`. Цей механізм описаний в IETF RFC 7231 і 7232.

[RSG-104] Використовуючи версифікацію відповідей, споживач послуг МОЖЕ реалізувати оптимістичне блокування.

Кешування

[9] OAS - це специфікація. Вона також підтримує Markdown, а RAML - ні. З іншого боку, хоча і OAS, і RAML підтримують перевірку схеми JSON для запитів і відповідей, OAS не підтримує XSD. Тому в майбутньому, коли OAS буде повнофункціональним, його можна буде рекомендувати.

72. Реалізація Web API має підтримувати роботу з кешем для економії пропускної здатності, відповідно до IETF RFC 7234.

[RSG-105] Web API ПОВИНЕН підтримувати кешування результатів `GET`; Web API МОЖЕ підтримувати кешування результатів і інших методів HTTP.

[RSG-106] МАЮТЬ використовуватись HTTP-заголовки відповіді `Cache-Control` і `Expires`. Останній МОЖЕ використовуватися для підтримки старих клієнтів.

Керування передачею файлів

73. Передача (тобто завантаження або вивантаження) великих файлів має високу ймовірність викликати переривання мережі або інші збої в передачі. Вона також споживає великий обсяг пам'яті як для постачальника, так і для споживача послуг. Тому рекомендується передавати великі файли кількома фрагментами з декількома запитами. Ця опція також дає змогу відстежувати загальний прогрес завантаження або вивантаження. Часткова передача великих файлів має відновити підтримку. Постачальник послуг має повідомити, чи підтримує він часткову передачу великих файлів.[10].

74. Існує два підходи для реалізації цього типу передачі: перший - використання заголовка `Transfer-Encoding: chunked` і другий - використання заголовка `Content-Length`. Ці заголовки не слід використовувати разом. `Content-Length` вказує на повний розмір файлу, що передається, тому одержувач знатиме довжину тіла і зможе оцінити час завершення завантаження. Заголовок `Transfer-Encoding: chunked` корисний для необмеженої потокової передачі обмежених даних, таких як аудіо чи відео, але не файлів. Рекомендується використовувати заголовок `Content-Length` для скачування, оскільки завантаження сервера є низьким порівняно з `Transfer-Encoding: chunked`. Для завантаження рекомендується використовувати заголовок `Transfer-Encoding: chunked`.

Web-API повинен повідомити, чи підтримує він часткове завантаження файлів, відповідаючи на запити *HEAD* і передаючи у відповідь заголовки HTTP-відповідей: *Accept-Ranges і Content-Length*. Перший повинен вказувати одиницю виміру, яку можна використовувати для визначення діапазону, і має ніколи не бути визначеним як *'none'*. Останній вказує повний розмір завантажуваного файлу.

[RSG-107] Web API МАЄ повідомляти, чи підтримується часткове завантаження файлів, відповідаючи на запити `HEAD` і передаючи у відповідь HTTP-заголовки `AcceptRanges` *і* `Content-Length`.

75. Web API, що підтримує завантаження великих файлів, має підтримувати часткові запити згідно з IETF RFC 7232, тобто:

- Споживач послуг, який запитує діапазон, повинен використовувати HTTP-заголовок `Range`;
- Відповідь постачальника послуг повинна містити HTTP-заголовки `Content-Range` і `Content-Length`; і

[10] Постачальник послуг може повернути розташування файлу, і тоді споживач послуг може зателефонувати до служби каталогів, щоб завантажити файл. Врешті-решт, потрібне часткове завантаження файлу. У цьому параграфі не розглядаються не-REST протоколи, такі як FTP, sFTP або rsync.

- У разі успішного запиту діапазону відповідь постачальника послуг повинна мати HTTP- статус `206 Partial Content`. У разі запиту діапазону, який виходить за межі (значення діапазону перекривають обсяг ресурсу), сервер відповідає *«*`416 Requested Range Not Satisfiable`*»*. У разі якщо запит діапазону не підтримується, сервер посилає у відповідь статус *«*`200 OK`*».*

  [RSG-108] Web API МАЄ підтримувати часткове завантаження файлів. МАЮТЬ підтримуватися діапазони, що складаються з декількох частин.

76. Діапазони з декількох частин також можуть бути запитані, якщо HTTP-заголовок `Content-Type: multipart/byteranges; boundary=XXXXX`. Запит діапазону може бути умовним, якщо він поєднується з HTTP-заголовками `ETag` або `IfRange`.

77. Не існує IETF RFC для заливки великих файлів. Тому в цьому стандарті не надається жодних рекомендацій щодо завантаження великих файлів.

> [RSG-109] Web API МАЄ повідомляти про те, що він підтримує часткове завантаження файлів.
>
> [RSG-110] Web API МАЄ підтримувати часткове завантаження файлів. Повинні підтримуватися діапазони, що складаються з декількох частин.

78. IETF RFC 2616 не накладає жодних обмежень на розмір запитів. У договорі про надання послуг API має бути зазначено максимальний ліміт для запитів. Крім того, під час виконання постачальник послуг має вказувати споживачеві послуг, чи було перевищено дозволений максимальний ліміт.

> [RSG-111] Постачальник послуг МАЄ повертати в заголовках відповіді HTTP-заголовок *«*`413 Request Entity Too Large`*»* у разі, якщо запит перевищив максимально допустимий ліміт. Користувацький HTTP-заголовок МОЖЕ використовуватися для зазначення максимального розміру запиту.

Обробка налаштувань

79. Постачальник послуг може дозволити споживачеві послуг налаштовувати значення та впливати на те, як перший обробляє запити другого. Стандартні засоби для реалізації обробки налаштувань викладені в IETF RFC 7240.

> [RSG-112] Якщо Web API підтримує обробку налаштувань, вона МАЄ бути реалізована згідно з IETF RFC 7240, тобто МАЄ використовуватись заголовок HTTP-запиту `Prefer`, у HTTP-заголовку відповіді НЕОБХІДНО повернути `Preference-Applied` (з повторенням вихідного запиту).
>
> [RSG-113] Якщо Web API підтримує обробку налаштувань, номенклатура налаштувань, які МОЖУТЬ бути встановлені за допомогою заголовка `Prefer`, ПОВИННА бути записана в сервісному договорі.

Переклад

80. Споживач послуги може запросити відповіді певною мовою, якщо постачальник послуги підтримує її. Стандартну специфікацію для обробки набору природних мов викладено в IETF TFC 7231.

[RSG-114] Якщо Web API підтримує локалізовані дані, HTTP-заголовок запиту `Accept-Language` ПОВИНЕН підтримуватися для зазначення набору природних мов, яким надається перевага у відповіді, як зазначено в IETF RFC 7231.

Довготривалі операції

81. Існують випадки, коли Web API може бути пов'язаний з тривалими операціями. Наприклад, створення PDF-файлу постачальником послуг може зайняти кілька хвилин. У цьому параграфі рекомендується типовий шаблон обміну повідомленнями для реалізації таких випадків, наприклад:

```
// (a)
GET https://wipo.int/api/v1/patents
Accept: application/pdf
…
// (b)
HTTP/1.1 202 Accepted
Location: https://wipo.int/api/v1/queues/12345
…
// (c1)
GET https://wipo.int/api/v1/queues/12345
…
HTTP/1.1 200 OK
…
// (c2)
GET https://wipo.int/api/v1/queues/12345
HTTP/1.1 303 See Other
Location: https://wipo.int/api/v1/path/to/pdf
…
// (c3)
GET https://wipo.int/api/v1/path/to/pdf
…
```

82. Якщо API підтримує довготривалі операції, то вони повинні виконуватися асинхронно, щоб користувач не був змушений чекати на відповідь. У наведеному нижче правилі викладено рекомендований підхід до реалізації

[RSG-115] Якщо API підтримує тривалі операції, вони МАЮТЬ бути асинхронними. МАЄ застосовуватися такий підхід:

a) Споживач послуги активує сервісну операцію;

b) Операція сервісу повертає код стану *«202 Accepted»* згідно з IETF RFC 7231 (розділ 6.3.3), тобто запит було прийнято до обробки, але обробка не була завершена. Розташування створеного завдання в черзі також повертається з HTTP-заголовком Location; a

c) Споживач послуги звертається до повернутого *Location* (розташування), щоб дізнатися, чи доступний ресурс. Якщо ресурс недоступний, у відповіді МАЄ бути код стану *«200 OK»,* міститися статус завдання (наприклад, очікування) і МОЖЕ міститись інша інформація (наприклад, індикатор виконання та/або посилання для скасування або видалення задачі за допомогою HTTP-методу *DELETE*). Якщо ресурс доступний, у відповіді МАЄ бути код стану *«303 See Other»,* а HTTP-заголовок *Location* МАЄ містити URL-адресу, за якою можна отримати результати завдання.

Модель безпеки

Загальні правила

83. У межах цього стандарту безпека API пов'язана з основними атрибутами безпеки, що забезпечують безпеку інформації, доступної через API, і самих API протягом усього їх життєвого циклу. Цими атрибутами є конфіденційність, цілісність, доступність, довіра, безвідмовність, поділ, автентифікація, авторизація та аудит.

[RSG-116] Конфіденційність: Різні API та інформація в API ПОВИННІ бути ідентифіковані, класифіковані та захищені від несанкціонованого доступу, розкриття та перехоплення в будь-який час.Повинні дотримуватися[11] принципів найменших привілеїв, нульової довіри, необхідності знати та необхідності ділитися.

[RSG-117] Забезпечення цілісності: Різні API та інформація в API ПОВИННІ бути захищені від несанкціонованої модифікації, дублювання, ушкодження та знищення. Інформація ПОВИННА модифікуватися через підтверджені транзакції та інтерфейси. Системи ПОВИННІ оновлюватися з використанням затверджених процесів управління конфігурацією, управління змінами та управління виправленнями.

[RSG-118] Доступність: Різні API та інформація в API ПОВИННІ бути доступні авторизованим користувачам у встановлений час згідно з угодами про рівень обслуговування (SLA), політиками контролю доступу та визначеними бізнес-процесами.

[RSG-119] Безвідмовність (не допускають відмов): Кожна оброблювана транзакція або дія, що виконується API, ПОВИННІ забезпечувати не допущення відмов за допомогою реалізації належного аудиту, авторизації, автентифікації, а також реалізації безпечних шляхів, а також сервісів і механізмів, що не допускають відмов.

[RSG-120] Автентифікація, авторизація, аудит: Користувачі, системи, API або пристрої, залучені до критичних транзакцій або дій, ПОВИННІ бути автентифіковані, авторизовані за допомогою служб контролю доступу на основі ролей або атрибутів і підтримувати поділ обов'язків. Крім того, всі дії ПОВИННІ реєструватися, а надійність автентифікації повинна збільшуватися з відповідним інформаційним ризиком.

Рекомендації щодо безпечного та стійкого до загроз управління API

84. API слід проєктувати, створювати, тестувати та впроваджувати з урахуванням вимог безпеки та ризиків. Відповідні заходи протидії та засоби контролю мають бути вбудовані безпосередньо в проєкт, а не розглядатися після його завершення. Рекомендується використовувати найкращі практики та стандарти, такі як OWASP.

[RSG-121] При розробці API ПОВИННІ ретельно враховуватися загрози, зловмисні випадки використання, безпечні методи кодування, безпеку транспортного рівня та тестування безпеки, особливо:

- `PUTs` і `POSTs` - тобто: які зміни внутрішніх даних потенційно можуть бути використані для атаки або неправдивої інформації;
- `DELETES` - тобто: може використовуватися для видалення вмісту внутрішнього сховища ресурсів;

[11] https://wiki.owasp.org/index.php/Security_by_Design_Principles

- Білий список допустимих методів - щоб гарантувати, що дозволені HTTP-методи належним чином обмежені, в той час як інші повертатимуть правильний код відповіді; і
- Добре відомі атаки повинні бути розглянуті на етапі моделювання загроз в процесі проєктування, щоб гарантувати, що ризик загроз не збільшиться. Загрози та способи їх усунення, визначені в OWASP Top Ten Cheat Sheet[12], ПОВИННІ бути взяті до уваги.

[RSG-122] Під час розроблення API слід дотримуватися стандартів і Кращих практик, перелічених нижче:

- Кращі практики безпечного кодування: Принципи безпечного кодування OWASP;
- Безпека Rest API: Пам'ятка з безпеки REST;
- Захист від несанкціонованого доступу та міжсайтового скриптінгу: OWASP XSS Cheat Sheet;
- Запобігання SQL-ін'єкцій: Пам'ятка OWASP щодо SQL-ін'єкцій, пам'ятка OWASP щодо Parameterization; і
- Безпека транспортного рівня: Пам'ятка OWASP щодо захисту транспортного рівня.

[RSG-123] Тестування безпеки та оцінка вразливостей ПОВИННІ проводитися для забезпечення безпеки та стійкості API до загроз. Ця вимога МОЖЕ бути виконана шляхом використання статичного та динамічного тестування безпеки додатків (SAST/DAST), автоматизовані інструменти управління вразливостями та тестування на проникнення.

Шифрування, цілісність і безвідмовність

85. Захищені сервіси повинні бути захищені для захисту облікових даних автентифікації під час передачі: наприклад, паролів, ключів API або JSON-Web Tokens. Також має бути гарантована цілісність даних, які передаються, і незаперечення вчинених дій. Безпечні криптографічні механізми можуть забезпечити конфіденційність, шифрування, гарантію цілісності та безвідмовність. Цілковита цілковита пряма секретність (*perfect forward security* - один з алгоритмів безпеки) є одним із засобів забезпечення неможливості компрометації сеансових ключів.

[RSG-124] Захищені служби ПОВИННІ надавати тільки кінцеві точки HTTPS, що використовують TLS 1.2 або вище, з набором шифрів, що включають *ECDHE* для обміну ключами.

[RSG-125] При розгляді протоколів автентифікації для забезпечення транспортної безпеки МАЄ використовуватися цілковита пряма секретність. НЕ МАЄ допускатися використання небезпечних криптографічних алгоритмів і зворотну сумісність із SSL 3 і TLS 1.0/1.1.

[RSG-126] Для забезпечення максимальної безпеки та довіри МАЄ встановлюватися з'єднання через VPN IPSEC між сайтами для додаткового захисту інформації, що передається через незахищені мережі.

[RSG-127] Користувацький додаток МАЄ перевіряти ланцюжок сертифікатів TLS під час виконання запитів до захищених ресурсів, включно з перевіркою списку відкликання

[12] https://owasp.org/www-project-top-ten/2017/

сертифікатів.

[RSG-128] У захищених сервісах МАЄ використовуватися тільки дійсні сертифікати, видані довіреним центром сертифікації (ЦС).

[RSG-129] Токени МАЮТЬ підписуватися з використанням безпечних алгоритмів підпису, що відповідають стандарту цифрового підпису (DSS) FIPS-186-4. МАЮТЬ розглядатися алгоритм цифрового підпису RSA або алгоритм ECDSA.

Автентифікація та авторизація

86. Авторизація - це акт виконання контролю доступу до ресурсу. Авторизація охоплює не лише застосування засобів контролю доступу, але й визначення цих засобів. Це включає в себе правила і політики доступу, які повинні визначати необхідний рівень доступу, прийнятний як для провайдера, так і для програми-споживача. Основою контролю доступу є надання або відмова провайдером у доступі додатку-споживачу та/або споживачеві до ресурсу з певним рівнем деталізації. Грубий доступ слід розглядати на рівні API або точки запиту шлюзу API, в той час як тонкий контроль слід розглядати на рівні внутрішньої служби, якщо це можливо. Можна розглянути модель управління доступом на основі ролей (RBAC) або управління доступом на основі атрибутів (ABAC).

87. Якщо сервіс захищений, то має віддаватися перевага Open ID Connect перед OAuth 2.0, оскільки він заповнює багато прогалин останнього і забезпечує стандартизований спосіб отримання даних профілю власника ресурсу, стандартизований формат токенів JSON Web Token (JWT) і криптографію. Не мають використовуватися інші схеми безпеки, такі як базова авторизація HTTP, яка вимагає, щоб клієнт зберігав десь пароль у вигляді відкритого тексту і надсилав його разом з кожним запитом. Крім того, перевірка цього пароля буде повільнішою, оскільки він повинен мати доступ до сховища облікових даних. OAuth 2.0 не визначає токен безпеки. Тому токен JWT має використовуватись в порівнянні, наприклад, з SAML 2.0, який є більш багатослівним.

[RSG-130] Анонімна автентифікація ПОВИННА використовуватися тільки тоді, коли клієнти і додаток, який вони використовують, отримують доступ до інформації або функцій з низьким рівнем чутливості, які не повинні вимагати автентифікації, наприклад, до публічної інформації.

[RSG-131] НЕ ПОВИННА дозволятися автентифікація з використанням імені користувача та пароля або хешу пароля.

[RSG-132] Якщо сервіс захищений, МАЄ використовуватися *Open ID Connect*.

[RSG-133] Якщо використовується JSON *Web Token (JWT),* секретний JWT МАЄ володіти високою ентропією, щоб збільшити коефіцієнт роботи атаки грубої сили; токени TTL і RTTL повинні бути якомога коротшими; і конфіденційна інформація НЕ МАЄ зберігатися в корисному навантаженні JWT.

88. Загальним вибором під час проєктування системи безпеки є централізація автентифікації користувачів. Вона має зберігатися в *Identity Provider* (провайдерах ідентифікації - IdP) або локально в кінцевих точках REST.

89. Сервіси мають ретельно стежити за тим, щоб запобігти витоку облікових даних. Паролі, маркери безпеки та ключі API не повинні з'являтися в URL, оскільки це може бути

зафіксовано в журналах вебсервера, що робить їх цінними за своєю суттю. Наприклад, наступне є некоректним (API-ключ в URL): https://wipo.int/api/patents?apiKey=a53f435643de32.

[RSG-134] У запитах `POST/PUT` конфіденційні дані МАЮТЬ передавати в тілі або в заголовках запиту.

[RSG-135] У запитах `GET` конфіденційні дані СЛІД передавати в заголовку HTTP.

[RSG-136] Для того щоб мінімізувати затримки і зменшити зв'язок між захищеними сервісами, рішення про управління доступом МАЄ прийматися на локальному рівні в кінцевих точках REST.

90. Автентифікація за API-ключами: API-ключі мають використовуватися там, де потрібна автентифікація між системами, і вони мають генеруватися автоматично і випадковим чином. Ризик, притаманний цьому режиму автентифікації, полягає в тому, що будь-хто, хто має копію ключа API, може використовувати його так, ніби він є легітимним додатком-споживачем. Отже, всі комунікації повинні відповідати RSG-124, щоб захистити ключ під час передачі. Розробник програми зобов'язаний належним чином захистити свою копію ключа API. Якщо ключ API вбудований у програму, яка його використовує, його можна декомпілювати та витягти. Якщо ж він зберігається у вигляді звичайного текстового файлу, його можна викрасти і повторно використати у зловмисних цілях. Тому ключ API повинен бути захищений сховищем облікових даних або механізмом секретного управління. Ключі API можуть використовуватися для контролю використання послуг, навіть для загальнодоступних послуг.

[RSG-137] API-ключі МАЮТЬ використовуватися для захищених і загальнодоступних сервісів, щоб запобігти перевантаженню постачальника послуг численними запитами (атакам на відмову в обслуговуванні). Для захищених сервісів ключі API МОЖНА використовувати для монетизації (придбаних тарифних планів), забезпечення дотримання політики використання (QoS) та моніторингу.

[RSG-138] Ключі API МОЖУТЬ бути об'єднані із заголовком HTTP запиту *user-agent* для розрізнення користувача-людини та програмного агента, як зазначено в IETF RFC 7231.

[RSG-139] Сервіс-провайдер МАЄ повертати разом із заголовками відповіді HTTP поточний статус використання. Наступні дані відповіді МОЖУТЬ бути повернуті:

- ліміт швидкості (*rate limit*) - ліміт швидкості (за хвилину), встановлений в системі;
- ліміт швидкості, що залишився (*rate limit remaining*) - залишкова кількість запитів, дозволених протягом поточного часового інтервалу (-1 означає, що ліміт перевищено); та
- скидання ліміту швидкості (*rate limit reset*) - час (у секундах), що залишився до скидання лічильника запитів.

[RSG-140] Сервіс-провайдер МАЄ повертати код стану «*`429 Too Many Requests`*», якщо запити надходять занадто швидко.

[RSG-141] Ключі API ПОВИННІ бути відкликані, якщо клієнт порушує угоду про використання, як зазначено в ВІВ (Відомстві інтелектуальної власності).

[RSG-142] Ключі API МАЮТЬ передаватися за допомогою користувацьких HTTP-заголовків. Вони НЕ МАЮТЬ передаватися за допомогою параметрів запиту.

[RSG-143] Ключі API МАЮТЬ генеруватися випадковим чином.

91. Хоча використання криптографії з відкритим ключем і сертифікатів пов'язане з накладними витратами, взаємну автентифікацію на основі сертифікатів слід використовувати у тих випадках, коли для забезпечення додаткової безпеки Web API потрібна більш надійна автентифікація, ніж та, яка забезпечується ключами API. Безпечні та надійні сертифікати мають бути видані взаємно довіреним центром сертифікації (ЦС) у процесі встановлення довіри або перехресної сертифікації. Для зниження ризиків безпеки ідентифікації, характерних для чутливих систем і привілейованих дій, можна використовувати сувору автентифікацію. Мають використовуватися загальні для клієнта і сервера сертифікати, наприклад, X.509.

[RSG-144] Безпечні та довірені сертифікати ПОВИННІ видаватися взаємно довіреним центром сертифікації (ЦС) через процес встановлення довіри або перехресної сертифікації.

[RSG-145] Сертифікати, спільно використовувані клієнтом і сервером, МАЮТЬ використовуватися для зниження ризиків безпеки ідентифікації, характерних для чутливих систем і привілейованих дій, наприклад, X.509.

[RSG-146] Для високопривілейованих служб у двосторонній взаємній автентифікації між клієнтом і сервером МАЮТЬ використовуватися сертифікати для забезпечення додаткового захисту.

[RSG-147] Для високоризикових додатків, систем, що обробляють дуже чутливу інформацію або виконують привілейовані дії, МАЄ реалізовуватися багатофакторна автентифікація для зниження ризиків ідентифікації.

Доступність і захист від загроз

92. Доступність у цьому контексті охоплює захист від загроз, щоб мінімізувати час простою API, розглядаючи, як можна зменшити загрози для відкритих API, використовуючи базові принципи проєктування. Доступність також охоплює масштабування для задоволення попиту, забезпечення стабільності хостингових середовищ тощо. Ці рівні доступності розглядаються в стеках апаратного та програмного забезпечення, які підтримують надання API. Зазвичай доступність розглядається в рамках стандартів безперервності бізнесу та аварійного відновлення, які рекомендують підхід до оцінки ризиків для визначення вимог до доступності.

Міждоменні запити

93. Деякі «міждоменні» запити, зокрема *Ajax*-запити, за замовчуванням заборонені політикою безпеки одного і того ж джерела. Відповідно до політики єдиного джерела веббраузер дозволяє сценаріям, що містяться на першій вебсторінці, доступ до даних на другій вебсторінці, тільки якщо обидві вебсторінки мають однакове походження (тобто поєднання схеми URI, імені хоста і номера порту).

94. Міждоменний обмін ресурсами (*CORS - Cross-Origin Resource Sharing*) - це стандарт W3C, що дозволяє гнучко визначати, які міждоменні запити дозволені. Передаючи відповідні

HTTP-заголовки міждоменного обміну, ваш REST API сигналізує браузеру, яким доменам або джерелам дозволено виконувати *JavaScript*-запити до REST- сервісу.

95. JSON з доповненням (*JSONP -JSON with padding*) - це метод відправки даних JSON, який не має проблем міждоменних запитів. Він вводить функції зворотного виклику для завантаження даних JSON з різних доменів. Ідея методу базується на тому, що тег HTML `<script>` не піддається впливу політики походження. Все, що імпортується через цей тег, негайно виконується в глобальному контексті. Замість того щоб передавати файл *JavaScript*, можна передати URL-адресу служби, яка повертає код *JavaScript*.

96. Для обходу цього обмеження зазвичай використовуються наступні підходи:

- JSONP - це обхідний шлях для міждоменних запитів. Він не пропонує жодного механізму виявлення помилок, тобто якщо виникла проблема, і сервіс не спрацював або відповів HTTP-помилкою, немає способу визначити, у чому була проблема зі сторони клієнта. У результаті AJAX-додаток просто «зависне». Більш того, сайт, що використовує JSONP, буде беззастережно довіряти JSON, наданому з іншого домену;
- *Iframe* - це альтернативне обхідне рішення для міждоменних запитів. Використовуючи метод *JavaScript window.postMessage (*`message`, `targetOrigin`*)* на об'єкті *iframe*, можна передати запит на сайт іншого домену. Підхід *iframe* має хорошу сумісність навіть у старих браузерах. Крім того, він підтримує тільки *GET*. Джерело сторінки *Iframe* завжди повинен бути перевіреним через питання безпеки; і
- CORS - це стандартизований підхід для виконання виклику до зовнішнього домену. Він може використовувати *XMLHttpRequest* для надсилання та отримання даних і має кращий механізм обробки помилок, ніж JSONP. Він підтримує безліч типів авторизації порівняно з JSONP, який підтримує тільки файли cookie.. Він також підтримує HTTP-методи порівняно з JSONP, який підтримує тільки `GET`. З іншого боку, не завжди можливо реалізувати CORS, оскільки браузери повинні його підтримувати, а споживачі API повинні бути внесені до білого списку CORS.

[RSG-148] Якщо REST API є загальнодоступним,, HTTP-заголовок *Access-Control-Allow-Origin* ПОВИНЕН мати значення '*'.

[RSG-149] Якщо REST API захищений, СЛІД використовувати CORS, якщо це можливо. В іншому випадку, як запасний варіант МОЖЕ використовуватися JSONP, але тільки для GET-запитів, наприклад, коли користувач отримує доступ за допомогою старого браузера. *Iframe* НЕ МАЄ використовуватись.

Модель зрілості API

97. Зазвичай REST API класифікують за моделлю зрілості. Хоча існують різні моделі, цей Стандарт посилається на модель зрілості Річардсона (Richardson Maturity Model, RMM). RMM визначає три рівні, і цей Стандарт рекомендує рівень 2 для REST API, оскільки рівень 3 є складним для реалізації і вимагає значних концептуальних інвестицій та інвестицій, пов'язаних з розробкою, від постачальників і споживачів послуг. У той же час, він не приносить негайної вигоди споживачам послуг.

98. Якщо Web API реалізує рівень 3 RMM, необхідно впровадити гіпермедійний формат.

Hypertext Application Language (HAL) [13] простий і сумісний із відповідями JSON і XML. Однак це лише проєкт рекомендації, поряд з іншими гіпермедійними форматами, такими як JSON-LD[14]. Має використовуватися JSON-Schema[15] оскільки, хоча наразі не існує специфікації для рівня 3 RMM, він вважається найбільш зрілим. Не слід розглядати такі формати гіпермедіа: IETF RFC 5988 та `Collection+JSON`.

99. Рекомендується, щоб в екземплярах, описаних схемою, надавалися посилання на завантажувану JSON-схему, з використанням зв'язку «`describedby`», як визначено в протоколі *Linked Data Protocol 1.0*, розділ 8.1 [W3C.REC-ldp-20150226][16].

У HTTP такі посилання можуть бути прикріплені до будь-якої відповіді за допомогою заголовка `Link` [RFC8288]. Прикладом такого заголовка може бути:

```
Link: <http://example.com/my-hyper-schema#>; rel="describedby"
```

[RSJ-150] Якщо використовуються екземпляри, що описують схему, для надання посилання на схему JSON, яку можна завантажити згідно з RFC8288, МАЄ використовуватися заголовок `Link`.

[RSJ-151] Web API МАЄ впровадити принаймні рівень 2 (властивості, притаманні транспорту) RMM. Рівень 3 (Гіпермедіа) МОЖЕ бути реалізований, щоб зробити API повністю інформативним.

100. Може бути розроблений власний формат гіпермедіа. У такому випадку рекомендується набір атрибутів. Наприклад:

```
{
   "link": {
      "href": "/patents",
      "rel": "self"
   },
   ...
 }
```

[RSJ-152] Для створення власного гіпермедійного формату МАЄ використовуватися наступний набір атрибутів, вкладений у посилання на атрибут:

- `href` - цільовий URI;
- `rel` - значення цільового URI;
- `self` - URI посилається на сам ресурс;
- `next` - URI посилання на попередню сторінку (якщо використовується при пагінації);
- `previous` - URI посилання на наступну сторінку (якщо використовується при пагінації); і
- довільне ім'я `v` позначає користувацьке значення відношення.

SOAP WEB-API

101. Цей стандарт рекомендує архітектурний стиль REST як найкращий підхід до розробки API. Архітектури RESTful, як правило, простіше розробляти, розширювати та інтегрувати, ніж

[13] https://tools.ietf.org/html/draft-kelly-json-hal-08t
[14] https://www.w3.org/TR/json-ld/
[15] https://json-schema.org/specification.html#specification-documents
[16] http://json-schema.org/latest/json-schema-core.html#hypermedia

SOAP. Для повноти викладу тут розглянуто SOAP; приклади та варіанти використання не наводяться.

102. SOAP Web API - це програмний застосунок, ідентифікований URI, інтерфейси та прив'язка якого можуть бути визначені, описані та виявлені за допомогою XML- сутностей. Він також підтримує пряму взаємодію з іншими програмними додатками, використовуючи повідомлення на основі XML, через інтернет-протоколи, такі як SOAP і HTTP.

103. Контракт на основі SOAP описується мовою визначення вебсервісів (WSDL), стандартним документом W3C. У цьому документі «Документ WSDL для контракту на вебсервіс» буде згадуватися як «WSDL».

104. При створенні вебсервісів існує два стилі розроблення: контракт в останню чергу і контракт у першу чергу. При застосуванні підходу «контракт-останній» ви починаєте з програмного коду і контракт вебсервісу генерується на його основі. При застосуванні підходу контракт-перший ви починаєте з контракту WSDL і використовуєте код для реалізації цього контракту.

Загальні правила

105. Профіль взаємодії вебслужб (WS-I) є одним із найважливіших стандартів щодо API-інтерфейсів на основі SOAP, і він забезпечує мінімальну основу для написання вебсервісів, які можуть працювати разом. WS-I надає керівництво по тому, як сервіси «відкриваються» один одному і як вони передають інформацію (це називається «обмін повідомленнями»). Це профіль для реалізації конкретних версій деяких із найважливіших стандартів вебслужб, таких як WSDL, SOAP, XML тощо. Приєднання до певних профілів неявно вказує на приєднання до конкретних версій цих стандартів вебсервісів. *WS-I Basic Profile v1.1* надає настанови щодо використання XML 1.0, HTTP 1.1, UDDI, SOAP 1.1, WSDL 1.1 і UDDI 2.0. *WS-I Basic Profile 2.0* надає посібник із використання SOAP 1.2, WSDL 1.1, UDDI 2.0, WS-Addressing і MTOM. SOAP 1.2 забезпечує чітку модель обробки і веде до кращої функціональної сумісності. WSDL 2.0 було розроблено для розв'язання проблем сумісності, виявлених у WSDL 1.1, шляхом використання вдосконалених прив'язок SOAP 1.2.

[WS-01] Всі WSDL ПОВИННІ відповідати WS-I Basic Profile 2.0. МОЖЕ використовуватися WSDL 1.2.

106. Прив'язка WSDL SOAP може бути або прив'язкою в стилі віддаленого виклику процедур (RPC), або прив'язкою в стилі документа. Прив'язка SOAP також може мати кодоване застосування або буквальне застосування. Прив'язка надає п'ять моделей стилю/використання: RPC/encoded, RPC/literal, document/encoded, document/literal, document/literal wrapped (RPC/закодований, RPC/літеральний, документ/закодований, документ/літеральний, документ/літеральний обгорнутий).

[WS-02] Сервіси ПОВИННІ використовувати прив'язку в стилі документа і моделі літерального використання (або document/literal, або document/literal wrapped). За наявності графів, ПОВИНЕН використовуватися стиль *RPC/encoded.*

[WS-03] Коли існують виняткові випадки використання, наприклад, коли у WSDL є перевантажені операції, МАЮТЬ застосовуватися всі інші стилі.

107. Для забезпечення більш модульного і гнучкого інтерфейсу конкретний WSDL має бути відокремлений від абстрактного WSDL. Абстрактний WSDL визначає типи даних,

повідомлення, операції і тип порту (port type). Конкретний WSDL визначає прив'язку, порт і сервіс.

[WS-04] WSDL МАЄ розділятися на абстрактну і конкретну частини.

[WS-05] Усі типи даних МАЮТЬ визначатися в XSD-файлі та імпортувати в абстрактну WSDL.

[WS-06] Конкретний WSDL ПОВИНЕН визначати лише один сервіс з одним портом.

Схеми (XML)

108. Схеми, що використовуються у WSDL, повинні відповідати стандарту BOIB ST.96. Для цілей повторного використання і модульності схема має бути окремим документом, який або включається, або імпортується в WSDL, замість того, щоб визначати її безпосередньо в WSDL. Це дозволить вносити зміни у структуру XML без зміни WSDL.

[WS-07] Схема, визначена в елементі `wsdl:types`, ПОВИННА бути імпортована з самостійного файлу схеми, щоб забезпечити модульність і повторне використання.

[WS-08] Імпорт зовнішньої схеми ПОВИНЕН бути реалізований за допомогою техніки `xsd:import`, а не `xsd:include`.

[WS-09] Елемент `xsd:any` НЕ ПОВИНЕН використовуватися для зазначення кореневого елемента в тілі повідомлення.

[WS-10] Цільовий простір імен для WSDL (*атрибут* `targetNamespace` на `wsdl:definitions`) ПОВИНЕН відрізнятися від цільового простору імен схеми (атрибут `targetNamespace` на `xsd:schema`).

[WS-11] Запити і відповіді (угода про іменування, формат повідомлень, структура даних і словник даних) МАЮТЬ братися зі стандарту BOIB ST.96.

Іменування та управління версіями

109. Відповідні угоди про іменування також повинні застосовуватися при іменуванні Служб та елементів WSDL. Угоди щодо іменування повинні відповідати тим, які впроваджені в стандарті BOIB ST.96.

[WS-12] Сервіси ПОВИННІ називатися у верхньому регістрі (*UpperCamelCase*) та мати суфікс *"Service"*, наприклад:
`https://wipo.int/PatentsService`.

[WS-13] WSDL-елементи *message, part, portType, operation, input, output ma binding* МАЮТЬ бути названі у верхньому регістрі (*UpperCamelCase*).

[WS-14] Назви повідомлень із запитами ПОВИННІ мати суфікс *"Request".*

[WS-15] Назви повідомлень-відповідей ПОВИННІ мати суфікс *"Response".*

[WS-16] Назви операцій МАЮТЬ відповідати формату `<Verb><Object>{<Qualifier>}`, де `<Verb>` вказує операцію (переважно Get, Create,

Update або Delete, де це можна застосувати) над <Object> операції, необов'язково після нього слідує <Qualifier> <Object>.

110. Усі назви операцій повинні складатися щонайменше з двох частин. Необов`язкова третя частина може бути включена для подальшого прояснення та/або уточнення ділової мети операції. Три частини: <Verb> <Object> <Qualifier - необов`язково>. Кожна частина буде детально описана нижче.

Дієслово (Verb) – Ім`я кожної операції починається з дієслова. Нижче наведені приклади дієслів, які найчастіше використовуються:

Дієслово (Verb)	Опис	Приклад
Get	Отримати окремий об'єкт (Get a single object)	GetBibData
Create	Отримати новий об'єкт (Get a new object)	CreateBibData
Update	Оновити об'єкт (Update an object)	UpdateBibData
Delete	Видалити об'єкт (Delete an object)	DeleteCustomer

Об'єкт (*object*)- іменник, що йде за дієсловом, являє собою короткий і однозначний опис бізнес-функції, яку забезпечує операція. Мета полягає в тому, щоб надати споживачам краще розуміння того, що виконує операція, без двозначності. Враховуючи, що визначення деяких сутностей не поширені серед різних центрів витрат, об'єкт може бути складеним полем, першим вузлом якого є центр витрат, а другим - сутність, наприклад, PatentCustomer .

Кваліфікатор (*qualifier)* - Мета атрибута класифікатора об'єкта (необов'язкового) полягає в тому, щоб уточнити ділову або предметну область, наприклад, GetCustomerList. Get - позначає операцію, яка повинна бути виконана над клієнтом («Customer»), а «List» додатково описує той факт, що намір полягає в отриманні списку клієнтів, а не тільки одного клієнта, як у випадку «GetCustomer».

111. Відповідно до принципів сервіс-орієнтованого проєктування, постачальники і споживачі сервісів повинні розвиватися незалежно один від одного. На споживача послуг не повинні впливати незначніх (зворотньо сумісні) зміни з боку постачальника послуг. Тому під час версіонування сервісів мають використовуватися тільки основні номери версій. Для внутрішніх API (наприклад, для розроблення та тестування) можуть також використовуватися мінорні версії, наприклад, *Semantic Versioning* (семантичне версіонування).

[WS-17] Ім'я файлу WSDL МАЄ відповідати наступному зразку: *<ім'я сервісу>_V<основний номер версії>(<service name>_V <major version number>).*

[WS-18] Простір імен файлу WSDL МАЄ містити версію сервісу; наприклад, https://wipo.int/PatentsService/V1".

112. Опис сервісу та його операцій надається у вигляді документації WSDL.

[WS-19] Елемент `wsdl:documentation` МАЄ використовуватися в WSDL з описом сервісу (як перший дочірній елемент `wsdl:definitions` WSDL) та його операцій.

Розроблення контракту вебсервісу

113. До контракту вебсервісу має бути включено технічний інтерфейс, що складається з мови визначення вебсервісу (WSDL), визначень схеми XML, опису політики WS, а також нетехнічний інтерфейс, який складається з одного або декількох документів опису сервісу.

114. WSDL, частина «Контракту сервісу», повинна бути спроєктована до розробки коду. Жоден WSDL не повинен автоматично генеруватися з коду. Девіз: «Спочатку контракт», а НЕ «Спочатку код». Усі контракти вебсервісів повинні відповідати базовому профілю взаємодії вебсервісів (WS-I BP). Будь-який проєкт, який автогенерується з коду, буде зобов'язаний внести поправки для забезпечення відповідності цим стандартам.

Приєднання політик до визначень WSDL

115. Контракти вебсервісів можуть бути розширені політиками безпеки, які виражають додаткові обмеження, вимоги та якості, що зазвичай стосуються поведінки сервісів. Політики безпеки можуть бути доступними для читання людиною і стають частиною додаткової угоди про рівень обслуговування, або ж можуть бути машинозчитуваними і оброблятися під час виконання. Машинозчитувані політики визначаються за допомогою мови WS-Policy та відповідних специфікацій WS-Policy.

[WS-20] Вирази політики ПОВИННІ бути виділені в окремий документ визначення WS-Policy, на який потім робиться посилання в документі WSDL через елемент `wsp:PolicyReference`.

[WS-21] Глобальні або специфічні для домену політики МАЮТЬ бути ізольованими і застосовуватися до декількох сервісів.

[WS-22] Точки приєднання політики МАЮТЬ відповідати WSDL 1.1 або пізнішій версії, бажано версії 2.0, елементам точки прив'язки та відповідним суб'єктам політики (сервіс, кінцева точка, операція і повідомлення).

SOAP - Безпека вебсервісів

116. Безпека вебсервісів (WSS): SOAP *Message Security* - це набір удосконалень для обміну повідомленнями SOAP, які забезпечують цілісність і конфіденційність повідомлень. WSS: SOAP *Message Security* є розширюваним і може працювати з різними моделями безпеки та технологіями шифрування. WSS: SOAP *Message Security* надає три основні механізми, які можна використовувати незалежно або разом:

- Здатність надсилати маркери безпеки як частину повідомлення, а також пов'язувати маркери безпеки зі змістом повідомлення;
- Здатність захистити вміст повідомлення від несанкціонованої та невиявленої зміни (цілісність повідомлення); і
- Здатність захистити вміст повідомлення від несанкціонованого розголошення (конфіденційність повідомлення).

WSS: SOAP Message Security може використовуватися в поєднанні з іншими розширеннями вебслужб і протоколами, специфічними для додатків, для задоволення різних вимог

безпеки.

[WS-23] Вебсервіси, в яких використовуються повідомлення SOAP, МАЮТЬ захищатися відповідно до рекомендацій стандарту WSS:SOAP.

ФОРМАТИ ТИПІВ ДАНИХ

117. Цей стандарт рекомендує формати примітивних типів даних, таких як час, дата та мова, відповідно до рекомендацій стандартів BOIB ST.96 та ST.97, що використовуються для запитів і відповідей XML та JSON відповідно і для параметрів запиту.

[CS-01] Запис часу ПОВИНЕН бути відформатований так, як зазначено в IETF RFC 3339 (це профіль ISO 8601).

[CS-02] Інформація про часовий пояс МАЄ використовуватися як зазначено у IETF RFC 3339. Наприклад: `20:54:21+00:00`

[CS-03] Дата ПОВИННА бути відформатована, як зазначено в IETF RFC 3339 (це профіль ISO 8601). Наприклад: `2018-10-19`

[CS-04] Об'єкти Datetime (тобто штампи часу) ПОВИННІ бути відформатовані так, як зазначено в IETF RFC 3339 (це профіль ISO 8601).

[CS-05] Відповідний часовий пояс МАЄ використовуватися, як зазначено в IETF RFC 3339. Наприклад: `2017-02-14T20:54:21+00:00`

[CS-06] Для кодів валют ПОВИНЕН використовуватися ISO 4217-Alpha (трибуквені коди валют). Точність значення (тобто кількість цифр після десяткової крапки) МОЖЕ варіюватися залежно від ділових вимог.

[CS-07] Двобуквені коди в стандарті BOIB ST.3 повинні використовуватися для представлення BIB, держав, інших суб'єктів, організацій, а також пріоритетних та визначених країн/організацій.

[CS-08] Елементи коду відповідно до стандарту ISO 3166-1-Alpha-2 (двобуквені коди назв країн світу) ПОВИННІ використовуватися для представлення назв країн, залежних територій та інших територій, що становлять особливий геополітичний інтерес, на основі списків назв країн, отриманих від Організації Об'єднаних Націй.

[CS-09] Для кодів мов ПОВИНЕН використовуватися стандарт ISO 639-1 (Двобуквені коди мов).

[CS-10] Одиниці виміру МАЮТЬ використовуватися в одиницях виміру, як описано в Уніфікованому кодексі одиниць виміру (на основі визначень ISO 80000). Наприклад, для вимірювання ваги використовують кілограми (кг)

[CSJ-11] Символи, що використовуються в значеннях перерахувань, ПОВИННІ бути обмежені таким набором:{a-z, A-Z, 0-9, крапка (.), кома (,), пробіли ( ), тире (-) і знак підкреслення (_).

[CSJ-12] Терміни представлення, наведені в Додатку VI, ПОВИННІ використовуватися для назв атомарних властивостей.

[CSJ-13] Акроніми та абревіатури, що з'являються на початку назви властивості, ПОВИННІ бути в нижньому регістрі. В іншому випадку всі значення переліку, акроніми та абревіатури ПОВИННІ відображатися у верхньому регістрі.

ВІДПОВІДНІСТЬ

118. Цей Стандарт розроблено як набір правил і угод, які можна накладати на існуючі або нові API вебсервіси, щоб забезпечити спільну функціональність. Не всі сервіси будуть підтримувати всі угоди, визначені в Стандарті, через бізнес (наприклад, QoS може бути непотрібним) або технічні обмеження (наприклад, може вже використовуватися OAuth 2.0).

119. Цей стандарт визначає два рівні відповідності: Рівні відповідності A та AA. Варто зауважити, що правила, в яких зазначено МОЖНА, не вважаються важливими при визначенні відповідності.

120. API вебсервісам рекомендується підтримувати стільки додаткових функцій, що виходять за рамки їхнього рівня відповідності, скільки необхідно для передбачуваного сценарію.

121. Визначено два рівні відповідності:

Рівень A: Для відповідності рівню A, API показує, що дотримуються необхідні загальні правила проєктування (RSG), які в цьому стандарті позначені як «ПОВИНЕН». Крім того, правила, специфічні для типу відповіді, що повертається, також повинні бути дотримані, іншими словами, вказується наступний підрівень відповідності:

- Рівень AJ: повернення відповіді у форматі відповідно до стандарту BOIB ST.97 JSON повинен відповідати всім загальним правилам рівня (RSG), визначеним як ПОВИНЕН, а також усім правилам, специфічним для JSON (RSJ), визначеним як ПОВИНЕН;

- Рівень AX: повертає екземпляр відповідно до стандарту BOIB ST.96 XML, повинен відповідати всім загальним правилам рівня (RSG), визначеним як ПОВИНЕН, а також усім правилам, специфічним для XML (RSX), визначеним як ПОВИНЕН; і

- Рівень A: повертаючи відповідь у форматі JSON або XML, необхідно відповідати всім загальним правилам рівня (RSG), позначеним як ПОВИНЕН, а також всім правилам для JSON (RSJ), позначеним як ПОВИНЕН, і всім правилам для XML (RSX), позначеним як ПОВИНЕН.

- **Рівень AA**: Для відповідності рівню AA API вказує, що він відповідає рівню A і всі рекомендовані правила проєктування, які в цьому стандарті позначені як «МАЄ», дотримані. Як і у випадку з рівнем A, існують підрівні, що залежать від типу реагування:

- Рівень AAJ: відповідність рівню AJ, а також рекомендовані правила МАЄ , що застосовуються до відповіді JSON; і

- Рівень AAX: відповідність рівню AX, а також рекомендованим правилам МАЄ, що застосовуються до відповіді XML.

122. Матриця простежуваності між правилами проєктування та рівнями відповідності наведена в Додатку I.

ПОСИЛАННЯ

Стандарти ВОІВ

Стандарт ВОІВ ST.3	Рекомендований стандарт стосовно двобуквених кодів для представлення держав, інших адміністративних одиниць та міжурядових організацій
Стандарт ВОІВ ST.96	Рекомендації щодо обробки інформації з інтелектуальної власності з використанням XML (розширюваної мови розмітки)
Стандарт ВОІВ ST.97	Рекомендації щодо обробки інформації про інтелектуальну власність за допомогою формату JSON

Стандарти та конвенції

Запит про надання коментарів Інженерної ради Інтернету (IEFT RFC) 2119	Ключові слова для використання в RFC для позначення рівнів вимог www.ietf.org/rfc/rfc2119.txt
IEFT RFC 3339	Дата і час в Інтернеті: Мітки часу – www.ietf.org/rfc/rfc3339.txt
IEFT RFC 3986	Уніфікований ідентифікатор ресурсу (URI): загальний синтаксис www.ietf.org/rfc/rfc3986.txt
IEFT RFC 5789	Метод PATCH для HTTP https://tools.ietf.org/rfc/rfc5789.txt
IEFT RFC 5988	Вебпосилання - https://www.rfc-editor.org/rfc/rfc5988.txt
IEFT RFC 6648	Зменшення значення «X-» Префікс та подібні конструкції у прикладних протоколах - https://www.rfc-editor.org/rfc/rfc6648.txt
IEFT RFC 6750	Фреймворк авторизації OAuth 2.0: Використання токенів на пред'явника - https://www.rfc-editor.org/rfc/rfc6750.txt
IEFT RFC 7231	Протокол передачі гіпертексту (HTTP/1.1): Семантика та зміст https://www.ietf.org/rfc/rfc7231.txt
EFT RFC 7232	Протокол передачі гіпертексту (HTTP/1.1): Умовні запити https://www.ietf.org/rfc/rfc7232.txt
IEFT RFC 7234	Протокол передачі гіпертексту (HTTP/1.1) – Кешування https://www.ietf.org/rfc/rfc7234.txt
IEFT RFC 7386	Patch для об'єднання JSON - https://www.ietf.org/rfc/rfc7386.txt
IEFT RFC 7240	Заголовок "Prefer" для HTTP -https://www.rfc-editor.org/rfc/rfc7240.txt
IEFT RFC 7519	JSON Web Token (JWT) https://www.ietf.org/rfc/rfc7519.txt
IEFT RFC 7540	Протокол передачі гіпертексту версії 2 (HTTP/2) - https://datatracker.ietf.org/doc/html/rfc7540
IEFT BCP-47	Теги для ідентифікації мов -https://www.rfc-editor.org/rfc/bcp/bcp47.txt
Міжнародний стандарт ISO	Коди для подання назв мов - https://en.wikipedia.org/wiki/List_of_ISO_639_language_codes

639-1

Міжнародний стандарт ISO 3166-1 alpha-2	Двобуквені коди для представлення держав – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
Міжнародний стандарт ISO 3166-1 alpha-3	Трибуквені коди для представлення держав – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
Міжнародний стандарт ISO 4217	Класифікація валют - https://www.iso.org/iso-4217-currency-codes.html
Міжнародний стандарт ISO 8601	Формати дати та часу - https://en.wikipedia.org/wiki/ISO_8601
OData	Протокол відкритих даних -https://www.odata.org/

OASIS OData Metadata Service Entity Model
http://docs.oasisopen.org/odata/odata/v4.0/os/models/MetadataService.edmx

OASIS OData JSON Format Version 4.0. Під редакціяєю Ральфа Хандля, Майкла Піццо і Марка Біамонте. Остання версія

http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html

OASIS OData Atom Format Version 4.0. Під редакцією Мартіна Цурмуєля, Майкла Піццо і Ральфа Хандля. Остання версія -

http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html

OASIS OData Версія 4.0

- Частина 1: Протокол – http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html
- Частина 2: Угоди URL - http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html
- Частина 3: Common Schema Definition Language (CSDL) - http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html

Компоненти OASIS ABNF: правила проєктування OData ABNF версія 4.0 та тестові випадки/завданняOData ABNF Test Cases –
http://docs.oasis-open.org/odata/odata/v4.0/os/abnf/

OASIS Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary –
http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/
Компоненти OASIS:

OASIS Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary – - http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/
XML- схеми OASIS:

OData EDMX XML Schema and OData EDM XML Schema - http://docs.oasis-open.org/odata/odata/v4.0/os/schemas/

OASIS SAML 2.0	http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html
RAML (ReSTful API Modeling Language) (Мова моделювання ReSTful API)	http://raml.org
OpenAPI Initiative	www.openapis.org
REST API Maturity Model - Richardson's	https://martinfowler.com/articles/richardsonMaturityModel.html
HAL	http://stateless.co/hal_specification.html
JSON-LD	https://json-ld.org
Collection+JSON (Формат документів)	http://amundsen.com/media-types/collection/format/
BadgerFish (формат обміну даними)	http://badgerfish.ning.com/
Semantic Versioning (Семантичне версіонування)	https://semver.org/
REST	https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
CQL	https://en.wikipedia.org/wiki/Contextual_Query_Language
Z39.50	https://www.loc.gov/z3950/agency/Z39-50-2003.pdf
WS-I Basic Profile 2.0	http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html
W3C SOAP 1.2 Частина 1: Структура обміну повідомленнями	https://www.w3.org/TR/soap12-part1/
W3C SOAP 1.2 частина 2 Додатки	https://www.w3.org/TR/soap12-part2/
W3C WSDL версії 2.0 частина 1 Мова основної частини	https://www.w3.org/TR/wsdl20/
W3C CORS	https://www.w3.org/TR/cors/
W3C Матричні параметри	https://www.w3.org/DesignIssues/MatrixURIs.html

REST API Відомств інтелектуальної власності:	
EPO Open Patent Services OPS v3.2	https://developers.epo.org
USPTO PatentsView	http://www.patentsview.org/api/doc.html
WIPO - ePCT v 1.1	https://pct.wipo.int/
EUIPO TMview, Designview, TMclas s	http://www.tm-xml.org/TM-XML/TM-XML_xml/TM-XML_TM-Search.xml
Галузеві REST API та рекомендації щодо проєктування:	
Facebook	https://developers.facebook.com/docs/graph-api/reference
GitHub	https://developer.github.com/v3
Google APIs Design Guide	https://cloud.google.com/apis/design/
Azure	https://docs.microsoft.com/en-us/rest/api/
OpenAPI	https://swagger.io/docs/specification/about/
OData	http://www.odata.org/documentation/
JSON API	http://jsonapi.org/format/
Microsoft API Design	https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
JIRA REST API	https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples
Confluence REST API	https://developer.atlassian.com/server/confluence/
Ebay API	https://developer.ebay.com/api-docs/static/ebay-rest-landing.html
Oracle REST Data Services	http://www.oracle.com/technetwork/developer-tools/rest-data-services/overview/index.html
PayPal REST API	https://developer.paypal.com/api/rest/
Найкращі практики роботи з даними в мережі	https://www.w3.org/TR/dwbp/#intro
Рекомендації SAP щодо гармонізації майбутніх REST API	https://d.dam.sap.com/m/xAUymP/54014_GB_54014_enUS.pdf
GitHub API	https://developer.github.com/v3/
Zalando	https://github.com/zalando/ReSTful-api-guidelines

Dropbox	https://www.dropbox.com/developers
TTwitter	https://developer.twitter.com/en/docs

Інші

CQRS	https://martinfowler.com/bliki/CQRS.html
ITU	https://www.itu.int/en/ITU-T/ipr/Pages/open.aspx
OWASP Rest Security Cheat Sheet	https://www.owasp.org/index.php/REST_Security_Cheat_Sheet
DDD	https://martinfowler.com/bliki/BoundedContext.html
REST Principles	https://en.wikipedia.org/wiki/Representational_state_transfer
Open/Closed Principle	- https://en.wikipedia.org/wiki/Open/closed_principle
Which style of WSDL should I use?	https://www.ibm.com/developerworks/library/ws-whichwsdl/

Уряд Нової Зеландії

Стандарт API та настанови	https://www.ict.govt.nz/guidance-and-resources/standards-compliance/api-standard-and-guidelines/
Посібник із запобігання міжсайтовому скриптингу	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
Серія довідників OWASP	https://cheatsheetseries.owasp.org/
Стандарт цифрового підпису (DSS)	https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf
SOAP-безпека повідомлень 1.0, Стандарт OASIS 200401	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
Принципи проєктування послуг SOA, Томас Ерл (2008)	

ДОДАТОК I

СПИСОК ПРАВИЛ І УГОД ЩОДО ПРОЄКТУВАННЯ ВЕБСЕРВІСІВ RESTFUL

Версія 1.1

Редакція затверджена Комітетом зі стандартів ВОІВ (КСВ) на його десятій сесії 25 листопада 2022 року

Наступні таблиці узагальнюють правила та угоди щодо проєктування сервісів, а також визначені основні вимоги стосовно відповідності з точки зору того, який рівень відповідності підтримує впровадження Web ServicesAPI. Нижче наведено рекомендації щодо наведених далі таблиць:

- Таблиця 1 містить стислий опис правил, яких необхідно дотримуватися для досягнення відповідності рівню AJ (для відповіді у форматі JSON);
- - Таблиця 2 містить стислий виклад правил проєктування, яких необхідно дотримуватися для досягненнявідповідності рівню AX (для відповіді у форматі XML);
- Таблиця 3 містить короткий опис правил проєктування, яких необхідно дотримуватися для досягнення відповідності рівню AAJ (для відповіді у форматі JSON); і
- Таблиця 4 містить стислий опис правил оформлення, яких необхідно дотримуватися для досягнення відповідності рівню AAX (для відповіді у форматі XML).

[Редакційна примітка: Для досягнення відповідності рівню A необхідно дотримуватися правил, наведених у Таблицях 1 і 2. Для досягнення відповідності рівню AA необхідно дотримуватися правил, наведених у Таблицях 3 і 4. Третя буква вказує на тип наданої відповіді.]

Таблиця 1: Таблиця відповідності JSON-відповіді

Ідентифікатор правила	**Опис правил**	**Перехресні посилання та примітки**
[RSG-01]	Символ скісної риски «/» ПОВИНЕН використовуватися в шляху до URI для позначення ієрархічних відносин між ресурсами, але шлях НЕ ПОВИНЕН закінчуватися скісною рискою, оскільки він не надає жодного семантичного значення і може викликати плутанину.	AJ, AX, AAJ, AAX
[RSG-02]	Назви ресурсів ПОВИННІ бути узгодженими за схемою іменування.	AJ, AX, AAJ, AAX
[RSG-04]	Параметри запиту ПОВИННІ бути узгодженими в шаблоні іменування	AJ, AX

[RSG-06]	Шаблон URL-адреси для веб-API ПОВИНЕН містити слово «api» в URI.	AJ, AX, AAJ, AAX
[RSG-07]	Параметри матриці НЕ МОЖНА використовувати.	AJ, AX, AAJ, AAX
[RSG-08]	Web API ПОВИНЕН послідовно застосовувати коди стану HTTP, як описано в IETF RFC	AJ, AX, AAJ, AAX
[RSG-10]	Якщо API виявляє невірні вхідні значення, він ПОВИНЕН повернути код стану HTTP *«*`400 Bad Request`*»*. Корисне навантаження помилки ПОВИННО вказувати на помилкове значення.	AJ, AX, AAJ, AAX
[RSG-12]	Якщо API виявляє допустимі значення, які вимагають, щоб функції не були реалізовані, він ПОВИНЕН повернути HTTP-код статусу «`501 Not Implemented`” Корисне навантаження помилки ПОВИННО вказувати на необроблене значення.	AJ, AX, AAJ, AAX
[RSG-14]	Якщо ресурс може бути самостійним, він ПОВИНЕН бути ресурсом верхнього рівня, в іншому випадку - підресурсом.	AJ, AX, AAJ, AAX
[RSG-15]	Параметри запиту ПОВИННІ використовуватися замість шляхів URL для отримання вкладених ресурсів.	AJ, AX, AAJ, AAX
[RSG-18]	Назви ресурсів, сегментів та параметри запитів ПОВИННІ складатися зі слів англійською мовою з використанням основного англійського правопису, представленого в Оксфордському словнику англійської мови. Назви ресурсів, які локалізовані у зв'язку з вимогами бізнесу, МОЖУТЬ бути іншими мовами.	AJ, AX, AAJ, AAX
[RSG-20]	Web API ПОВИНЕН підтримувати узгодження типів вмісту відповідно до IETF RFC 7231.	AJ, AX, AAJ, AAX
[RSG-21]	Формат JSON ПОВИНЕН використовуватися, якщо не запитується певний тип вмісту.	AJ, AX, AAJ, AAX
[RSG-27]	Web API ПОВИНЕН підтримувати як мінімум XML або JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Методи HTTP ПОВИННІ бути обмежені стандартними методами HTTP `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` і `HEAD`, як вказано в IETF RFC 7231 та 5789.	AJ, AX, AAJ, AAX
[RSG-33]	Для кінцевої точки, яка витягує один ресурс, якщо ресурс не знайдено, метод `GET` ПОВИНЕН повертати код стану *«*`404 Not Found`». Кінцеві точки, які повертають списки ресурсів, просто повертають пустий список.	AJ, AX, AAJ, AAX
[RSG-34]	Якщо ресурс отримано успішно, метод *GET* ПОВИНЕН повернути *«*`200 OK`*»*.	AJ, AX, AAJ, AAX
[RSG-35]	Запит `GET` ПОВИНЕН бути ідемпотентним (незалежним від кількості виконання запиту).	AJ, AX, AAJ, AAX
[RSG-37]	Запит `HEAD` ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-39]	`POST`-запит НЕ ПОВИНЕН бути ідемпотентним відповідно до IETF RFC 2616.	AJ, AX, AAJ, AAX
[RSG-43]	Запит `PUT` ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX

[RSG-44]	Якщо ресурс не знайдено, `PUT` ПОВИНЕН повернути код стану «`404 Not Found`».	AJ, AX, AAJ, AAX
[RSG-45]	Якщо ресурс успішно оновлено, `PUT` ПОВИНЕН повернути код стану «`200 OK`», якщо оновлений ресурс повернутий, або «`204 No Content`», якщо він не повернутий.	AJ, AX, AAJ, AAX
[RSG-46]	Запит `PATCH` НЕ ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-48]	Якщо ресурс не знайдено, `PATCH` ПОВИНЕН повернути код стану «`404 Not Found`».	AJ, AX, AAJ, AAX
[RSJ-49]	Якщо Web API реалізує часткові оновлення за допомогою `PATCH`, він ПОВИНЕН використовувати формат JSON Merge Patch для опису часткового набору змін, як описано в IETF RFC 7386, використовуючи тип вмісту `application/merge-patch+json`.	AJ, AAJ
[RSG-50]	Запит `DELETE` НЕ ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-51]	Якщо ресурс не знайдено, `DELETE` ПОВИНЕН повернути код стану «`404 Not Found`».	AJ, AX, AAJ, AAX
[RSG-52]	Якщо ресурс видалено успішно, `DELETE` ПОВИНЕН повернути статус «`200 OK`», якщо видалений ресурс повернуто, або «`204 No Content`», якщо його не повернуто.	AJ, AX, AAJ, AAX
[RSG-53]	Кінцевим одержувачем є або вихідний сервер, або перший проксі або шлюз, який отримав у запиті значення Max-Forwards, що дорівнює нулю. Запит `TRACE` НЕ ПОВИНЕН містити тіло (запиту).	AJ, AX, AAJ, AAX
[RSG-54]	Запит `TRACE` НЕ ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-55]	Значення поля заголовка `Via` HTTP ПОВИННО служити для відстеження ланцюжка запитів.	AJ, AX, AAJ, AAX
[RSG-56]	Поле заголовка HTTP Max-Forwards ПОВИННО використовуватися для того, щоб клієнт міг обмежити довжину ланцюжка запитів.	AJ, AX, AAJ, AAX
[RSG-58]	Відповіді на `TRACE` НЕ ПОВИННІ кешуватися.	AJ, AX, AAJ, AAX
[RSG-60]	Запит `OPTIONS` ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-70]	Web API ПОВИНЕН використовувати параметри запиту для реалізації пагінації (розбивки на сторінки).	AJ, AX, AAJ, AAX
[RSG-71]	Web API НЕ ПОВИНЕН використовувати HTTP - заголовки для реалізації пагінації.	AJ, AX, AAJ, AAX
[RSG-75]	Для зазначення критерію сортування за кількома атрибутами ПОВИНЕН використовуватися параметр запиту. Значення цього параметра являє собою список ключів сортування, розділених комами, а напрямки сортування або «asc» для висхідного, або «desc» для низхідного МОЖУТЬ бути додані до кожного ключа сортування, розділені символом двокрапки «:». Напрямок за замовчуванням ПОВИНЕН бути визначений сервером у разі, якщо для ключа не вказано напрямок сортування.	AJ, AX, AAJ, AAX
[RSG-76]	Web API МАЄ повертати критерії сортування у відповіді.	AJ, AX, AAJ, AAX

[RSG-79]	Web API ПОВИНЕН підтримувати повернення кількості елементів у колекції.	AJ, AX, AAJ, AAX
[RSG-80]	Для підтримки повернення кількості елементів у колекції. ПОВИНЕН використовуватися параметр запиту	AJ, AX, AAJ, AAX
[RSG-82]	Web API МОЖЕ підтримувати повернення кількості елементів у колекції в режимі *inline,* тобто як частину відповіді, що містить саму колекцію. При цьому ПОВИНЕН використовуватися параметр запиту.	AJ, AX, AAJ, AAX
[RSG-86]	У договорі про надання послуг ПОВИННА бути вказана граматика, що підтримується (наприклад, поля, функції, ключові слова та оператори)	AJ, AX, AAJ, AAX
[RSG-87]	ПОВИНЕН використовуватися параметр запиту "q".	AJ, AX, AAJ, AAX
[RSG-88]	На рівні протоколу Web API ПОВИНЕН повертати відповідний код стану HTTP, вибраний зі списку стандартних кодів стану HTTP.	AJ, AX, AAJ, AAX
[RSJ-89]	На рівні програми Web API ПОВИНЕН повертати корисне навантаження, що повідомляє про помилку з достатнім ступенем деталізації. Атрибути code і *message* є обов'язковими, атрибут *details* є умовно обов'язковим, атрибути `target`, `status`, `moreInfo` *і* `internalMessage` є необов'язковими.	AJ, AX, AAJ, AAX
[RSG-90]	Помилки НЕ ПОВИННІ розкривати критично важливі для безпеки дані або внутрішні технічні деталі, такі як стеки викликів у повідомленнях про помилку.	AJ, AX, AAJ, AAX
[RSG-91]	Заголовок HTTP `Header: «Reason-Phrase»` (описаний в RFC 2616) НЕ ПОВИНЕН використовуватися для передачі повідомлень про помилки.	AJ, AX, AAJ, AAX
[RSG-93]	Формат договору про надання послуг ОБОВ'ЯЗКОВО повинен включати наступне: – Версію API; – Інформацію про семантику елементів API; – Ресурси – Атрибути ресурсу; – Параметри запиту; – Методи ; – Типи носіїв; – Граматика пошуку (якщо вона підтримується); – Коди стану HTTP; – Методи HTTP; – Обмеження та відмінні функції; і – Безпека (за наявності).	AJ, AX, AAJ, AAX
[RSG-95]	REST API ПОВИНЕН надавати документацію API у вигляді сервісного контракту.	AJ, AX, AAJ, AAX
[RSG-96]	Реалізація Web API, що відхиляється від цього стандарту, ПОВИННА бути чітко задокументована в сервісному контракті. Якщо правило відхилення не зазначено в сервісному контракті, ПОВИННО передбачатися дотримання цього	AJ, AX, AAJ, AAX

	стандарту	
[RSG-97]	Сервісний контракт ПОВИНЕН дозволяти генерувати скелетний код клієнта API.	AJ, AX, AAJ, AAX
[RSG-105]	Web API ПОВИНЕН підтримувати кешування результатів GET; Web API МОЖЕ підтримувати кешування результатів і інших методів HTTP.	AJ, AX, AAJ
[RSG-113]	Якщо Web API підтримує обробку налаштувань (preference handling), номенклатура налаштувань, які МОЖУТЬ бути встановлені за допомогою заголовка `Prefer`, ПОВИННА бути записані в сервісному контракті.	AAJ, AAX, AJ, AX
[RSG-114]	Якщо Web API підтримує локалізовані дані, HTTP -заголовок запиту `Accept-Language` ПОВИНЕН підтримуватися для вказівки набору природних мов, яким у відповіді надається перевага, як зазначено в IETF RFC 7231.	AJ, AX, AAJ, AAX
[RSG-116]	Конфіденційність: різні API та інформація в API ПОВИННІ бути ідентифіковані, класифіковані та захищені від несанкціонованого доступу, розголошення та перехоплення в будь-який час. При цьому ПОВИННІ дотримуватися принципи найменших привілеїв, нульової довіри, необхідності знати та необхідності ділитися.	AJ, AX, AAJ, AAX
[RSG-117]	Забезпечення цілісності: різні API та інформація в API ПОВИННІ бути захищені від несанкціонованої модифікації, дублювання, пошкодження та знищення. Інформація ПОВИННА модифікуватися через підтверджені транзакції та інтерфейси. Системи ПОВИННІ оновлюватися з використанням затверджених процесів управління конфігурацією, управління змінами та управління виправленнями.	AJ, AX, AAJ, AAX
[RSG-118]	Доступність: різні API та інформація в API ПОВИННІ бути доступними авторизованим користувачам у встановлений час згідно з угодами про рівень обслуговування (SLA), політиками контролю доступу та визначеними бізнес-процесами.	AJ, AX, AAJ, AAX
[RSG-119]	Безвідмовність: Кожна оброблювана транзакція або дія, що виконується API, ПОВИННІ забезпечувати невідказованість через реалізацію належного аудиту, авторизації, автентифікації, а також реалізації безпечних шляхів, а також сервісів і механізмів невідказованості.	AJ, AX, AAJ, AAX
[RSG-120]	Автентифікація, авторизація, аудит: Користувачі, системи, API або пристрою, що беруть участь у критичних транзакціях або діях, ПОВИННІ бути автентифіковані, авторизовані за допомогою служб контролю доступу на основі ролей або атрибутів і підтримувати розділення обов'язків. Крім того, всі дії ПОВИННІ реєструватися, а надійність автентифікації повинна збільшуватися згідно з відповідним інформаційним ризиком.	AJ, AX, AAJ, AAX
[RSG-121]	При розробці API НЕОБХІДНО ретельно враховувати загрози, випадки шкідливого використання, методи безпечного кодування, безпеку транспортного рівня та тестування безпеки, особливо:	AJ, AX, AAJ, AAX

	– PUTs і POSTs - тобто: які зміни внутрішніх даних потенційно можуть бути використані для атаки або неправдивої інформації; – DELETES - тобто: може використовуватися для видалення вмісту внутрішнього сховища ресурсів; – Білий список допустимих методів - для забезпечення того, щоб допустимі методи HTTP були належним чином обмежені, в той же час як інші будуть повертати правильний код відповіді; і – Добре відомі атаки слід враховувати на етапі моделювання загроз у процесі проєктування, щоб не допустити збільшення ризику загроз. ПОВИННІ враховуватися загрози та заходи щодо їх зниження, визначені в OWASP Top Ten Cheat.	
[RSG-122]	При розробці API слід дотримуватися стандартів та найкращих практик, перелічених нижче: – Найкращі практики безпечного кодування: Принципи безпечного кодування OWASP; – Безпека Rest API: Пам'ятка з безпеки REST; – Екранування введення та захист від міжсайтового скриптингу: OWASP XSS Cheat Sheet; – Запобігання SQL-ін'єкцій: Пам'ятка OWASP щодо SQL-ін'єкцій, пам'ятка OWASP щодо Parameterization; і – Безпека транспортного рівня: Пам'ятка OWASP щодо захисту транспортного рівня	AJ, AX, AAX, AAJ
[RSG-123]	Тестування безпеки та оцінювання вразливостей ПОВИННІ проводитися для забезпечення безпеки та стійкості API до загроз. Ця вимога МОЖЕ бути виконана шляхом використання статичного та динамічного тестування безпеки додатків(SAST/DAST), автоматизованих інструментів управління вразливостями та тестування на проникнення	AJ, AX, AAX, AAJ
[RSG-124]	Захищені служби ПОВИННІ надавати кінцеві точки HTTPS тільки з використанням TLS 1.2 або вище з набором шифрів, що включає ECDHE для обміну ключами.	AJ, AX, AAJ, AAX
[RSG-130]	Анонімна автентифікація ПОВИННА використовуватись тільки тоді, коли клієнти та використовуваний ними додаток отримують доступ до інформації або функцій з низьким рівнем чутливості, які не повинні вимагати автентифікації, наприклад, до публічної інформації.	AJ, AX, AAJ, AAX
[RSG-131]	Автентифікація з використанням імені користувача і паролю або хеш-паролю НЕ ПОВИННА дозволятись.	AJ, AX, AAJ, AAX
[RSG-141]	Ключі API ПОВИННІ бути відкликані, якщо клієнт порушує угоду про використання, як визначено у BIB.	AJ, AX, AAJ, AAX
[RSG-144]	Безпечні та надійні сертифікати ПОВИННІ видаватися взаємно довіреним центром сертифікації (ЦС) через процес встановлення довіри або перехресної сертифікації.	AJ, AX, AAJ, AAX

[RSG-145]	Сертифікати, що спільно використовуються клієнтом і сервером, НЕОБХІДНО використовувати для зниження ризиків безпеки ідентифікації, характерних для чутливих систем та привілейованих дій, наприклад, X.509	AJ, AX, AAJ, AAX
[RSG-148]	Якщо REST API є публічним, HTTP-заголовок Access-Control-Allow-Origin ПОВИНЕН мати значення «*».	AJ, AX, AAJ, AAX

Таблиця 2: **Таблиця відповідності XML-відповідь**

Ідентифікатор правила	**Опис правил**	**Перехресні посилання та примітки**
[RSG-01]	Символ скісної риски "/" ПОВИНЕН використовуватися в шляху до URI для позначення ієрархічного зв'язку між ресурсами, але шлях НЕ ПОВИНЕН закінчуватися прямою похилою рискою, оскільки він не надає жодного семантичного значення і може викликати плутанину.	AJ, AX, AAJ, AAX
[RSG-02]	Назви ресурсів ПОВИННІ бути узгодженими за схемою іменування.	AJ, AX, AAJ, AAX
[RSG-04]	Параметри запиту ПОВИННІ бути узгодженими в шаблоні іменування.	AJ, AX
[RSG-06]	Шаблон URL для Web API ПОВИНЕН містити слово «`api`» в URI.	AJ, AX, AAJ, AAX
[RSG-07]	Матричні параметри НЕ ПОВИННІ використовуватися.	AJ, AX, AAJ, AAX
[RSG-08]	Web API ПОВИНЕН послідовно застосовувати коди стану HTTP, як описано в IETF RFC	AJ, AX, AAJ, AAX
[RSG-10]	ЯКЩО API виявляє невірні вхідні значення, він ПОВИНЕН повернути код стану HTTP «`400 Bad Request`». Корисне навантаження помилки ПОВИННО вказувати на помилкове значення.	AJ, AX, AAJ, AAX
[RSG-12]	Якщо API виявляє допустимі значення, які вимагають, щоб функції не були реалізовані, він ПОВИНЕН повернути HTTP-код статусу «`501 Not Implemented`». Корисне навантаження помилки ПОВИННО вказувати на необроблене значення.	AJ, AX, AAJ, AAX
[RSG-14]	Якщо ресурс може бути самостійним, він ПОВИНЕН бути ресурсом верхнього рівня, в іншому випадку - підресурсом.	AJ, AX, AAJ, AAX
[RSG-15]	Параметри запиту ПОВИННІ використовуватися замість шляхів URL для отримання вкладених ресурсів.	AJ, AX, AAJ, AAX
[RSG-18]	Назви ресурсів, сегментів і параметрів запитів ПОВИННІ складатися зі слів англійською мовою, використовуючи основні англійські написання, наведені в Оксфордському словнику англійської мови. Назви ресурсів, які локалізуються через бізнес-вимоги, МОЖУТЬ бути іншими мовами.	AJ, AX, AAJ, AAX
[RSG-20]	Web API ПОВИНЕН підтримувати узгодження типів контенту відповідно до IETF RFC 7231.	AJ, AX, AAJ, AAX
[RSG-21]	Формат JSON ПОВИНЕН передбачатися, якщо не запитується певний тип вмісту.	AJ, AX, AAJ, AAX
[RSG-27]	Web API ПОВИНЕН підтримувати, принаймні, XML або JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Методи HTTP ПОВИННІ бути обмежені стандартними методами HTTP `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` і `HEAD`, як вказано в IETF RFC 7231 та 5789.	AJ, AX, AAJ, AAX

[RSG-33]	Для кінцевої точки, яка отримує один ресурс, якщо ресурс не знайдено, метод `GET` ПОВИНЕН повернути код статусу «`404 Not Found`». Кінцеві точки, які повертають списки ресурсів, просто повернуть порожній список.	AJ, AX, AAJ, AAX
[RSG-34]	Якщо ресурс успішно отримано, метод `GET` ПОВИНЕН повернути код стану «`200 OK`».	AJ, AX, AAJ, AAX
[RSG-35]	Запит `GET` ПОВИНЕН бути ідемпотентним (незалежним від кількості виконань запиту)	AJ, AX, AAJ, AAX
[RSG-37]	Запит `HEAD` ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-39]	`POST`- запит НЕ ПОВИНЕН бути ідемпотентним згідно з IETF RFC 2616.	AJ, AX, AAJ, AAX
[RSG-43]	Запит `PUT` ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-44]	Якщо ресурс не знайдено, *PUT* ПОВИНЕН повернути код стану «`404 Not Found`».	AJ, AX, AAJ, AAX
[RSG-45]	Якщо ресурс успішно оновлено, PUT ПОВИНЕН повернути код стану «`200 OK`», якщо оновлений ресурс буде повернуто, або «`204 No Content`», якщо його не буде повернуто.	AJ, AX, AAJ, AAX
[RSG-46]	Запит `PATCH` НЕ ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-48]	Якщо ресурс не знайдено, *PATCH* ПОВИНЕН повернути код стану «`404 Not Found`».	AJ, AX, AAJ, AAX
[RSG-50]	Запит `DELETE` НЕ ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-51]	Якщо ресурс не знайдено, `DELETE` ПОВИНЕН повернути код стану «`404 Not Found`».	AJ, AX, AAJ, AAX
[RSG-52]	Якщо ресурс успішно видалено, *DELETE* ПОВИНЕН повернути статус «`200 OK`», якщо видалений ресурс повернуто або «`204 No Content`», якщо видалений ресурс не повернуто.	AJ, AX, AAJ, AAX
[RSG-53]	Кінцевим одержувачем є або вихідний сервер,або перший проксі чи шлюз, який отримав у запиті значення Max-Forwards рівне нулю. Запит `TRACE` НЕ ПОВИНЕН включати тіло (запиту).	AJ, AX, AAJ, AAX
[RSG-54]	Запит `TRACE` НЕ ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-55]	Значення поля заголовка *«Via»* HTTP ПОВИННО служити для відстеження ланцюжка запитів.	AJ, AX, AAJ, AAX
[RSG-56]	Поле заголовка HTTP *«Max-Forwards»* ПОВИННО використовуватися для того, щоб клієнт міг обмежити довжину ланцюжка запитів..	AJ, AX, AAJ, AAX
[RSG-58]	Відповіді на `TRACE` НЕ ПОВИННІ кешуватися.	AJ, AX, AAJ, AAX
[RSG-60]	Запит `OPTIONS` ПОВИНЕН бути ідемпотентним.	AJ, AX, AAJ, AAX
[RSG-70]	Web API ПОВИНЕН використовувати параметри запиту для реалізації пагінації (розбивки на сторінки).	AJ, AX, AAJ, AAX
[RSG-71]	Web API НЕ ПОВИНЕН використовувати HTTP-заголовки для реалізації пагінації.	AJ, AX, AAJ, AAX
[RSG-75]	Для зазначення критерію сортування за кількома атрибутами ПОВИНЕН використовуватися параметр запиту. Значення цього параметра являє собою список ключів сортування, розділених комами, а напрямки сортування або `'asc'` для висхідного, або `'desc'` для низхідного МОЖУТЬ бути додані до кожного ключа сортування, розділені символом двокрапки ':'. Напрямок за замовчуванням ПОВИНЕН бути визначений сервером у разі, якщо для ключа не вказано напрямок сортування.	AJ, AX, AAJ, AAX

[RSG-76]	Web API МАЄ повертати критерії сортування у відповіді.	AJ, AX, AAJ, AAX
[RSG-79]	WebAPI ПОВИНЕН підтримувати повернення кількості елементів у колекції.	AJ, AX, AAJ, AAX
[RSG-80]	Параметр запиту ПОВИНЕН використовуватися для підтримки повернення кількості елементів у колекції.	AJ, AX, AAJ, AAX
[RSG-82]	Web API МОЖЕ підтримувати повернення кількості елементів у колекції в режимі inline, тобто як частину відповіді, що містить саму колекцію. Для цього ПОВИНЕН використовуватися параметр запиту.	AJ, AX, AAJ, AAX
[RSG-86]	У договорі про надання послуг (сервісний контракт) ПОВИННА бути вказана граматика, що підтримується (наприклад, поля, функції, ключові слова та оператори).	AJ, AX, AAJ, AAX
[RSG-87]	ПОВИНЕН використовуватися параметр запиту *«*`q`*»*	AJ, AX, AAJ, AAX
[RSG-88]	На рівні протоколу Web API ПОВИНЕН повертати відповідний код стану HTTP, вибраний із списку стандартних кодів стану HTTP.	AJ, AX, AAJ, AAX
RSJ-89]	На рівні додатку WebAPI ПОВИНЕН повертати частину, що повідомляє про помилку з достатнім ступенем деталізвції. Атрибути *«code»* і *«message»* є обов'язковими, атрибут *«details»* є умовно обов'язковим, атрибути *«target,» «status», "*`moreInfo`*," і "*`internalMessage`*"* є необов'язковими.	AJ, AX, AAJ, AAX
[RSG-90]	Помилки НЕ ПОВИННІ розкривати критично важливі для безпеки дані або внутрішні технічні деталі, такі як стеки викликів у повідомленнях про помилку.	AJ, AX, AAJ, AAX
[RSG-91]	Заголовок HTTP `Header: Reason-Phrase` (описаний в RFC 2616) НЕ ПОВИНЕН використовуватися для передачі повідомлень про помилки.	AJ, AX, AAJ, AAX
[RSG-93]	У формат сервісного контракту НЕОБХІДНО включати наступне: – Версія API; – Інформація про семантику елементів API; – Ресурси ; – Атрибути ресурсу; – Параметри запиту; – Методи; – Типи носіїв; – Граматика пошуку (якщо вона підтримується); – Коди стану HTTP; – Методи HTTP; – Обмеження та відмінні особливості; і – Безпека (якщо є).	AJ, AX, AAJ, AAX
[RSG-95]	REST API ПОВИНЕН надавати документацію API у вигляді сервісного контракту.	AJ, AX, AAJ, AAX
[RSG-96]	Реалізація Web API, що відхиляється від цього стандарту, ПОВИННА бути чітко задокументована в сервісному контракті. Якщо правило відхилення в сервісному контракті не вказано, ПОВИННО вважатися, що дотримується цей Стандарт.	AJ, AX, AAJ, AAX
[RSG-97]	Сервісний контракт ПОВИНЕН дозволяти генерувати скелетний код клієнта API.	AJ, AX, AAJ, AAX
[RSG-105]	Web API ПОВИНЕН підтримувати кешування результатів *GET*; Web API МОЖЕ підтримувати кешування результатів інших методів HTTP.	AJ, AX, AAJ
[RSG-113]	Якщо Web API підтримує обробку налаштувань, номенклатура налаштувань, які МОЖУТЬ бути встановлені за допомогою заголовка *«*`Prefer`*»,* ПОВИННА бути записана в сервісному	AAJ, AAX, AJ, AX

	контракті.	
[RSG-114]	Якщо Web API підтримує локалізовані дані, HTTP- заголовок запиту «`Accept-Language`» ПОВИННА підтримуватися вказівка на набір природних мов, яким надається перевага у відповіді, як зазначено в IETF RFC 7231	AAJ, AAX, AJ, AX
[RSG-116]	Конфіденційність: різні API та інформація в API ПОВИННІ бути ідентифіковані, класифіковані та захищені від несанкціонованого доступу, розкриття та перехоплення в будь-який час. ПОВИННІ дотримуватися принципи найменших привілеїв, нульової довіри, необхідності знати та необхідності ділитися.	AAJ, AAX, AJ, AX
[RSG-117]	Забезпечення цілісності: API та інформація про API ПОВИННІ бути захищені від несанкціонованої модифікації, дублювання, пошкодження та знищення. Інформація ПОВИННА модифікуватися за допомогою затверджених транзакцій та інтерфейсів. Системи ПОВИННІ оновлюватися за допомогою затверджених процесів управління конфігурацією, управління змінами та управління виправленнями.	AAJ, AAX, AJ, AX
[RSG-118]	Доступність:API та інформація про API ПОВИННІ бути доступними для авторизованих користувачів у потрібний час, як визначено в Угодах про рівень обслуговування (SLA), політиках контролю доступу та визначених бізнес-процесах.	AAJ, AAX, AJ, AX
[RSG-119]	Невідмовність: Кожна транзакція, що обробляється або виконується за допомогою API, ПОВИННА забезпечувати неможливість відмови через реалізацію належного аудиту, авторизації, автентифікації та впровадження безпечних шляхів, а також послуг і механізмів, що не допускають відмови.	
[RSG-120]	Автентифікація, авторизація, аудит: Користувачі, системи, API або пристрої, що беруть участь у критичних транзакціях або діях, ПОВИННІ бути автентифіковані, авторизовані за допомогою служб контролю доступу на основі ролей або атрибутів і підтримувати розподіл обов'язків. Крім того, всі дії ПОВИННІ реєструватися, а надійність автентифікації повинна збільшуватися з відповідним інформаційним ризиком.	AAJ, AAX, AJ, AX
[RSG-121]	При розробці API ПОТРІБНО ретельно враховувати загрози, випадки шкідливого використання, методи безпечного кодування, безпека транспортного рівня та тестування безпеки, особливо: – *PUTs та POSTs* – тобто: які зміни у внутрішніх даних можуть бути потенційно використані для атак або дезінформації; – *DELETE* – тобто: може використовуватися для видалення вмісту внутрішнього сховища ресурсів; – Білий список допустимих методів – для забезпечення того, щоб допустимі методи HTTP були належними чином обмежені, тоді як інші будуть повертати правильний код відповіді; і – Добре відомі атаки слід враховувати на етапі моделювання загроз у процесі проєктування, щоб не допустити збільшення ризику загроз. Загрози та заходи щодо їх зниження, визначені в OWASP Top Ten Cheat Sheet, ПОВИННІ бути прийняті до уваги.	AAJ, AAX, AJ, AX

[RSG-122]	При розробці API НЕОБХІДНО дотримуватися стандартів та найкращих практик, наведених нижче: – Кращі практики безпечного кодування: Принципи безпечного кодування OWASP; – Безпека REST API: Пам`ятка з безпеки REST; – Екранування введення та захист від міжсайтового скрипнінгу: OWASP XSS Cheat Sheet; – Запобігання SQL-ін'єкціям: Пам'ятка OWASP щодо SQL-ін'єкцій, пам'ятка OWASP щодо Parameterization; і – Безпека транспортного рівня: пам'ятка OWASP щодо захисту транспортного рівня.	AJ, AX, AAX, AAJ
[RSG-123]	Тестування безпеки та оцінка вразливостей ПОВИННІ проводитися, щоб гарантувати, що API є безпечними та стійкими до загроз. Ця вимога МОЖЕ бути досягнута за допомогою статичного та динамічного тестування безпеки додатків (SAST/DAST), автоматизованих інструментів управління вразливостями та тестування на проникнення.	AJ, AX, AAJ, AAX
[RSG-124]	Захищені служби ПОВИННІ надавати кінцеві точки HTTPS тільки з використанням TLS 1.2 або вище з набором шифрів, який включає ECDHE для обміну ключами.	AJ, AX, AAJ, AAX
[RSG-130]	Анонімна автентифікація ПОВИННА використовуватися лише тоді, коли клієнти та додаток, який вони використовують, одержують доступ до інформації або функцій із низьким рівнем чутливості, які не повинні вимагати автентифікації, наприклад до публічної інформації.	AJ, AX, AAJ, AAX
[RSG-131]	Автентифікація з використанням імені користувача і пароля або хеш-пароля НЕ ПОВИННА дозволятися.	AJ, AX, AAJ, AAX
[RSG-141]	Ключі API ПОВИННІ бути відкликані, якщо клієнт порушує угоду про використання, як зазначено в BIB.	AJ, AX, AAJ, AAX
[RSG-144]	Безпечні та надійні сертифікати ПОВИННІ видаватися взаємно довіреним центром сертифікації (ЦС) через процес встановлення довіри або перехресної сертифікації.	AJ, AX, AAJ, AAX
[RSG-145]	Сертифікати, спільні для клієнта та сервера, ПОВИННІ використовуватися для зменшення ризиків безпеки ідентичності, особливо для чутливих систем та привілейованих дій, наприклад, X.509.	AJ, AX, AAJ, AAX
[RSG-148]	Якщо REST API є публічним, HTTP-заголовок *Access-Control-Allow-Origin* ПОВИНЕН мати значення '*'.	AJ, AX, AAJ, AAX

Таблиця 3: Таблиця відповідності рівня AAJ,

Ідентифікатор правила	**Правило**	**Перехресні посилання та примітки**
[RSG-01]	Символ скісної риски «/» ПОВИНЕН використовуватися в шляху URI для зазначення ієрархічних зв'язків між ресурсами, але шлях НЕ ПОВИНЕН закінчуватися скісною рискою, оскільки він не надає жодного семантичного значення і може викликати плутанину.	AAJ, AAX, AX, AJ
[RSG-02]	Назви ресурсів ПОВИННІ бути узгодженими за схемою іменування.	AAJ, AAX, AX, AJ
[RSG-03]	Назви ресурсів ПОВИННІ використовувати малі або великі літери. Назви ресурсів МОЖНА скорочувати.	AAJ, AAX

[RSG-05]	Параметри запиту ПОВИННІ використовувати угоду про нижній регістр (lowerCamelCase), і вони МОЖУТЬ бути скороченими.	AAJ, AAX
[RSG-06]	Шаблон URL-адреси для Web API ПОВИНЕН містити слово "api" в URI.	AAJ, AAX, AX, AJ
[RSG-07]	Матричні параметри НЕ ПОВИННІ використовуватися.	AAJ, AAX, AX, AJ
[RSG-08]	Web API ПОВИНЕН послідовно застосовувати коди стану HTTP, як описано в RFC IETF.	AAJ, AAX, AX, AJ
[RSG-09]	Рекомендовані коди в Додатку V МАЮТЬ використовувати Web API для класифікації помилок.	AAX, AAJ
[RSG-10]	Якщо API виявляє неприпустимі значення введення, він ПОВИНЕН повернути код стану HTTP *"400 Bad Request"*. Корисне навантаження помилки ПОВИННО вказувати на помилкове значення.	AAJ, AAX, AX, AJ
[RSG-11]	Якщо API виявляє синтаксично правильні імена аргументів (у запиті або параметрах запиту), які не є очікуваними, він МАЄ їх ігнорувати.	AAJ, AAX
[RSG-12]	Якщо API виявляє допустимі значення, які вимагають, щоб функції не були реалізовані, він ПОВИНЕН повернути HTTP-код статусу *«501 Not Implemented».* Корисне навантаження помилки ПОВИННО вказувати на необроблене значення.	AAJ, AAX, AX, AJ
[RSG-13]	Web API НЕОБХІДНО використовувати тільки ресурси верхнього рівня. Якщо є підресурси, вони мають бути колекціями і мати на увазі асоціацію. Об'єкт повинен бути доступним або як ресурс верхнього рівня, або як підресурс, але не використовувати обидва способи.	AAJ, AAX
[RSG-14]	Якщо ресурс може бути відокремленим, він ПОВИНЕН бути ресурсом верхнього рівня, або, інакше, підресурсом.	AAJ, AAX, AX, AJ
[RSG-15]	Параметри запиту ПОВИННІ використовуватися замість шляхів URL для отримання вкладених ресурсів.	AAJ, AAX, AX, AJ
[RSG-16]	Назви ресурсів МАЮТЬ бути іменниками для CRUD Web API і дієсловами для Intent Web API.	AAJ, AAX
[RSG-17]	Якщо назва ресурсу є іменником, він МАЄ завжди використовувати форм множини. НЕ МАЮТЬ використовуватися неправильні форми іменників. Наприклад, замість */people* слід використовувати */persons.*	AAJ, AAX
[RSG-18]	Назви ресурсів, сегментів і параметрів запитів ПОВИННІ складатися зі слів англійською мовою з використанням основного правопису, наведеного в Оксфордському словнику англійської мови. Назви ресурсів, які локалізуються за бізнес-вимогами, МОЖУТЬ бути іншими мовами.	AAJ, AAX, AX, AJ
[RSG-19]	Web API МАЄ використовувати для узгодження типу вмісту HTTP-заголовок запиту *Accept* і HTTP-заголовок відповіді *Content-Type*.	AAJ, AAX
[RSG-20]	Web API ПОВИНЕН підтримувати узгодження типів вмісту відповідно до IETF RFC 7231.	AAJ, AAX, AX, AJ
[RSG-21]	Формат JSON МАЄ передбачатися, якщо не запитується певний тип вмісту.	AAJ, AAX, AX, AJ
[RSG-22]	Web API МАЄ повертати код стану *"406 Not Acceptable*", якщо запитаний формат не підтримується.	AAJ, AAX
[RSG-23]	Web API МАЄ відхиляти запити, що містять неочікувані або відсутні заголовки типу вмісту з кодом стану HTTP *"406 Not Acceptable"* або *"415 Unsupported Media Type"*.	AAJ, AAX

[RSG-24]	Запити та відповіді (угода про іменування, формат повідомлень, структура даних і словник даних) МАЮТЬ посилатися на стандарт BOIB ST.96 для XML або стандарт BOIB ST.97 для JSON.	AAX, AAJ
[RSJ-25]	Назви властивостей об'єктів JSON МАЮТЬ бути вказані в нижньому регістрі (lowerCamelCase) наприклад, *applicantName.*	AAJ
[RSG-27]	Web API ПОВИНЕН підтримувати принаймі XML або JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Методи HTTP ПОВИННІ бути обмежені стандартними методами HTTP *POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE і HEAD*, як зазначено в IETF RFC 7231 і 5789.	AAJ, AAX, AX, AJ
[RSG-29]	HTTP-методи МОЖУТЬ слідувати принципу «pick-and-choice», який полягає в тому, що слід реалізовувати лише ту функціональність, яка необхідна для цільового сценарію використання.	AAJ, AAX
[RSG-30]	Деякі проксі-сервери підтримують тільки методи *POST* і *GET*. Щоб подолати ці обмеження, Web API МОЖЕ використовувати метод POST зі спеціальним HTTP- заголовком, який «тунелює» справжній HTTP- метод. МАЄ використовуватися спеціальний HTTP-заголовок X-HTTP-Method.	AAJ, AAX
[RSG-31]	Якщо метод HTTP не підтримується, МАЄ повернутися код стану HTTP *«405 Method Not Allowed».*	AAJ, AAX
[RSG-32]	Web API МАЄ підтримувати пакетні операції (вони ж масові операції) замість кількох окремих запитів для скорочення затримки. Така ж семантика повинна використовуватися для методів HTTP і кодів стану HTTP. У змістовній частині відповіді СЛІД мати інформацію щодо всіх пакетних операцій. Якщо виникає кілька помилок, змістовній частині помилки СЛІД мати інформацію про всі помилки (в атрибуті *details*). Всі пакетні операції МАЮТЬ виконуватися атомарно.	AAJ, AAX
[RSG-33]	Для кінцевої точки, яка отроимує один ресурс, якщо ресурс не знайдено, метод GET ПОВИНЕН повернути кодстану *404 Not Found".* Кінцеві точки, які повертають списки ресурсів, просто повертають порожній список.	AAJ, AAX, AX, AJ
[RSG-34]	Якщо ресурс отримано успішно, метод *GET* ПОВИНЕН повернути *"200 OK"*.	AAJ, AAX, AX, AJ
[RSG-35]	Запит *GET* ПОВИНЕН бути ідемпотентним (незалежним від кількості виконань запиту).	AAJ, AAX, AX, AJ
[RSG-36]	Коли довжина URI перевищує 255 байт, МАЄ використовуватися метод POST замість GET через обмеження GET, або створювати іменовані запити, якщо це можливо.	AAJ, AAX
[RSG-37]	Запит *HEAD* ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-38]	Деякі проксі-сервери підтримують тільки методи *POST* і *GET*. Web API МАЄ підтримувати спеціальний (користувацький) заголовок запиту HTTP для перевизначення методу HTTP, щоб подолати ці обмеження	AAJ, AAX
[RSG-39]	*POST*-запит НЕ ПОВИНЕН бути ідемпотентним відповідно до IETF RFC 2616.	AAJ, AAX, AX, AJ
[RSG-40]	Якщо створення ресурсу було успішним, HTTP-заголовок Location МАЄ містити URI (абсолютний або відносний), що вказує на створений ресурс.	AAJ, AAX
[RSG-41]	Якщо створення ресурсу пройшло успішно, відповідь МАЄ містити код стану *«201 Created».*	AAJ, AAX

[RSG-42]	Якщо створення ресурсу пройшло успішно, в корисній частині відповіді МАЄ за замовчуванням міститися тіло створеного ресурсу, щоб клієнт міг використовувати його без додаткового виклику HTTP.	AAJ, AAX
[RSG-43]	Запит *PUT* ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-44]	Якщо ресурс не знайдений, *PUT* ПОВИНЕН повернути код стану *«404 Not Found».*	AAJ, AAX, AX, AJ
[RSG-45]	Якщо ресурс успішно оновлений, *PUT* ПОВИНЕН повернути код стану *«200 OK»,* якщо оновлений ресурс буде повернуто, або «204 No Content», якщо його не буде повернуто.	AAJ, AAX, AX, AJ
[RSG-46]	Запит *PATCH* НЕ ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-47]	Якщо Web API реалізує часткові оновлення, то ідемпотентні характеристики *PATCH* МАЮТЬ прийматися до уваги. Для того щоб зробити його ідемпотентним, API МОЖЕ дотримуватися пропозиції IETF RFC 5789 про використання оптимістичного блокування.	AAJ, AAX
[RSG-48]	Якщо ресурс не знайдений, *PATCH* ПОВИНЕН поверути код стану *«404 Not Found».*	AAJ, AAX, AX, AJ
[RSJ-49]	Якщо Web API реалізує часткові оновлення за допомогою *PATCH*, він ПОВИНЕН використовувати формат *JSON Merge Patch* для опису часткового набору змін, як описано в IETF RFC 7386, використовуючи тип вмісту *application/merge-patch+json.*	AAJ, AJ
[RSG-50]	Запит *DELETE* НЕ ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-51]	Якщо ресурс не знайдено, *DELETE* ПОВИНЕН повернути код стану HTTP *«404 Not Found».*	AAJ, AAX, AX, AJ
[RSG-52]	Якщо ресурс видалено успішно, *DELETE* ПОВИНЕН повернути статус *«200 OK»,* якщо видалений ресурс повернуто, або *«204 No Content*», якщо його не повернуто.	AAJ, AAX, AX, AJ
[RSG-53]	Кінцевим одержувачем є або вихідний сервер, або перший проксі чи шлюз, який отримав у запиті значення Max-Forwardsґ, що дорівнює нулю. Запит *TRACE* НЕ ПОВИНЕН включати тіло (запиту).	AAJ, AAX, AX, AJ
[RSG-54]	Запит *TRACE* НЕ ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-55]	Значення поля заголовка *«Via»* HTTP ПОВИННЕ служити для відстеження ланцюжка запитів.	AAJ, AAX, AX, AJ
[RSG-56]	Поле Max-Forwards HTTP заголовка ПОВИННО використовуватися, щоб дозволити клієнту обмежити довжину ланцюжка запитів.	AAJ, AAX, AX, AJ
[RSG-57]	Якщо запит коректний, відповідь ПОВИННА містити повне повідомлення запиту в тілі відповіді, з Content-Type *«message/http».*	AAJ, AAX
[RSG-58]	Відповіді на *TRACE* НЕ ПОВИННІ кешуватися.	AAJ, AAX, AX, AJ
[RSG-59]	Код стану *«200 OK»* МАЄ повернутися в *TRACE.*	AAJ, AAX
[RSG-60]	Запит *OPTIONS* ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-61]	НЕ МАЮТЬ використовуватися нестандартні (користувацькі) HTTP-заголовки, що починаються з префікса *"X-"*.	AAJ, AAX

[RSG-62]	Користувацькі HTTP-заголовки НЕ МАЮТЬ використовувати для зміни поведінки HTTP-методів, за винятком випадків, коли це необхідно для усунення існуючих технічних обмежень (наприклад, див. [RSG-39]).	AAJ, AAX
[RSG-63]	Угода про імена для користувацьких HTTP заголовків - <організація>-<ім'я заголовка> *(<organization>-<header name>*), де поля <організація> і <заголовок> МАЮТЬ відповідати угоді про kebab-case стиль іменування.	AAJ, AAX
[RSG-64]	Web API МАЄ підтримувати єдиний метод версіонування сервісу з використанням версіонування URI, наприклад `/api/v1/inventors` або версіонування заголовка, наприклад `Accept-version:v1` або версіонування типу даних, наприклад `Accept: application/vnd.v1+json`. НЕ МАЄ використовуватися управління версіями рядка запиту.	AAJ, AAX
[RSG-65]	МАЄ дотримуватися схеми нумераці та управління версіями версій, враховуючи лише основний номер версії (наприклад, `/v1`).	AAJ, AAX
[RSG-66]	Сервісні контракти API МОЖУТЬ включати функцію перенаправлення кінцевих точок. Коли споживач послуг намагається викликати послугу, може бути повернута відповідь про перенаправлення, щоб повідомити споживачу послуг про необхідність повторно надіслати запит на нову кінцеву точку. Перенаправлення МОЖУТЬ бути тимчасовими або постійними: – Тимчасове перенаправлення - з використанням заголовка HTTP-відповіді Location і коду стану HTTP «302 Found» відповідно до IETF RFC 7231; або – Постійне перенаправлення - з використанням заголовка HTTP-відповіді Location і коду стану HTTP «301 Moved Permanently» відповідно до IETF RFC 7238.	AAJ, AAX
[RSG-67]	Розробники МАЮТЬ публікувати стратегії життєвого циклу API , щоб допомогти користувачам зрозуміти, як довго буде підтримуватися та чи інша версія.	AAJ, AAX
[RSG-68]	Web API МАЄ підтримувати пагінацію (розбивку на сторінки).	AAJ, AAX
[RSG-69]	Пагіновані запити НЕ МОЖУТЬ бути ідемпотентними.	AAJ, AAX
[RSG-70]	Web API ПОВИНЕН використовувати параметри запиту для реалізації пагінації.	AAJ, AAX, AX, AJ
[RSG-71]	Web API НЕ ПОВИНЕН використовувати HTTP- заголовки для реалізації пагінації.	AAJ, AAX, AX, AJ
[RSG-72]	МАЮТЬ використовуватися параметри запиту `limit=<кількість елементів для доставки>` *і* `offset=<кількість елементів для пропуску>` (`limit=<number of items to deliver>` and `offset=<number of items to skip>`), де limit - кількість елементів, що повертаються (розмір сторінки), а пропуск (skip) – кількість елементів, які повинні бути пропущені (зміщення). Якщо обмеження на розмір сторінки не вказано, МАЄ визначатися значення за замовчуванням - глобальне або для кожної колекції; зміщення за замовчуванням ПОВИННО дорівнювати нулю «0»: Наприклад, наступний URL є допустимим: https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73]	Значення параметрів limit і skip МАЮТЬ бути включені у відповідь.	AAJ, AAX
[RSG-74]	Web API ПОВИНЕН підтримувати сортування.	AAJ, AAX

[RSG-75]	Для того, щоб задати багатоатрибутивний критерій сортування, ПОВИНЕН використовуватися параметр запиту. Значенням цього параметра є список ключів сортування, розділених комами, до кожного ключа сортування МОЖНА додавати напрямок сортування за зростанням або за спаданням, відокремлений двокрапкою ':'. Напрямок за замовчуванням ПОВИНЕН бути визначений сервером, якщо напрямок сортування не вказано для ключа.	AAJ, AAX, AX, AJ
[RSG-76]	Web API МАЄ повертати критерії сортування у відповіді.	AAJ, AAX, AX, AJ
[RSG-77]	Web API МОЖЕ підтримувати розширення тіла вмісту, що повертається. МАЄ використовувати параметр запиту `expand=<розділений комами список імен атрибутів>`.	AAJ, AAX
[RSG-78]	Параметр запиту МАЄ використовувати замість шляхів URL у разі, якщо Web API підтримує проєкцію у форматі: «`fields=«<розділений комами список імен атрибутів>`.	AAJ, AAX
[RSG-79]	Web API ПОВИНЕН підтримувати повернення кількості елементів у колекції.	AAJ, AAX, AX, AJ
[RSG-80]	Параметр запиту ПОВИНЕН використовуватися для підтримки повернення кількості елементів у колекції.	AAJ, AAX, AX, AJ
[RSG-81]	МАЄ використовуватися параметр запиту *count* для повернення кількості елементів у колекції.	AAJ, AAX
[RSG-82]	Web API МОЖЕ підтримувати повернення кількості елементів у колекції всередині, тобто як частину відповіді, яка містить саму колекцію. При цьому ПОВИНЕН використовуватися параметр запиту.	AAJ, AAX, AX, AJ
[RSG-83]	МАЄ використовуватися параметр запиту `count=true`. Якщо він не вказаний, то за замовчуванням *count* слід встановити в *false*.	AAJ, AAX
[RSG-84]	Якщо Web API підтримує пагінацію, то МАЄ підтримуватися повернення всередині відповіді розміру колекції (тобто загальної кількості елементів колекції).	AAJ, AAX
[RSG-85]	Якщо Web API підтримує складні пошукові вирази, МАЄ вказуватися мова запитів, наприклад CQL (*Contextual Query Language).*	AAJ, AAX
[RSG-86]	Сервісний контракт ПОВИНЕН визначати підтримувану граматику (наприклад, поля, функції, ключові слова та оператори).	AAJ, AAX, AX, AJ
[RSG-87]	ПОВИНЕН використовуватися параметр запиту *«q».*	AAJ, AAX, AX, AJ
[RSG-88]	На рівні протоколу, Web API ПОВИНЕН повертати відповідний код стану HTTP, вибраний із списку стандартних кодів стану HTTP.	AAJ, AAX, AX, AJ
[RSJ-89]	На рівні програми Web API ПОВИНЕН повертати частину, що повідомляє про помилку з достатнім ступенем деталізації. Атрибути `code` *і* `message` є обов'язковими, атрибут *details* є умовно обов'язковим, атрибути `target`, `status`, `moreInfo` *та* `internalMessage` є необов'язковими.	AAJ, AAX, AX, AJ
[RSG-90]	Помилки НЕ ПОВИННІ розкривати критично важливі для безпеки дані або внутрішні технічні деталі, такі як стеки викликів у повідомленнях про помилки.	AAJ, AAX, AX, AJ
[RSG-91]	Заголовок HTTP «`Reason-Phrase`» (описаний в RFC 2616) НЕ ПОВИНЕН використовуватися для передачі повідомлень про помилки.	AAJ, AAX, AX, AJ

[RSG-92]	Кожна зареєстрована помилка МАЄ отримати унікальний ідентифікатор кореляції. СЛІД використовувати користувацький HTTP-заголовок, якому СЛІД мати ім'я *Correlation-ID.*	AAJ, AAX
[RSG-93]	Формат Службового контракту ПОВИНЕН містити наступне: – Версія API; – Інформація про семантику елементів API; – Ресурси; – Атрибути ресурсу; – Параметри запиту; – Методи; – Типи носіїв; – Граматика пошуку (якщо вона підтримується); – Коди стану HTTP; – Методи HTTP; – Обмеження та відмінні риси; і – Безпека (якщо є).	AAJ, AAX, AX, AJ
[RSG-94]	Формат сервісного контракту МАЄ включати запити та відповіді у вигляді XML-схеми або JSON-схеми, а також приклади використання API у підтримуваних форматах, тобто XML або JSON.	AAJ, AAX
[RSG-95]	REST API ПОВИНЕН надавати документацію API у вигляді сервісного контракту.	AAJ, AAX, AX, AJ
[RSG-96]	Реалізація Web API, що відхиляється від цього стандарту, ПОВИННА бути чітко задокументована в сервісному контракті. Якщо в сервісному контракті не вказано правило відхилення, ПОВИННО вважатися, що дотримується цей Стандарт.	AAJ, AAX, AX, AJ
[RSG-97]	Сервісний контракт ПОВИНЕН надавати можливість генерувати скелетний код клієнта API.	AAJ, AAX, AX, AJ
[RSG-98]	Сервісний контракт МАЄ дозволяти генерацію скелетного (каркасного) коду сервера.	AAJ, AAX
[RSG-99]	Документація до Web API МАЄ бути написана у форматі RAML або OAS. Користувацькі формати документації НЕ МАЮТЬ використовуватися.	AAJ, AAX
[RSG-100]	У споживача Web API МАЄ бути можливість вказати час очікування сервера для кожного запиту; МАЄ використовуватися спеціальний HTTP-заголовок. Максимальний час очікування сервера МАЄ також використовуватися для захисту ресурсів сервера від надмірного використання.	AAJ, AAX
[RSG-101]	Web API МАЄ підтримувати умовне отримання даних, щоб гарантувати, що будуть отримані тільки ті дані, які були змінені. МАЄ використовуватися перевірка ресурсів на основі вмісту, оскільки вона є більш точною.	AAJ, AAX
[RSG-102]	Для реалізації *Content-based Resource Validation* (перевірка коректності ресурсу за вмістом) HTTP-заголовок *ETag* СЛІД використовувати у відповіді для кодування стану даних. Після цього це значення СЛІД використовувати в наступних запитах в умовних HTTP-заголовках (таких як *If-Match* або *If-None-Match* - Якщо відповідає або Якщо не відповідає). Якщо дані не були змінені після того, як запит повернув *ETag,* сервер МАЄ повернути код стану *«304 Not Modified»* (якщо дані не змінено). Цей механізм описано в IETF RFC 7231 и 7232.	AAJ, AAX
[RSG-103]	Для того, щоб реалізувати перевірку ресурсів за часом, МАЄ використовуватися останній змінений HTTP-заголовок (*Last-Modified*). Цей механізм описаний в IETF RFC 7231 і 7232.	AAJ, AAX

[RSG-104]	Використовуючи версійність відповіді, споживач послуг МОЖЕ реалізувати оптимістичне блокування.	AAJ, AAX
[RSG-105]	Web API ПОВИНЕН підтримувати кешування результатів *GET;* Web API МОЖЕ підтримувати кешування результатів інших методів HTTP.	AAJ, AJ, AX
[RSG-106]	МАЮТЬ використовуватися HTTP-заголовки відповіді *Cache-Control* і *Expires*. Останній МОЖЕ використовуватися для підтримки старих клієнтів.	AAJ, AAX
[RSG-107]	Web API МАЄ повідомляти, чи підтримується часткове завантаження файлів, відповідаючи на запити *HEAD* і надаючи заголовки HTTP-відповіді *Accept-Ranges і Content-Length.*	AAJ, AAX
[RSG-108]	Web API МАЄ підтримувати завантаження файлів частинами. Необхідно підтримувати діапазон з декількох частин.	AAJ, AAX
[RSG-109]	Web API МАЄ повідомляти, чи підтримує він заливку файлів частинами.	AAJ, AAX
[RSG-110]	Web API МАЄ підтримувати заливку файлів частинами. Діапазони, що складаються з декількох частин, ПОВИННІ підтримуватися.	AAJ, AAX
[RSG-111]	Постачальнику послуг МАЄ повертати в заголовках відповіді HTTP заголовок «`413 Request Entity Too Large`» у випадку, якщо запит перевищив максимально допустимий ліміт. Для вказівки максимального розміру запиту МОЖНА використовувати спеціальний HTTP-заголовок.	AAJ, AAX
[RSG-112]	Якщо Web API підтримує обробку налаштувань, вона МАЄ реалізуватися відповідно до IETF RFC 7240, тобто слід використовувати HTTP-заголовок запиту `Prefer`, у HTTP-заголовку відповіді МАЄ повернутися `Preference-Applied` (з повторенням вихідного запиту).	AAJ, AAX
[RSG-113]	Якщо Web API підтримує обробку налаштувань, номенклатура налаштувань, які МОЖУТЬ бути встановлені за допомогою заголовка `Prefer`, ПОВИННА бути записана в сервісному контракті.	AAJ, AAX, AJ, AX
[RSG-114]	Якщо Web API підтримує локалізовані дані, HTTP-заголовок запиту `Accept-Language` ПОВИНЕН підтримуватися, щоб вказати набір природних мов, яким надається перевага у відповіді, як зазначено в IETF RFC 7231.	AAJ, AAX, AJ, AX
[RSG-115]	Якщо API підтримує довгострокові операції, то вони МАЮТЬ бути асинхронними. МАЄ застосовуватися наступний підхід: a.Споживач сервісу активує сервісну операцію; b. Операція сервісу повертає код стану *«202Accepted»* відповідно до IETF RFC 7231 (розділ 6.3.3), тобто запит було прийнято до обробки, але обробка не була завершена. Місце розташування створеного завдання в черзі також повертається з HTTP-заголовком *Location*; і c. Споживач послуги звертається до місця розташування, що повертається, щоб дізнатися, чи доступний ресурс. Якщо ресурс недоступний, у відповіді МАЄ бути код стану*»200 OK»*, міститись статус завдання (наприклад, очікування) та МОЖЕ міститися інша інформація (наприклад, індикатор виконання та/або посилання для відміни або видалення завдання з допомогою HTTP- методу *DELETE*). Якщо ресурс доступний, у відповіді МАЄ наводитись код стану *«303 See Other»,* а в заголовку HTTP *Location* МАЄ наводитись URL для отримання результатів задачі.	AAJ, AAX
[RSG-116]	Конфіденційність: різні API та інформація в API ПОВИННІ бути ідентифіковані, класифіковані та захищені від несанкціонованого доступу, розкриття та перехоплення в будь-який час. Повинні дотримуватися принципи найменших привілеїв, нульової довіри, необхідності знати та необхідності ділитися.	AAJ, AAX, AJ, AX

[RSG-117]	Забезпечення цілісності: API та інформація про API ПОВИННІ бути захищені від несанкціонованої модифікації, дублювання, пошкодження та знищення. Інформація ПОВИННА модифікуватися за допомогою затверджених транзакції та інтерфейсів. Системи ПОВИННІ оновлюватися з використанням затверджених процесів управління конфігурацією, управління змінами та управління виправленнями.	AAJ, AAX, AJ, AX
[RSG-118]	Доступність: API та інформація про API ПОВИННІ бути доступними для авторизованих користувачів у встановлений час відповідно до угоди про рівень обслуговування (SLA), політики контролю доступу та визначеними бізнес-процесами.	AAJ, AAX, AJ, AX
[RSG-119]	Безвідмовність: Кожна транзакція, що обробляється або виконується за допомогою API, ПОВИННА забезпечувати неможливість відмови через реалізацію належного аудиту, авторизації, автентифікації та впровадження безпечних шляхів, а також послуг і механізмів, що не допускають відмови.	AAJ, AAX, AJ, AX
[RSG-120]	Автентифікація, авторизація, аудит: Користувачі, системи, API або пристрої, залучені до критичних транзакцій або дії, ПОВИННІ бути автентифіковані, авторизовані за допомогою служб контролю доступу на основі ролей або атрибутів та підтримувати поділ обов'язків. Крім того, всі дії ПОВИННІ реєструватися, а надійність автентифікації повинна збільшуватися з відповідним інформаційним ризиком.	AAJ, AAX, AJ, AX
[RSG-121]	При розробці API ПОТРІБНО ретельно враховувати загрози, випадки шкідливого використання, методи безпечного кодування, безпека транспортного рівня та тестування безпеки, особливо: - *PUTs і POSTs* – тобто: які зміни у внутрішніх даних можуть бути потенційно використані для атак або дезінформації; - *DELETEs* – тобто: можна використовувати для видалення вмісту внутрішнього сховища ресурсів; - Білий список дозволених методів - щоб гарантувати, що дозволені HTTP-методи належним чином обмежені, в той час як інші повертатимуть правильний код відповіді; і - Добре відомі атаки повинні бути враховані на етапі моделювання загроз у процесі проєктування, щоб не допустити збільшення ризику загроз. Загрози та заходи щодо їх усунення, визначені в OWASP *Top Ten Cheat Sheet*, ПОВИННІ бути взяті до уваги.	AAJ, AAX, AJ, AX
[RSG-122]	При розробці API слід дотримуватись стандартів та кращих практик, перелічених нижче: - Кращі практики безпечного кодування: Принципи безпечне кодування OWASP; - Безпека Rest API: Пам'ятка з безпеки REST; - Захист від несанкціонованого доступу та захист від міжсайтового скриптингу: OWASP XSS Cheat Sheet; - Запобігання SQL-ін'єкцій: Пам'ятка OWASP щодо SQL-ін'єкцій, пам'ятка OWASP щодо *Parameterization*; і - Безпека транспортного рівня: Пам'ятка OWASP із захисту транспортного рівня.	AAJ, AAX, AJ, AX
[RSG-123]	Тестування безпеки та оцінка вразливостей ПОВИННІ проводитися для забезпечення безпеки та стійкості API до загроз. Ця вимога МОЖЕ бути виконана шляхом використання статичного та динамічного тестування безпеки додатків (SAST/DAST), автоматизованих інструментів управління вразливістю та тестування на проникнення.	AAJ, AAX, AJ, AX
[RSG-124]	Захищені сервіси ПОВИННІ надавати тільки кінцеві точки HTTPS, що використовують TLS 1.2 або вище, з набором шифрів, який включає ECDHE для обміну ключами.	AAJ, AAX, AJ, AX

[RSG-125]	Під час розгляду протоколів автентифікації для забезпечення безпеки транспортування РЕКОМЕНДОВАНО дотримання секретності передачі. Використання небезпечних криптографічних алгоритмів і зворотньої сумісності із SSL 3 та TLS 1.0/1.1 НЕ МАЮТЬ БУТИ дозволені.	AAX, AAJ
[RSG-126]	Для забезпечення максимального рівня безпеки і довіри, НЕОБХІДНО встановити з`єднання через VPN IPSEC між сайтами для додаткового захисту інформації, що передається через незахищені мережі.	AAX, AAJ
[RSG-127]	Користувацькому додатку МАЮТЬ перевірятися ланцюжки сертифікатів TLS під час виконання запитів до захищених ресурсів, включно з перевіркою списку відкликання сертифікатів.	AAX, AAJ
[RSG-128]	Захищені служби МАЮТЬ використовувати тільки дійсні сертифікати, які видані довіреним центром сертифікації (ЦС).	AAX, AAJ
[RSG-129]	Токени МАЮТЬ підписуватися з використанням безпечних алгоритмів підпису, що відповідають стандарту цифрового підпису (DSS) FIPS-186-4. Слід розглядати алгоритм цифрового підпису RSA або алгоритм ECDSA	AAX, AAJ
[RSG-130]	Анонімна автентифікація ПОВИННА використовуватися лише тоді, коли клієнти та програми, якими вони користуються, отримують доступ до інформації або функцій з низьким рівнем чутливості, які не потребують автентифікації, наприклад, загальнодоступної інформації.	AAJ, AAX, AJ, AX
[RSG-131]	НЕ ПОВИННА дозволятися автентифікація за допомогою імені користувача та пароля або хеш-пароля.	AAJ, AAX, AJ, AX
[RSG-132]	Якщо сервіс захищений, МАЄ використовуватися *Open ID Connect.*	AAX, AAJ
[RSG-133]	Якщо використовується JSON Web Token (JWT), секретний код JWT ПОВИНЕН володіти високою ентропією, щоб збільшити коефіцієнт роботи атаки грубої сили; TTL і RTTL токенів повинні бути якомога коротшими; і конфіденційна інформація НЕ МАЄ зберігатися в корисному навантаженні JWT.	AAX, AAJ
[RSG-134]	У `POST/PUT`-запитах конфіденційні дані МАЮТЬ передаватися в тілі запиту або в заголовках запиту.	AAX, AAJ
[RSG-135]	У запитах `GET` конфіденційні дані МАЮТЬ передаватися в заголовку HTTP.	AAX, AAJ
[RSG-136]	Щоб мінімізувати затримку і зменшити зв'язок між захищеними сервісами, рішення про контроль доступу МАЮТЬ прийматися локально кінцевими точками REST.	AAX, AAJ
[RSG-137]	API-ключі МАЮТЬ використовувати для захищених та публічних сервісів для запобігання перевантаженню постачальника послуг множинними запитами (атаки типу «відмова в обслуговуванні»). Для захищених сервісів API-ключі МОЖУТЬ використовуватися для монетизації (придбані тарифні плани), застосування політики використання (QoS) та моніторингу.	AAX, AAJ
[RSG-138]	Ключі API МОЖНА комбінувати із заголовком HTTP-запиту user-agent, щоб відрізнити користувача-людину від програмного агента, як зазначено в IETF RFC 7231.	AAX, AAJ
[RSG-139]	Постачальник послуг МАЄ повертати разом із заголовками HTTP-відповіді поточний стан використання. У відповідь МОЖУТЬ бути повернуті наступні дані: - ліміт швидкості (rate limit) - ліміт швидкості (за хвилину), встановлений у системі; - rate limit remaining - кількість запитів, що залишилось, дозволених	AAX, AAJ

	протягом поточного часового інтервалу (-1 означає, що ліміт перевищено); та - скидання ліміту швидкості (rate limit reset) - час (у секундах), що залишився до обнулення лічильника запитів.	
[RSG-140]	Постачальник послуг ПОВИНЕН повертати код статусу «`429 Too Many Requests`», якщо запити надходять занадто швидко.	AAX, AAJ
[RSG-141]	Ключі API ПОВИННІ бути відкликані, якщо клієнт порушує угоду про використання, як це визначено у BIB.	AAJ, AAX, AJ, AX
[RSG-142]	Ключі API СЛІД передавати за допомогою користувацьких HTTP-заголовків. Вони НЕ МАЮТЬ передаватись за допомогою параметрів запиту.	AAX, AAJ
[RSG-143]	Ключі API МАЮТЬ бути згенеровані випадковим чином.	AAX, AAJ
[RSG-144]	Безпечні та надійні сертифікати ПОВИННІ випускатися взаємно довіреним центром сертифікації (ЦС) за допомогою процесу встановлення довіри або перехресної сертифікації.	AAJ, AAX, AJ, AX
[RSG-145]	Сертифікати, спільні для клієнта і сервера, ПОВИННІ використовуватися для зменшення ризиків безпеки ідентифікації, особливо для чутливих систем і привілейованих дій, наприклад, X.509.	AAJ, AAX, AJ, AX
[RSG-146]	Для високопривілейованих служб у двосторонній взаємній автентифікації між клієнтом і сервером СЛІД використовувати сертифікати для забезпечення додаткового захисту.	AAX, AAJ
[RSG-147]	Багатофакторна автентифікація ПОВИННА бути впроваджена для зменшення ризиків ідентифікації для додатків з високим профілем ризику, систем, що обробляють дуже конфіденційну інформацію або привілейовані дії.	AAX, AAJ
[RSG-148]	Якщо REST API є публічним, HTTP-заголовок *Access-Control-Allow-Origin* ПОВИНЕН мати значення '*'.	AAJ, AAX, AJ, AX
[RSG-149]	Якщо REST API захищений, НЕОБХІДНО використовувати *CORS*, якщо це можливо. В іншому випадку, JSONP МОЖЕ використовуватися як запасний варіант, але тільки для *GET*-запитів, наприклад, коли користувач отримує доступ за допомогою старого браузера. НЕ СЛІД використовувати *Iframe.*	AAX, AAJ
[RSJ-150]	При використанні екземплярів, що описані схемою, згідно з RFC8288, для надання посилання на завантажувану JSON-схему СЛІД використовувати заголовок Link.	AAJ
[RSJ-151]	Web API ПОВИНЕН реалізовувати щонайменше рівень 2 (властивості нативних транспортних засобів) RMM. Рівень 3 (гіпермедіа) МОЖЕ бути реалізований, щоб зробити API повністю відкритим.	AAJ
[RSJ-152]	Для створення власного гіпермедійного формату ПОВИНЕН використовуватися наступний набір атрибутів, вкладений у посилання на атрибут: – `href` – цільовий URI (Uniform Resource Identifier); – `rel` – значення цільового URI – `self` – URI посилається на сам ресурс; – `next` – URI посилається на попередню сторінку (якщо використовується під час пагінації); – `previous` – URI посилається на наступну сторінку (якщо використовується під час пагінації); - довільне ім'я `v` позначає користувацьке значення відношення.	AAJ

Таблиця 4: Рівень відповідності AAX

Ідентифікатор правила	Правило	Перехресні посилання та примітки
[RSG-01]	Символ скісної риски «/» ПОВИНЕН використовуватися в шляху до URI для позначення ієрархічного зв'язку між ресурсами, але шлях НЕ ПОВИНЕН закінчуватися скісною рискою, оскільки він не надає жодного семантичного значення і може призвести до плутанини.	AAJ, AAX, AX, AJ
[RSG-02]	Назви ресурсів ПОВИННІ дотримуватись свого зразка іменування.	AAJ, AAX, AJ, AX
[RSG-03]	В назвах ресурсів ПОВИННІ використовуватися малі або великі літери. Назви ресурсів МОЖНА скорочувати.	AAJ, AAX
[RSG-05]	Параметри запиту ПОВИННІ використовувати нижній регістр. Параметр запиту МОЖЕ бути скороченим.	AAJ, AAX
[RSG-06]	Шаблон URL для WEB API ПОВИНЕН містити слово «api» в URI.	AAJ, AAX, AX, AJ
[RSG-07]	Матричні параметри НЕ ПОВИННІ використовуватися.	AAJ, AAX, AX, AJ
[RSG-08]	Web API ПОВИНЕН послідовно застосовувати коди стану HTTP, як описано в IETF RFC	AAJ, AAX, AX, AJ
[RSG-09]	Для класифікування помилки в WEB API НЕОБХІДНО використовувати рекомендовані коди в Додатку V.	AAX, AAJ
[RSG-10]	Якщо API виявляє невірні вхідні значення, він ПОВИНЕН повернути код стану HTTP *«400 Bad Request».* Корисне навантаження помилки ПОВИННО вказувати на помилкове значення.	AAJ, AAX, AX, AJ
[RSG-11]	Якщо API виявляє синтаксично правильні імена аргументів (у запиті або параметрах запиту), які не є очікуваними, він ПОВИНЕН їх ігнорувати.	AAJ, AAX
[RSG-12]	Якщо API виявляє допустимі значення, які вимагають, щоб функції не були реалізовані, він ПОВИНЕН повернути HTTP-код статусу *«501 Не реалізовано».* Корисне навантаження помилки ПОВИННО вказувати на необроблене значення.	AAJ, AAX, AX, AJ
[RSG-13]	Web API ПОВИНЕН використовувати тільки ресурси верхнього рівня. Якщо є підресурси, вони повинні бути колекціями і передбачати асоціацію. Сутність має бути доступною як ресурс верхнього рівня або як підресурс, але не можна використовувати обидва способи.	AAJ, AAX
[RSG-14]	Якщо ресурс може бути самостійним, то він ПОВИНЕН бути ресурсом верхнього рівня, або інакше – підресурсом.	AAJ, AAX, AX, AJ
[RSG-15]	Параметри запиту ПОВИННІ використовуватися замість шляхів URL для отримання вкладених ресурсів.	AAJ, AAX, AX, AJ
[RSG-16]	Назви ресурсів МАЮТЬ бути іменниками для CRUD Web API і дієсловами для Intent Web API.	AAJ, AAX
[RSG-17]	Якщо назва ресурсу є іменником, вона повинна завжди	AAJ, AAX

	використовуватися у формі множини. НЕ МОЖНА використовувати неправильні форми іменників. Наприклад, замість */people* слід використовувати */persons.*	
[RSG-18]	Назви ресурсів, сегментів і параметрів запитів ПОВИННІ складатися зі слів англійською мовою, з використанням основного англійського написання, наведеного в Оксфордському словнику англійської мови. Назви ресурсів, які локалізуються через бізнес-вимоги, МОЖУТЬ бути іншими мовами.	AAJ, AAX, AX, AJ
[RSG-19]	Web API ПОВИНЕН використовувати для узгодження типу контенту HTTP-заголовок запиту *Accept* і HTTP-заголовок відповіді *Content-Type*.	AAJ, AAX
[RSG-20]	Web API ПОВИНЕН підтримувати узгодження типів вмісту відповідно до IETF RFC 7231.	AAJ, AAX, AX, AJ
[RSG-21]	Формат JSON ПОВИНЕН використовуватися, якщо не запитується певний тип вмісту.	AAJ, AAX, AX, AJ
[RSG-22]	Web API ПОВИНЕН повертати код стану *«406 Not Acceptable»*, якщо запитаний формат не підтримується.	AAJ, AAX
[RSG-23]	Web API ПОВИНЕН відхиляти запити, що містять несподівані або відсутні заголовки типу вмісту, видаючи кодом стану HTTP *«406 Not Acceptable» або «415 Unsupported Media Type».*	AAJ, AAX
[RSG-24]	Запити та відповіді (угода про імена, формат повідомлень, структура даних і словник даних) ПОВИННІ відповідати стандарту BOIB ST.96 для XML або стандарту BOIB ST.97 для JSON.	AAX
[RSX-26]	Компоненти XML ПОВИННІ бути представлені у верхньому регістрі відповідно до стандарту BOIB ST.96.	AAX
[RSG-27]	Web API ПОВИНЕН підтримувати принаймні формат XML або JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Методи HTTP ПОВИННІ бути обмежені стандартними методами HTTP *POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE і HEAD*, як зазначено в IETF RFC 7231 і 5789.	AAJ, AAX, AX, AJ
[RSG-29]	Методи HTTP МОЖУТЬ слідувати принципу «pick-and-choice», який полягає в тому, що слід реалізовувати лише ту функціональність, яка необхідна для цільового сценарію використання.	AAJ, AAX
[RSG-30]	Деякі проксі-сервери підтримують тільки методи *POST і GET*. Щоб подолати ці обмеження,Web API МОЖЕ використовувати метод *POST* з користувацьким заголовком HTTP, що «тунелює» справжній метод HTTP. НЕОБХІДНО використовувати користувацький HTTP-*заголовок X-HTTP-Method.*	AAJ, AAX
[RSG-31]	Якщо метод HTTP не підтримується, НЕОБХІДНО повернути код стану HTTP *«405 Method Not Allowed».*	AAJ, AAX
[RSG-32]	Web API ПОВИНЕН підтримувати пакетні операції (також відомі як масові операції) замість кількох окремих запитів, щоб зменшити затримку. Однакова семантика повинна використовуватися для методів HTTP і кодів стану HTTP. Корисне навантаження відповіді ПОВИННО містити інформацію про всі пакетні операції. Якщо	AAJ, AAX

	виникає декілька помилок, корисне навантаження помилки ПОВИННО містити інформацію про всі випадки (в атрибуті details). Всі масові операції ПОВИННІ виконуватися в атомарній операції.	
[RSG-33]	Для кінцевої точки, яка отримує один ресурс, якщо ресурс не знайдено, метод GET ПОВИНЕН повернути код стану *«404 Not Found».* Кінцеві точки, які повертають списки ресурсів, просто повертають порожній список.	AAJ, AAX, AX, AJ
[RSG-34]	Якщо ресурс отримано успішно, метод *GET* ПОВИНЕН повернути *200 OK.*	AAJ, AAX, AX, AJ
[RSG-35]	*GET* -запит ПОВИНЕН бути ідемпотентним (незалежним від кількості виконань запиту)..	AAJ, AAX, AX, AJ
[RSG-36]	Якщо довжина URI перевищує 255 байт, СЛІД використовувати метод *POST* замість *GET* через обмеження *GET*, або створювати іменовані запити, якщо це можливо.	AAJ, AAX
[RSG-37]	Запит *HEAD* ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-38]	Деякі проксі-сервери підтримують тільки методи *POST* і *GET*. Web API ПОВИНЕН підтримувати спеціальний заголовок HTTP-запиту, щоб перевизначити метод HTTP, щоб подолати ці обмеження.	AAJ, AAX
[RSG-39]	*POST* - запит НЕ ПОВИНЕН бути ідемпотентним відповідно до IETF RFC 2616.	AAJ, AAX, AX, AJ
[RSG-40]	Якщо створення ресурсу було успішним, HTTP-заголовок *Location* ПОВИНЕН містити URI (абсолютний або відносний), який вказує на створений ресурс.	AAJ, AAX
[RSG-41]	Якщо створення ресурсу пройшло успішно, відповідь ПОВИННА містити код стану *«201 Created».*	AAJ, AAX
[RSG-42]	Якщо створення ресурсу пройшло успішно, корисне навантаження відповіді ПОВИННО за замовчуванням містити тіло створеного ресурсу, щоб дозволити клієнту використовувати його без додаткового HTTP-виклику.	AAJ, AAX
[RSG-43]	Запит *PUT* ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-44]	Якщо ресурс не знайдено, *PUT* ПОВИНЕН повернути код стану *«404 Not Found».*	AAJ, AAX, AX, AJ
[RSG-45]	Якщо ресурс оновлено успішно, *PUT* ПОВИНЕН повернути код стану *«200 OK»,* якщо оновлений ресурс повернуто, або *«204 No Content»*, якщо його не повернуто.	AAJ, AAX, AX, AJ
[RSG-46]	Запит *PATCH* НЕ ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-47]	Якщо Web API реалізує часткові оновлення, то ідемпотентні характеристики *PATCH* СЛІД брати до уваги. Для того щоб зробити його ідемпотентним, API МОЖЕ слідувати рекомендаціям IETF RFC 5789 про використання оптимістичного блокування.	AAJ, AAX

[RSG-48]	Якщо ресурс не знайдено, *PATCH* ПОВИНЕН повернути код стану *«404 Not Found»*.	AAJ, AAX, AX, AJ
[RSG-50]	Запит *DELETE* НЕ ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-51]	Якщо ресурс не знайдено, *DELETE* ПОВИНЕН повернути код стану *«404 Not Found».*	AAJ, AAX, AX, AJ
[RSG-52]	Якщо ресурс видалено успішно*, DELETE* ПОВИНЕН повернути статус *«200 OK»,* якщо видалений ресурс повертається, або *«204 No Content»*, якщо його не повернуто.	AAJ, AAX, AX, AJ
[RSG-53]	Кінцевим одержувачем є або вихідний сервер, або перший проксі чи шлюз, який отримав у запиті значення *Max-Forwards* рівне нулю. Запит *TRACE* НЕ ПОВИНЕН містити тіло (запиту).	AAJ, AAX, AX, AJ
[RSG-54]	Запит *TRACE* НЕ ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-55]	Значення поля заголовка *Via* HTTP ПОВИННЕ служити для відстеження ланцюжка запитів.	AAJ, AAX, AX, AJ
[RSG-56]	Поле заголовка HTTP Max-Forwards ПОВИННЕ використовуватися для того, щоб клієнт міг обмежити довжину ланцюжка запитів.	AAJ, AAX, AX, AJ
[RSG-57]	Якщо запит дійсний, відповідь ПОВИННА містити повне повідомлення запиту в тілі відповіді, з Content-Type *«message/http».*	AAJ, AAX
[RSG-58]	Відповіді на *TRACE* НЕ ПОВИННІ кешуватися.	AAJ, AAX, AX, AJ
[RSG-59]	Код стану *«200 OK»* СЛІД повернути в *TRACE*.	AAJ, AAX
[RSG-60]	Запит *OPTIONS* ПОВИНЕН бути ідемпотентним.	AAJ, AAX, AX, AJ
[RSG-61]	НЕ СЛІД використовувати користувацькі заголовки HTTP, що починаються з префікса *«X-»*,.	AAJ, AAX
[RSG-62]	НЕ СЛІД використовувати користувацькі заголовки HTTP для зміни поведінки HTTP-методів, за винятком випадків, коли це необхідно для усунення існуючих технічних обмежень (наприклад, див. [RSG-39]).	AAJ, AAX
[RSG-63]	Угода про іменування користувацьких HTTP-заголовків – має вигляд *<організація>-<ім'я заголовка> (<organization>-<header name>)*, де полям <організація> і <заголовок> рекомендується дотримуватися угоди про kebab-case стилі іменування.	AAJ, AAX
[RSG-64]	Web API ПОВИНЕН підтримувати єдиний метод версіонування сервісу з використанням версіонування URI, наприклад /api/v1/inventors або версіонування заголовка, наприклад Accept-version: v1 або версіонування типу даних, наприклад Accept: application/vnd.v1+json. НЕ СЛІД використовувати версіонування рядків запитів.	AAJ, AAX
[RSG-65]	Схему нумерації версій СЛІД застосовувати з урахуванням тільки	AAJ, AAX

	основного номера версії (наприклад, */v1*).	
[RSG-66]	Сервісні контракти API МОЖУТЬ включати функцію перенаправлення кінцевих точок. Коли споживач послуг намагається викликати послугу, може бути повернута відповідь про перенаправлення, щоб повідомити споживачу послуг про необхідність повторно надіслати запит на нову кінцеву точку. Перенаправлення МОЖУТЬ бути тимчасовими або постійними: -Тимчасове перенаправлення - з використанням заголовка HTTP відповіді *Location* і кода статусу HTTP *«302 Found»* відповідно до IETF RFC 7231; чи – Постійне перенаправлення – з використанням заголовка HTTP-відповіді *Location* і кода статусу HTTP *«301 Moved Permanently»* відповідно до IETF RFC 7238.	AAJ, AAX
[RSG-67]	Розробникам СЛІД опублікувати стратегії життєвого циклу API , щоб допомогти користувачам зрозуміти, як довго підтримуватиметьсясь та або інша версія.	AAJ, AAX
[RSG-68]	Web API СЛІД підтримувати пагінацію (розбивки на сторінки).	AAJ, AAX
[RSG-69]	Пагіновані запити НЕ МОЖУТЬ бути ідемпотентними.	AAJ, AAX
[RSG-70]	Web API ПОВИНЕН використати параметри запиту реалізації пагінації.	AAJ, AAX, AX, AJ
[RSG-71]	Web API НЕ ПОВИНЕН використовувати HTTP- заголовки для реалізації пагінації.	AAJ, AAX, AX, AJ
[RSG-72]	СЛІД використати параметри запиту *limit=<кількість елементів для доставки> і offset=<кількість елементів для пропуску> (limit=<number of items to deliver> and offset=<number of items to skip>),* де limit - кількість повернутих елементів (розмір сторінки), а пропуск (skip) кількість елементів, які повинні бу ти пропущені (зміщення). Якщо обмеження на розмір сторінки не вказано, СЛІДУЄ визначити значення за замовчуванням - глобальне або для кожної колекції; зміщення за замовчуванням ПОВИННЕ дорівнювати нулю «0»: Наприклад, наступний URL є допустимим: https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73]	Значення параметрів *limit і skip* НЕОБХІДНО включати у відповідь.	AAJ, AAX
[RSG-74]	Web API ПОВИНЕН підтримувати сортування.	AAJ, AAX
[RSG-75]	Для того, щоб задати багатоатрибутивний критерій сортування, НЕОБХІДНО використовується параметр запиту. Значенням цього параметра є список ключів сортування, розділених комами, до кожного ключа сортування ДОЗВОЛЕНО додавати напрямок сортування за зростанням або за спаданням, відокремлений двокрапкою ':'. Напрямок за замовчуванням ПОВИНЕН бути визначений сервером, якщо напрямок сортування не вказано для ключа.	AAJ, AAX, AX, AJ

[RSG-76]	Web API ПОВИНЕН повертати критерії сортування у відповіді.	AAJ, AAX, AX, AJ
[RSG-77]	Web API МОЖЕ підтримувати розширення тіла вмісту, що повертається. НЕОБХІДНО використати параметр запиту *expand=<розділений комами список імен атрибутів>.*	AAJ, AAX
[RSG-78]	Параметр запиту СЛІД використати замість шляхів URL у разі, якщо Web API підтримує проєкцію у форматі: *"fields="<розділений комами список імен атрибутів>.*	AAJ, AAX
[RSG-79]	Web API ПОВИНЕН підтримувати повернення кількості елементів у колекції.	AAJ, AAX, AX, AJ
[RSG-80]	Параметр запиту ПОВИНЕН використовуватись для підтримки повернення кількості елементів у колекції.	AAJ, AAX, AX, AJ
[RSG-81]	СЛІД використовувати параметр запиту *count* для повернення кількості елементів у колекції.	AAJ, AAX
[RSG-82]	Web API МОЖЕ підтримувати повернення кількості елементів у колекції всередині, тобто як частину відповіді, яка містить саму колекцію. Параметр запиту ОБОВ'ЯЗКОВО має бути використаний.	AAJ, AAX, AX, AJ
[RSG-83]	Повинен використовуватися параметр запиту *count=true*. Якщо його не вказано, то за замовчуванням *count* слід встановити в *false*.	AAJ, AAX
[RSG-84]	Якщо Web API підтримує пагінацію, він ПОВИНЕН повертати у відповіді номер колекції (тобто загальну кількість елементів колекції).	AAJ, AAX
[RSG-85]	Якщо Web API підтримує складні пошукові вирази, СЛІД вказувати мову запитів, наприклад, CQL.	AAJ, AAX
[RSG-86]	У договорі про надання послуг ПОВИННА бути вказана граматика, що підтримується (наприклад, поля, функції, ключові слова та оператори)	AAJ, AAX, AX, AJ
[RSG-87]	ПОВИНЕН використовуватись параметр запиту *«q».*	AAJ, AAX, AX, AJ
[RSG-88]	На рівні протоколу Web API ПОВИНЕН повертати відповідний код стану HTTP, вибраний зі списку стандартних кодів стну HTTP.	AAJ, AAX, AX, AJ
[RSJ-89]	На рівні додатку Web API ПОВИНЕН повертати частину, що повідомляє про помилку з достатнім ступенем деталізації. Атрибути *code і message* є обов'язковими, атрибут *details* є умовно обов'язковим, атрибути *target, status, moreInfo ma internalMessage* є необов'язковими.	AAJ, AAX, AX, AJ
[RSG-90]	Помилки НЕ ПОВИННІ розкривати критично важливі для безпеки дані або внутрішні технічні деталі, такі як стеки викликів у повідомленнях про помилку.	AAJ, AAX, AX, AJ
[RSG-91]	Заголовок HTTP: *Reason-Phrase* (описаний в RFC 2616) НЕ ПОВИНЕН використовуватися для передачі повідомлень про помилки.	AAJ, AAX, AX, AJ

[RSG-92]	Кожній зареєстрованій помилці СЛІД мати унікальний ідентифікатор кореляції. НЕОБХІДНО використовувати користувацький HTTP-заголовок, якому СЛІД мати ім'я *Correlation-ID.*	AAJ, AAX
[RSG-93]	У формат сервісного контракту НЕОБХІДНО включати наступне: Версію API; Інформацію про семантику елементів API; Ресурси; Атрибути ресурсу; Параметри запиту; Методи; Типи носіїв; Граматику пошуку (якщо вона підтримується); Коди стану HTTP; Методи HTTP; Обмеження та відмінні риси; і Безпеку (якщо є).	AAJ, AAX, AX, AJ
[RSG-94]	У формат сервісного контракту НЕОБХІДНО включати запити і відповіді в XML-схемі або JSON-схемі та приклади використання API у підтримуваних форматах, тобто XML або JSON.	AAJ, AAX
[RSG-95]	REST API ПОВИНЕН надавати документацію API у вигляді сервісного контракту.	AAJ, AAX, AX, AJ
[RSG-96]	Впровадження Web API, що відхиляється від рекомендацій цього стандарту, ПОВИННЕ бути чітко задокументоване в сервісному контракті. Якщо правил, що відхиляються від рекомендацій цього стандарту не зазначено в сервісному контракті, ПОВИННО вважатися, що Web-API дотримується рекомендацій цього стандарту.	AAJ, AAX, AX, AJ
[RSG-97]	Сервісний контракт ПОВИНЕН дозволяти генерацію каркасного (скелетного) коду сервера.	AAJ, AAX, AX, AJ
[RSG-98]	Сервісний контракт РЕКОМЕНДУЄТЬСЯ, що може дозволяти генерувати скелетний код для сервера.	AAJ, AAX
[RSG-99]	Документація до Web API ПОВИННА бути написана у форматі RAML або OAS. Користувацькі формати документації НЕ ПОВИННІі використовуватися.	AAJ, AAX
[RSG-100]	Споживач Web API ПОВИНЕН мати можливість вказати таймаут сервера для кожного запиту; ПОВИНЕН використовуватися спеціальний HTTP-заголовок. Максимальний таймаут сервера ПОВИНЕН також використовуватися для захисту ресурсів сервера від надмірного використання.	AAJ, AAX
[RSG-101]	Web API ПОВИНЕН підтримувати умовне отримання даних, щоб гарантувати, що будуть отримані тільки ті дані, які були змінені. ПОВИННА використовуватися перевірка ресурсів на основі вмісту, оскільки вона є більш точною.	AAJ, AAX

[RSG-102]	Для того, щоб реалізувати перевірку ресурсів на основі вмісту, HTTP-заголовок ETag ПОВИНЕН використовуватися у відповіді для кодування стану даних. Після цього це значення ПОВИННО використовуватися в наступних запитах в умовних HTTP-заголовках (таких як *If-Match або If-None-Match*). Якщо дані не були змінені після того, як запит повернув ETag, сервер ПОВИНЕН повернути код стану «*304 Not Modified*» (якщо вони не були змінені). Цей механізм описано в IETF RFC 7231 та 7232.	AAJ, AAX
[RSG-103]	Для реалізації перевірки ресурсів за часом НЕОБХІДНО використовувати HTTP-заголовок *Last-Modified*. Цей механізм описано в IETF RFC 7231 і 7232.	AAJ, AAX
[RSG-104]	Використовуючи версійність відповіді, споживач послуг МОЖЕ реалізувати оптимістичне блокування.	AAJ, AAX
[RSG-106]	НЕОБХІДНО використовувати HTTP-заголовки відповіді *Cache-Control та Expires*. Останні МОЖНА використовувати для підтримки застарілих клієнтів (legacy clients).	AAJ, AAX
[RSG-107]	Web API.ПОВИНЕН повідомляти, чи підтримує він завантаження файлів частинами, відповідаючи на запити HEAD і використовуючи заголовки HTTP-відповіді *Accept-Ranges* і *Content-Length.*	AAJ, AAX
[RSG-108]	Web API НЕОБХІДНО підтримувати завантаження файлів частками. Повинні підтримуватися діапазони, що складаються з декількох частин.	AAJ, AAX
[RSG-109]	Web API ПОВИНЕН повідомляти, якщо він підтримує завантаження файлів частинами.	AAJ, AAX
[RSG-110]	Web API СЛІД підтримувати завантаження файлу частинами. ПОВИННІ підтримуватися діапазони, що складаються з декількох частин	AAJ, AAX
[RSG-111]	Постачальник послуг ПОВИНЕН повертати в заголовоках відповіді HTTP заголовок *«413 Request Entity Too Large»* у випадку, якщо запит перевищив максимально дозволений ліміт. Для позначення максимального розміру запиту МОЖНА використовувати спеціальний HTTP-заголовок.	AAJ, AAX
[RSG-112]	Якщо Web API підтримує обробку налаштувань, її СЛІД реалізувати відповідно до IETF RFC 7240, тобто слід використовувати HTTP-заголовок запиту *Prefer,* у HTTP-заголовку відповіді СЛІД повернути *Preference-Applied* (з повторенням початкового запиту).	AAJ, AAX
[RSG-113]	Якщо Web API підтримує обробку налаштувань, номенклатура налаштувань, які МОЖУТЬ бути встановлені за допомогою заголовка *Prefer*, ПОВИННА бути записана в сервісному контракті.	AAJ, AAX, AJ, AX
[RSG-114]	Якщо Web API підтримує локалізовані дані, то ПОВИНЕН підтримуватися HTTP-заголовок запиту *Accept-Language*, щоб вказати набір природних мов, яким надається перевага у відповіді, як зазначено в IETF RFC 7231.	AAJ, AAX, AJ, AX
[RSG-115]	Якщо API підтримує довготривалі операції, вони ПОВИННІ бути асинхронними. Має застосовуватися такий підхід:	AAJ, AAX

	Споживач послуги активує сервісну операцію; Операція сервісу повертає код стану *«202 Accepted»* відповідно до IETF RFC 7231 (розділ 6.3.3), тобто запит було прийнято до обробки, але обробка не була завершена. Споживач послуги звертається до повернутої локації, щоб дізнатися, чи доступний ресурс.Якщо ресурс недоступний, відповідь ПОВИННА мати код статусу «200 OK», містити статус завдання (наприклад, на розгляді) і МОЖЕ містити іншу інформацію (наприклад, посилання для скасування або видалення завдання за допомогою методу DELETE HTTP). Якщо ресурс доступний, відповідь ПОВИННА мати код стану «303 See Other», а HTTP-заголовок Location ПОВИНЕН містити URL-адресу, за якою можна отримати результати виконання завдання.	
[RSG-116]	**Конфіденційність:** різні API та інформація в API ПОВИННІ бути ідентифіковані, класифіковані та захищені від несанкціонованого доступу, розкриття та перехоплення в будь-який час. Повинні дотримуватися принципи найменших привілеїв, нульової довіри, необхідності знати та необхідності ділитися.	AAJ, AAX, AJ, AX
[RSG-117]	**Забезпечення цілісності:** різні API та інформація в API ПОВИННІ бути захищені від несанкціонованої модифікації, дублювання, пошкодження та знищення. Інформація ПОВИННА модифікуватися через підтверджені транзакції та інтерфейси. Системи ПОВИННІ оновлюватися з використанням затверджених процесів управління конфігурацією, управління змінами та управління виправленнями.	AAJ, AAX, AJ, AX
[RSG-118]	**Доступність:** різні API та інформація в API ПОВИННІ бути доступні авторизованим користувачам у встановлений час відповідно до угод про рівень обслуговування (SLA), політики контролю доступу та визначених бізнес-процесів.	
[RSG-119]	**Неможливість відмови (безвідмовність)**: кожна транзакція, що обробляється або виконується за допомогою API, ПОВИННА забезпечувати неможливість відмови через реалізацію належного аудиту, авторизації, автентифікації та впровадження безпечних шляхів, а також послуг і механізмів, що не допускають відмови.	AAJ, AAX, AJ, AX
[RSG-120]	**Автентифікація, авторизація, аудит:** користувачі, системи, API або пристрої, що беруть участь у критично важливих транзакціях або діях, ПОВИННІ бути автентифіковані, авторизовані за допомогою служб контролю доступу на основі ролей або атрибутів і підтримувати розподіл обов'язків. Крім того, всі дії ПОВИННІ реєструватися, а надійність автентифікації повинна зростати разом з відповідним інформаційним ризиком.	AAJ, AAX, AJ, AX
[RSG-121]	При розробці API ПОТРІБНО ретельно враховувати загрози, зловмисні випадки використання, безпечні методи кодування, безпеку транспортного рівня та тестування безпеки, особливо: PUTs та POSTs - тобто: які зміни у внутрішніх даних можуть бути потенційно використані для атак або дезінформації; DELETES – тобто: може використовуватись для видалення вмісту внутрішнього сховища ресурсів; - Білий список дозволених методів - щоб гарантувати, що дозволені HTTP-методи належним чином обмежені, в той час як інші повертатимуть правильний код відповіді; і	AAJ, AAX, AJ, AX

	- Добре відомі атаки повинні бути розглянуті на етапі моделювання загроз в процесі проєктування, щоб гарантувати, що ризик загрози не збільшиться. Загрози та способи їх зменшення, визначені в OWASP Top Ten Cheat Sheet , ПОВИННІ бути взяті до уваги.	
[RSG-122]	При розробці API слід дотримуватися стандартів і кращих практик, наведених нижче: Кращі практики безпечного кодування: Принципи безпечного кодування OWASP; Безпека REST API: Пам’ятка з безпеки REST; Захист від несанкціонованого доступу та міжсайтового скриптингу: пам`ятка OWASP XSS Cheat Sheet; Запобігання SQL-ін'єкціям: Пам'ятка OWASP щодо SQL-ін'єкцій, пам'ятка OWASP щодо Parameterization; і Безпека транспортного рівня: Пам'ятка OWASP щодо захисту транспортного рівня.	AAJ, AAX, AJ, AX
[RSG-123]	Тестування безпеки та оцінювання вразливостей ПОВИННІ проводитися для забезпечення безпеки та стійкості API до загроз. Ця вимога МОЖЕ бути виконана шляхом використання статичного та динамічного тестування безпеки додатків (SAST/DAST), автоматизованих інструментів управління вразливостями та тестування на проникнення.	AAJ, AAX, AJ, AX
[RSG-124]	Захищені послуги ПОВИННІ надавати кінцеві точки HTTPS тільки з використанням TLS 1.2 або вище з набором шифрів, який включає ECDHE для обміну ключами.	AAJ, AAX, AJ, AX
[RSG-125]	Під час розгляду протоколів автентифікації для забезпечення транспортної безпеки НЕОБХІДНО використовувати досконалу пряму секретність. Не СЛІД допускати використання небезпечних криптографічних алгоритмів і зворотну сумісність із SSL 3 і TLS 1.0/1.1.	AAX, AAJ
[RSG-126]	Для забезпечення максимальної безпеки та довіри НЕОБХІДНО встановлювати з'єднання через VPN IPSEC між сайтами для додаткового захисту інформації, що передається через незахищені мережі.	AAX, AAJ
[RSG-127]	Користувацькому додатку НЕОБХІДНО перевіряти ланцюжок сертифікатів TLS під час виконання запитів до захищених ресурсів, включно з перевіркою списку відкликання сертифікатів.	AAX, AAJ
[RSG-128]	Захищені послуги ПОВИННІ використовувати тільки дійсні сертифікати, видані довіреним центром сертифікації (ЦС).	AAX, AAJ
[RSG-129]	Токени НЕОБХІДНО підписувати з використанням безпечних алгоритмів підпису, що відповідають стандарту цифрового підпису (DSS) FIPS-186-4. Слід розглядати алгоритм цифрового підпису RSA або алгоритм ECDSA.	AAX, AAJ
[RSG-130]	Анонімна автентифікація ПОВИННА використовуватися лише тоді, коли клієнти та програми, якими вони користуються, отримують доступ до інформації або функцій з низьким рівнем чутливості, які не потребують автентифікації, наприклад, загальнодоступної інформації.	AAJ, AAX, AJ, AX

[RSG-131]	Автентифікація з використанням імені користувача і пароля або хеш-пароля НЕ ПОВИННА дозволятися.	AAJ, AAX, AJ, AX
[RSG-132]	Якщо сервіс захищений, СЛІД використовувати Open ID Connect.	AAX, AAJ
[RSG-133]	Якщо використовується JSON Web Token (JWT), секретний JWT повинен мати високу ентропію, щоб збільшити коефіцієнт роботи атаки грубої сили; токени TTL і RTTL повинні бути якомога коротшими; і конфіденційна інформація НЕ повинна зберігатися в корисному навантаженні JWT.	AAX, AAJ
[RSG-134]	У запитах *POST/PUT* конфіденційні дані НЕОБХІДНО передавати в тілі або в заголовку запиту.	AAX, AAJ
[RSG-135]	У запитах GET конфіденційні дані НЕОБХІДНО передавати в заголовку HTTP.	AAX, AAJ
[RSG-136]	Щоб мінімізувати затримки та зменшити зв'язок між захищеними сервісами, рішення про управління доступом СЛІД приймати на локальному рівні в кінцевих точках REST.	AAX, AAJ
[RSG-137]	API-ключі ПОВИННІ використовуватися для захищених і загальнодоступних сервісів, щоб запобігти перевантаженню постачальника послуг численними запитами (атакам на відмову в обслуговуванні). Для захищених сервісів ключі API МОЖНА використовувати для монетизації (придбаних планів), забезпечення дотримання політики використання (QoS) та моніторингу.	AAX, AAJ
[RSG-138]	Ключі API МОЖУТЬ бути об'єднані із заголовком HTTP запиту *user-agent* для розрізнення користувача-людини та програмного агента, як зазначено в IETF RFC 7231.	AAX, AAJ
[RSG-139]	Постачальник послуг ПОВИНЕН повертати разом із заголовками HTTP-відповіді поточний стан використання. У відповідь МОЖУТЬ бути повернуті наступні дані: – ліміт швидкості (rate limit) - ліміт швидкості (у хвилину), встановлена в системі; – rate limit remaining - кількість запитів, що залишилось, дозволених протягом поточного часового інтервалу (-1 означає, що ліміт перевищено); та – скидання ліміту швидкості (rate limit reset) - час (у секундах), що залишився до скидання лічильника запитів.	AAX, AAJ
[RSG-140]	Постачальник послуг ПОВИНЕН повертати код статусу *«429 Too Many Requests»*, якщо запити надходять занадто швидко.	AAX, AAJ
[RSG-141]	Ключі API ПОВИННІ бути відкликані, якщо клієнт порушує угоду про використання, як це визначено BIB.	AAJ, AAX, AJ, AX
[RSG-142]	Ключі API ПОВИННІ передаватися за допомогою спеціальних HTTP-заголовків. Їх НЕ СЛІД передавати через параметри запиту.	AAX, AAJ
[RSG-143]	Ключі API ПОВИННІ бути згенеровані випадковим чином.	AAX, AAJ

[RSG-144]	Безпечні та надійні сертифікати ПОВИННІ видаватися взаємно довіреним центром сертифікації (ЦС) через процес встановлення довіри або перехресної сертифікації.	AAJ, AAX, AJ, AX
[RSG-145]	Сертифікати, які спільно використовуються клієнтом і сервером, СЛІД використовувати для зниження ризиків безпеки ідентифікації, характерних для чутливих систем і привілейованих дій, наприклад, X.509.	AAJ, AAX, AJ, AX
[RSG-146]	Для високопривілейованих служб двостороння взаємна автентифікація між клієнтом і сервером ПОВИННА використовувати сертифікати для забезпечення додаткового захисту.	AAX, AAJ
[RSG-147]	Багатофакторна автентифікація ПОВИННА бути впроваджена для зменшення ризиків ідентифікації для додатків з високим профілем ризику, систем, що обробляють дуже конфіденційну інформацію або виконують привілейовані дії.	AAX, AAJ
[RSG-148]	Якщо REST API є публічним, HTTP-заголовок *Access-Control-Allow-Origin* ПОВИНЕН мати значення'*'.	AAJ, AAX, AJ, AX
[RSG-149]	Якщо REST API захищений, слід використовувати CORS, якщо це можливо. В іншому випадку, JSONP МОЖЕ використовуватися як запасний варіант, але тільки для GET-запитів, наприклад, коли користувач отримує доступ за допомогою старого браузера. НЕ СЛІД використовувати Iframe.	AAX, AAJ

[Додаток II далі]

ДОДАТОК II

СЛОВНИК REST для ІВ

Версія 1.1

Редакція, схвалена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

1. У Таблиці 5 наведено наступний словник в галузі інтелектуальної власності як приклади параметрів /basic RESTful Service Request. ВІВ, ймовірно, зіткнуться з необхідністю розробляти більш складні запити та різноманітне корисне навантаження відповідей відповідно до своїх бізнес-потреб. Параметри в цій таблиці є прикладами елементів стандарту ВОІВ ST.96 у нижньому регістрі, які використовуються для відповіді JSON. Повний словник даних стандарту ВОІВ ST.96 IP та XML-схеми в галузі інтелектуальної власності можна отримати за цим посиланням:
https://www.wipo.int/standards/en/st96/v5-0 /.

[Примітка редактора: У майбутньому планується надати посилання на більш повний список словників REST IP XML і JSON, який буде динамічно підтримуватися на постійній основі, оскільки елементи і лексика в галузі інтелектуальної власності продовжують розвиватися.]

Таблиця 5: Приклад бізнес-лексики API у нижньому регістрі (lowerCamelCase) відповідно до стандарту ВОІВ ST.96 XSDs

Сфера застосування (бізнес-домен(и))	**Назва ресурсу (ів)**	**Назва параметра**	**Опис**
ВСІ	/торговельні марки /патенти /промислові зразки	*St13ApplicationNumber*	Номер заявки на поданий об'єкт інтелектуальної власності, використовуючи формат стандарту ВОІВ ST.13, який являє собою рядок з декількох значень, включаючи номер національної заявки, тип об'єкта інтелектуальної власності та країну/організацію
ВСІ	/торговельні марки /патенти /промислові зразки	*applicationNumber*	Номер заявки на поданий об'єкт інтелектуальної власності у форматі національного відомства.
МНОЖИНА	/торговельні марки /промислові зразоки	*internationalRegistration Number*	Міжнародний реєстраційний номер права інтелектуальної власності. Для торговельних марок це стосується Мадридської системи. Для промислових зразків це стосується Гаазької системи

BCI	/торговельні марки /патенти /промислові зразки	*availableDocument*	Єдиний запис про документ, що відповідає критеріям пошуку, наданий в API DocList
BCI	/торговельні марки /патенти /промислові зразки	*sortingCriteria*	Критерій сортування, який використовується API DocList
BCI	/торговельні марки /патенти /промислові зразки	*receivingOfficeCode*	Код відомства інтелектуальної власності у форматі відповідно до стандарту BOIB ST.2
BCI	/торговельні марки /патенти /промислові зразки	*receivingOfficeDate*	Дата отримання у Відомстві інтелектуальної власності
Торговельні марки	/торговельні марки	*registrationDate*	Дата реєстрації у Відомстві інтелектуальної власності
		applicationDate	Дата подання заяви
		markCurrentStatusCode	Код поточного правового статусу заявки
		markCurrentStatusCode	Дата поточного правового статусу заявки
Патент	/патенти	*filingDate*	Дата подання заявки
		grantPublicationDate	Дата, коли патент був опублікований
		fileReferenceIdentifier	Реєстраційний номер заявки
		applicationBodyStatus	Статус органу, що розглядає заявку
		statusEventData	Дані, пов'язані з подією правового статусу стосовно конкретної патентної заявки
		keyEventData	Код, що позначає широку, високорівневу подію, яка охоплює найбільш загальні та важливі ситуації в категорії
Промислові зразки	/промислові зразки	*applicationDate*	Дата подання заяви
		designApplicationCurrentStatus	Категорія поточного правового статусу заявки на промисловий зразок
		designApplicationCurrentStatusDate	Дата поточного правового статусу заявки на промисловий зразок

2. Наступні технічні параметри запиту, що визначені в таблиці 6, повинні застосовуватися до всіх сервісів REST API:

Таблиця 6: Технічний словник API

Параметр запиту/ шляху	**Значення параметра Тип даних**	**Обмеження**	**Формат**	**Опис**	**Правило проєктування**
format	string		type/subtype; parameter=value відповідно до RFC7231, 3.1.1.1. Тип носія	Використовується для узгодження типу вмісту (переважно заголовок запиту HTTP)	[RSG-19]
v	string		v%, де % - є натуральним числом	Використовується для керування версіями сервісу (бажано вказувати версію як сегмент шляху до URL-адреси)	[RSG-64]
limit	integer	Позитивне число	*limit =10*	Розмір сторінки, що використовується для пагінації (розбивки на сторінки)	[RSG-73]
offset	integer	Позитивне число; за замовчуванням 0	*offset=5*	Зміщення, що використовується для пагінації	[RSG-73]
sort	список рядків, розділених комами	Можливі значення: d. – asc e. desc	*sort=key1:asc, key2:desc*	Багатоатрибутний критерій сортування	[RSG-74] – [RSG-76]
expand	список рядків, розділе них комами		*expand=key1,key2*	Використовується для розширення тіла, повернутого вмісту	[RSG-77]
count	boolean	За замовчуванням false	*count=true*	Повертає кількість елементів у колекції (може бути послідовною)	[RSG-81]
apiKey	string		*apiKey=abcdef12345*	Використовується для зазначення ключа Web API (бажано використовувати HTTP-заголовок).	[RSG-137]-[RSG-138]

[Додаток III далі]

ДОДАТОК III

КЕРІВНИЦТВО З RESTFUL WEB API ТА ТИПОВИЙ СЕРВІСНИЙ КОНТРАКТ

Версія 1.1

Редакція схвалена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

1. У Додатку III наведено два приклади моделей специфікацій API, що відповідають Стандарту, які мають на меті забезпечити керівництво для відомств інтелектуальної власності (ВІВ), які бажають розробляти вебсервіси відповідно до цього Стандарту. Детальна інформація щодо двох прикладів моделей наведена нижче, а також у Доповненнях A і B.

2. Слід зазначити, що приклади моделей були створені з використанням гібридного підходу, що поєднує в собі підходи «contract-first» та «code-first».

Приклад моделі DocList

3. Перший із прикладів моделей було створено на основі набору вебсервісів відомств IP5[17] Open Portal Dossier (OPD - досьє відкритого порталу) з однойменною назвою. API DocList надає список відповідних патентних документів, пов'язаних принаймні з номером заявки або публікації.

Приклад моделі правового статусу патенту

4. Друга модель - це API правового статусу патенту, який надає або історію подій правового статусу для конкретного номера заявки, або деталі конкретної події правового статусу.

[Доповнення A і B до Додатка III наведено далі]

[17] До складу відомств IP5 входять Китайська національна адміністрація інтелектуальної власності (CNIPA), Європейське патентне відомство (ЄПВ), Японське патентне відомство (JPO), Корейське відомство інтелектуальної власності (KIPO) та Відомство США з патентів і торговельних марок (USPTO).

ДОПОВНЕННЯ A

ПРИКЛАД МОДЕЛІ DOCLIST

1. У Доповненні A наведено посилання на zip-файл, який містить документ з вимогами, що описує формати запиту та відповіді, специфікацію YAML та компоненти XSD.

2. Доповнення A доступне за посиланням: https://www.wipo.int/standards/en/st90/annex_iii_appendix_a_V1_0.zip

ДОПОВНЕННЯ B

ПРИКЛАДУ МОДЕЛІ ПРАВОВОГО СТАТУСУ ПАТЕНТУ

1. У Доповненні B наведено посилання на zip-файл, що містить специфікацію API, надану в RAML, приклади даних і списки переліків відповідно до стандарту ВОІВ ST.96.

2. Доповнення B доступне за посиланням: https://www.wipo.int/standards/en/st90/annex-iii_appendix_b_V1_0.zip

[Додаток IV наведено далі]

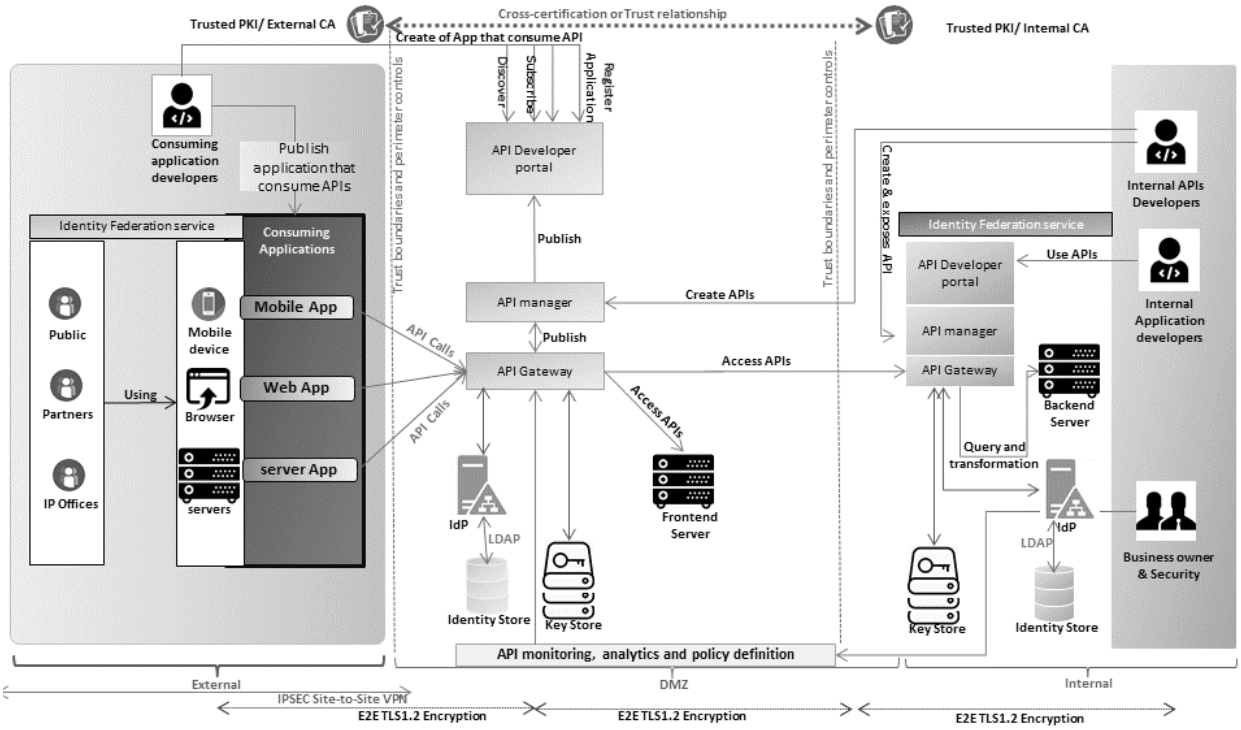
ДОДАТОК IV

КРАЩІ ПРАКТИКИ АРХІТЕКТУРИ БЕЗПЕКИ ВИСОКОГО РІВНЯ

Версія 1.1

Редакція схвалена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

1. Архітектура безпеки визначає сервіси та механізми, які повинні бути реалізовані для забезпечення дотримання встановлених політик і правил, а також забезпечує основу для подальшої стандартизації та автоматизації безпеки. Основні сервіси та механізми цієї архітектури безпеки API (портал розробки, менеджер API і шлюз API) забезпечують групування функціональних можливостей. Ці функції можуть бути надані окремими додатками, розробкою коду на замовлення, за допомогою COTS-продуктів або шляхом використання існуючих технологій, які можуть бути налаштовані для надання цих функцій/послуг. Деякі з цих функцій можуть перетинатися або об'єднуватися в один або декілька продуктів залежно від постачальника.



2. Рекомендованій архітектурі безпеки СЛІД мати наступні сервіси та механізми безпеки API:

- Web-портал API, що надає такі функції, як виявлення API, аналітика API, доступ до специфікацій та описів, включаючи SLA, соціальну мережу та поширені запитання (FAQ);
- Менеджер Web API для забезпечення централізованого адміністрування API та управління каталогами API, управління реєстрацією та приєднанням до різних спільнот розробників API, управління життєвим циклом API, застосування попередньо визначених профілів безпеки та управління життєвим циклом політик безпеки;
- Шлюз Web API для забезпечення можливостей автоматизації безпеки, включаючи, але не обмежуючись цим, централізований захист від загроз,

централізовану автентифікацію API, авторизацію, ведення журналів, впровадження політики безпеки, шифрування повідомлень, моніторинг та аналітику;

- Служба моніторингу та аналітики Web API яка надає такі функції, як розширений моніторинг API-сервісів, аналітика, використання профілю для базових рівнів безпеки, зміни у використанні та попиті;
- Сховище облікових даних для безпечного зберігання ключів API, секретів, сертифікатів тощо;
- Довірений центр сертифікації (ЦС) який видає безпечні сертифікати та уможливлює встановлення довіри між різними відомствами;
- Система управління інформацією та подіями безпеки (SIEM) для кореляції журналів безпеки та розширеної аналітики та моніторингу безпеки;
- Постачальник ідентифікаційних даних для управління ідентифікаційними даними, що зберігаються в LDAP-каталогах, і забезпечення автентифікації; і
- Продукт для сканування вебдодатків, який виконує регулярне сканування безпеки та аналіз на основі довіреної базової лінії безпеки, наприклад, OWASP Top 10.

[Додаток V наведено далі]

ДОДАТОК V

КОДИ СТАНУ HTTP

Версія 1.1

Редакція схвалена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

1. Важливо узгоджувати відповіді з відповідним кодом стану HTTP та слідувати стандартним кодам HTTP. Окрім відповідного коду стану, у тілі відповіді HTTP рекомендується мати корисний та короткий опис помилки. Відповіді мають бути конкретними та чіткими, щоб споживачі могли швидко зробити висновок при використанні API.

2. Набір кодів стану HTTP визначено на основі RFC7231. Коди стану, перераховані нижче, необхідно використовувати в API, де це може бути застосовано.

3. Визначено такі категорії кодів стану відповіді:

- 1xx: Informational - передає інформацію на рівні протоколу передачі;
- 2xx: Success - вказує на те, що запит клієнта був успішно прийнятий;
- 3xx: Redirection – вказує на те, що клієнт повинен зробити деякі додаткові дії, щоб завершити свій запит;
- 4xx: Client Error - ця категорія кодів стану помилки вказує на клієнтів; і
- 5xx: Server Error - коди помилок, за які відповідає сервер.

4. У наступній таблиці об'єднані коди стану HTTP і наведені посилання на відповідні RFC IETF.

Значення	**Опис**	**Посилання**
100	Continue – Продовжити	[RFC7231, розділ 6.2.1]
101	Switching Protocols – Протоколи комутації	[RFC7231, розділ 6.2.2]
102	Processing – Йде обробка	[RFC2518]
103	Early Hints-Ранні підказки(перед остаточною відповіддю)	[RFC8297]
104-199	Не описано	
200	OK – Успішно	[RFC7231, розділ 6.3.1]
201	Created – Створено	[RFC7231, розділ 6.3.2]
202	Accepted - Прийнято	[RFC7231, розділ 6.3.3]
203	Non-Authoritative Information-Неофіційна інформація	[RFC7231, розділ 6.3.4]

204	No Content – Немає вмісту	[RFC7231, розділ 6.3.5]
205	Reset Content – Скидання вмісту	[RFC7231, розділ 6.3.6]
206	Partial Content – Частина вмісту	[RFC7233, розділ 4.1]
207	Multi-Status – Мульти-статус	[RFC4918]
208	Already Reported – Вже повідомлено	[RFC5842]
209-225	Не описано	
226	IM Used – Використовуваний для управління екземпляром	[RFC3229]
227-299	Не описані	
300	Multiple Choices – Множинний вибір	[RFC7231, розділ 6.4.1]
301	Moved Permanently – Ресурс переміщено назавжди	[RFC7231, розділ 6.4.2]
302	Found – Знайдено	[RFC7231, розділ 6.4.3]
303	See Other – Див.інше	[RFC7231, розділ 6.4.4]
304	Not Modified – Не змінено	[RFC7232, розділ 4.1]
305	Use Proxy –Використовувати проксі-сервер	[RFC7231, розділ 6.4.5]
306	Не використовується	[RFC7231, розділ 6.4.6]
307	Temporary Redirect – Тимчасове перенаправлення	[RFC7231, розділ 6.4.7]
308	Permanent Redirect – Постійне перенаправлення	[RFC7538]
309-399	Не описані	
400	Bad Request – Поганий запит	[RFC7231, розділ 6.5.1]
401	Unauthorized – Неавторизований(потребує авторизації)	[RFC7235, розділ 3.1]
402	Payment Required – Потрібна оплата	[RFC7231, розділ 6.5.2]
403	Forbidden – Заборонено	[RFC7231, розділ 6.5.3]
404	Not Found – Не знайдено	[RFC7231, розділ 6.5.4]

405	Method Not Allowed – Метод не дозволений	[RFC7231, розділ 6.5.5]
406	Not Acceptable – Неприйнятний запит	[RFC7231, розділ 6.5.6]
407	Proxy Authentication Required – Необхідна автентифікація проксі-сервера	[RFC7235, раздел 3.2].
408	Request Timeout – Перевищено час очікування для запиту	[RFC7231, розділ 6.5.7]
409	Conflict – Конфлікт	[RFC7231, розділ 6.5.8]
410	Gone – Зникло(доступ більше неможливий)	[RFC7231, розділ 6.5.9]
411	Length Required – Необхідна довжина	[RFC7231, розділ 6.5.10]
412	Precondition Failed – Необхідна умова не виконана	[RFC7232, розділ 4.2] [RFC8144, розділ 3.2]
413	Payload Too Large – Занадто великий корисний вміст	[RFC7231, розділ 6.5.11]
414	URI Too Long – URI Занадто довгий URI	[RFC7231, розділ 6.5.12]
415	Unsupported Media Type – Тип носія не підтримується	[RFC7231, розділ 6.5.13] [RFC7694, розділ 3]
416	Range Not Satisfiable – Діапазон не задовільний	[RFC7233, розділ 4.4]
417	Expectation Failed – Очікування не вдалося	[RFC7231, розділ 6.5.14]
418-420	Не описані	
421	Misdirected Request – Неправильно надісланий запит	[RFC7540, розділ 9.1.2]
422	Unprocessable Entity- Необроблювана сутність	[RFC4918]
423	Locked – Заблоковано (закрито)	[RFC4918]
424	Failed Dependency – Невдала залежність	[RFC4918]
425	Не описаний	
426	Upgrade Required –	[RFC7231, розділ 6.5.15]

	Потрібне оновлення	
427	Unassigned – Не описаний	
428	Precondition Required – Потрібна попередня умова	[RFC6585]
429	Too Many Requests – Занадто багато запитів	[RFC6585]
430	Unassigned- Не описаний	
431	Request Header Fields Too Large – Занадто великі поля заголовка запиту	[RFC6585]
432-450	Unassigned – Не описані	
451	Unavailable For Legal Reasons – Недоступно з юридичних причин	[RFC7725]
452-499	Unassigned – Не описані	
500	Internal Server Error – Внутрішня помилка сервера	[RFC7231, розділ 6.6.1]
501	Not Implemented – Не підтримується	[RFC7231, розділ 6.6.2]
502	Bad Gateway – Помилка шлюзу	[RFC7231, розділ 6.6.3]
503	Service Unavailable – Сервіс недоступний	[RFC7231, розділ 6.6.4]
504	Gateway Timeout – Шлюз не відповідає	[RFC7231, розділ 6.6.5]
505	HTTP Version Not Supported – Версія HTTP не підтримується	[RFC7231, розділ 6.6.6]
506	Variant Also Negotiates– Варіант також проводить узгодження	[RFC2295]
507	Insufficient Storage – Недостатньо місця для зберігання	[RFC4918]
508	Loop Detected – Виявлено зациклення	[RFC5842]
509	Unassigned – Не описаний	
510	Not Extended Не розширено (не вдався розширений запит)	[RFC2774]
511	Network Authentication Required – Потрібна мережева автентифікація	[RFC6585]
512-599	Unassigned – Не описані	

[Додаток VI наведено далі]

ДОДАТОК VI

ТЕРМІНИ ПРЕДСТАВЛЕННЯ

Версія 1.1

Редакція, схвалена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

Термін	Визначення	Тип даних
Amount	Грошовий вираз	Number – число
Category	Конкретно визначений підрозділ або підмножина в системі класифікації, в якій всі елементи мають однакову схему класифікації.	String - рядок
Code	Комбінація з однієї або декількох цифр, букв або спеціальних символів, що підставляється для конкретного значення. Являє собою кінцеві, заздалегідь визначені значення або вільний формат	String - рядок
Date	Подання конкретного моменту часу, вираженого роком, місяцем і днем.	String - рядок
Directory	Завжди передує PATH	String - рядок
Document	CLOB розшифровується як «Character Large Object» (великий символьний об'єкт), що є специфічним типом даних майже для всіх баз даних. Простіше кажучи, CLOB - вказівник на текст, що зберігається поза таблицею в спеціальному блоці. Використовується для XML документів. Складається з текстової інформації про обмінювану міжнародну реєстрацію торговельної марки. XML-теги ідентифікують елементи даних, що належать до такої інформації. Команда розробників TIS - Madrid може визначити атрибут XML_DOC як CLOB, покажчик на дані з тегами, що зберігаються поза таблицею у виділеному блоці	String - рядок
Identifier	Комбінація з одного або декількох цілих чисел, букв, спеціальних символів, яка однозначно ідентифікує конкретний екземпляр бізнес-об'єкта, але може не мати чітко визначеного значення.	String– рядок
Indicator	Сигнал про наявність, відсутність або необхідність чого-небудь. Рекомендовані значення: «Y», «N» і, за необхідності, “?”.	Boolean – булевий (логічний)
Measure	Міра (Вимірювання) -це числове значення, яке визначається шляхом вимірювання об'єкта разом із зазначеною одиницею вимірювання. `MeasureType` використовується для представлення фізичного виміру, наприклад, температури, довжини, швидкості, ширини, ваги, об'єму, широти об'єкта. Точніше	Number - номер

Термін	Визначення	Тип даних
	кажучи, MeasureType слід використовувати для вимірювання внутрішніх або фізичних властивостей об'єкта, що розглядається як єдине ціле.	
Name	Позначення об'єкта, виражене словом або виразом/в слові або фразі.	String - рядок
Number	Рядок цифрових або буквено-цифрових символів, що виражає етикетку, значення, кількість або ідентифікацію.	Number, String - число, рядок
Percent	Число, яке представляє собою частину цілого, що ділиться на 100.	Number- число
Quantity	Кількість - це підрахована кількість негрошових одиниць, можливо, включаючи дроби. Кількість використовується для позначення підрахованої кількості речей. Кількість слід використовувати для простих властивостей об'єкта, що розглядається як складова частина, колекція або контейнер, для кількісної оцінки або підрахунку його компонентів. Кількість завжди має виражати підраховану кількість предметів, а властивість буде такою: всього, відвантажено, завантажено, зберігається *QuantityType* необхідно використовувати для компонентів, які вимагають інформації про одиниці виміру; а *xsd:nonNegativeInteger* необхідно використовувати для лічильних компонентів, які не потребують інформації про одиниці виміру	Number - число
Rate	Кількість або сума, виміряна по відношенню до іншої кількості або суми.	Number - номер
Text	Неформатований рядок символів, зазвичай у вигляді слів. (Включає: абревіатура, коментарі)	String - рядок
Time	Позначення певної хронологічної точки в межах періоду.	Date - дата
DateTime	Зафіксовані дата і час події, коли вона відбувається.	Date - дата
URI	Уніфікований ідентифікатор ресурсу, який визначає, де знаходиться файл.	String - рядок

[Додаток VII наведено далі]

ДОДАТОК VII

ПУБЛІКАЦІЯ ПРО УПРАВЛІННЯ ЖИТТЄВИМ ЦИКЛОМ API

Версія 1.1

Редакція схвалена Комітетом зі стандартів ВОІВ (КСВ)
на його десятій сесії 25 листопада 2022 року

1. У цьому додатку представлено короткий огляд управління життєвим циклом API і запропоновано ключові фрагменти інформації, які повинні бути опубліковані в програмному документі відомства ІВ, щоб допомогти споживачам API зрозуміти, як краще використовувати ці API.

2. Управління життєвим циклом API є критично важливим аспектом стратегії API, оскільки воно забезпечує основу для життя API від створення до видалення. Управління життєвим циклом API корисне як для внутрішніх розробників та операційних команд, так і для зовнішніх споживачів API. Управління життєвим циклом API допомагає внутрішнім розробникам створити структуру та встановити очікування щодо розробки API, а операційним командам - зрозуміти вимоги до підтримки. Для споживачів API, як внутрішніх, так і зовнішніх, управління життєвим циклом API забезпечує неформальний договір про очікування щодо використання певного API. На зображенні нижче зрозуміло представлений кожен етап життєвого циклу

3. Опубліковані життєві цикли API можуть складатися з простих 4-етапних процесів або більш складних, що включають до 10 і більше етапів. Однак, здебільшого, життєві цикли з більшою кількістю кроків вважаються більш детальними версіями життєвих циклів з меншою кількістю кроків. Таким чином, цей документ зосередиться на базовому 4-етапному процесі, необхідному для фіксації життєвого циклу API: Створено -> Опубліковано -> Застаріло -> Виведено. Будь-який опублікований документ про життєвий цикл API повинен містити принаймні опис цих чотирьох етапів, якими керує Відомство.

Створено

4. Створення API фокусується на проєктуванні, реалізації та документуванні API. На етапі створення API важливо враховувати призначення API та загальну структуру, необхідну для того, щоб зробити API максимально «перспективним». Оптимально, якщо API відповідає набору внутрішніх і зовнішніх стандартів, таких як рекомендації, що включені до цього Стандарту. Якщо API планується монетизувати, то на цьому етапі слід визначити стратегію монетизації.

Опубліковано

5. Після створення API його потрібно опублікувати. Він має бути версифікований з використанням стандартної стратегії управління версіями, а також має бути надана документація, включаючи специфікацію API та зразки запитів і відповідей (див. [RSG-64]-[RSG-65]).Після публікації API використовується програмами. Варто зауважити, що на етапі публікації можуть бути внесені виправлення та покращення.

Застаріло

6. У якийсь момент API втрачає корисність. Він або замінюється новішою версією API, або втрачає актуальність через певні зовнішні чи внутрішні чинники. Необхідно зв'язатися зі споживачами API і підготуватися до видалення API з каталогу. На цьому етапі, ймовірно, будуть виправлені лише основні помилки API.

Виведено

7. На цьому етапі API виводиться з експлуатації. Це має включати відключення доступу до API та видалення його з платформи API. Слід розглянути питання про те, чи буде запропонована «розширена підтримка», чи є випадки, коли виведення з експлуатації може бути відтерміноване.

8. Останні два етапи є найбільш важливими для документування з точки зору управління життєвим циклом - етапи застарівання і виведення з експлуатації. Для користувачів API дуже важливо розуміти очікування, які покладаються на них, коли вони починають використовувати API, щоб уникнути невдач або проблем при спробі видалення API з каталогу. Документування повинно включати, наприклад, управління основними і вторинними версіями, а також будь-які терміни повідомлення про зміни. На високому рівні, як правило, існує два підходи до застарівання/видалення API: або збереження раніше заявленої кількості версій, або збереження старих версій протягом певного періоду часу. Можна також використовувати комбінацію цих підходів, але в опублікованому документі життєвого циклу має бути чітко зазначено або кількість старих версій, які підтримуватимуться, або період часу, протягом якого зберігатимуться старі версії.

[Кінець Додатка VII і Стандарту]