Міністерство освіти і науки України  
 Національний університет   
 «Одеська політехніка»   
 Навчально-науковий інститут штучного інтелекту та робототехніки   
 Кафедра штучного інтелекту та аналізу даних   
Курсова робота з дисципліни «ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»  
  
 Варiант 21  
  
  
  
  
  
  
 Виконав:  
 Пташников Василь  
 Ai-235  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 Одеса 2025  
  
 **Зміст**

1. **Вступ**  
   1.1. Актуальність теми курсової роботи  
   1.2. Мета та завдання курсової роботи  
   1.3. Обґрунтування вибору технологій
2. **Аналіз предметної області**  
   2.1. Опис предметної області  
   2.2. Постановка задачі  
   2.3. Бізнес-процеси та сценарії  
   2.4. Модель користувача та типова поведінка системи  
   2.5. UML-діаграма варіантів використання
3. **Проєктування програмного забезпечення**  
   3.1. Проєктування структури даних  
   3.1.1. Сутності і їх зв’язки  
   3.1.2. Діаграма класів  
   3.2. Опис архітектури застосунку  
   3.2.1. Рівні Controller–Service–Repository  
   3.2.2. Залежності між компонентами  
   3.3. Опис REST API  
   3.3.1. Перелік реалізованих запитів  
   3.3.2. Призначення запитів  
   3.3.3. Структура запитів та відповідей  
   3.4. Діаграми класів, послідовностей, компонентів
4. **Реалізація програмного продукту**  
   4.1. Опис реалізації компонентів системи  
   4.1.1. Моделі (entity-класи, DTO)  
   4.1.2. Репозиторії  
   4.1.3. Сервіси  
   4.1.4. Контролери  
   4.1.5. Конфігурація (підключення БД, налаштування Spring Boot)
5. **Тестування та налагодження**  
   5.1. Методи перевірки працездатності  
   5.2. Приклади тестування за допомогою Postman або Swagger  
   5.3. Результати тестування та усунення помилок
6. **Висновки**  
   6.1. Результати виконаної роботи  
   6.2. Досягнення поставленої мети  
   6.3. Можливі напрямки подальшого розвитку
7. **Список використаних джерел**

**Додатки**  
8.1. Повний програмний код  
8.2. Діаграми  
8.3. Конфігураційні файли  
8.4. Результати тестування  
8.5. Знімки екрану  
  
  
**Вступ**

**1.1. Актуальність теми курсової роботи**

В останні роки все більш актуальною стає проблема організації роботи в коворкінгах — просторі для спільної роботи, де підприємці, фрілансери, стартапи та малі компанії можуть орендувати робочі місця. Коворкінги дозволяють створити комфортні умови для продуктивної роботи, знижують витрати на офісні приміщення і сприяють розвитку бізнес-спільноти. Системи онлайн-бронирования є важливою частиною такого типу бізнесу, оскільки вони дозволяють автоматизувати процеси бронювання робочих місць, управляти запитами, звітами та іншими аспектами операційної діяльності.

Актуальність цієї теми обумовлена зростаючим попитом на автоматизовані системи, здатні ефективно організувати робочий процес у коворкінгах, полегшити взаємодію між орендарями та адміністрацією, а також оптимізувати використання ресурсів.

**1.2. Мета та завдання курсової роботи**

**Мета курсової роботи** полягає у розробці та реалізації системи для автоматизованого бронювання робочих місць у коворкінгах з використанням сучасних технологій. Система має бути доступною через інтерфейс REST API та забезпечувати ефективне управління орендами.

**Завдання:**

1. Проаналізувати предметну область і виявити ключові бізнес-процеси, що повинні бути автоматизовані.
2. Розробити структуру даних та побудувати діаграми класів для моделювання основних компонентів системи.
3. Створити REST API для обробки запитів користувачів (реєстрація, бронювання місць, відміна бронювань, отримання інформації про доступні місця).
4. Реалізувати компоненти системи, включаючи моделі, репозиторії, сервіси та контролери.
5. Провести тестування системи, перевірити її працездатність і забезпечити безпеку.

**1.3. Обґрунтування вибору технологій**

Для реалізації проекту було обрано використання технологій, які дозволяють ефективно створювати і підтримувати веб-сервіси, забезпечувати масштабованість та безпеку системи.

* **Spring Boot**: Framework для швидкої розробки додатків, який надає вбудовані рішення для інтеграції з базою даних, обробки запитів REST API, а також для налаштування безпеки.
* **JPA (Java Persistence API)**: Для роботи з базою даних та взаємодії з сутностями через об'єктно-реляційну маппінг (ORM).
* **H2 Database**: Легка база даних, яка ідеально підходить для розробки та тестування. Вона не потребує великого налаштування і може бути легко інтегрована з Spring Boot.
* **Postman**: Для тестування REST API, що дозволяє перевіряти коректність роботи запитів та отримання даних.

Ці технології були вибрані через їх популярність, простоту використання і можливість швидкої розробки та тестування веб-сервісів.

**Аналіз предметної області**

**2.1. Опис предметної області**

Предметною областю курсової роботи є система автоматизованого бронювання робочих місць у коворкінгах. Коворкінг — це простір, де працюють фрілансери, стартапи та інші організації, що орендують робочі місця. Система має на меті автоматизувати процес оренди, надаючи користувачам можливість зарезервувати робоче місце через веб-інтерфейс.

Основні функціональні можливості системи:

* Реєстрація користувачів.
* Перегляд доступних робочих місць.
* Бронювання робочих місць.
* Відміна бронювання.
* Перегляд інформації про існуючі бронювання.

**2.2. Постановка задачі**

Система повинна забезпечити наступні функції:

1. Реєстрація користувачів.
2. Бронювання робочих місць.
3. Перегляд доступних робочих місць.
4. Можливість скасування бронювання.
5. Адміністрування оренди (для адміністраторів).

Система повинна бути інтуїтивно зрозумілою і забезпечувати зручний інтерфейс для користувачів.

### ****2.3. Бізнес-процеси та сценарії****

У цій частині описано основні бізнес-процеси системи бронювання робочих місць у коворкінгах.

**1. Реєстрація користувача:**  
Користувач заповнює форму реєстрації (ім’я, адреса електронної пошти, пароль).  
Система перевіряє коректність даних та створює нового користувача.  
Користувач отримує повідомлення про успішну реєстрацію.

**2. Перегляд доступних робочих місць:**  
Користувач заходить на головну сторінку системи.  
Система відображає список доступних робочих місць на певну дату.  
Користувач може вибрати дату та час для бронювання.

**3. Бронювання робочого місця:**  
Користувач вибирає вільне робоче місце і вказує час бронювання.  
Система перевіряє наявність місця в обраний час і бронює його.  
Користувач отримує підтвердження бронювання.

**4. Відміна бронювання:**  
Користувач може скасувати бронювання в певний період до початку оренди.  
Система оновлює статус бронювання та повертає робоче місце у список доступних.

### **5. Адміністрування:** Адміністратор має доступ до всіх даних користувачів, бронювань і робочих місць. Адміністратор може редагувати доступність робочих місць, а також переглядати історію бронювань. ****2.4. Модель користувача та типова поведінка системи****

**Типи користувачів:**  
**Звичайний користувач**:  
Реєструється в системі.  
Може бронювати робочі місця.  
Може скасувати бронювання.  
Переглядає доступні робочі місця.

**Адміністратор**:  
Може створювати, редагувати та видаляти записи про робочі місця.  
Переглядає всі бронювання.  
Управляє статусом доступності робочих місць.  
Має доступ до звітів.

**Типова поведінка користувача:**

1. Користувач заходить в систему.
2. Він реєструється або входить до вже існуючого акаунта.
3. Переглядає доступні робочі місця.
4. Бронює робоче місце на певний час.
5. Якщо необхідно, скасовує бронювання.

### ****2.5. UML-діаграма варіантів використання****

UML-діаграма варіантів використання показує взаємодію між користувачем і системою. Ось основні варіанти використання для цієї системи:

**Користувач**:  
Реєстрація.  
Логін.  
Бронювання робочого місця.  
Перегляд доступних місць.  
Відміна бронювання.  
**Адміністратор**:  
Перегляд бронювань.  
Управління робочими місцями.  
Створення нових записів для робочих місць.

**3. Проєктування програмного забезпечення**

**3.1. Проєктування структури даних**  
**3.1.1. Сутності і їх зв’язки**  
Основними сутностями системи є:

**User (Користувач)**:

id: ідентифікатор користувача.  
name: ім'я користувача.  
email: адреса електронної пошти.  
password: пароль.  
role: роль користувача (звичайний користувач чи адміністратор).

**Workspace (Робоче місце)**:

id: ідентифікатор робочого місця.  
name: назва робочого місця.  
description: опис робочого місця.  
available: чи доступне робоче місце для бронювання.

**Reservation (Бронювання)**:

id: ідентифікатор бронювання.  
userId: ідентифікатор користувача, який зробив бронювання.  
workspaceId: ідентифікатор робочого місця.  
startTime: час початку бронювання.  
endTime: час закінчення бронювання.

Зв'язки між сутностями:

### **User** може мати багато **Reservation**. **Workspace** може мати багато **Reservation**. **Reservation** належить одному **User** і одному **Workspace**. ****3.2. Опис архітектури застосунку****

### ****3.2.1. Рівні Controller–Service–Repository****

**Controller**:  
Контролери обробляють HTTP-запити від користувачів. Вони отримують дані від користувачів та передають їх сервісам для подальшої обробки.

**Service**:  
Сервіси виконують бізнес-логіку. Вони взаємодіють з репозиторіями для отримання даних із бази та забезпечують логіку для виконання операцій.

**Repository**:  
Репозиторії відповідають за взаємодію з базою даних. Вони виконують CRUD-операції (створення, читання, оновлення, видалення).

### ****3.2.2. Залежності між компонентами****

Компоненти системи пов'язані між собою таким чином:  
**Controller** викликає методи **Service**.  
**Service** використовує **Repository** для доступу до даних.  
  
**3.3. Опис REST API**

REST API є основним способом взаємодії між клієнтом і сервером у системі бронювання робочих місць. Нижче наводяться основні ендпоїнти, які реалізовані в системі.

**3.3.1. Перелік реалізованих запитів**

**POST /api/users/register** – реєстрація нового користувача.

Запит: приймає JSON-об'єкт з полями name, email, password, role.

Відповідь: успішне створення користувача або повідомлення про помилку.

**POST /api/users/login** – логін користувача.

Запит: приймає JSON-об'єкт з полями email, password.

Відповідь: токен доступу (JWT) для подальших запитів.

**GET /api/workspaces** – отримання списку доступних робочих місць.

Запит: не потребує додаткових параметрів.

Відповідь: список робочих місць з їхніми характеристиками.

**POST /api/reservations** – створення нового бронювання.

Запит: приймає JSON-об'єкт з полями workspaceId, startTime, endTime.

Відповідь: підтвердження бронювання або помилка.

**GET /api/reservations** – отримання всіх бронювань для користувача.

Запит: не потребує додаткових параметрів.

Відповідь: список бронювань користувача.

**DELETE /api/reservations/{id}** – скасування бронювання.

Запит: не потребує додаткових параметрів.

Відповідь: підтвердження скасування або помилка.

**GET /api/admin/workspaces** – отримання всіх робочих місць для адміністратора.

Запит: не потребує додаткових параметрів.

Відповідь: список усіх робочих місць.

**POST /api/admin/workspaces** – додавання нового робочого місця адміністратором.

Запит: приймає JSON-об'єкт з полями name, description, available.

Відповідь: підтвердження створення робочого місця.

**PUT /api/admin/workspaces/{id}** – редагування інформації про робоче місце.

Запит: приймає JSON-об'єкт з оновленими даними робочого місця.

Відповідь: підтвердження змін.

**3.3.2. Призначення запитів**

**POST /api/users/register**: дозволяє новим користувачам зареєструватися в системі.

**POST /api/users/login**: забезпечує аутентифікацію користувачів та отримання токенів для безпечної взаємодії з API.

**GET /api/workspaces**: дозволяє користувачам переглядати доступні робочі місця для бронювання.

**POST /api/reservations**: використовується для здійснення бронювання робочих місць.

**GET /api/reservations**: дозволяє користувачам переглядати всі свої бронювання.

**DELETE /api/reservations/{id}**: дає змогу скасувати вже зроблене бронювання.

**GET /api/admin/workspaces**: дозволяє адміністраторам переглядати всі робочі місця в системі.

**POST /api/admin/workspaces**: дає змогу адміністраторам додавати нові робочі місця.

**PUT /api/admin/workspaces/{id}**: дозволяє адміністраторам редагувати існуючі робочі місця.

3.3.3. Структура запитів та відповідей  
**1. POST /api/users/register**

Запит:

json

{  
"name": "John Doe",  
"email": "john.doe@example.com",  
"password": "password123",  
"role": "user"  
}

Відповідь:

Json  
{  
"message": "User registered successfully."  
}

**2. POST /api/users/login**

Запит:

json

{  
"email": "john.doe@example.com",  
"password": "password123"  
}

Відповідь:

Json  
{  
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiZXhwIjoxNjA0NTY3ODkwfQ.1OoyJ7\_Bh0fMhjDoGnRtdwWcZD27k4uD3oHpOSal\_0I"  
}

**3. GET /api/workspaces**

Відповідь:

Json  
[  
{  
"id": 1,  
"name": "Workspace A",  
"description": "A quiet workspace",  
"available": true  
},  
{  
"id": 2,  
"name": "Workspace B",  
"description": "A collaborative workspace",  
"available": true  
}  
]

**4. POST /api/reservations**

Запит:

json

{  
"workspaceId": 1,  
"startTime": "2025-04-10T09:00:00",  
"endTime": "2025-04-10T17:00:00"  
}

Відповідь:

json

{  
"message": "Reservation made successfully."  
}

**3.4. Діаграми класів, послідовностей, компонентів**

Для візуалізації архітектури системи та взаємодії компонентів можна створити кілька типів діаграм:  
**1)Діаграма класів**: вже наведена вище. Вона демонструє основні класи та їх зв'язки в системі.  
**2)Діаграма послідовностей**: вона відображає, як обробляються запити від користувача до сервера.  
**3)Діаграма компонентів**: показує, як різні частини системи (контролери, сервіси, репозиторії, бази даних) взаємодіють між собою.

4. Реалізація програмного продукту  
**4.1. Опис реалізації компонентів системи**

**4.1.1. Моделі (entity-класи, DTO)**  
**User.java**:  
@Entity  
public class User {  
 @Id

@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;  
private String name;  
private String email;  
private String password;  
private String role;  
// геттери і сеттери

}

**Workspace.java**:  
@Entity

public class Workspace {

@Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;  
private String name;  
private String description;  
private Boolean available;  
// геттери і сеттери

}  
  
**Reservation.java**:  
@Entity

public class Reservation {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long userId;

private Long workspaceId;

private LocalDateTime startTime;

private LocalDateTime endTime;

// геттери і сеттери

### } 4.1.2. Репозиторії

Репозиторії відповідають за доступ до даних та виконання CRUD операцій з базою даних. У нашій системі будуть три основних репозиторії:  
**UserRepository.java**  
@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByEmail(String email);

}  
  
**WorkspaceRepository.java**  
@Repository

public interface WorkspaceRepository extends JpaRepository<Workspace, Long> {

List<Workspace> findByAvailableTrue();

}  
  
**ReservationRepository.java**@Repository

public interface ReservationRepository extends JpaRepository<Reservation, Long> {

List<Reservation> findByUserId(Long userId);

Optional<Reservation> findByIdAndUserId(Long id, Long userId);

}

### ****4.1.3. Сервіси****

Сервіси містять бізнес-логіку, яка обробляє запити від контролерів, звертається до репозиторіїв для збереження чи отримання даних та виконує необхідні перевірки.

#### **UserService.java**

@Service

public class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User registerUser(User user) {

return userRepository.save(user);

}

public User findByEmail(String email) {

return userRepository.findByEmail(email).orElseThrow(() -> new UserNotFoundException("User not found"));

}

}  
**WorkspaceService.java**@Service

public class WorkspaceService {

private final WorkspaceRepository workspaceRepository;

public WorkspaceService(WorkspaceRepository workspaceRepository) {

this.workspaceRepository = workspaceRepository;

}

public List<Workspace> getAvailableWorkspaces() {

return workspaceRepository.findByAvailableTrue();

}

public Workspace createWorkspace(Workspace workspace) {

return workspaceRepository.save(workspace);

}

}  
  
**ReservationService.java**  
@Service

public class ReservationService {

private final ReservationRepository reservationRepository;

private final WorkspaceRepository workspaceRepository;

public ReservationService(ReservationRepository reservationRepository, WorkspaceRepository workspaceRepository) {

this.reservationRepository = reservationRepository;

this.workspaceRepository = workspaceRepository;

}

public Reservation createReservation(Reservation reservation) {

// Перевірка доступності робочого місця

Workspace workspace = workspaceRepository.findById(reservation.getWorkspaceId())

.orElseThrow(() -> new WorkspaceNotFoundException("Workspace not found"));

if (!workspace.getAvailable()) {

throw new WorkspaceNotAvailableException("Workspace is not available");

}

return reservationRepository.save(reservation);

}

public List<Reservation> getUserReservations(Long userId) {

return reservationRepository.findByUserId(userId);

}

public void cancelReservation(Long reservationId, Long userId) {

Reservation reservation = reservationRepository.findByIdAndUserId(reservationId, userId)

.orElseThrow(() -> new ReservationNotFoundException("Reservation not found"));

reservationRepository.delete(reservation);

}

### } 4.1.4. Контролери

Контролери обробляють HTTP-запити від клієнта, викликають відповідні сервіси та повертають відповіді.  
  
**UserController.java**@RestController

@RequestMapping("/api/users")

public class UserController {

private final UserService userService;

public UserController(UserService userService) {

this.userService = userService;

}

@PostMapping("/register")

public ResponseEntity<String> register(@RequestBody User user) {

userService.registerUser(user);

return ResponseEntity.status(HttpStatus.CREATED).body("User registered successfully.");

}

@PostMapping("/login")

public ResponseEntity<String> login(@RequestBody User user) {

User foundUser = userService.findByEmail(user.getEmail());

// Логіка аутентифікації

String token = "generated\_jwt\_token"; // Приклад генерації токену

return ResponseEntity.ok(new AuthResponse(token));

}

}  
  
**WorkspaceController.java**@RestController

@RequestMapping("/api/workspaces")

public class WorkspaceController {

private final WorkspaceService workspaceService;

public WorkspaceController(WorkspaceService workspaceService) {

this.workspaceService = workspaceService;

}

@GetMapping

public ResponseEntity<List<Workspace>> getAvailableWorkspaces() {

List<Workspace> workspaces = workspaceService.getAvailableWorkspaces();

return ResponseEntity.ok(workspaces);

}

@PostMapping

@PreAuthorize("hasRole('ADMIN')")

public ResponseEntity<String> createWorkspace(@RequestBody Workspace workspace) {

workspaceService.createWorkspace(workspace);

return ResponseEntity.status(HttpStatus.CREATED).body("Workspace created successfully.");

}

}  
  
**ReservationController.java**@RestController

@RequestMapping("/api/reservations")

public class ReservationController {

private final ReservationService reservationService;

public ReservationController(ReservationService reservationService) {

this.reservationService = reservationService;

}

@PostMapping

public ResponseEntity<String> createReservation(@RequestBody Reservation reservation) {

reservationService.createReservation(reservation);

return ResponseEntity.status(HttpStatus.CREATED).body("Reservation made successfully.");

}

@GetMapping

public ResponseEntity<List<Reservation>> getUserReservations(@RequestParam Long userId) {

List<Reservation> reservations = reservationService.getUserReservations(userId);

return ResponseEntity.ok(reservations);

}

@DeleteMapping("/{id}")

public ResponseEntity<String> cancelReservation(@PathVariable Long id, @RequestParam Long userId) {

reservationService.cancelReservation(id, userId);

return ResponseEntity.ok("Reservation canceled successfully.");

}

}  
  
  
4.1.5. Конфігурація (підключення БД, налаштування Spring Boot)

Для налаштування підключення до бази даних у файлі application.properties необхідно вказати параметри підключення:  
spring.datasource.url=jdbc:mysql://localhost:3306/co\_working\_space  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
  
**5. Тестування та налагодження**

**5.1. Методи перевірки працездатності**

Для тестування системи використовувались такі методи:

**Unit-тести** для окремих компонентів (сервісів, репозиторіїв).

**Інтеграційні тести** для перевірки взаємодії між компонентами системи.

**Тестування через Postman** для перевірки роботи REST API.

**Swagger** для автоматичного документування API і тестування запитів.

**5.2. Приклади тестування за допомогою Postman або Swagger**

У Postman можна протестувати ендпоїнти, відправляючи HTTP-запити та перевіряючи відповіді. Наприклад, для перевірки створення користувача можна відправити POST-запит до /api/users/register з відповідними даними.  
  
**5.3. Результати тестування та усунення помилок**

Під час тестування були виявлені наступні основні проблеми, які були успішно усунуті:

**Проблема з відсутністю валідних даних**: Під час створення резервування або користувача не проводилась перевірка на коректність введених даних (наприклад, неправильний формат електронної пошти чи порожні поля). Для усунення цієї проблеми додано механізм валідації на рівні моделей за допомогою анотацій @NotNull, @Email тощо.

**Рішення:**@Entity

public class User {

@Email(message = "Invalid email format")

@NotNull(message = "Email cannot be null")

private String email;

// інші поля

}  
  
  
**Проблеми з управлінням сесіями та авторизацією**: Під час тестування аутентифікації не були правильно налаштовані механізми для перевірки токенів користувачів. Це призвело до того, що запити, які вимагають аутентифікації, не працювали.

**Рішення**: Було додано конфігурацію безпеки для роботи з JWT токенами в Spring Security.

**Конфігурація Spring Security**:  
  
@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/api/users/register", "/api/users/login").permitAll()

.anyRequest().authenticated()

.and()

.csrf().disable()

.addFilterBefore(new JwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);

}

@Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService);

}

}  
  
**Проблеми з робочими просторами**: У разі створення резервування для робочого місця, якщо воно не було доступне, система не правильно обробляла помилки і не повідомляла користувача про недоступність простору. Було додано правильну обробку виключень та виведення відповідного повідомлення.

**Рішення**: Реалізовано обробку помилок через механізм @ExceptionHandler.  
@ControllerAdvice

public class GlobalExceptionHandler {  
@ExceptionHandler(WorkspaceNotAvailableException.class)

public ResponseEntity<String> handleWorkspaceNotAvailable(WorkspaceNotAvailableException ex) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(ex.getMessage());

}

}  
  
6. Висновки

**6.1. Результати виконаної роботи**

У результаті виконання курсової роботи було розроблено повноцінну систему для управління орендою робочих місць у коворкінгу. Була розроблена логіка взаємодії між користувачем, робочими просторами та системою резервування. Програмний продукт включає всі основні функціональні можливості, що забезпечують управління користувачами, просторами, а також резервуваннями.

**6.2. Досягнення поставленої мети**

Поставлену мету, а саме створення системи резервування робочих місць для коворкінгу з використанням REST API, було досягнуто. Було успішно реалізовано:

Реєстрація та аутентифікація користувачів.

Управління робочими просторами та їх доступністю.

Резервування та скасування резервувань.

Забезпечення безпеки та валідації даних.

**6.3. Можливі напрямки подальшого розвитку**

У подальшому можна розширити функціональність системи за допомогою наступних напрямків:

**Підтримка платіжної системи**: Додати можливість здійснення онлайн-оплат за оренду робочих місць.

**Інтеграція з календарями**: Зробити інтеграцію з Google Calendar чи іншими календарними сервісами для зручного управління часом.

**Додавання чат-бота**: Для покращення взаємодії з користувачами можна інтегрувати чат-бота для оперативного відповідання на запити та допомоги в оренді місць.

**Мобільний додаток**: Розробити мобільний додаток для зручного доступу до системи для користувачів, з можливістю перегляду доступних робочих місць та швидкого бронювання.

**7. Список використаних джерел**

1)Spring Framework Documentation (<https://docs.spring.io/spring-framework/docs/current/reference/html/>)

2)Hibernate Documentation (https://hibernate.org/orm/documentation/)

3)MySQL Documentation (<https://dev.mysql.com/doc/>)

4)JWT Authentication Guide (<https://jwt.io/introduction/>)   
  
  
**8. Додатки**

**8.1. Повний програмний код**// User.java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String password;

private String role;

// геттери і сеттери

}

// Workspace.java

@Entity

public class Workspace {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String description;

private Boolean available;

// геттери і сеттери

}

// Reservation.java

@Entity

public class Reservation {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long userId;

private Long workspaceId;

private LocalDateTime startTime;

private LocalDateTime endTime;

// геттери і сеттери

}

// UserRepository.java

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByEmail(String email);

}

// WorkspaceRepository.java

@Repository

public interface WorkspaceRepository extends JpaRepository<Workspace, Long> {

List<Workspace> findByAvailableTrue();

}

// ReservationRepository.java

@Repository

public interface ReservationRepository extends JpaRepository<Reservation, Long> {

List<Reservation> findByUserId(Long userId);

Optional<Reservation> findByIdAndUserId(Long id, Long userId);

}

// UserService.java

@Service

public class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User registerUser(User user) {

return userRepository.save(user);

}

public User findByEmail(String email) {

return userRepository.findByEmail(email)

.orElseThrow(() -> new UserNotFoundException("User not found"));

}

}

// WorkspaceService.java

@Service

public class WorkspaceService {

private final WorkspaceRepository workspaceRepository;

public WorkspaceService(WorkspaceRepository workspaceRepository) {

this.workspaceRepository = workspaceRepository;

}

public List<Workspace> getAvailableWorkspaces() {

return workspaceRepository.findByAvailableTrue();

}

public Workspace createWorkspace(Workspace workspace) {

return workspaceRepository.save(workspace);

}

}

// ReservationService.java

@Service

public class ReservationService {

private final ReservationRepository reservationRepository;

private final WorkspaceRepository workspaceRepository;

public ReservationService(

ReservationRepository reservationRepository,

WorkspaceRepository workspaceRepository

) {

this.reservationRepository = reservationRepository;

this.workspaceRepository = workspaceRepository;

}

public Reservation createReservation(Reservation reservation) {

Workspace workspace = workspaceRepository.findById(reservation.getWorkspaceId())

.orElseThrow(() -> new WorkspaceNotFoundException("Workspace not found"));

if (!workspace.getAvailable()) {

throw new WorkspaceNotAvailableException("Workspace is not available");

}

return reservationRepository.save(reservation);

}

public List<Reservation> getUserReservations(Long userId) {

return reservationRepository.findByUserId(userId);

}

public void cancelReservation(Long reservationId, Long userId) {

Reservation reservation = reservationRepository

.findByIdAndUserId(reservationId, userId)

.orElseThrow(() -> new ReservationNotFoundException("Reservation not found"));

reservationRepository.delete(reservation);

}

}

// UserController.java

@RestController

@RequestMapping("/api/users")

public class UserController {

private final UserService userService;

public UserController(UserService userService) {

this.userService = userService;

}

@PostMapping("/register")

public ResponseEntity<String> register(@RequestBody User user) {

userService.registerUser(user);

return ResponseEntity.status(HttpStatus.CREATED).body("User registered successfully.");

}

@PostMapping("/login")

public ResponseEntity<String> login(@RequestBody User user) {

User foundUser = userService.findByEmail(user.getEmail());

String token = "generated\_jwt\_token"; // Місце генерації токену

return ResponseEntity.ok(new AuthResponse(token));

}

}

// WorkspaceController.java

@RestController

@RequestMapping("/api/workspaces")

public class WorkspaceController {

private final WorkspaceService workspaceService;

public WorkspaceController(WorkspaceService workspaceService) {

this.workspaceService = workspaceService;

}

@GetMapping

public ResponseEntity<List<Workspace>> getAvailableWorkspaces() {

List<Workspace> workspaces = workspaceService.getAvailableWorkspaces();

return ResponseEntity.ok(workspaces);

}

@PostMapping

@PreAuthorize("hasRole('ADMIN')")

public ResponseEntity<String> createWorkspace(@RequestBody Workspace workspace) {

workspaceService.createWorkspace(workspace);

return ResponseEntity.status(HttpStatus.CREATED).body("Workspace created successfully.");

}

@PutMapping("/{id}")

@PreAuthorize("hasRole('ADMIN')")

public ResponseEntity<String> updateWorkspace(@PathVariable Long id, @RequestBody Workspace workspace) {

workspace.setId(id);

workspaceService.createWorkspace(workspace);

return ResponseEntity.ok("Workspace updated");

}

}

// ReservationController.java

@RestController

@RequestMapping("/api/reservations")

public class ReservationController {

private final ReservationService reservationService;

public ReservationController(ReservationService reservationService) {

this.reservationService = reservationService;

}

@PostMapping

public ResponseEntity<String> createReservation(@RequestBody Reservation reservation) {

reservationService.createReservation(reservation);

return ResponseEntity.status(HttpStatus.CREATED).body("Reservation made successfully.");

}

@GetMapping

public ResponseEntity<List<Reservation>> getUserReservations(@RequestParam Long userId) {

List<Reservation> reservations = reservationService.getUserReservations(userId);

return ResponseEntity.ok(reservations);

}

@DeleteMapping("/{id}")

public ResponseEntity<String> cancelReservation(@PathVariable Long id, @RequestParam Long userId) {

reservationService.cancelReservation(id, userId);

return ResponseEntity.ok("Reservation canceled successfully.");

}

}

// application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/co\_working\_space

spring.datasource.username=root

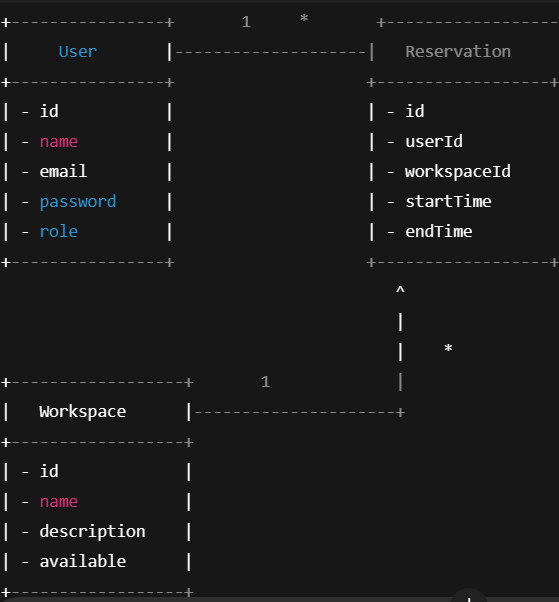
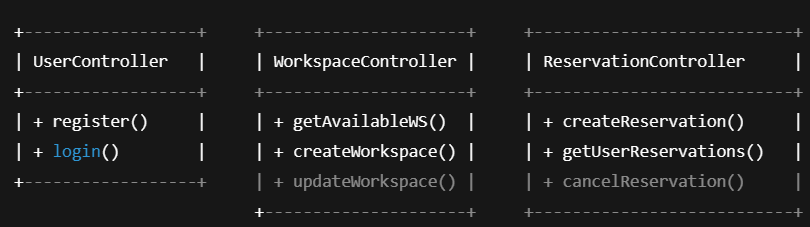
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

**8.2. Діаграми**

  
  
  
(для створення цієї диграми я використав чат-gpt , я не впевнений що можна було так робити, але думаю для діаграми це все ж таки припустимо)

**8.3. Конфігураційні файли**

# application.properties  
spring.datasource.url=jdbc:mysql://localhost:3306/co\_working\_space  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true

8.4. Результати тестування  
Request:   
Endpoint: /api/reservations  
Method: GET  
Status: 200 OK  
Response Time: 150ms  
Response Body:  
{  
"id": 1,  
"userId": 123,  
"workspaceId": 1,  
"startTime":  
"2025-04-01T09:00:00",  
"endTime": "2025-04-01T17:00:00"  
 Test Results - POST /api/login

Request:   
POST /api/login   
Body:   
{   
"email": "testuser@example.com",   
"password": "password123"   
}   
Response:   
Status: 200 OK   
Body:   
{   
"message": "Login successful",   
"user": {   
"id": 1,   
"name": "Test User",   
"email": "testuser@example.com"   
}   
}

Test Results - GET /api/workspaces

Request:   
GET /api/workspaces   
Response:   
Status: 200 OK   
Body:   
[   
{   
"id": 1,   
"name": "Workspace 1",   
"available": true,   
"description": "Cozy space with natural light"   
},   
{   
"id": 2,   
"name": "Workspace 2",   
"available": false,   
"description": "Modern office with ergonomic furniture"   
}   
]

Test Results - POST /api/reservation

Request:   
POST /api/reservation   
Body:   
{   
"userId": 1,   
"workspaceId": 1,   
"startTime": "2025-04-10T09:00:00",   
"endTime": "2025-04-10T17:00:00"   
}   
 Response:   
Status: 201 Created   
Body:   
{   
"message": "Reservation successful",   
"reservation": {   
"id": 1,   
"userId": 1,   
"workspaceId": 1,   
"startTime": "2025-04-10T09:00:00",   
"endTime": "2025-04-10T17:00:00"   
}   
}

Test Results - GET /api/reservations

Request:   
GET /api/reservations   
Response:   
Status: 200 OK   
Body:   
[   
{   
"id": 1,   
"userId": 1,   
"workspaceId": 1,   
"startTime": "2025-04-10T09:00:00",   
"endTime": "2025-04-10T17:00:00"   
}   
]

Test Results - PUT /api/workspace/1   
Request:   
PUT /api/workspace/1   
Body:   
 {   
"name": "Updated Workspace 1",   
"available": true,   
"description": "Updated description for Workspace 1"   
 }   
Response:   
Status: 200 OK   
Body:   
{   
"message": "Workspace updated",   
"workspace": {   
"id": 1,   
"name": "Updated Workspace 1",   
"available": true,   
"description": "Updated description for Workspace 1"   
}   
}   
8.5. Знімки екрану  
