УДК 004.415.2
DOI https://doi.org/10.32782/2663-5941/2024.3.1/29

Ронський С.І.
Державний університет «Житомирська політехніка»

ПРАКТИЧНЕ ЗАСТОСУВАННЯ СТАТУС КОДІВ RFC 9110 У КОНТЕКСТІ ВАЛІДАЦІЇ HTTP ЗАПИТУ

Сучасний світ важко уявити без веб застосунків. Однією з ключових задач при розробці архітектури для таких застосунків є валідація вхідних даних та коректних стандартизованих HTTP відповідей. І хоча протокол HTTP існує на протязі більше як двох десятиліть, досі виникають спірні питання при розробці веб додатків у контексті статус коду відповіді.

Дана стаття розглядає використання HTTP статус кодів. Зокрема досліджені певні спірні прикладах використання, та як ці питання вирішили в крайньому стандарті RFC 9110. На початку, стаття розглядає хронологію змін в стандартах семантики HTTP, базове використання статус кодів. Більш детально розглядаються коди які в різних випадках використовуються як коди помилок валідації HTTP запиту, а саме 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found) та 409 (Conflict). Окремо розглядається використання коду 422 (Unprocessable Entity) який введений в крайнрьому стандарті, та потенційно вирішує суперечливі питання.

Для опису проблемних питань розглядається приклад коду контроллера веб-додатку на платформі Node.js з використанням фреймворку NestJS. В статті описані потенційні обробники виключень повязаних з валідацією вхідних даних та валідацією бізнес правил. Стаття включає аналіз обговорень спірних питань формування HTTP відповідей у розробницькій спільноті та порівняння різних підходів до використання HTTP статусів. Перше спірне питання яке розглядається – це формування відповіді у випадку відсутності пов'язаної сутності у POST та PUT запитах. Тобто випадок, у якому клієнт вказує ідентифікатор сутності у тілі запиту, але дана сутність відсутня в системі. Друге питання що розглядається, формування відповіді у випадку виключення пов'язаним з правилами бізнес валідації. У статті вказано на переваги та недоліки різних підходів і надано рекомендації до використання стандарту RFC 9110, зокрема коду статусу 422.

Ключові слова: *hypertext transfer protocol, веб-застосунок, валідація, архітектура, статус код.*

Постановка проблеми. При розробці веб-додатків однією з ключових задач є обробка HTTP запитів та правильна валідація вхідних даних. Незважаючи на наявність різноманітних інструментів та бібліотек, іноді розробники стикаються з викликом неоднозначності кодів статусу при валідації даних та управлінні бізнес-правилами веб-застосунків.

В мережі інтернет існують обговорення певних питань коректності повернення кодів 400, 404 та 409 для різних випадків. Документ RFC 9110 стандартизує статус коду 422. Цей статус код вирішує певні спірні питання, такі як валідація бізнес-правил та обробка відсутності пов'язаної сутності, але виникає питання про його практичне використання у веб-додатках, зокрема на платформі Node.js.

У цій публікації ми розглянемо спірні питання використання статус кодів валідації та практичне застосування статусу кодів згідно з RFC 9110 у контексті валідації HTTP запитів в Node.js додатках. Ми звернемо особливу увагу на новий код статусу 422 та його роль у розв'язанні вищезазначених проблем валідації та управління даними в веб-додатках.

Аналіз останніх досліджень і публікацій. Теоретичні аспекти HTTP протоколу закладені Тімом Бернерсом-Лі ще у 1989 році. Але протокол та стандарти оновлюються різними групами інженерів, зокрема над стандартом який ми розглядаємо в даній статті працювали R. Fielding, M. Nottingham та J. Reschke. Стандарти лягають в основу як веб розробки в цілому, так і архітектури веб додатків. Серед вітчизняних науковців питання розробки веб додатків є доволі популярним, наприклад роботи на дану тематику пишуть наступні науковці: Пономарьов Ігор Володимирович, Поперешняк Світлана Володимирівна.

Постановка завдання. Метою статті є демонстрація можливостей практичного використання кодів зі стандарту RFC 9110, а саме питання використання кодів що пов'язані з помилками валідації HTTP запитів.

Виклад основного матеріалу. HTTP (Hypertext Transfer Protocol) є основним протоколом для обміну даними в Інтернеті, що використовується

для передачі гіпертекстових документів, зображень, відео та іншого вмісту між веб-серверами та клієнтськими програмами. Розроблений у 1989 році [1] і вдосконалений з часом, HTTP є основним механізмом комунікації в мережі Інтернет, забезпечуючи швидкий та ефективний обмін інформацією між веб-ресурсами.

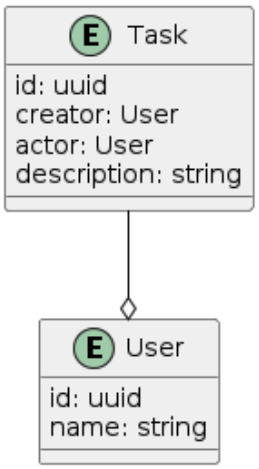
Для розуміння сучасного стану HTTP протоколу і його розвитку важливо розглянути ключові стандарти та специфікації, що змінювалися з часом. В червні 2022 року Інтернет інженерна група (IETF) опублікувала новий стандарт RFC 9110, який замінює застарівші 2818, 7230, 7231, 7232, 7233, 7235, 7538, 7615, 7694 [2]. Дані стандарти визначають HTTP статус код як частину відповіді на HTTP запит. Він використовується для передачі інформації про результат виконання запиту між клієнтом і сервером. Кожен HTTP статус код вказує на певний стан або результат запиту Ці статуси можуть вказувати на успішне виконання запиту (наприклад, статус 200 “OK”), перенаправлення (наприклад, статус 301 “Moved Permanently”), помилки клієнта (наприклад, статус 404 “Not Found”) або помилки сервера (наприклад, статус 500 “Internal Server Error”).

Загалом, статус коди HTTP поділяються на п’ять основних груп, кожна з яких вказує на певний аспект результату виконання запиту. Перша група, яку представляють статуси від 1xx до 1xx, використовується для інформаційних повідомлень, які підтверджують, що запит був отриманий і оброблений. Друга група статусів, від 2xx до 2xx, позначає успішне завершення запиту, коли сервер успішно зрозумів і виконав запит клієнта. Третя група, від 3xx до 3xx, вказує на те, що для завершення запиту клієнт повинен виконати додаткові дії, такі як перенаправлення. Четверта група, від 4xx до 4xx, вказує на помилки, які виникли через неправильні або некоректні запити з боку клієнта. Нарешті, п’ята група, від 5xx до 5xx, вказує на помилки, які виникли на стороні сервера під час обробки запиту, такі як внутрішня помилка сервера чи перевантаження [3]. Ці групи статусів допомагають розробникам та адміністраторам розуміти характер проблеми та вживати відповідні заходи для її вирішення.

В контексті даної статті нас цікавлять наступні статус коди:

- 400 – Bad Request: Цей код статусу вказує на те, що сервер не може обробити запит через помилку у синтаксисі запиту, що зробив клієнт.
- 401 – Unauthorized: Цей код статусу означає, що клієнт не авторизований для доступу до ресурсу. Це може стати результатом недостатньої авторизації або відсутності валідних облікових даних.
- 403 – Forbidden: Сервер повідомляє клієнта, що доступ до запитаного ресурсу заборонений. Це може бути через недостатні права доступу або обмеження, встановлені на сервері.
- 404 – Not Found: Цей код статусу означає, що запитаний ресурс не знайдений на сервері. Це може бути через невірну URL адресу або через відсутність ресурсу на сервері.
- 409 – Conflict: Цей код статусу вказує на конфлікт у виконанні запиту, який може виникнути, наприклад, при спробі створити ресурс, який вже існує.
- 422 – Unprocessable Entity: Цей код статусу означає, що сервер не може обробити запит через неправильний формат або невірні дані у вмісті запиту, незважаючи на те, що сам запит був коректним.

Опишемо гіпотетичний приклад додатку на мові TypeSctipt з використанням фреймворку NestJS [4]. Нехай в нашій системі існують сутності користувач (User) та задача (Task). ідентифікацію сутностей виконується за допомогою UUID (Universally Unique Identifier) [5], кожна задача має 2 посилання на сутність користувача: створювач (creator) та виконавець (actor). UML схема зображена на рис. 1.



Поле creator заповнюється у відповідності до поточного користувача. Поле actor заповнюється користувачем. Відповідно у нашій системі мають існувати REST ресурс [6] */api/tasks* . Розглянемо приклад коду для кінцевих точок (endpoint) *POST /api/tasks* – для створення сутності завдання та *PUT /api/tasks/{uuid}* – для редагування сутності завдання. Розглянемо частину потенційного спрощеного лістингу коду контроллеру:

```
@Controller('api/tasks')
@UseGuards(JwtAuthGuard)
export class Tasks {
  constructor(
    private readonly taskGateway: ITaskGateway,
    private readonly userGateway: IUserGateway,
    private readonly canBeUpdatedSpec: CanBeUpdatedSpecification,
    private readonly validDataSpec: ValidDataSpecification,
  ) {}

  // ...
  @Post()
  public async create(
    @Body() data: TaskRequest,
    @CurrentUser() userRetriever,
  ): Promise<TaskResponse> {
    const entity = data.toEntity(new Task());
    entity.creator = await userRetriever();
    entity.actor = await this.getUserOrFail(data.actorId);
    await this.validDataSpec.isSatisfiedOrFail(entity);
    const task = await this.taskGateway.create(entity);

    return TaskResponse.fromEntity(task);
  }

  @Put(':id')
  public async update(
    @Param('id') id: string,
    @Body() data: TaskRequest,
  ): Promise<Action> {
    const dbTask = await this.getTaskOrFail(id);
    await this.canBeUpdatedSpec.isSatisfiedOrFail(dbTask);
    const entity = data.toEntity(dbTask);
    if (dbTask.actor.id !== data.actorId) {
      entity.actor = await this.getUserOrFail(data.actorId);
    }
    await this.validDataSpec.isSatisfiedOrFail(entity);
    await this.gateway.update(entity);

    return TaskResponse.fromEntity(task);
  }

  private async getTaskOrFail(id: string): Promise<Action> {
    const entity = await this.taskGateway.getOneById(id);
    if (!entity) throw new NotFoundException();
    return entity;
  }

  private async getUserOrFail(id: string): Promise<Action> {
    const entity = await this.userGateway.getOneById(id);
    if (!entity) throw new LinkedEntityNotFoundException();
    return entity;
  }
}
```

Додатково розглянемо використані класи:

- *UseGuards, JwtAuthGuard* – декоратор що відповідає за аутинтентифікацію контроллера та відповідна стратегія аутентифікації.
- *ITaskGateway, IUserGateway* – абстрактні шлюзи, які відповідають за збереження даних у системі.
- *CurrentUser* – декоратор отримання поточного користувача.
- *TaskResponse, TaskRequest* – DTO, що репрезентують дані запиту та відповіді для веб запиту.
- *CanBeUpdatedSpecification, ValidDataSpecification* – сервіси створені за паттерном специфікація [7].

Наш код на рівні додатку має повертати виключення у наступних випадках:

- Користувач не автентифікований – 401 статус.
- Неправильно сформовані дані запиту – 400 статус.
- Відсутнє завдання у базі даних – 404 статус.

Більшість випадків на сьогоднішній день покриті стандартами та рекомендаціями доволі однозначно. Розглянемо випадки у яких виникають неоднозначності, та як стандарт RFC 9110 вирішує їх вирішує.

Відсутній користувач з ідентифікатором при присвоєнні виконавця. В нашому лістингу, код поверне 404 статус. Зазначимо, що дане питання обговорюється протягом довгого часу у професійній спільності розробників програмного забезпечення. Ми можемо знайти наступні обговорення даного та споріднених питань на популярному ресурсі для розробників Stack Overflow:

- "StackOverflow: 400 vs 422 response to POST that references an unknown entity" [8];
- "Should POST request return 404 if reference to other entity fails?" [9];
- "StackOverflow: 400 vs 422 response to POST of data" [10];
- "StackOverflow: RFC – 404 or 400 for relation of entity not found in PUT request" [11];
- "StackOverflow: RestApi: 404, 422 or 500?" [12].

Звертаємо увагу, що для більшості обговорені і відповідей актуальним RFC є 7231. Дехто з відповідачів посилається на більш старий стандарт RFC 2616. Розглянемо основні варіанти відповідей:

- 404. Частина розробників використовує 404 (Not Found) статус. Тим не менше стандарт говорить про використання даного статусу у випадках коли не знайдено цільовий ресурс. В нашому випадку це ендпойнт на створення чи редагування завдання, тому використання даного статусу не відповідає стандарту.
- 400. Популярним варіантом серед запитувачів також є 400 (Bad Request) статус. Даний статус використовується для невірно сформованих запитів. У нашому випадку, цей статус у контексті поля запиту actorId ми могли б використати якщо поле відсутнє, або не відповідає формату uuid.
- 409. Непопулярним та все ж непоганим кандидатом є 409 (Conflict). Але знову ж, стандарт говорить про неможливість завершити запит через наявність конфлікту стану цільового ресурсу. Тобто такий статус краще використовувати для валідації бізнес правил.
- 422. Саме даний варіант набрав найбільшу кількість голосів у найпопулярнішому обговоренні. Даний код говорить про те, що сервер розуміє суть запиту, але не може виконати поточних інструкцій. Тим не менше на момент даного обговорення статус 422 (Unprocessable Entity) був відсутній в стандарті HTTP, та був описаний в розширенні для WebDAV (RFC 4918) [13]. Саме цим фактом контраргументують інші користувачі. Також автор даної відповіді вказує на даний факт і рекомендує використовувати код 400, який є в RFC 7231 [14] у випадку, якщо використання 422 не виглядає прийнятним.

Таким чином ми можемо бачити, що довгий час існувала неоднозначність навколо даного питання, яку вирішує стандарт RFC 9110, додавши статус 422 в стандарт: *"The 422 (Unprocessable Content) status code indicates that the server understands the content type of the request content (hence a 415 (Unsupported Media Type) status code is inappropriate), and the syntax of the request content is correct, but it was unable to process the contained instructions."*

Друге спірне питання це статус код при виникненні помилки валідації бізнес правил. В контексті нашого прикладу це виклики функції *isSatisfiedOrFail* інтерфейсу специфікації. Ми можемо знайти питання бізнес валідації обговорюється у наступних гілках форумів:

- "422 or 409 status code for existing email during signup" [15];
- "Which HTTP response code for "This email is already registered"?" [16];
- "Should HTTP status codes be used to represent business logic errors on a server?" [17].

Запропоновані варіанти кодів є 400, 409 та 422. Аргументація помилковості коду 400 така ж як і у попередньому випадку – хоча 400 помилка і вважається загальною, та згідно стандарту вона

говорить про некоректно сформований запит. Розглянемо детальніше 409 та 422 коди. 409 говорить про конфлікт з поточним станом додатку. У випадку з бізнес валідаціями, ми зазвичай говоримо про неможливість виконати певну операцію за поточних умов, тобто ми маємо конфлікт з поточним станом, отже 409 код нас влаштовує. Водночас 422 код говорить про те, що сервер розуміє суть запиту, але не може виконати поточних інструкцій. Тобто обидва коди 409 та 422 можуть бути застосовані для помилок бізнес валідацій. Одним з варіантів диференціації даних кодів може бути наш приклад:

- *CanBeUpdatedSpecification* поверне викине помилку що трансформується в 409 код оскільки ми входимо в конфлікт з поточним станом системи.
- *ValidDataSpecification* поверне викине помилку що трансформується в 422 код оскільки ми не можемо виконати дію з новими даними і змінити стан системи на новий.

Висновки. Стаття спрямована на демонстрацію практичних можливостей використання HTTP статус кодів, що визначені в стандарті RFC 9110, зокрема кодів, пов'язаних з помилками валідації HTTP запитів. Шляхом аналізу спірних питань використання HTTP статусів та подальшого розгляду їх практичного застосування в контексті веб-додатків на платформі Node.js, ми показали, як новий стандарт може вплинути на розробку та управління даними в сучасних веб-додатках. Дана стаття спрямована на покращення розуміння можливостей і переваг використання стандартизованих HTTP статус кодів та допоможе розробникам зробити кращі рішення у своїх проектах.

Список літератури:

1. Weaving the Web: the original design and ultimate destiny of the World Wide Web by its inventor / Berners-Lee, T. Harper Business, 2000. 246 c.

2. RFC 9110 HTTP Semantics [Електронний ресурс] / R. Fielding, M. Nottingham, J. Reschke. URL: https://www.rfc-editor.org/rfc/rfc9110.html

3. HTTP response status codes [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

4. NestJS documentation [Електронний ресурс]. URL: https://docs.nestjs.com.

5. A Universally Unique IDentifier (UUID) URN Namespace [Електронний ресурс] / P. Leach, M. Mealling, R. Salz / Липень 2005. URL: https://www.ietf.org/rfc/rfc4122.txt

6. Architectural Styles and the Design of Network-based Software Architectures / R. Fielding – University of California, Irvine. 2000

7. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans. Addison-Wesley Professional, 2003. 560 p.

8. "422 or 409 status code for existing email during signup" [Електронний ресурс]. URL: https://stackoverflow.com/questions/50946698/422-or-409-status-code-for-existing-email-during-signup

9. "Should POST request return 404 if reference to other entity fails?" [Електронний ресурс]. URL: https://stackoverflow.com/questions/20250409/should-post-request-return-404-if-reference-to-other-entity-fails

10. "StackOverflow: 400 vs 422 response to POST of data" [Електронний ресурс]. URL: https://stackoverflow.com/questions/16133923/400-vs-422-response-to-post-of-data

11. "StackOverflow: RFC – 404 or 400 for relation of entity not found in PUT request" [Електронний ресурс]. URL: https://stackoverflow.com/questions/31773088/rfc-404-or-400-for-relation-of-entity-not-found-in-put-request

12. "StackOverflow: RestApi: 404, 422 or 500?" [Електронний ресурс]. URL: https://stackoverflow.com/questions/55166006/restapi-404-422-or-500

13. RFC 4918 HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV) [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc4918

14. RFC 7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc7231

15. "422 or 409 status code for existing email during signup" [Електронний ресурс]. URL: https://stackoverflow.com/questions/50946698/422-or-409-status-code-for-existing-email-during-signup

16. "Which HTTP response code for "This email is already registered"?" [Електронний ресурс]. URL: https://stackoverflow.com/questions/9269040/which-http-response-code-for-this-email-is-already-registered

17. "Should HTTP status codes be used to represent business logic errors on a server?" [Електронний ресурс]. URL: https://softwareengineering.stackexchange.com/questions/341732/should-http-status-codes-be-used-to-represent-business-logic-errors-on-a-server

Ronskyi S.I. PRACTICAL APPLICATION OF RFC 9110 STATUS CODES IN THE CONTEXT OF HTTP REQUEST VALIDATION

It is hard to imagine the modern world without web applications. One of the key tasks in developing an architecture for such applications is the validation of input data and correct standardized HTTP responses. And although the HTTP protocol has existed for more than two decades, controversial issues still arise when developing web applications in the context of the response code status.

This article discusses the use of HTTP status codes. In particular, certain controversial use cases are investigated, and how these issues were resolved in the final standard RFC 9110. At the beginning, the article examines the chronology of changes in HTTP semantics standards, the basic use of status codes. The codes that are used as HTTP request validation error codes in various cases are considered in more detail, namely 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found) and 409 (Conflict). The use of code 422 (Unprocessable Entity), which is introduced in the latest standard, is considered separately and potentially resolves controversial issues.

To describe the problematic issues, an example of the controller code of a web application on the Node.js platform using the NestJS framework is considered. The article describes potential exception handlers related to input data validation and business rule validation. The article includes an analysis of discussions on controversial issues of HTTP response generation in the development community and a comparison of different approaches to using HTTP statuses. The first controversial issue under consideration is the formation of a response in the absence of a related entity in POST and PUT requests. That is, the case in which the client specifies an entity identifier in the request body, but this entity does not exist in the system. The second issue under consideration is the formation of an answer in the event of an exception related to business validation rules. The article points out the pros and cons of the different approaches and provides recommendations for using RFC 9110, including status code 422.

Key words: *hypertext transfer protocol, web application, validation, architecture, code status.*