Специфікація вимог та шляхи реалізації компонент Веб-застосунку «JIPS»

ІПС-41, підгрупа №3

**Бекенд**

Бекенд повинен бути зроблений у із використанням Spring Boot, фреймворком для мови Java. Бекенд притримується тришарової архітектури застосунку MVC. В якості сховища даних використовується реляційна база даних PostgreSQL. Схема бази даних має вигляд, який описаний у відповідному пункті.

У межах Spring Application мають бути такі типи класів:

**Entity-класи** - це класи, які відображають сутності у предметній області. Кожен з таких класів відповідає певній таблиці в базі даних, і один об'єкт цього класу відповідає одному рядку в відповідній таблиці. Такі класи ідентифікуються за допомогою анотацій @Entity та @Table, де вказується відповідна таблиця. Передбачається створення таких класів для Користувача, Замовлення, Товару із замовлення, Товару, Товару з кошика.

**DTO-класи** – класи, що призначені для передавання даних між фронтендом та бекендом. Кожен DTO-клас визначає кількість інформації, яка може передаватися між фронтендом та бекендом.

**Repository-класи** – це інтерфейси, за допомогою яких відбуваються взаємодія бекенду із базою даних. Для кожного Entity-класу має бути описаний Repository.

**Service-класи** – це класи бізнес-логікі застосунку. Вони, використовуючи Repository-класи, для взаємодії із базою даних, збереження, зміна або читання даних, сформованих бізнес-логікою.

**Controller-класи** – це класи, які приймають HTTP-запити від фронтенду та відповідають на них. Вони є інтерфейсами взаємодії користувачів із застосунком. Вони приймають вхідні об’єкти в тілі запиту, зміні шляху та параметри запиту. Усю отриману інформацію вони передають на відповідні сервіси для виконання бізнес-логіка. Отриманий від сервісу результат відправляється у відповідь на фронтенд. API сервера повинно надавати можливість створювати, переглядати та змінювати сутності системи.

**Логування**

На бекенді має бути реалізовано логування помилок у відповідний файл. Все це можливо за допомогою ExceptionHandler-класів. Це такі класи, які перехоплюють виняткові ситуації, Exception, на їх шляху по стеку виклики та дозволяють їх обробити на високому рівні, як, наприклад, залогувати.

**Захист інформації**

Також на бекенді необхідно передбачити захист API від доступу непередбачених користувачів, таким чином, щоб користувачі однієї ролі не мали змогу виконувати функціонал, передбачений для іншої ролі. Для цього необхідно реалізувати реєстрацію та автентифікацію. Для авторизації використовується JWT токен. JWT (JSON Web Token) - це структурований токен, який містить інформацію у JSON форматі і використовується для безпечної передачі даних між двома сторонами.

В системі існує два типи ендпоінтів – які не потребують та потребують авторизації. До тих, що не потребують, відносяться шляхи до реєстрації та автентифікації (sign-up та sign-in). До тих, що потребують, відносяться всі інші. Запит від неавторизованого користувача або від користувача, що не має прав доступу до ресурсу, має відхилятися із з HTTP-кодом 401 (Unauthorized) або 403 (Forbidden), відповідно.

**Обмін даними, валідація вхідних даних та обробка помилок**

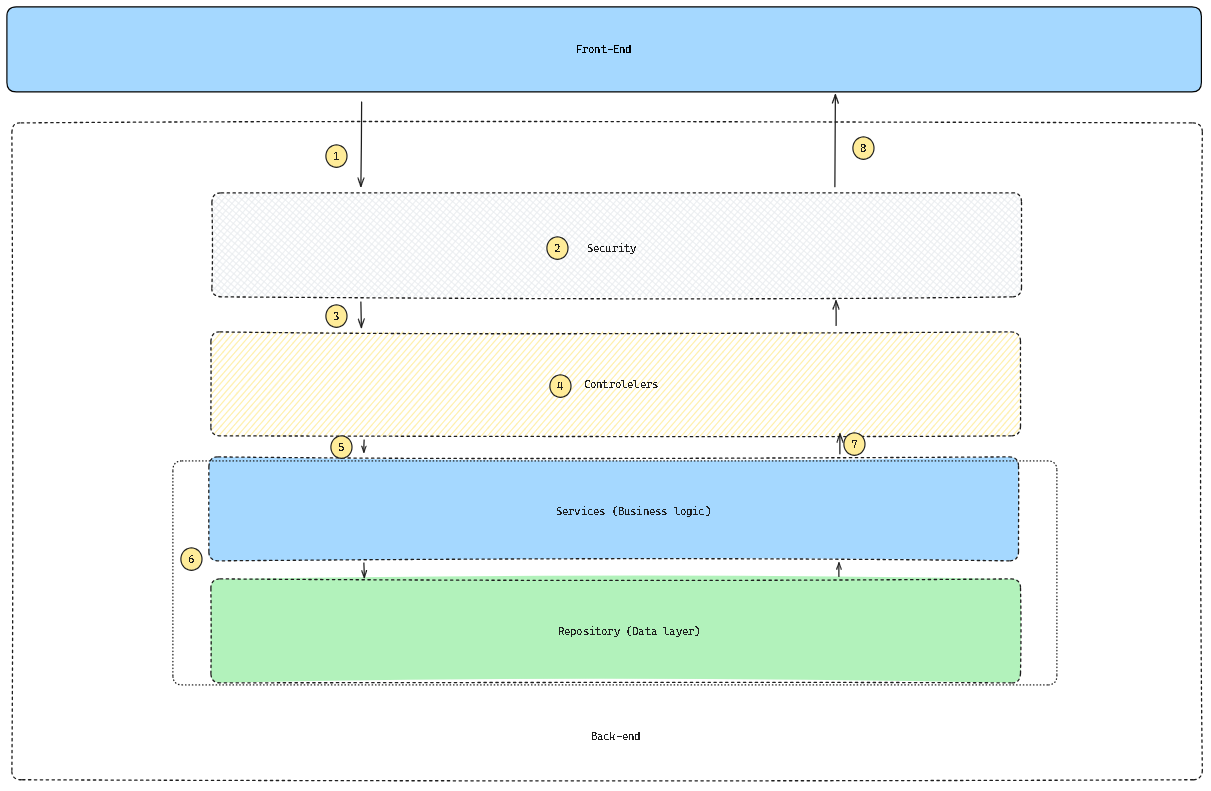
Об’єкти між фронтендом та бекендом передаються у форматі JSON. Їхня серіалізація та десеріалізація відбувається неявно засобами Spring Boot.

Вхідні дані мають обов’язково перевірятися на прості обмеження – наявність важливих полів та їх коректність відносно бізнес-логіки застосунку. Це досягається за допомогою бібліотеки Spring Boot Starter Validation та перевірки правил застосунку на стороні Service-класів. В разі, якщо вхідні дані не проходять перевірку, то на фронтенд відправляється відповідь з HTTP-кодом 400 (Bad Request) та з текстовим описом причини. Фронтенд має взяти до уваги цю відповідь та коректно і зрозуміло представити для користувача суть помилки.

Ту ж саму відповідь буде надано фронтенду в випадку, коли були порушені правила бізнес логіки – відповідь з HTTP-кодом 403. Тобто, бекенд не може повернути на фронтенд відповідь із HTTP-кодом 500 (Internal Server Error).

Загальна схема відповіді бекенда на запит

1. Бекенд отримує запит
2. Перевірка авторизації, прав доступу до ресурсу
3. Вхідні дані десеріалізовуються та валідуються
4. Викликається Controller, який відповідає даному запиту
5. Controller викликає потрібний Service
6. Service перевіряє дані відносно бізнес-логіки та виконує необхідну функцію із залученням Repository-класів
7. Service результат та надсилає його на Contorller
8. Controller отримує відповідь від Service та надсилає її фронтенду.



**Ролі користувачів у Веб-застосунку**

У Веб-застосунку для користувачів передбачено три ролі: неавторизований користувач, користувач.

Механізм набуття ролей наступний:

* **Неавторизований користувач** – неявна роль, яка надається будь-якому не автентифікованому користувачу.
* **Користувач** – це користувач, що зареєстрований у Веб-застосунку. Він може створювати проекти та приймати участь в проектах, виконувати, редагувати завдання, додавати коментарі та прикріплювати файли.

**Фронтенд**

Фронтенд має бути реалізований у вигляді React Single-page Application на мові JavaScript із залученням таких технологій: React Router, Material UI, фреймворк React.

**Управління даними та взаємодія з бекендом**

Основна мета фронтенду полягає в наданні користувачам зручного веб-інтерфейсу для взаємодії з системою. Таким чином, основною функцією фронтенду є відправлення HTTP-запитів на бекенд відповідно до дій користувача, отримання відповідей з бекенду та їхнє відображення у зрозумілому вигляді на веб-сторінці. Також обробка помилок та їхнє відображення, які можуть надходити від сервера у відповідь на запити фронтенду.

Для взаємодії із серверним REST API зазвичай використовують бібліотеку Axios. Згідно стандартам RESTful, кожна кінцева точка API сервера повинна мати відповідний запит Axios для обміну даними.

**Перелік необхідних сторінок**

Проект має містити такі веб-сторінки:

1. Сторінка реєстрації
   1. Можливість вводити свої дані для реєстрації ( пошта, ім’я, прізвище, пароль)
2. Сторінка авторизації
   1. Можливість вводити свої логін та пароль для авторизації (пошта та пароль)
3. Сторінка з інформацією користувача:
   1. Відображати користувачу інформацію про нього
   2. Можливість користувача редагувати цю інформацію (ім’я та прізвище)
4. Сторінка активних проектів:
   1. Відображати користувачу список проектів, в яких він бере участь
   2. Відображення модального вікна для створення нового проекту
5. Сторінка завдання:
   1. Можливість переглядати активні завдання
   2. Можливість видаляти завдання
   3. Можливість переглядати завдання, які відсортовані за пріоритетом
6. Сторінка користувачів проекту:
   1. Можливість додавати нових користувачів до проекту (для власника проекту)
   2. Можливість змінювати ролі користувачів (для власника проекту і менеджера)
   3. Можливість додати нове завдання користувачу, якщо у завдання немає виконавця.
7. Сторінка завдання:
   1. Можливість переглядати інформацію про завдання (назву, опис, дату створення, дату очікуваного завершення, хто працює над завданням, хто створив завдання, пріоритет, стан)
   2. Можливість змінювати інформацію про завдання (назву, опис, дату, очікуваного завершення, хто працює над завданням, хто створив завдання, пріоритет, стан)
   3. Можливість додавати коментарі до завдання та прикріплювати файли.
8. Пошук користувача

Важливою вимогою до вищевказаних веб-сторінок є те, що для кожного типу користувача повинні бути доступні тільки ті елементи керування, які передбачені його роллю.

**Вимоги до хостингу**

Фронтенд буде розгорнутий на сервері, який наявний у замовника.

**Документація**

Бекенд і фронтенд повинні мати документацію у вигляді коментарів у коді. Ці коментарі повинні бути зрозумілими, лаконічними, актуальними і відображати доречність. Також допускається винесення пояснень ключових принципів роботи коду в окремі файли.  
Всі запити від фронтенду до бекенду при успіху повинні повертати код 200.

**База даних**

База даних реалізована за допомогою діалекта мови реляціійних баз даних PostgreSQL. Розгортання відбувається на орендованому сервері.

**Таблиці сутностей бази даних**

user – користувач застосунку.

project – проект, який створює користувач. Має зв’язок з таблицею користувачів, вказуючи на творця проекту.

role – роль користувача в проекті.

project\_user\_role – об’єднує в собі інформацію про зв’язок користувача з курсом, так яку роль він в ньому має.

issue – завдання, яке створює користувач в курс. Має посилання на проект, в якому це завдання створено, а також на користувача, на якого це завдання призначено.

comment – коментар користувача в завдані. Має посилання на користувача та на завдання, в якому він залишив коментар.

attachment – прикріплений файл. Має посилання на завдання та на користувача, який прикріпив файл.

