Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE »**
Система ведення нотаток

Виконав:
Студент групи ІА-24
Шкарніков А. С.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Мета:
Ознайомитися з основними принципами та шаблонами взаємодії між додатками, такими як «Client-Server», «Peer-to-Peer» та «Service-Oriented Architecture», дослідити їхні особливості та способи реалізації для створення гнучких, масштабованих і ефективних програмних систем, зокрема у розробці музичного програвача.

Завдання:
Система ведення нотаток (State, Builder, Observer, Façade, Composite, Client-Server)
В якості нотаток має бути, як мінімум, текст або зображення, додатково можна добавити підтримку файлів розміром до 2Мб, збереження посилань на веб-сторінки, збереження html-тексту з коректним його відображенням. Система повинна дозволяти вести нотатки онлайн, заводити блокноти, прикріпляти до нотаток мітки, давати можливість працювати в системі одночасно багатьом користувачам і одному користувачу працювати з різних комп'ютерів. Користувач повинен мати можливість дати доступ до свого блокноту з замітками іншому користувачу з різними правами (тільки перегляд; перегляд та редагування; перегляд, редагування, додавання та видалення нотаток)

Хід роботи

1. Реалізувати функціонал для роботи в розподіленому оточенні (логіку роботи)

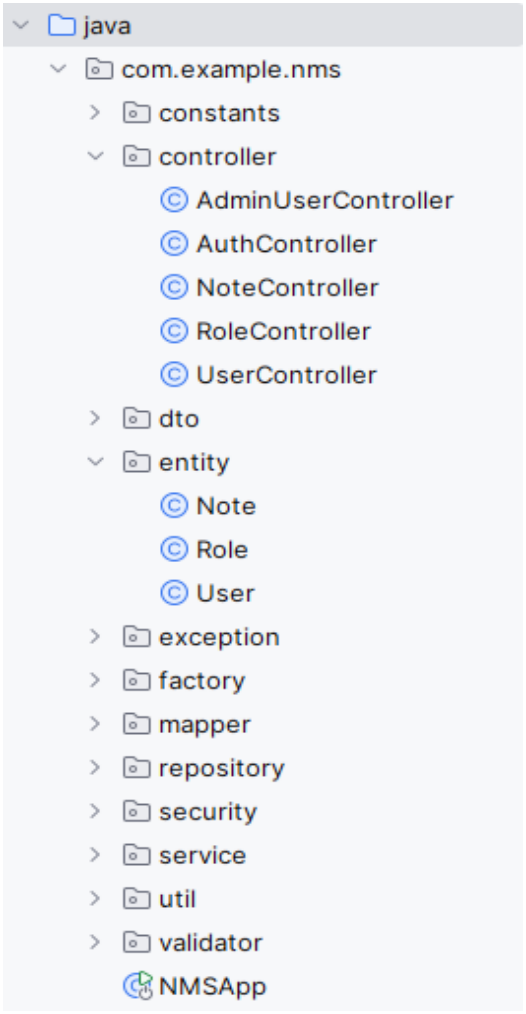

Рис. 1 — Структура проекту сервера

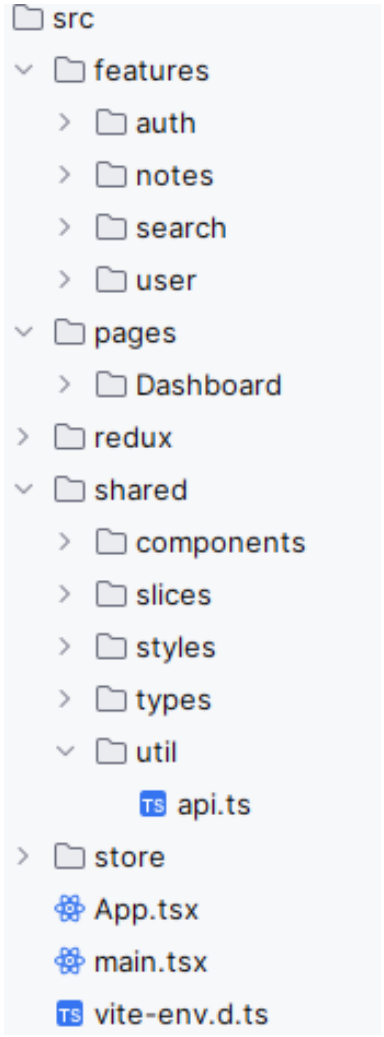

Рис. 2 — Структура проекту клієнта

У наведених структурах проектів ми реалізували серверну (за допомогою Spring Boot) та клієнтську (на React) частини для системи ведення нотаток, які працюють у розподіленому оточенні. Ось деталізований опис кожної частини:

Серверна частина (Spring Boot, Java)

Серверна частина побудована за архітектурою багатошарового додатку з чітким розподілом відповідальностей, що забезпечує модульність, масштабованість та легкість підтримки.

Шари серверної архітектури:

1. Entity (Сутності):
    - Note, User, Role: Сутності, що представляють таблиці бази даних. Вони включають ключові атрибути, такі як заголовок нотатки, ім'я користувача, роль, часові мітки тощо. Зв'язки між сутностями відображають відносини: "один-до-багатьох" (користувачі та їх нотатки) і "багато-до-багатьох" (користувачі та ролі).

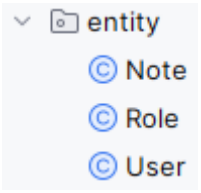

Рис. 3 – Шар Entity

2. Repository (Репозиторії):
   - NoteRepository, UserRepository, RoleRepository: Інтерфейси для доступу до бази даних, які використовують Spring Data JPA для виконання CRUD-операцій. Вони дозволяють працювати з нотатками, користувачами та ролями через спеціалізовані методи, наприклад, пошук нотаток за назвою чи користувачем.

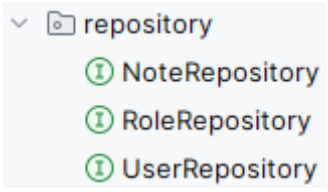

Рис. 4 – Шар Repository

3. Service (Сервіси):
   - Інтерфейси (NoteService, UserService, AuthService) та їх реалізації (NoteServiceImpl, UserServiceImpl, AuthServiceImpl) забезпечують бізнес-логіку, включаючи створення нотаток, пошук, автентифікацію та управління користувачами. Ці сервіси абстрагують взаємодію з репозиторіями та обробляють логіку, необхідну для роботи програми.

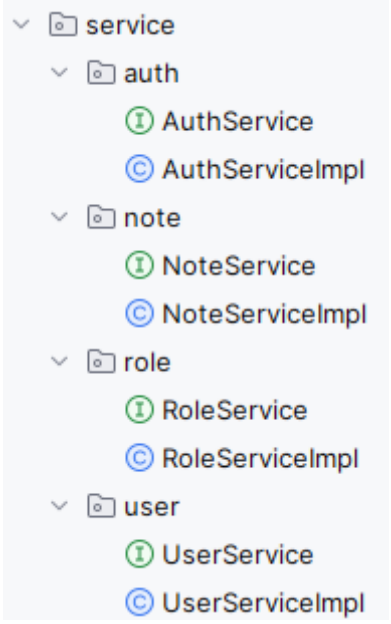

Рис. 5 – Шар Service

4. Security (Безпека):

- Використовуються компоненти для забезпечення автентифікації та авторизації, такі як JwtTokenProvider (для генерації та валідації токенів) та SecurityConfig (для визначення правил доступу до ресурсів).

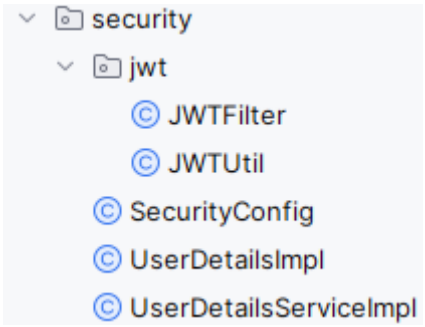

Рис. 6 – Шар Security

5. Web (Веб-шар):
   - Контролери (NoteController, UserController, AuthController) відповідають за обробку HTTP-запитів від клієнта. Наприклад, NoteController виконує операції з нотатками, такі як створення, редагування та видалення.

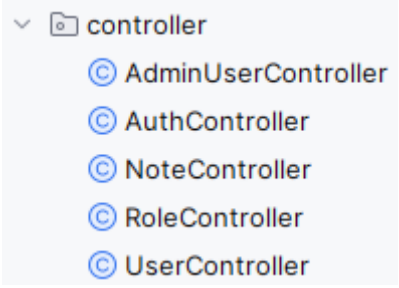

Рис. 7 – Шар Web

6. DTO (Об'єкти передачі даних):
   - NoteDTO, UserDTO, AuthResponseDTO використовуються для передачі даних між клієнтом і сервером, спрощуючи структуру запитів та відповідей.

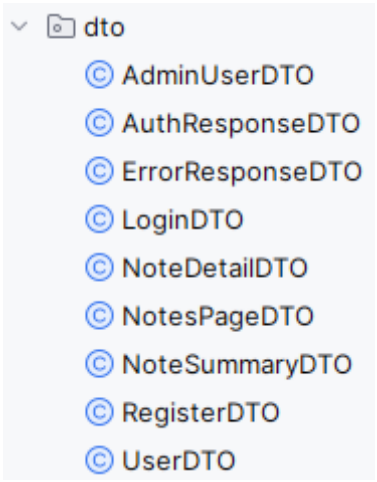

Рис. 8 – Шар DTO

7. Основний клас:

- NMSApp: Відповідає за ініціалізацію сервера, конфігурацію залежностей і запуск програми.

Функціонал серверу:

- Управління нотатками: створення, редагування, видалення, пошук.
- Автентифікація та авторизація користувачів через JWT-токени.
- Обробка REST-запитів для інтерактивної роботи з клієнтською частиною.

Клієнтська частина (Kotlin + Java)

Клієнтська частина реалізована для взаємодії із сервером через API та забезпечення інтуїтивного користувацького інтерфейсу. Вона побудована на основі React з використанням Redux для управління станом і забезпечує функціональність створення, редагування, перегляду нотаток, а також керування доступом до блокнотів.

Шари клієнтської архітектури:

1. **Features (Особливості):**
   - **Auth:**
     Відповідає за авторизацію та автентифікацію користувачів. Містить компоненти для форм входу/реєстрації та управління токенами доступу. Реалізує виклики API для входу, реєстрації та оновлення сесії.
   - **Notes:**
     Забезпечує роботу з нотатками: створення, редагування, видалення та відображення списку. Включає компоненти для роботи з текстом.
   - **Search:**
     Реалізує функціонал пошуку нотаток та блокнотів за ключовими словами. Забезпечує швидку інтеграцію з іншими компонентами, наприклад, відображення результатів у списку нотаток.
   - **User:**
     Відповідає за відображення інформації про користувача, редагування профілю.
2. **Redux (Управління станом):**
   - Використовує Redux для централізованого управління станом програми.
   - Включає слайси для окремих модулів, таких як authSlice (для роботи з токенами та сесіями користувачів), notesSlice (для зберігання нотаток), та userSlice (для управління інформацією про користувачів).

- Забезпечує підтримку асинхронних запитів через middleware, наприклад, Redux Thunk.

3. **Shared (Спільні компоненти):**
   - **Components:**
     Містить спільні UI-компоненти, такі як кнопки, модальні вікна, текстові поля і повідомлення про помилки. Ці компоненти перевикористовуються в різних частинах додатку.
   - **Slices:**
     Окремі фрагменти стану (state) для кожного модуля, такі як authSlice, notesSlice. Вони забезпечують логіку дій (actions) і оновлення стану (reducers).
   - **Util:**
     Містить утилітарні функції для роботи з API та обробки помилок. Реалізує базову конфігурацію клієнта axios для виконання HTTP-запитів до сервера, включаючи автоматичне додавання JWT-токена до заголовків запитів. Також забезпечує функції для обробки неочікуваних помилок і помилок сервера (handleUnexpectedError, handleBackendError), які дозволяють логувати помилки та інформувати користувачів через сповіщення. Цей файл є ключовим для уніфікації викликів API та обробки помилок у додатку.
4. **Store (Стан додатку):**
   - Центральне сховище Redux, яке інтегрує всі слайси програми.
   - Забезпечує єдину точку зберігання даних, до яких мають доступ усі компоненти.

Функціонал клієнта:

- Інтуїтивний інтерфейс для створення, редагування, перегляду та видалення нотаток.
- Авторизація користувачів і управління доступом до нотаток.
- Пошук нотаток за заголовками та вмістом.
- Взаємодія із сервером через REST API для отримання, оновлення та зберігання даних.
- Використання Redux для управління станом та синхронізації між компонентами.

2. Реалізувати взаємодію розподілених частин

```
@RestController
@RequestMapping("/api/notes")
@RequiredArgsConstructor
public class NoteController {

    private final NoteService noteService;
    private final NoteMapper noteMapper;
    private final UserService userService;
    private final NoteDTOValidator noteDTOValidator;

    @GetMapping("/title/{title}")
    public ResponseEntity<NoteDetailDTO> findByTitle(@PathVariable("title") String title) {...}

    @GetMapping
    public ResponseEntity<NotesPageDTO> findAll(
            @RequestParam(name = "page", defaultValue = "0") int page,
            @RequestParam(name = "size", defaultValue = "20") int size,
            @RequestParam(name = "direction", defaultValue = "asc") String direction,
            @RequestParam(name = "sortBy", defaultValue = "updatedAt") String sortBy) {...}

    @GetMapping("/search")
    public ResponseEntity<NotesPageDTO> searchNotes(
            @RequestParam(name = "term") String term,
            @RequestParam(name = "searchInContents", defaultValue = "false") boolean searchInContent
            @RequestParam(name = "page", defaultValue = "0") int page,
            @RequestParam(name = "size", defaultValue = "20") int size,
            @RequestParam(name = "direction", defaultValue = "asc") String direction,
            @RequestParam(name = "sortBy", defaultValue = "updatedAt") String sortBy) {...}

    @PostMapping
    public ResponseEntity<Object> createNote(@RequestBody @Valid NoteDetailDTO noteDTO, BindingResul

    @PatchMapping("/title/{title}")
    public ResponseEntity<NoteDetailDTO> updateNote(@PathVariable("title") String titleOfNoteToBeUpd
            @RequestBody @Valid NoteDetailDTO updatedNoteDTO, BindingResult br) {...}

    @DeleteMapping("/title/{title}")
    public ResponseEntity<Void> deleteNote(@PathVariable("title") String title) {...}
```

Рис. 9 — Реалізація АПІ нотаток на стороні сервера

```
import ...

export const NoteService = {  Show usages
  async createNote(note: NoteSummaryDTO | NoteDetailDTO) : Promise<AxiosResponse<any, any>>  {
    return api.post(`/notes`, note);
  },

  async getNotes(pagination: PaginationOptions) : Promise<NotesPageDTO>  {
    const { page, size, direction, sortBy } = pagination;
    const response : AxiosResponse<NotesPageDTO, any>  = await api.get<NotesPageDTO>(
      `/notes?page=${page}&size=${size}&direction=${direction}&sortBy=${sortBy}`
    );
    return response.data;
  },

  async getNoteByTitle(title: string) : Promise<NoteDetailDTO>  {...},

  async searchNotes(
    term: string,
    searchType: SearchType,
    pagination: PaginationOptions
  ) : Promise<NotesPageDTO>  {...},

  async updateNote(title: string, note: NoteSummaryDTO | NoteDetailDTO) : Promise<NoteDetailDTO>  {...},

  async deleteNote(title: string) : Promise<any>  {...},
};
```

Рис. 4 — Реалізація АПІ нотаток на стороні клієнта

У цьому прикладі взаємодія клієнта та сервера реалізована через REST API. Сервер надає кінцеві точки (endpoints) для роботи з ресурсами нотаток, а клієнт звертається до них через HTTP-запити. Розглянемо детально, як ця взаємодія працює..

Серверна частина: NoteController

Серверна частина реалізує REST API через клас NoteController, який:

1. Обробляє HTTP-запити (GET, POST, PATCH, DELETE) до /api/notes.
2. Використовує сервіси NoteService і UserService для виконання бізнес-логіки та роботи з базою даних.

Основні функції сервера:

- **Отримання списку нотаток**: Метод findAll() обробляє запит GET /api/notes, отримує список нотаток поточного користувача із сервісу, застосовує мапер для перетворення їх у DTO та повертає результат у форматі JSON.

- **Отримання нотатки за заголовком**: Метод findByTitle(@PathVariable String title) обробляє запит GET /api/notes/title/{title}, знаходить нотатку за заголовком через сервіс і повертає деталі нотатки у форматі JSON.
- **Створення нової нотатки**: Метод createNote(@RequestBody NoteDetailDTO noteDTO) обробляє запит POST /api/notes, перевіряє валідацію через NoteDTOValidator, створює нову нотатку через сервіс і повертає статус 201 Created з посиланням на ресурс.
- **Оновлення нотатки**: Метод updateNote(@PathVariable String title, @RequestBody NoteDetailDTO updatedNoteDTO) обробляє запит PATCH /api/notes/title/{title}, знаходить існуючу нотатку, оновлює її через сервіс і повертає оновлену нотатку у відповідь.
- **Видалення нотатки**: Метод deleteNote(@PathVariable String title) обробляє запит DELETE /api/notes/title/{title}, видаляє нотатку через сервіс і повертає статус 204 No Content.

Сервер забезпечує повний набір CRUD-операцій із ресурсом нотаток, повертаючи відповіді у форматі JSON.

Клієнтська частина: NoteService

Клієнт реалізує логіку доступу до серверу через об’єкт NoteService, який:

1. Використовує HTTP-клієнт axios для виконання запитів до серверу.
2. Забезпечує асинхронну модель виконання запитів із обробкою відповідей.

Основні функції клієнта:

- **Отримання списку нотаток:** Метод getNotes(pagination) надсилає запит GET /api/notes із параметрами пагінації (page, size, direction, sortBy) та повертає список нотаток у форматі NotesPageDTO.
- **Отримання нотатки за заголовком:** Метод getNoteByTitle(title) надсилає запит GET /api/notes/title/{title} для отримання детальної інформації про нотатку у форматі NoteDetailDTO.
- **Створення нової нотатки:** Метод createNote(note) надсилає запит POST /api/notes із даними нової нотатки у форматі NoteDetailDTO або NoteSummaryDTO.
- **Оновлення нотатки:** Метод updateNote(title, note) надсилає запит PATCH /api/notes/title/{title} із оновленими даними нотатки.
- **Видалення нотатки:** Метод deleteNote(title) надсилає запит DELETE /api/notes/title/{title} для видалення нотатки за заголовком.

Клієнт отримує відповіді у форматі JSON, які десеріалізуються в DTO-об’єкти для зручності використання.

Як реалізована взаємодія:

1. **Загальна схема:**
   - Клієнт (NoteService) викликає методи, що генерують HTTP-запити до кінцевих точок (NoteController) на сервері.
   - Сервер обробляє запити через сервіси та мапери, повертаючи відповіді у форматі JSON.
2. **Формат передачі даних:**
   - Дані передаються у форматі JSON через HTTP:
     - При POST /api/notes клієнт передає JSON із полями нотатки.
     - При GET /api/notes сервер повертає JSON зі списком нотаток.
3. **Асинхронність:** Клієнтська частина використовує асинхронні функції (async/await), забезпечуючи зручне виконання запитів без блокування основного потоку.
4. **Протокол HTTP:** Використання стандартних методів HTTP (GET, POST, PATCH, DELETE) забезпечує сумісність і стандартизовану роботу між клієнтом і сервером.
5. **Розподіленість:** Клієнт і сервер працюють незалежно, обмінюючись даними через REST API, що дозволяє масштабувати їх окремо.

Висновки:

У ході лабораторної роботи було реалізовано клієнт-серверну архітектуру для взаємодії між клієнтом і сервером у застосунку. Це дозволило забезпечити чіткий розподіл обов’язків між компонентами, спростити обробку запитів, полегшити додавання нового функціоналу та створити масштабовану й гнучку систему для розробки системи ведення нотаток.

Код проекту:

https://github.com/Atl4sDev/TRPZ_CS