Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра інформаційних систем

Власійчук Данило Олександрович
студент групи АІ-235

КУРСОВА РОБОТА

Розробити backend для застосунку оренди приміщень

Спеціальність:
122 Комп’ютерні науки

Освітня програма: Комп’ютерні науки

Керівник:
Годовіченко Микола Анатолійович,
доцент кафедри

Одеса – 2025

ЗМІСТ

ВСТУП

У сучасному світі технології активно впроваджуються у сферу організації заходів. Оренда приміщень для проведення подій, таких як конференції, семінари, презентації чи святкування, потребує ефективних інструментів для автоматизації процесів бронювання, управління подіями, координації організаторів і гостей. Це зумовлює актуальність створення програмного забезпечення, яке забезпечує зручний доступ до даних, оперативну взаємодію між учасниками та можливість аналітики для прийняття управлінських рішень. Предметна область включає процеси бронювання приміщень, створення подій, облік організаторів і гостей, а також керування запрошеннями.

Для реалізації обрано мову програмування Java завдяки її поширеності в корпоративній розробці та широкій підтримці в екосистемі Spring. Метою даної курсової роботи є розробка бекенд-застосунку для системи оренди приміщень для подій, який реалізує REST API, забезпечує автентифікацію та авторизацію користувачів, управління ключовими сутностями системи (приміщення, події, організатори, гості, запрошення) та надає інструменти для адміністрування.

Реалізований REST API забезпечує взаємодію з фронтендом або сторонніми сервісами відповідно до принципів архітектури клієнт-сервер.
Застосунок реалізовано з використанням: Spring Boot, PostgreSQL, JWT, OAuth2, MapStruct. Архітектура побудована із чітким розподілом між рівнями контролерів, сервісів та репозиторіїв відповідно до принципів інверсії управління та впровадження залежностей .

У межах курсової роботи було реалізовано проєктування об’єктної моделі відповідно до предметної області, організацію взаємозв’язків між сутностями налаштування безпечної автентифікації користувачів за допомогою JWT та OAuth2, тестування основних функцій застосунку, а також розгортання у хмарному середовищі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Система оренди приміщень для подій призначена для автоматизації процесів, пов’язаних із організацією заходів у спеціалізованих залах. Основною метою є створення ефективного цифрового інструменту, що забезпечує зручну взаємодію між адміністраторами, організаторами подій та запрошеними гостями. Програмне забезпечення дозволяє здійснювати повний цикл управління подіями — від планування та бронювання зали до контролю запрошень і аналізу результатів участі.

У межах предметної області система оперує такими основними сутностями: Room, Event , Organizer , Guest та Invitation. Кожна подія пов’язується з конкретною залою, має дату та час проведення, а також організатора. Організатор зберігає контактні дані та відповідає за ініціацію заходів. Гості реєструються в системі та отримують персональні запрошення на події. Кожне запрошення містить інформацію про статус, що дозволяє відслідковувати участь гостей у режимі реального часу.

Система підтримує широкий спектр функцій, зокрема: управління списком залів та перевірка їх доступності на певну дату, створення й редагування подій, додавання організаторів і гостей, надсилання запрошень, а також перегляд статистики по відвідуваності та кількості запрошених. Завдяки цьому реалізується гнучкий та масштабований підхід до управління заходами будь-якого типу.

Для адміністраторів надається інтерфейс керування всіма сутностями, що дозволяє швидко додавати, змінювати або видаляти дані про приміщення, організаторів, події та гостей. Крім того, доступна функція генерації аналітичних звітів, яка дозволяє виявити найбільш популярні події, підрахувати кількість гостей, відстежити динаміку бронювань і приймати обґрунтовані управлінські рішення.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування структури даних

У даному додатку присутні такі сутності:

– Room. Містить такі поля: id: Long, name: String, capacity: Integer, location: String. Визначає інформацію про доступні зали для проведення подій.

– Event. Містить такі поля: id: Long, name: String, dateTime: LocalDateTime, а також зв’язки багато до одного з сутностями Room та Organizer. Представляє подію, яка проводиться у певній залі та організовується конкретним користувачем.

– Organizer. Містить такі поля: id: Long, name: String, contactEmail: String. Визначає особу або організацію, відповідальну за створення та проведення подій.

– Guest. Містить такі поля: id: Long, name: String, email: String. Представляє користувача, який може бути запрошений на певну подію.

– Invitation. Містить такі поля: id: Long, status: String, а також зв’язки багато до одного з сутностями Guest та Event. Визначає запрошення на подію, яке має статус участі.

Зв’язки багато до одного між сутністю Event та сутностями Room і Organizer необхідні для встановлення чіткої відповідності між кожною подією та залом, у якому вона проводиться, а також організатором, який відповідає за її створення. Це забезпечує централізоване керування розкладом подій у конкретних приміщеннях і дозволяє відстежувати активність кожного організатора.

Зв’язок багато до одного між сутністю Invitation та сутностями Event і Guest реалізує механізм запрошення учасників на події. Кожне запрошення пов’язує певного гостя з конкретною подією та зберігає інформацію про статус участі. Це дозволяє контролювати, хто запрошений, хто підтвердив участь або відмовився.

2.2 Опис архітектури застосунку

У даному програмному забезпеченні використано трирівневу архітектуру, яка дозволяє чітко організувати логіку обробки запитів і взаємодії з базою даних. Це забезпечує гнучкість, масштабованість та легкість супроводу застосунку.

Перший рівень — контролери. Вони відповідають за обробку запитів, що надходять через HTTP-протокол. Контролери приймають вхідні параметри, передають їх до відповідних сервісних методів та повертають результат у вигляді JSON-відповіді. Наприклад, контролери дозволяють створювати події, переглядати зали або надсилати запрошення гостям.

Другий рівень — сервіси. Тут реалізується основна бізнес-логіка системи. Сервіси відповідають за перевірку вхідних даних, обробку інформації, виконання обчислень та взаємодію з базою даних через репозиторії. Саме на цьому рівні визначається логіка створення події, перевірки доступності залів, формування списків гостей тощо.

Третій рівень — репозиторії. Вони забезпечують доступ до бази даних за допомогою Spring Data JPA. Репозиторії містять методи для збереження, пошуку, оновлення та видалення сутностей, таких як Room, Event, Guest, Organizer, Invitation. Цей рівень абстрагує застосунок від конкретної реалізації сховища даних.

Взаємозалежність між компонентами є ієрархічною: контролери використовують сервіси для обробки логіки, сервіси, у свою чергу, звертаються до репозиторіїв для виконання операцій з даними. Репозиторії безпосередньо працюють із об’єктами, що відповідають таблицям у базі даних, що забезпечує структуровану та прозору логіку збереження інформації.

Якщо потрібно розуміти на прикладі ,то це буде вигляди якось так

POST /api/events

→ EventController.createEvent(eventDTO)

→ EventService.createEvent(eventDTO)

→ EventRepository.save(new Event(...))

2.3 Опис REST API

У моєму застосунку реалізовано REST API, яке забезпечує взаємодію між клієнтом і сервером через HTTP-запити. Система дозволяє працювати з основними сутностями: Room, Event, Organizer, Guest, Invitation. Кожен запит має чітке призначення, структуру URL, відповідну HTTP-методу, а також формат запиту і відповіді у вигляді JSON.

Нижче наведено приклади запитів, реалізованих у межах проєкту:

1.Створення нової зали:

POST /api/rooms

Приймає JSON-об’єкт з даними про залу (name, capacity, location) і зберігає її в базу.

2.Отримання списку всіх подій:

GET /api/events

Повертає перелік усіх створених подій з їхньою датою, залою та організатором.

3.Додавання нового організатора:

POST /api/organizers

Дозволяє створити нового організатора, передаючи name та contactEmail.

4.Надсилання запрошення гостю:

POST /api/invitations

Приймає guestId, eventId і створює відповідне запрошення зі статусом "очікується".

5.Зміна статусу запрошення:

PATCH /api/invitations/{id}

Дозволяє оновити статус запрошення (наприклад, підтверджено чи відхилено).

Отримання гостей конкретної події:

6.GET /api/events/{eventId}/guests

Повертає список гостей, запрошених на певну подію.

Усі запити супроводжуються перевіркою вхідних даних, обробкою винятків і поверненням відповідей у форматі JSON з HTTP-статусами (200, 201, 400, 404)

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Моделі

В проєкті реалізовані наступні моделі:

– Event. Нижче наведений java-код :

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Event {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private LocalDateTime dateTime;

    @ManyToOne
    @JoinColumn(name = "room_id")
    private Room room;

    @ManyToOne
    @JoinColumn(name = "organizer_id")
    private Organizer organizer;
}
```

– Guest:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Guest {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String email;
}
```

– Invitation:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Invitation {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @ManyToOne
   @JoinColumn(name = "guest_id")
   private Guest guest;

   @ManyToOne
   @JoinColumn(name = "event_id")
   private Event event;

   private String status; //  "INVITED", "CONFIRMED", "DECLINED"
}
```

– Organizer:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Organizer {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String name;

   private String contactEmail;
}
```

– Room:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Room {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
```

```
    private String name;
    private Integer capacity;
    private String location;
}
```

– User:

```
@Entity
@Table(name = "app_user")
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true)
    private Long id;

    private String username;
    private String password;
    private String role;
}
```

3.2 Репозиторії

У створеному застосунку для доступу до бази даних використовується Spring Data JPA. Для кожної сутності реалізовано окремий інтерфейс репозиторію, який розширює інтерфейс JpaRepository. Це дозволяє автоматично використовувати базові CRUD-операції без необхідності явно реалізовувати їх у коді.

Окрім стандартних методів, у репозиторіях були створені додаткові, специфічні для логіки системи. Spring автоматично генерує SQL-запити для методів, названих за відповідними шаблонами. У деяких випадках було реалізовано власні запити з використанням JPQL або анотацій @Query.

Нижче наведено короткий опис реалізованих репозиторіїв та приклади додаткових методів:

– UserRepository. Додано метод findByEmail(String email), який дозволяє знайти користувача за адресою електронної пошти. Повертає результат у вигляді Optional<User>, що зручно для перевірки при автентифікації.

– OrganizerRepository. Додано метод для пошуку організаторів за іменем — findByName(String name), що використовується при створенні або перевірці подій.

– GuestRepository. Створено методи для пошуку гостей за іменем або email-адресою: findByName(String name), findByEmail(String email).

– RoomRepository. Додатково реалізовано метод findByLocationAndCapacity(String location, Integer capacity), який дозволяє знаходити зали за місткістю і розташуванням. Це корисно при плануванні подій.

– EventRepository. Реалізовано кілька спеціалізованих методів:

- findByRoomId(Long roomId)
- findByOrganizerId(Long organizerId)
- findByDateTimeBetween(LocalDateTime from, LocalDateTime to) — для пошуку подій за часовим діапазоном.

– InvitationRepository. Додано методи:

- findByEventId(Long eventId) — повертає всі запрошення на подію,
- findByGuestId(Long guestId) — дозволяє отримати список подій, на які запрошено конкретного гостя.

Ці методи дозволяють будувати гнучкі запити до бази даних без необхідності писати SQL вручну, а у випадках складної логіки — використовувати власноруч задані запити.

3.3 Сервіси

У застосунку використовується сервісний рівень, де кожен клас позначений анотацією @Service. Сервіси містять основну бізнес-логіку: виконують перевірки, працюють з репозиторіями.У разі помилок обробляються стандартні та власні винятки. Також у сервісах використовуються анотації безпеки для обмеження доступу до окремих операцій.

EventService

Цей сервіс відповідає за управління подіями. Він реалізує створення, оновлення, видалення подій, а також надає можливість пошуку за різними

параметрами: зала, організатор, дата.

Метод створення події

```
public Event createEvent(Event event) {
   return eventRepository.save(event);
}
```

Метод оновлення події

```
public Event updateEvent(Long id, Event updatedEvent) {
   return eventRepository.findById(id).map(event -> {
      event.setName(updatedEvent.getName());
      event.setDateTime(updatedEvent.getDateTime());
      event.setRoom(updatedEvent.getRoom());
      event.setOrganizer(updatedEvent.getOrganizer());
      return eventRepository.save(event);
   }).orElseThrow();
}
```

GuestService

Сервіс відповідає за управління інформацією про гостей. Він реалізує базові CRUD-операції: додавання нового гостя, отримання всіх зареєстрованих гостей та видалення за ідентифікатором.

Метод додавання гостя

```
public Guest addGuest(Guest guest) {
   return guestRepository.save(guest);
}
```

Метод отримання всіх гостей

```
public List<Guest> getAllGuests() {
   return guestRepository.findAll();
}
```

InvitationService

Цей сервіс відповідає за управління запрошеннями на події. Він дозволяє створювати, оновлювати та видаляти запрошення, а також отримувати інформацію про участь гостей у подіях та зворотну — події, на які запрошено певного гостя. Додатково реалізовано підрахунок кількості гостей для події.

Метод оновлення статусу запрошення

```
public Invitation updateStatus(Long id, String status) {
   Invitation invitation = invitationRepository.findById(id).orElseThrow();
   invitation.setStatus(status);
   return invitationRepository.save(invitation);
}
```

Метод отримання гостей події

```
public List<Guest> getGuestsByEventId(Long eventId) {
```

```
    Event event = eventRepository.findById(eventId).orElseThrow();
    List<Invitation> invitations = invitationRepository.findByEvent(event);
    return invitations.stream()
            .map(Invitation::getGuest)
            .toList();
}
```

OrganizerService

Сервіс відповідає за управління організаторами подій. Реалізовано базові операції: створення нового організатора, перегляд усіх зареєстрованих та видалення за ідентифікатором.

Метод створення організатора

```
public Organizer addOrganizer(Organizer organizer) {
    return organizerRepository.save(organizer);
}
```

Метод отримання всіх організаторів

```
public List<Organizer> getAllOrganizers() {
    return organizerRepository.findAll();
}
```

RoomService

Цей сервіс відповідає за управління інформацією про приміщення (зали). Він дозволяє створювати, оновлювати, переглядати, видаляти зали, а також отримувати перелік доступних приміщень на вказану дату з урахуванням уже запланованих подій.

Метод створення нової зали

```
public Room addRoom(Room room) {
    return roomRepository.save(room);
}
```

Метод оновлення зали

```
public Room updateRoom(Long id, Room updatedRoom) {
    return roomRepository.findById(id).map(room -> {
        room.setName(updatedRoom.getName());
        room.setCapacity(updatedRoom.getCapacity());
        room.setLocation(updatedRoom.getLocation());
        return roomRepository.save(room);
    }).orElseThrow(() -> new RuntimeException("Room not found"));
}
```

3.4 Контролери

Контролери відповідають за обробку HTTP-запитів REST API та делегують

обробку сервісному шару. Кожен метод має чітко визначений шлях, HTTP-метод, вхідні дані та формат відповіді

AuthController – автентифікація та авторизація

Цей контролер відповідає за реєстрацію користувачів та вхід до системи. Реалізовано механізми кодування пароля, перевірки облікових даних та генерації JWT-токенів.

POST /api/auth/register

– Приймає: об'єкт типу User (JSON з полями username і password)

– Повертає: повідомлення про успішну реєстрацію або помилку (рядок)

– Призначення: створює нового користувача з роллю USER, якщо такий ще не існує

POST /api/auth/login

– Приймає: об'єкт типу User (JSON з полями username і password)

– Повертає: JSON-об’єкт з JWT-токеном або повідомлення про помилку

– Призначення: автентифікує користувача, видає JWT-токен у випадку успіху

EventController – управління подіями

Цей контролер відповідає за створення, оновлення, видалення та пошук подій за різними параметрами, такими як зала, організатор або дата. Всі маршрути працюють через REST API.

POST /api/events

– Приймає: Event (тіло запиту)

– Повертає: створена подія

– Призначення: створює нову подію в системі

GET /api/events

– Приймає: нічого

– Повертає: список усіх подій

– Призначення: повертає всі наявні події з бази

GET /api/events/room/{roomId}

– Приймає: roomId (шляхова змінна)

– Повертає: список подій, що відбуваються у вказаній залі

– Призначення: фільтрація подій за ідентифікатором зали

GET /api/events/organizer/{organizerId}

– Приймає: organizerId (шляхова змінна)

– Повертає: список подій певного організатора

– Призначення: відображення активності обраного організатора

GET /api/events/date?dateTime=yyyy-MM-ddTHH:mm:ss

– Приймає: параметр dateTime у форматі LocalDateTime

– Повертає: список подій, які точно відповідають заданій даті й часу

– Призначення: точний пошук подій за часом

GET /api/events/by-date?date=yyyy-MM-dd

– Приймає: параметр date у форматі yyyy-MM-dd

– Повертає: список подій, які відбуваються протягом зазначеного дня

– Призначення: зручний добовий пошук подій

PUT /api/events/{id}

– Приймає: id події (шляхова змінна) + оновлена подія (Event у тілі)

– Повертає: оновлений об’єкт події

– Призначення: редагування існуючої події

DELETE /api/events/{id}

– Приймає: id події

– Повертає: нічого

– Призначення: видалення події з бази

GuestController – управління гостями

Цей контролер забезпечує додавання, перегляд та видалення гостей у системі. Гості можуть бути запрошені на події через інші механізми системи.

POST /api/guests

– Приймає: Guest (тіло запиту)

– Повертає: створений об'єкт Guest

– Призначення: додає нового гостя до системи

GET /api/guests

– Приймає: нічого

– Повертає: список усіх гостей

– Призначення: виводить усіх зареєстрованих гостей

DELETE /api/guests/{id}

– Приймає: id гостя (шляхова змінна)

– Повертає: нічого

– Призначення: видаляє гостя з бази за його ідентифікатором

InvitationController – керування запрошеннями

Цей контролер дозволяє створювати та оновлювати запрошення, отримувати список гостей події або подій гостя, а також визначати кількість запрошених. Використовується для організації участі у заходах.

POST /api/invitations

– Приймає: Invitation (тіло запиту)

– Повертає: створене запрошення

– Призначення: створює нове запрошення для конкретного гостя на конкретну подію

PUT /api/invitations/{id}/status

– Приймає: id запрошення + параметр status (рядок)

– Повертає: оновлене запрошення

– Призначення: змінює статус участі (наприклад, "підтверджено" або "відхилено")

DELETE /api/invitations/{id}

– Приймає: id запрошення

– Повертає: нічого

– Призначення: видаляє запрошення за його ідентифікатором

GET /api/invitations/event/{eventId}

– Приймає: eventId

– Повертає: список запрошень на конкретну подію

– Призначення: дозволяє переглянути всі запрошення на захід

GET /api/invitations/guest/{guestId}

– Приймає: guestId

– Повертає: список запрошень, які були надіслані певному гостю

– Призначення: відображення участі гостя у подіях

GET /api/invitations/event/{eventId}/guests

– Приймає: eventId

– Повертає: список гостей, запрошених на подію

– Призначення: використовується для формування списку учасників заходу

GET /api/invitations/guest/{guestId}/events

– Приймає: guestId

– Повертає: список подій, на які запрошено цього гостя

– Призначення: дозволяє переглянути історію або план участі гостя

GET /api/invitations/event/{eventId}/guest-count

– Приймає: eventId

– Повертає: ціле число (кількість гостей)

– Призначення: визначає кількість запрошених на захід, може використовуватись для перевірки заповненості зали

OrganizerController – керування організаторами подій

Цей контролер відповідає за додавання, перегляд і видалення організаторів. Організатори відповідають за створення та адміністрування подій у системі.

POST /api/organizers

– Приймає: Organizer (тіло запиту)

– Повертає: створений об’єкт Organizer

– Призначення: додає нового організатора до системи

GET /api/organizers

– Приймає: нічого

– Повертає: список усіх організаторів

– Призначення: перегляд наявних організаторів

DELETE /api/organizers/{id}

– Приймає: id організатора

– Повертає: нічого

– Призначення: видаляє організатора за вказаним ідентифікатором

RoomController – керування приміщеннями

Цей контролер відповідає за операції з приміщеннями (залами), які можна використовувати для організації подій. Передбачено додавання, оновлення, видалення, перегляд усіх залів, а також пошук доступних залів на конкретну дату.

POST /api/rooms

– Приймає: Room (тіло запиту)

– Повертає: створене приміщення

– Призначення: додає нову залу до системи

GET /api/rooms

– Приймає: нічого

– Повертає: список усіх залів

– Призначення: вивід усього переліку приміщень у системі

PUT /api/rooms/{id}

– Приймає: id зали + оновлений об'єкт Room (у тілі запиту)

– Повертає: оновлену залу

– Призначення: редагування наявної зали (назва, місткість, розташування)

DELETE /api/rooms/{id}

– Приймає: id зали

– Повертає: нічого

– Призначення: видаляє залу з бази даних за її ідентифікатором

GET /api/rooms/available?date=yyyy-MM-dd

– Приймає: параметр date у форматі yyyy-MM-dd

– Повертає: список доступних залів на цю дату

– Призначення: визначає вільні приміщення, які ще не зайняті подіями на зазначену дату

UserController – керування користувачами системи

Цей контролер відповідає за адміністративні операції з користувачами: перегляд списку всіх користувачів та видалення облікового запису за ідентифікатором. Може використовуватись для контролю доступу або модерації.

GET /users

– Приймає: нічого

– Повертає: список усіх користувачів (User)

– Призначення: перегляд зареєстрованих у системі користувачів (наприклад, адміністратором)

DELETE /users/{id}

– Приймає: id користувача

– Повертає: нічого

– Призначення: видаляє обліковий запис користувача за заданим ідентифікатором

TestController – перевірка захищеного доступу

Цей контролер містить тестову кінцеву точку, яка використовується для перевірки доступу до захищених ресурсів після автентифікації. Застосовується для діагностики та демонстрації роботи системи безпеки.

GET /api/secure

– Приймає: нічого

– Повертає: текстове повідомлення (String)

– Призначення: повертає повідомлення «Ви успішно отримали доступ!», якщо користувач автентифікований і має дозвіл

3.5 Конфігурація та залежності

Застосунок реалізовано на основі фреймворку Spring Boot з використанням бази даних PostgreSQL. Усі параметри, необхідні для запуску, налаштовуються через конфігураційний файл application.yml, що дозволяє централізовано керувати середовищем виконання.

Підключення до бази даних

У конфігураційному файлі вказано дані для підключення до PostgreSQL: посилання на сервер (url), облікові дані (username та password), а також необхідні параметри драйвера та SQL-платформи. Завдяки цьому Spring автоматично встановлює з’єднання з базою при старті та може ініціалізувати таблиці відповідно до описаних моделей.

Параметри JPA/Hibernate

Конфігурація JPA дозволяє автоматично оновлювати схему бази даних при запуску застосунку (ddl-auto: update), виводити SQL-запити в консоль під час роботи, а також застосовує SQL-діалект, оптимізований під PostgreSQL. Такий підхід суттєво спрощує процес розробки та діагностики.

Безпека та OAuth2

У проєкті реалізовано підтримку авторизації через Google OAuth2. У файлі конфігурації задано необхідні ідентифікатори клієнта (client-id, client-secret), а також URL для редиректу після успішної авторизації. Це дозволяє користувачам здійснювати вхід через обліковий запис Google як альтернативу традиційному логіну з паролем.

JWT-токенізація

Для забезпечення безпеки REST API використовується токенізація з використанням JWT. У конфігурації зберігається секретний ключ, який застосовується для створення та перевірки токенів при автентифікації. Це дозволяє реалізувати безпечну систему авторизації без збереження сесій на сервері.

Системне логування

Включено розширене логування подій безпеки, зокрема спроб входу, доступу до ресурсів, відмов у доступі. Це дозволяє ефективно моніторити та аналізувати поведінку системи під час виконання.

Залежності, що використовуються в проєкті:

- spring-boot-starter-web – для створення REST API;
- spring-boot-starter-data-jpa – для роботи з базою через JPA;
- spring-boot-starter-security – для реалізації механізмів автентифікації та обмеження доступу;
- spring-security-oauth2-client – для інтеграції входу через Google;
- postgresql – драйвер бази даних;
- jjwt – для генерації та обробки JWT-токенів;

4 РЕЄСТРАЦІЯ ТА АВТЕНТИФІКАЦІЯ

У моїй програмі реалізовано 2 види автентифікації користувача: JWT-автентифікація та OAuth2-автентифікація через Google

JWT-автентифікація

Система авторизації у застосунку побудована на основі JWT (JSON Web Token). Після успішної автентифікації користувач отримує токен, який використовується для доступу до захищених ресурсів. Паролі зберігаються у базі в зашифрованому вигляді за допомогою BCrypt. Аутентифікація здійснюється за email (username) і паролем.

Генерація токена відбувається у класі JWTService. У токен включається:

- email користувача (як subject),
- час створення (iat),
- строк дії – 24 години (exp),
- підпис HmacSHA256, сформований на основі секретного ключа з конфігурації (jwt.secret).

Обробка запитів реалізована у фільтрі JwtAuthFilter, який перевіряє наявність токена у заголовку Authorization, витягує з нього ім’я користувача, і, якщо токен дійсний — встановлює автентифікацію в контексті Spring Security.

Фільтр JwtAuthFilter

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain) throws ServletException, IOException {

    final String authHeader = request.getHeader("Authorization");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    final String jwt = authHeader.substring(7);
    final String username = jwtService.extractUsername(jwt);

    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);
```

```
        if (jwtService.isTokenValid(jwt, userDetails)) {
            UsernamePasswordAuthenticationToken authToken =
                    new UsernamePasswordAuthenticationToken(
                            userDetails, null, userDetails.getAuthorities());
            authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }

    filterChain.doFilter(request, response);
}
```

Клас JWTService – генерація і валідація токена

```
public String generateToken(UserDetails userDetails) {
    return Jwts.builder()
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
            .signWith(Keys.hmacShaKeyFor(secret.getBytes()), SignatureAlgorithm.HS256)
            .compact();
}
public boolean isTokenValid(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}
```

OAuth2-автентифікація через Google

У проєкті реалізована можливість авторизації користувачів через обліковий запис Google. Після підтвердження доступу користувач отримує access_token та id_token, які використовуються для автентифікації в системі. Якщо користувач авторизується вперше — його обліковий запис створюється автоматично. Після цього генерується JWT-токен і виконується перенаправлення на клієнтську сторону із вбудованим токеном.

Маршрут:

http://localhost:8080/oauth2/authorization/google

CustomOAuth2UserService

Цей клас реалізує інтерфейс OAuth2UserService та відповідає за завантаження користувача після авторизації через Google. Якщо користувача з таким email не існує в базі — створюється новий із роллю USER. Пароль задається умовно,оскільки він не використовується для входу.

```
@Override
public OAuth2User loadUser(OAuth2UserRequest request) {
    OAuth2User oAuth2User = new DefaultOAuth2UserService().loadUser(request);

    String email = oAuth2User.getAttribute("email");

    Optional<User> existingUser = userRepository.findByUsername(email);
    if (existingUser.isEmpty()) {
        User user = User.builder()
                .username(email)
                .password(passwordEncoder.encode("oauth2user"))
                .role("USER")
                .build();
        userRepository.save(user);
    }

    return oAuth2User;
}
```

5 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Для перевірки працездатності системи використовувалися:

– Ручне тестування через Postman: усі основні REST-запити (GET, POST, PUT, DELETE) були протестовані вручну;

– Автоматичне тестування: були написані unit-тести для сервісів, контролерів, які перевіряли правильність бізнес-логіки, правильність роботи запитів

Інструменти тестування:

– Postman – для надсилання HTTP-запитів і перевірки відповідей сервера;

– GitHub Actions – для автоматичного запуску тестів.

– JUnit 5 з Mockito — для створення unit-тестів сервісів і контролерів із перевіркою логіки обробки запитів, обробки помилок та авторизації.

Postman

Приклад перевірки методу отримання подій за id організатора

URL: http://localhost:8080/api/events/organizer/1

HTTP-метод: GET

Успішне виконання запиту

У цьому випадку ми передаємо id організатора, який дійсно існує у базі. Отримуємо **статус 200 OK**, що свідчить про успішну обробку запиту. Відповідь містить масив подій у форматі JSON (рис. 1):


```
{
  "name": "Spring Conference 2025",
  "dateTime": "2025-07-15T10:00:00",
  "room": {
    "id": 1
  },
  "organizer": {
    "id": 1
  }
}
```

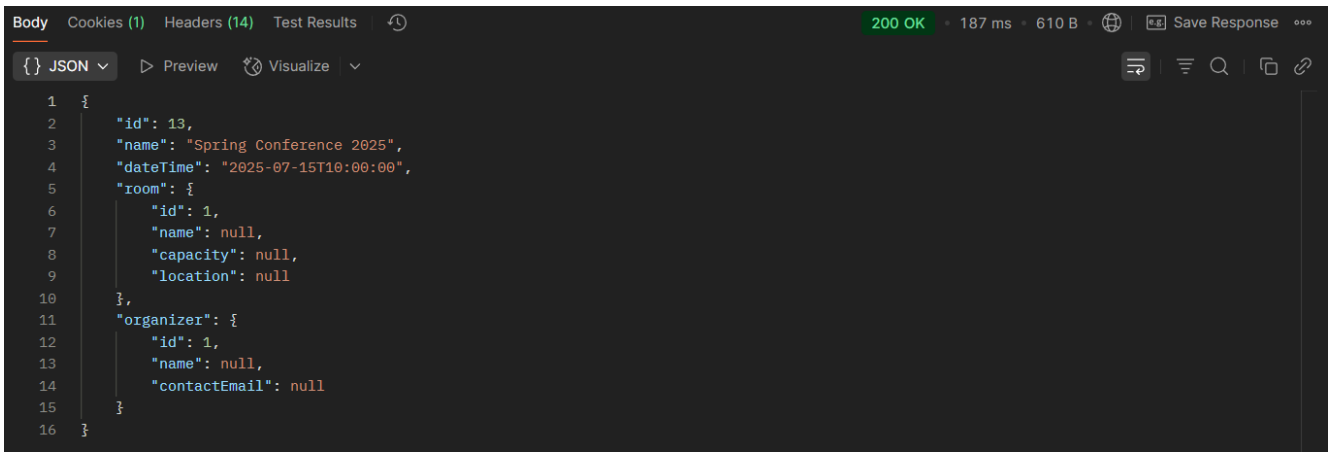

Рисунок 1 – Скріншот успішного виконання запиту

– Помилка 401, тому що не передано JWT токен.

У цьому випадку ми надсилаємо запит до POST /api/events без заголовка Authorization. Система безпеки Spring Security блокує запит, оскільки користувач неавтентифікований. Отримуємо статус 401 UNAUTHORIZED – це означає, що доступ заборонено без JWT токена.

Нижче наведено відповідь у форматі JSON (рис. 2):

Рисунок 2 – Скріншот виконання запиту без JWT токена

– Помилка 400, тому що організатора не існує в базі даних.

У цьому випадку ми надсилаємо POST-запит на створення події з organizer.id = 100, але в таблиці organizer такого запису немає. Це порушує зовнішній ключ у таблиці event. Система повертає 400 BAD REQUEST, оскільки у глобальному обробнику помилок (@ControllerAdvice) виняток DataIntegrityViolationException перехоплюється та перетворюється на зрозуміле повідомлення.

Нижче наведено відповідь у форматі JSON (рис. 3):

Рисунок 3 – Скріншот помилки порушення зовнішнього ключа при створенні події

– Помилка 500, тому що дата передана у неправильному форматі.
Запит містить поле dateTime зі значенням "aba", яке не відповідає очікуваному формату yyyy-MM-dd'T'HH:mm:ss. Jackson (модуль десеріалізації JSON у Spring Boot) не зміг перетворити цей рядок на об'єкт типу LocalDateTime. У результаті ми отримуємо статус 500 INTERNAL SERVER ERROR, тому що виникає внутрішня помилка на етапі перетворення JSON у Java-об'єкт.
Нижче наведено відповідь у форматі JSON (рис. 4):

Рисунок 4 – Скріншот помилки при неправильному форматі дати

У процесі реалізації серверного застосунку було проведено модульне тестування основних компонентів системи. Основна увага приділялася перевірці роботи REST-контролерів та окремих методів сервісного шару. Для тестування використовувались засоби JUnit 5, Mockito та бібліотека MockMvc.

Тестування REST-контролерів

Тестування контролерів виконувалося в ізольованому режимі за допомогою конструкції MockMvcBuilders.standaloneSetup(...). Такий підхід дозволив перевірити HTTP-логіку без запуску повного Spring-контексту, що значно пришвидшило тестування.

Для всіх основних контролерів — RoomController, EventController,

GuestController, InvitationController, OrganizerController, AuthController — були реалізовані окремі класи тестів. У них перевіряється:

- коректність обробки HTTP-запитів (GET, POST, PUT, DELETE);
- обробка вхідних параметрів;
- повернення статус-кодів (200 OK, 400 Bad Request, 401 Unauthorized тощо);
- коректність форматування JSON-відповідей.

Приклад 1 – тест успішного створення кімнати:

```
@Test
void testAddRoom() throws Exception {
    Room room = new Room();
    room.setName("Test Room");
    room.setCapacity(20);
    room.setLocation("Main Hall");

    Room saved = new Room();
    saved.setId(1L);
    saved.setName("Test Room");
    saved.setCapacity(20);
    saved.setLocation("Main Hall");

    when(roomService.addRoom(any(Room.class))).thenReturn(saved);

    mockMvc.perform(post("/api/rooms")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(objectMapper.writeValueAsString(room)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value(1));
}
```

Приклад 2 – тест авторизації користувача з JWT-токеном:

```
@Test
void testLoginSuccess() throws Exception {
    ua.opnu.practice1_template.model.User user = new ua.opnu.practice1_template.model.User();
    user.setUsername("testuser");
    user.setPassword("password");

    User springUser = new User("testuser", "encodedPassword", java.util.Collections.emptyList());

    Authentication authentication = Mockito.mock(Authentication.class);

when(authenticationManager.authenticate(any(UsernamePasswordAuthenticationToken.class))).thenR
eturn(authentication);
    when(authentication.getPrincipal()).thenReturn(springUser);
    when(jwtService.generateToken(any())).thenReturn("fake-jwt-token");

    mockMvc.perform(post("/api/auth/login")
                    .contentType(MediaType.APPLICATION_JSON)
```

```
                .content(objectMapper.writeValueAsString(user)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.token").value("fake-jwt-token"));
}
```

У тестах також перевіряються спеціалізовані запити, зокрема підрахунок кількості гостей на події, отримання подій за датою або залом, оновлення статусу запрошення тощо.

Тестування сервісного шару

Сервісний рівень також частково охоплений юніт-тестами. Для більшості сервісів (EventService, GuestService, InvitationService, OrganizerService) надано базові тести, які перевіряють наявність ін'єкції залежностей та коректну ініціалізацію.

Як приклад повноцінні тести реалізовано для RoomService, де перевіряються такі сценарії:

- створення та оновлення об'єктів;
- видалення кімнати за ідентифікатором;
- отримання всіх кімнат;
- перевірка доступних кімнат на конкретну дату (враховуючи зайнятість зала подіями).

Приклад – тест логіки вибору вільних кімнат:

```
@Test
void testGetAvailableRoomsByDate() {
   LocalDate date = LocalDate.now();

   Room room1 = new Room(1L, "Room1", 100, "A");
   Room room2 = new Room(2L, "Room2", 150, "B");
   Event event = new Event(1L, "Some Event", date.atTime(10, 0), room1, null);

   when(eventRepository.findByDateTimeBetween(any(), any())).thenReturn(List.of(event));
   when(roomRepository.findAll()).thenReturn(List.of(room1, room2));

   List<Room> result = roomService.getAvailableRoomsByDate(date);

   assertEquals(1, result.size());
   assertEquals("Room2", result.get(0).getName());
}
```

Висновки щодо тестування

Результати тестування підтверджують працездатність основних функціональних

модулів застосунку. Зокрема:

- перевірено поведінку всіх REST-запитів;
- протестовано обробку помилок та нестандартних ситуацій;
- тести реалізовані згідно з принципами модульності та ізольованості.

ВИСНОВКИ

У результаті виконання курсової роботи було розроблено серверний застосунок для управління процесами оренди приміщень для проведення подій. Система реалізує основні функціональні модулі, які відповідають поставленому технічному завданню: створення та адміністрування подій, збереження інформації про організаторів і гостей, формування запрошень, а також бронювання залів з урахуванням їхньої доступності.

Застосунок побудовано з використанням сучасного стеку технологій Java та Spring Boot. Логіка реалізована згідно з архітектурним поділом на шари: модель, репозиторії, сервіси та контролери. Для взаємодії з клієнтською частиною створено RESTful API, що забезпечує доступ до функціоналу системи за допомогою HTTP-запитів.

Особливу увагу приділено безпеці доступу до ресурсів: реалізовано автентифікацію за допомогою логіну й паролю з використанням JWT-токенів, а також підтримано авторизацію через протокол OAuth2 (Google Sign-In). Забезпечено контроль доступу на основі ролей користувачів.

Верифікація працездатності системи проводилась як вручну через API (Postman), так і автоматизовано — за допомогою юніт тестів, реалізованих із використанням JUnit, Mockito та MockMvc. Тестами покриті як контролери, так і окремі методи сервісного рівня.

Таким чином, поставлені в роботі задачі були повністю виконані. Отриманий результат є стабільним, розширюваним і придатним для впровадження в реальні прикладні рішення. У майбутньому систему можливо доповнити функціями пошуку, фільтрації за параметрами, підтримкою повідомлень та розширеним інтерфейсом адміністратора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java Documentation. URL: https://docs.oracle.com/en/java/ (дата звернення: 08.06.2025)

2. DEV. URL: https://dev.to/ayshriv/oauth-20-authentication-in-spring-boot-a-guide-to-integrating-google-and-github-login-2hga (дата звернення 08.06.2025)

3. Spring. URL: https://spring.io/ (дата звернення 08.06.2025)

4. Official Spring Security Documentation. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення 08.06.2025)

5. Spring. URL: https://spring.io/guides/tutorials/spring-boot-oauth2 (дата звернення 08.06.2025)

6. OAuth 2.0 Authorization Framework. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення 08.06.2025)

7. JUnit 5 User Guide. URL: https://junit.org/junit5/docs/current/user-guide (дата звернення 08.06.2025)