Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних систем

Пояснювальна записка
до курсової роботи з дисципліни
«Об’єктно-орієнтоване програмування»
на тему: «Система управління проєктами в компанії»

Виконав:
Студент групи АІ-233
Космін Ю. А.
Перевірив:
Годовиченко М.А.

Одеса 2025

Вступ

У сучасних умовах розвитку бізнесу спостерігається стрімке зростання потреби в цифровізації та автоматизації процесів управління проєктами. Це зумовлено необхідністю оперативного контролю великої кількості проєктів, ефективної координації роботи команд, прозорого розподілу ресурсів і своєчасного прийняття управлінських рішень. Проєктна діяльність охоплює широкий спектр завдань — від розробки концепції та планування до контролю виконання, управління бюджетом, ризиками й комунікацією між усіма учасниками, що вимагає використання гнучких і надійних інструментів.

Традиційні підходи до обліку та координації, засновані на паперових документах чи неструктурованих електронних таблицях, вже не відповідають сучасним вимогам до швидкості, прозорості та якості роботи. Впровадження сучасної системи управління проєктами (Project Management System, PMS) дозволяє автоматизувати ключові бізнес-процеси: планування та розподіл завдань, контроль термінів і бюджету, моніторинг виконання, аналітику ефективності, а також забезпечити якісну комунікацію між усіма учасниками проєкту.

Сучасна система управління проєктами повинна підтримувати інтеграцію з іншими інформаційними платформами, гарантувати захист даних, бути масштабованою й адаптивною до потреб різних компаній і типів проєктів. Її впровадження дозволяє створити єдиний інформаційний простір, підвищити прозорість та контроль усіх процесів, зменшити кількість помилок, оптимізувати витрати й підвищити результативність роботи компанії.

Метою цієї роботи є розробка ефективної, надійної та гнучкої системи управління проєктами для компанії, яка дозволить автоматизувати основні етапи підготовки та реалізації проєктів, підвищити якість обслуговування клієнтів і забезпечити прозорість та контроль усіх процесів. У пояснювальній записці розглядаються основні етапи створення такого програмного

забезпечення: аналіз предметної області, проєктування архітектури, реалізація функціональних модулів і оцінка ефективності впровадження системи.

1. Аналіз предметної області

Система управління проєктами розроблена з метою автоматизації ключових процесів у сфері планування, координації та контролю проєктної діяльності в компанії. Основна задача такої системи — забезпечити ефективну взаємодію між проєктами, задачами, працівниками та командами, а також оптимізувати розподіл ресурсів, відстеження виконання завдань і аналітику продуктивності. У системі зберігається та обробляється інформація про проєкти, задачі, працівників, команди та їхні ролі, що дозволяє підвищити якість управління проєктами, зменшити ризики помилок і забезпечити прозорість усіх етапів роботи.

Ключовими об’єктами системи є проєкти, задачі, працівники, команди та призначення працівників на проєкти (Assignment). Проєкт — це ініціатива з унікальним ідентифікатором, назвою, описом та датами початку і завершення. Задача — це конкретна робота в межах проєкту, яка має заголовок, опис, виконавця та статус виконання. Працівник — це учасник компанії з персональними даними та посадою, який може брати участь у кількох проєктах одночасно. Команда об’єднує групу працівників для спільної роботи над проєктами. Призначення (Assignment) визначає роль працівника в конкретному проєкті, що дозволяє гнучко керувати відповідальністю.

Життєвий цикл управління проєктом у системі включає кілька основних етапів. Спочатку адміністратор створює або оновлює інформацію про проєкти та працівників. Далі формуються задачі, які призначаються виконавцям і пов’язуються з проєктами. Паралельно створюються та управляються команди, до яких додаються працівники. Система забезпечує можливість контролю статусів задач, моніторингу прогресу проєктів, а також отримання статистики по продуктивності працівників і ефективності командної роботи.

Функціональні можливості системи охоплюють автоматизацію додавання, оновлення та видалення даних про проєкти, задачі, працівників і команди, призначення працівників на проєкти з ролями, отримання списків задач за

проєктом чи виконавцем, а також генерацію аналітичних звітів. Звіти включають статистику по виконанню задач, кількість задач у проєктах, середній час виконання, активність працівників, що сприяє стратегічному плануванню і підвищенню якості управління.

Для адміністраторів система надає зручний інтерфейс для управління всіма сутностями: проєктами, задачами, працівниками, командами та призначеннями, а також інструменти для аналітики та контролю ефективності роботи. Керівники проєктів можуть оперативно оновлювати інформацію, призначати задачі, отримувати статистику щодо виконання та завантаженості працівників, що дозволяє приймати обґрунтовані рішення щодо розподілу ресурсів і оптимізації процесів. Система також дозволяє відстежувати активність команд, статуси проєктів та інші ключові показники.

Загалом, впровадження системи управління проєктами забезпечує прозорість і контроль усіх етапів планування та виконання робіт, зменшує вплив людського фактора, підвищує продуктивність працівників і сприяє підвищенню ефективності роботи компанії в цілому.

2. Проєктування системи

2.1 Опис сутностей

У системі управління проєктами визначено шість основних сутностей, які відображають ключові об’єкти предметної області:

Project (Проєкт) — представляє ініціативу або задачу, яку необхідно виконати у визначені строки. Кожен проєкт має унікальний ідентифікатор (id), назву (name), опис (description), дату початку (startDate) та дату завершення (endDate).

Employee (Працівник) — представляє учасника компанії, який може брати участь у виконанні проєктів. Працівник має унікальний ідентифікатор (id), ім’я (name), посаду (position) та електронну адресу (email).

Task (Задача) — конкретна робота або активність у межах проєкту, що має унікальний ідентифікатор (id), заголовок (title), опис (description), посилання на проєкт (project), виконавця (assignee) та статус виконання (status).

Team (Команда) — група працівників, об’єднаних для спільної роботи над проєктами. Команда має унікальний ідентифікатор (id), назву (name) та список учасників (members).

Assignment (Призначення) — зв’язує працівника з проєктом, визначаючи його роль у цьому проєкті. Містить унікальний ідентифікатор (id), посилання на працівника (employee), проєкт (project) та роль (role).

2.2 Опис архітектури застосунку

Застосунок системи управління проєктами в компанії побудований за трирівневою архітектурою, що включає три основні рівні:

Контролер (Controller) — цей рівень відповідає за прийом і обробку HTTP-запитів від клієнтів. Контролери отримують параметри запитів, викликають відповідні методи сервісного шару для виконання бізнес-логіки та повертають результати у вигляді JSON-відповідей. Вони виступають посередниками між користувацьким інтерфейсом і внутрішніми компонентами системи, забезпечуючи маршрутизацію запитів і валідацію вхідних даних.

Сервіс (Service) — на цьому рівні реалізована основна бізнес-логіка системи управління проєктами. Сервісний шар відповідає за обробку даних, координацію операцій, перевірку бізнес-правил, управління проєктами, задачами, командами та працівниками. Він взаємодіє з репозиторіями для отримання або збереження інформації в базі даних і повертає результати у вигляді спеціалізованих об’єктів передачі даних (DTO), що дозволяє відокремити внутрішню структуру даних від зовнішніх представлень.

Репозиторій (Repository) — цей рівень відповідає за роботу з базою даних. Використовуючи технології ORM (наприклад, Spring Data JPA), репозиторії

забезпечують абстракцію від написання SQL-запитів, надаючи готові методи для створення, оновлення, видалення та пошуку даних про проєкти, задачі, працівників і команди. Репозиторії гарантують цілісність і консистентність даних у сховищі.

Взаємодія між рівнями організована послідовно: контролер приймає запити від користувача і передає їх сервісу для обробки; сервіс виконує бізнес-логіку і звертається до репозиторіїв для роботи з даними; репозиторії безпосередньо взаємодіють із базою даних, забезпечуючи збереження та отримання інформації. Така архітектура сприяє чіткому розділенню відповідальностей, підвищенню масштабованості, гнучкості та підтримуваності системи управління проєктами.

2.3 Опис REST API

Опис REST-запитів для системи організації виставок:

1. Створити новий проєкт — створює новий проєкт у системі, зберігаючи його назву, опис, дату початку та дату завершення.
2. Отримати список проєктів — повертає перелік усіх проєктів, що зберігаються у базі даних.
3. Оновити проєкт — дозволяє змінити інформацію про конкретний проєкт, наприклад, оновити опис або дати проведення.
4. Видалити проєкт — видаляє проєкт із системи за його унікальним ідентифікатором.
5. Додати нового працівника — створює запис про нового працівника з інформацією про ім’я, посаду та електронну адресу.
6. Отримати всіх працівників — повертає список усіх працівників компанії.
7. Оновити інформацію про працівника — дозволяє змінити дані конкретного працівника, наприклад, оновити посаду або контактні дані.
8. Видалити працівника — видаляє працівника з системи за його ідентифікатором.
9. Створити нову задачу — додає нову задачу до певного проєкту із заголовком, описом, виконавцем та статусом.
10. Отримати задачі проєкту — повертає список усіх задач, які належать до конкретного проєкту.
11. Отримати задачі працівника — повертає всі задачі, призначені певному працівнику.
12. Оновити задачу — оновлює інформацію про задачу, включаючи статус, опис або виконавця.
13. Видалити задачу — видаляє задачу з системи за її ідентифікатором.
14. Призначити працівника на проєкт — створює зв’язок між працівником і проєктом із визначенням ролі.

15. Отримати команду проєкту — повертає список учасників команди, які працюють над певним проєктом.
16. Отримати всі проєкти працівника — повертає перелік проєктів, у яких бере участь конкретний працівник.
17. Додати команду — створює нову команду з назвою та початковим складом учасників.
18. Додати учасника до команди — додає працівника до існуючої команди.
19. Видалити учасника з команди — видаляє працівника зі складу команди.
20. Отримати всі команди — повертає список усіх команд у компанії.
21. Отримати список задач у статусі "виконується" — повертає всі задачі, які наразі перебувають у статусі виконання.
22. Отримати кількість задач у проєкті — повертає загальну кількість задач, пов’язаних із конкретним проєктом.
23. Отримати середній час виконання задач — повертає середній час, витрачений на виконання задач у системі або в межах проєкту.
24. Завершити проєкт — позначає проєкт як завершений, фіксуючи дату закриття та підсумковий статус.
25. Отримати статистику працівника за задачами — повертає аналітичні дані по задачах, виконаних конкретним працівником, включаючи кількість, статуси та час виконання.

3. Реалізація програмного продукту

3.1 Моделі даних

Для відображення сутностей предметної області у програмному коді системи управління проєктами створюються відповідні моделі (класи), які відповідають таблицям у базі даних. Кожна модель містить поля, що відображають атрибути сутності, а також зв’язки між сутностями (наприклад, зв’язок "один-до-багатьох" між проєктом та задачами). У проєкті реалізовано такі основні моделі: Project, Employee, Task, Assignment та Team.

Клас Project представляє проєкт, який реалізується в компанії. Він містить основні атрибути проєкту, такі як назва, опис, дати початку і завершення, статус, а також зв’язки із задачами та призначеннями працівників.

java-код для класу Project:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Project {

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

    private String name;
    private String description;
    private LocalDate startDate;
    private LocalDate endDate;

    private String status; // Наприклад: "ACTIVE", "COMPLETED"

    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Task> tasks = new ArrayList<>();
```

```
    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Assignment> assignments = new ArrayList<>();
}
```

Клас Employee відображає працівника компанії, який може виконувати задачі у різних проєктах. Модель містить основні атрибути працівника, а також зв'язки із задачами та призначеннями (Assignment).

java-код для класу Employee:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Employee {

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

    private String name;
    private String position;
    private String email;

    @OneToMany(mappedBy = "assignee", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Task> tasks = new ArrayList<>();

    @OneToMany(mappedBy = "employee", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Assignment> assignments = new ArrayList<>();
}
```

Клас Task описує задачу, що виконується у межах конкретного проєкту. Задача має заголовок, опис, статус, дати створення та завершення, а також посилання на проєкт і виконавця.

java-код для класу Task:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Task {

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

    private String title;
    private String description;
    private String status;

    private LocalDateTime createdAt;
    private LocalDateTime completedAt;

    @ManyToOne
    @JoinColumn(name = "project_id")
    private Project project;

    @ManyToOne
    @JoinColumn(name = "assignee_id")
    private Employee assignee;
}
```

Клас Assignment зв’язує працівника з проєктом і визначає його роль у цьому проєкті. Це дозволяє одному працівнику брати участь у кількох проєктах із різними ролями.

java-код для класу Assignment:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Assignment {

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

    private String role;

    @ManyToOne
    @JoinColumn(name = "employee_id")
    private Employee employee;

    @ManyToOne
    @JoinColumn(name = "project_id")
    private Project project;
}
```

Клас Team представляє команду працівників, які можуть спільно працювати над одним або кількома проєктами. Команда має унікальний ідентифікатор, назву та список учасників.

java-код для класу Team:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Team {

    @Id
```

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

    private String name;

    @ManyToMany
    @JoinTable(
            name = "team_members",
            joinColumns = @JoinColumn(name = "team_id"),
            inverseJoinColumns = @JoinColumn(name = "employee_id")
    )
    private List<Employee> members = new ArrayList<>();
}
```

Також для кожної з моделей було створено DTO класи. Нижче будуть наведені приклади DTO класів:

```
public class AssignmentDTO {
    private Long id;
    private String role;
    private Long employeeId;
    private Long projectId;
}
public class EmployeeDTO {
    private Long id;
    private String name;
    private String position;
    private String email;
}
public class ProjectDTO {
    private Long id;
    private String name;
    private String description;
    private LocalDate startDate;
    private LocalDate endDate;
```

```
    private String status;
}
public class TaskDTO {
    private Long id;
    private String title;
    private String description;
    private String status;
    private LocalDateTime createdAt;
    private LocalDateTime completedAt;
    private Long projectId;
    private Long assigneeId;
}
public class TeamRequestDTO {
    private String name;
    private List<Long> memberIds;
}
public class TeamResponseDTO {
    private Long id;
    private String name;
    private List<EmployeeDTO> members;
}
```

3.2 Репозиторії

Для забезпечення доступу до даних у системі управління проєктами використовується шар репозиторіїв. Репозиторії реалізовані за допомогою Spring Data JPA, що дозволяє абстрагуватися від написання SQL-запитів і надає стандартні CRUD-операції (створення, читання, оновлення, видалення) для кожної сутності. Окрім базових методів, у репозиторіях можуть бути визначені додаткові запити для специфічних операцій пошуку або підрахунку.

У системі реалізовано такі основні репозиторії:

AssignmentRepository

Репозиторій для роботи з призначеннями (Assignment). Окрім стандартних CRUD-операцій, містить методи для отримання всіх призначень певного працівника або проєкту.

```
public interface AssignmentRepository extends
JpaRepository<Assignment, Long> {

    // Отримати всі призначення працівника
    List<Assignment> findByEmployeeId(Long employeeId);

    // Отримати всі призначення проєкту
    List<Assignment> findByProjectId(Long projectId);
}
```

EmployeeRepository

Репозиторій для роботи з працівниками (Employee). Надає стандартний набір CRUD-операцій для управління інформацією про працівників.

```
public interface EmployeeRepository extends
JpaRepository<Employee, Long> {
}
```

ProjectRepository

Репозиторій для роботи з проєктами (Project). Дозволяє виконувати основні операції над проєктами, такі як створення, оновлення, видалення та пошук.

```
public interface ProjectRepository extends JpaRepository<Project,
Long> {
}
```

TaskRepository

Репозиторій для роботи із задачами (Task). Окрім стандартних CRUD-операцій, містить додаткові методи для пошуку задач за проєктом, виконавцем, статусом, а також для підрахунку кількості задач.

```
public interface TaskRepository extends JpaRepository<Task, Long>
{

    // Знайти задачі за проєктом
    List<Task> findByProject(Project project);

    // Знайти задачі за виконавцем
```

```
    List<Task> findByAssignee(Employee employee);

    // Знайти задачі за статусом
    List<Task> findByStatus(String status);

    // Підрахувати кількість задач у проєкті
    Long countByProject(Project project);

    // Підрахувати кількість задач у працівника
    Long countByAssignee(Employee employee);

    // Підрахувати кількість задач певного статусу у працівника
    Long countByAssigneeAndStatus(Employee employee, String
status);
}
```

TeamRepository

Репозиторій для роботи з командами (Team). Дозволяє виконувати основні операції над командами, такі як створення, оновлення, видалення та пошук.

```
public interface TeamRepository extends JpaRepository<Team, Long>
{
}
```

3.3 Сервіси

Сервісний шар у системі управління проєктами реалізує основну бізнес-логіку застосунку, забезпечує обробку даних, виконання перевірок, координацію операцій та взаємодію з шаром репозиторіїв. Сервіси працюють із DTO (Data Transfer Object) для передачі даних між різними рівнями системи, що дозволяє відокремити внутрішню структуру моделей від зовнішніх представлень.

У системі реалізовано такі основні сервіси:

AssignmentService

Сервіс для управління призначеннями працівників на проєкти. Дозволяє створювати нові призначення, отримувати призначення за проєктом чи працівником, видаляти призначення та виконувати перетворення між сутністю Assignment і DTO.

Основні методи:

`createAssignment(AssignmentDTO dto)` — створює новий запис призначення, зберігаючи роль, працівника та проєкт.

`getAssignmentsByProject(Long projectId)` — повертає всі призначення для вказаного проєкту.

`getAssignmentsByEmployee(Long employeeId)` — повертає всі призначення для вказаного працівника.

`getAssignmentById(Long id)` — отримує призначення за ідентифікатором.

`deleteAssignment(Long id)` — видаляє призначення із системи.

EmployeeService

Сервіс для управління працівниками. Забезпечує створення, оновлення, видалення та отримання інформації про працівників.

Основні методи:

`createEmployee(EmployeeDTO dto)` — створює нового працівника.

`getAllEmployees()` — повертає список усіх працівників.

`getEmployeeById(Long id)` — отримує працівника за ідентифікатором.

`updateEmployee(Long id, EmployeeDTO dto)` — оновлює дані працівника.

`deleteEmployee(Long id)` — видаляє працівника із системи.

ProjectService

Сервіс для управління проєктами. Дозволяє створювати, оновлювати, завершувати, видаляти проєкти та отримувати інформацію про них.

Основні методи:

`createProject(ProjectDTO dto)` — створює новий проєкт.

`getAllProjects()` — повертає список усіх проєктів.

`getProjectById(Long id)` — отримує проєкт за ідентифікатором.

`updateProject(Long id, ProjectDTO dto)` — оновлює дані проєкту.

`completeProject(Long id)` — позначає проєкт як завершений.

`deleteProject(Long id)` — видаляє проєкт із системи.

TaskService

Сервіс для управління задачами. Дозволяє створювати, оновлювати, видаляти задачі, отримувати задачі за проєктом, працівником чи статусом, а також формувати статистику.

Основні методи:

`createTask(TaskDTO dto)` — створює нову задачу.

`getTasksByProject(Long projectId)` — повертає задачі певного проєкту.

`getTasksByEmployee(Long employeeId)` — повертає задачі певного працівника.

`updateTask(Long id, TaskDTO dto)` — оновлює дані задачі.

`deleteTask(Long id)` — видаляє задачу із системи.

`getTasksInProgress()` — повертає задачі зі статусом "виконується".

`countTasksByProject(Long projectId)` — підраховує кількість задач у проєкті.

`getAverageCompletionTime()` — обчислює середній час виконання задач.

`getEmployeeTaskStatistics(Long employeeId)` — повертає статистику задач для працівника (загальна кількість, у роботі, завершені).

TeamService

Сервіс для управління командами. Дозволяє створювати команди, додавати та видаляти учасників, отримувати інформацію про команди.

Основні методи:

`createTeam(TeamRequestDTO dto)` — створює нову команду з початковим складом учасників.

`addMemberToTeam(Long teamId, Long employeeId)` — додає працівника до команди.

`removeMemberFromTeam(Long teamId, Long employeeId)` — видаляє працівника з команди.

`getAllTeams()` — повертає список усіх команд у системі.

3.4 Контролери

Контролери у системі управління проєктами відповідають за прийом та обробку HTTP-запитів від клієнтів, взаємодіючи із сервісним шаром для виконання бізнес-логіки та повернення результатів у форматі JSON. Вони виступають посередниками між користувацьким інтерфейсом і внутрішніми компонентами системи, забезпечуючи чітку маршрутизацію запитів та валідацію вхідних даних.

У системі реалізовано такі основні контролери:

AssignmentController

Контролер для управління призначеннями працівників на проєкти. Дозволяє створювати нові призначення, отримувати призначення за проєктом

чи працівником, отримувати окреме призначення за ідентифікатором та видаляти призначення.

Основні ендпоінти:

— POST /api/assignments — створити нове призначення працівника на проєкт.

— GET /api/assignments/project/{projectId} — отримати всі призначення для певного проєкту.

— GET /api/assignments/employee/{employeeId} — отримати всі призначення для певного працівника.

— GET /api/assignments/{id} — отримати одне призначення за ідентифікатором.

— DELETE /api/assignments/{id} — видалити призначення.

EmployeeController

Контролер для управління працівниками. Дозволяє створювати, оновлювати, видаляти працівників, отримувати інформацію про всіх або одного працівника, а також отримувати всі проєкти, у яких бере участь працівник.

Основні ендпоінти:

— POST /api/employees — додати нового працівника.

— GET /api/employees — отримати список усіх працівників.

— GET /api/employees/{id} — отримати одного працівника за ID.

— PUT /api/employees/{id} — оновити інформацію про працівника.

— DELETE /api/employees/{id} — видалити працівника.

— GET /api/employees/{id}/projects — отримати всі проєкти, у яких бере участь працівник.

ProjectController

Контролер для управління проєктами. Дозволяє створювати, оновлювати, завершувати, видаляти проєкти та отримувати інформацію про всі проєкти.

Основні ендпоінти:

— POST /api/projects — створити новий проєкт.

— GET /api/projects — отримати список усіх проєктів.

— PUT /api/projects/{id} — оновити дані проєкту.

— PUT /api/projects/{id}/complete — завершити проєкт.

— DELETE /api/projects/{id} — видалити проєкт.

TaskController

Контролер для управління задачами. Дозволяє створювати, оновлювати, видаляти задачі, отримувати задачі за проєктом, працівником чи статусом, а також отримувати статистику.

Основні ендпоінти:

— POST /api/tasks — створити нову задачу.

— GET /api/tasks/project/{projectId} — отримати задачі певного проєкту.

— GET /api/tasks/employee/{employeeId} — отримати задачі певного працівника.

— PUT /api/tasks/{id} — оновити задачу.

— DELETE /api/tasks/{id} — видалити задачу.

— GET /api/tasks/status/in-progress — отримати задачі зі статусом "виконується".

— GET /api/tasks/project/{projectId}/count — отримати кількість задач у проєкті.

— GET /api/tasks/average-completion-time — отримати середній час виконання задач.

— GET /api/tasks/employee/{employeeId}/statistics — отримати статистику задач працівника.

TeamController

Контролер для управління командами. Дозволяє створювати команди, додавати та видаляти учасників, отримувати інформацію про всі команди.

Основні ендпоінти:

— POST /api/teams — створити нову команду.

— POST /api/teams/{teamId}/members/{employeeId} — додати учасника до команди.

— DELETE /api/teams/{teamId}/members/{employeeId} — видалити учасника з команди.

— GET /api/teams — отримати список усіх команд.

3.5 Обробка помилок і валідація

У системі управління проєктами особлива увага приділяється коректній обробці помилок і валідації вхідних даних для забезпечення надійності, безпеки та зручності користувачів. Валідація та обробка помилок реалізовані на кількох рівнях застосунку:

Валідація вхідних даних:

Валідація на рівні DTO: для передачі даних між клієнтом і сервером використовуються об’єкти DTO (Data Transfer Object). Вони містять анотації валідації (наприклад, @NotNull, @Size, @Email), що дозволяє автоматично перевіряти коректність полів під час отримання HTTP-запитів. Це запобігає збереженню некоректних або неповних даних у базі.

Валідація в контролерах: контролери приймають вхідні дані у вигляді DTO, які проходять валідацію за допомогою механізму Spring Validation. У разі виявлення помилок валідації контролер повертає клієнту відповідь із кодом помилки (наприклад, HTTP 400 Bad Request) та детальним описом проблем.

Перевірка у сервісному шарі: додаткові бізнес-правила та логічні перевірки виконуються у сервісах. Наприклад, перевірка унікальності назви проєкту, коректності дат (дата початку не повинна бути пізнішою за дату завершення), наявності пов’язаних сутностей перед оновленням чи видаленням, перевірка ролі працівника при призначенні на проєкт тощо.

Обробка помилок:

Глобальний обробник помилок: для централізованої обробки виключень у застосунку реалізований глобальний обробник помилок (наприклад, за допомогою анотації @ControllerAdvice), який перехоплює винятки, що виникають у контролерах і сервісах. Це дозволяє формувати уніфіковані відповіді з інформацією про помилки, логувати їх та уникати витоку внутрішньої інформації.

Обробка специфічних виключень:

У сервісах системи передбачена обробка типових помилок, таких як:

EntityNotFoundException — коли шуканий запис у базі не знайдено (наприклад, при спробі отримати неіснуючий проєкт, задачу чи працівника);

DataIntegrityViolationException — порушення цілісності даних (наприклад, спроба видалити працівника, який призначений на задачі або проєкти);

RuntimeException з повідомленнями про бізнес-логіку (наприклад, дублікати назв проєктів, некоректні дати, спроба призначити неіснуючого працівника).

Повернення коректних HTTP-статусів:

Контролери повертають відповідні HTTP-коди залежно від результату обробки запиту:

200 OK — успішна операція;

201 Created — успішне створення ресурсу;

204 No Content — успішне видалення;

400 Bad Request — помилки валідації або некоректні дані;

404 Not Found — ресурс не знайдено;

409 Conflict — конфлікти даних (наприклад, дублікати);

500 Internal Server Error — несподівані помилки серверу.

Безпека та захист від некоректних дій:

Аутентифікація та авторизація: усі вхідні запити можуть проходити перевірку прав доступу (наприклад, через Spring Security), що запобігає несанкціонованим діям (редагування чи видалення даних без відповідних ролей).

Обробка помилок аутентифікації: при невірному введенні логіна або пароля користувач отримує інформативне повідомлення без розкриття деталей безпеки.

4. Реєстрація та автентифікація

У системі управління проєктами реалізовано сучасні механізми реєстрації, автентифікації та авторизації користувачів для забезпечення безпеки, контролю доступу й персоналізації функціоналу. Система підтримує як класичну форму реєстрації/логіну, так і автентифікацію через JWT та OAuth2.

Реєстрація користувача:

Реєстрація нового користувача можлива через веб-інтерфейс або REST API:

—Веб-реєстрація: користувач заповнює форму, вказуючи ім’я користувача, пароль і роль (наприклад, USER або ADMIN). Пароль шифрується за допомогою BCrypt, а роль зберігається у вигляді ROLE_USER або ROLE_ADMIN. Якщо ім’я користувача вже існує, система повертає повідомлення про помилку;

— REST-реєстрація: через ендпоінт /api/register користувач надсилає POST-запит із логіном і паролем. Якщо логін унікальний, створюється новий користувач із роллю USER.

Автентифікація користувача:

Система підтримує кілька способів автентифікації:

— Форма логіну: користувач вводить логін і пароль на сторінці /login. Після успішної автентифікації користувач перенаправляється на сторінку відповідно до своєї ролі (/admin для адміністраторів, /user для звичайних користувачів);

— JWT-автентифікація:через ендпоінт /api/login користувач надсилає логін і пароль. У разі успіху система повертає JWT-токен, який використовується для подальших захищених запитів (у заголовку Authorization: Bearer ...);

— OAuth2: підтримується вхід через сторонні сервіси (наприклад, Google). Якщо користувач вперше входить через OAuth2, для нього автоматично створюється обліковий запис із роллю USER.

Авторизація та розмежування прав доступу

Права доступу реалізовані на основі ролей:

— Адміністратор (ROLE_ADMIN) має доступ до адміністративних сторінок і функцій;

— Користувач (ROLE_USER) має доступ до власного профілю, задач, проєктів, у яких бере участь.

Доступ до ендпоінтів контролюється через Spring Security (@PreAuthorize, @Secured), а також у конфігурації SecurityConfig:

— /admin — лише для адміністраторів;

— /user — лише для користувачів;

— /register, /login, /api/register, /api/login, /oauth2/** — відкриті для всіх.

Захист паролів і токенів

— Паролі зберігаються лише у хешованому вигляді (BCrypt);

— JWT-токени мають обмежений термін дії (наприклад, 1 година) і перевіряються фільтром TokenFilter для кожного запиту;

— При виході з системи токен стає недійсним (логічно — шляхом видалення з клієнта).

Глобальна безпека

— Усі конфіденційні запити виконуються через HTTPS;

— Захист від brute-force забезпечується стандартними механізмами Spring Security;

— При невірному логіні чи паролі користувач отримує інформативне, але безпечне повідомлення.

Обробка сесії та вихід

— Після логіну користувач отримує токен або сесію;

— Вихід із системи (logout) очищає сесію чи токен, перенаправляючи користувача на сторінку логіну.

Структура компонентів безпеки

— AppUser — модель користувача з полями username, password, role.

— UserRepository — зберігає користувачів і дозволяє пошук за username.

— AppService — логіка реєстрації користувачів з перевіркою унікальності логіна.

— AppProfileService — реалізує UserDetailsService для Spring Security.

— TokenGenerator — створює та перевіряє JWT-токени.

— TokenFilter — інтегрується у фільтри Spring Security для перевірки токенів у кожному запиті.

— OAuth2Service — підтримка входу через сторонні сервіси.

— SecurityConfig — конфігурує правила доступу, логіку авторизації, фільтри, шифрування паролів.

5. Тестування програмного продукту

Для перевірки правильності роботи функціоналу обробки зображень у системі було проведено ручне тестування через Postman та автоматичне тестування за допомогою юніт-тестів. Тестування охоплювало: завантаження файлів, перегляд, видалення, перевірку формату файлу та його розміру, перевірку авторизації доступу.

Реєстрація нового користувача

Метод: POST

URL: http://localhost:8080/api/register

Очікування: Користувач з вказаним логіном, паролем та роллю створюється в системі. Пароль зберігається у зашифрованому вигляді, а роль перетворюється на ROLE_USER та додається до списку авторитетів. Також створений обліковий запис позначено як активний, не заблокований, не протермінований.

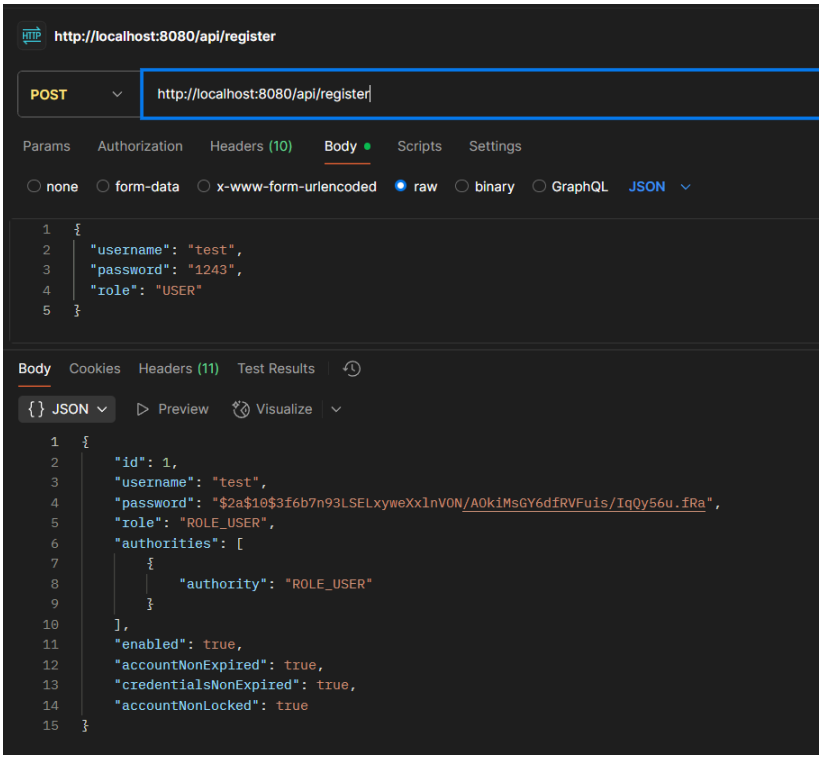

Рисунок 1 – Успішна реєстрація нового користувача через Postman

Авторизація користувача (JWT login)
Метод: POST

URL: http://localhost:8080/api/login

Очікування: Користувач вводить ім’я користувача та пароль для авторизації. У разі успішної перевірки облікових даних сервер повертає JWT-токен (JSON Web Token), який використовується для подальшої автентифікації запитів.

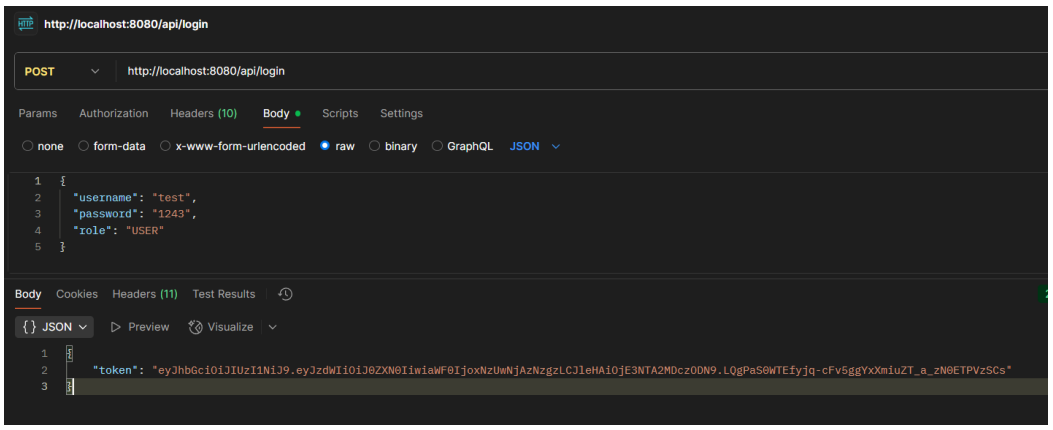

Рисунок 2 – Успішна авторизація та отримання JWT токена

Створення нового проєкту

Метод: POST

URL: http://localhost:8080/api/projects

Очікування: Створення нового проєкту. У тілі запиту передаються поля: name, description, startDate, endDate. Після успішного створення очікується, що сервер поверне JSON-об’єкт із присвоєним id та переданими даними.

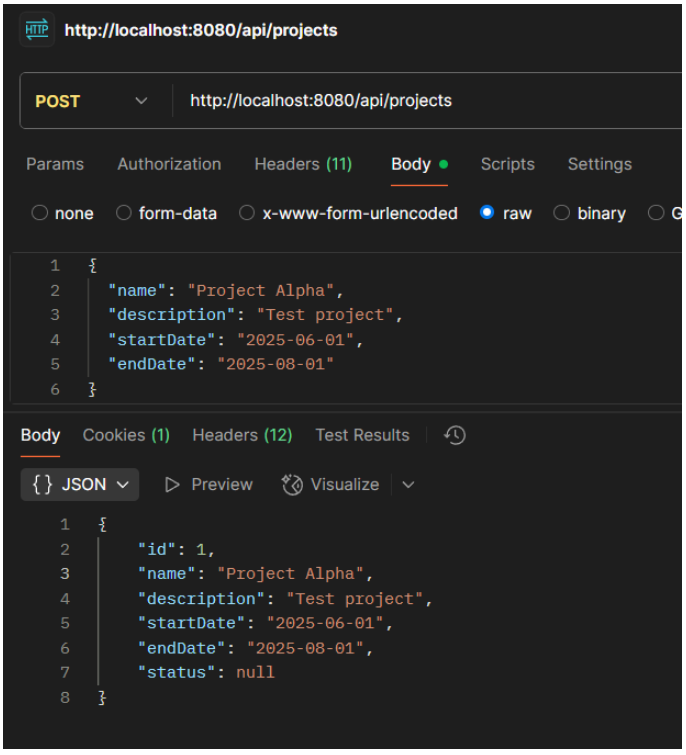

Рисунок 3 – Успішне створення нового проєкту

Отримання списку проєктів
Метод: GET

URL: http://localhost:8080/api/projects

Очікування: У відповідь повертається список проєктів із бази даних. Кожен об'єкт містить id, назву, опис, дати початку та завершення, а також статус (який може бути null, якщо не задано).

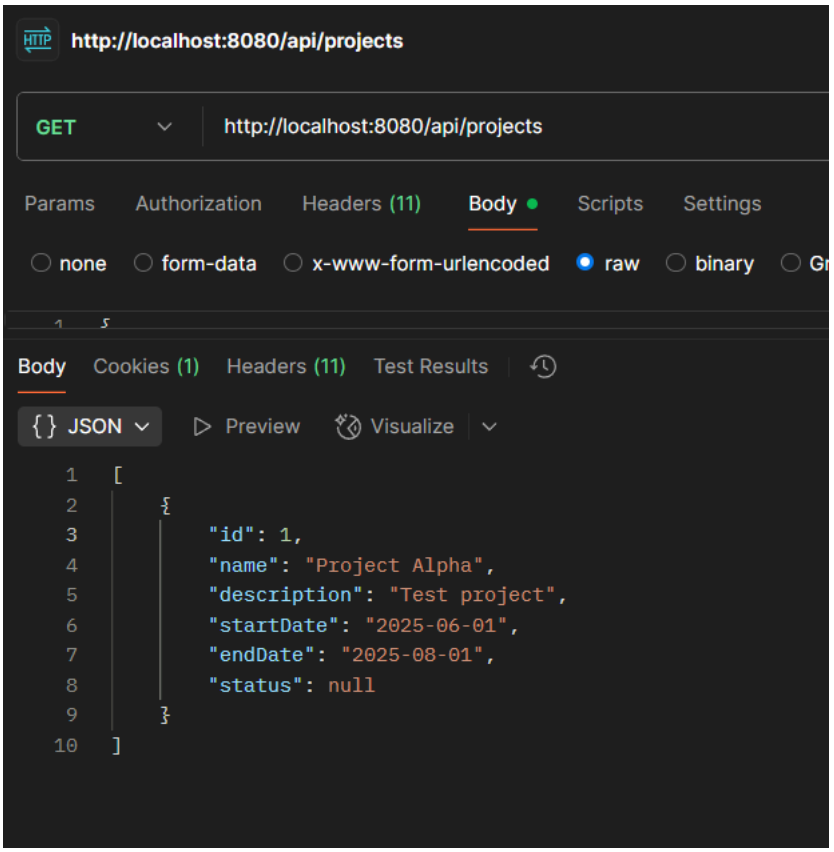

Рисунок 4 – Отримання списку проєктів з бази даних

Оновлення проєкту

Метод: PUT

URL: http://localhost:8080/api/projects/1

Очікування: Оновлення існуючого проєкту з ідентифікатором 1. У тілі запиту передаються нові значення для назви, опису та дат. У відповіді повертається оновлений об'єкт з актуальними даними.

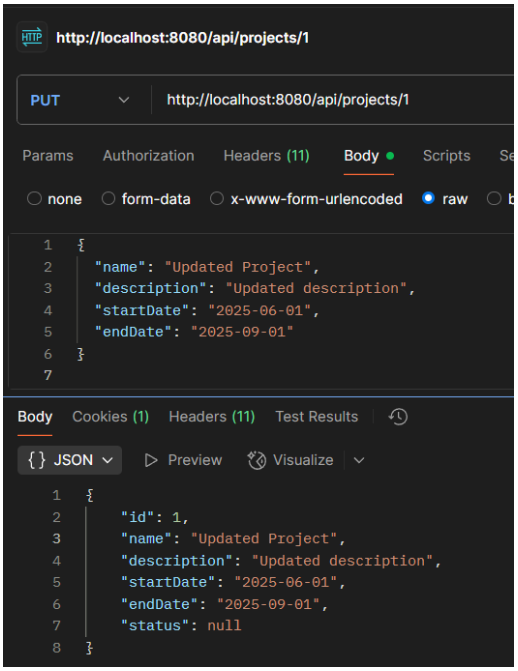

Рисунок 5 – Успішне оновлення проєкту з ID 1

Додавання нового працівника

Метод: POST

URL: http://localhost:8080/api/employees

Очікування: Створюється новий співробітник з вказаним ім’ям, посадою та електронною адресою. У відповіді повертається створений об’єкт із присвоєним унікальним ідентифікатором.

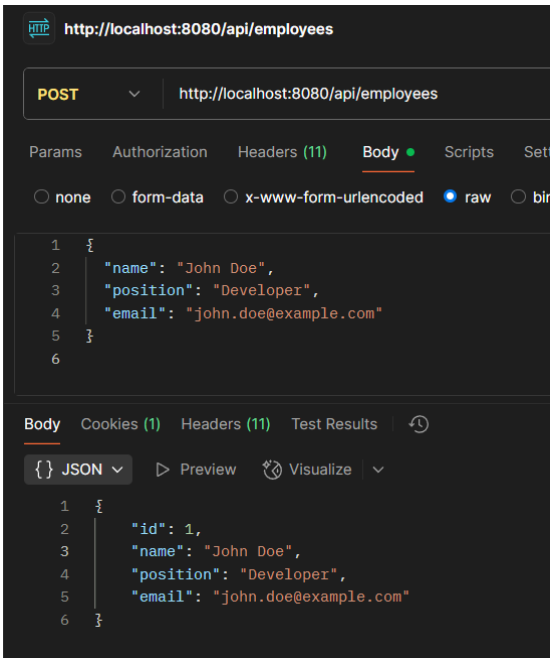

Рисунок 6 – Успішне створення нового співробітника

Видалення працівника
Метод: POST

URL: http://localhost:8080/api/employees

Очікування: Створюється новий запис співробітника з вказаними даними: ім’я, посада та електронна пошта. У відповіді повертається створений об’єкт із унікальним ідентифікатором id.

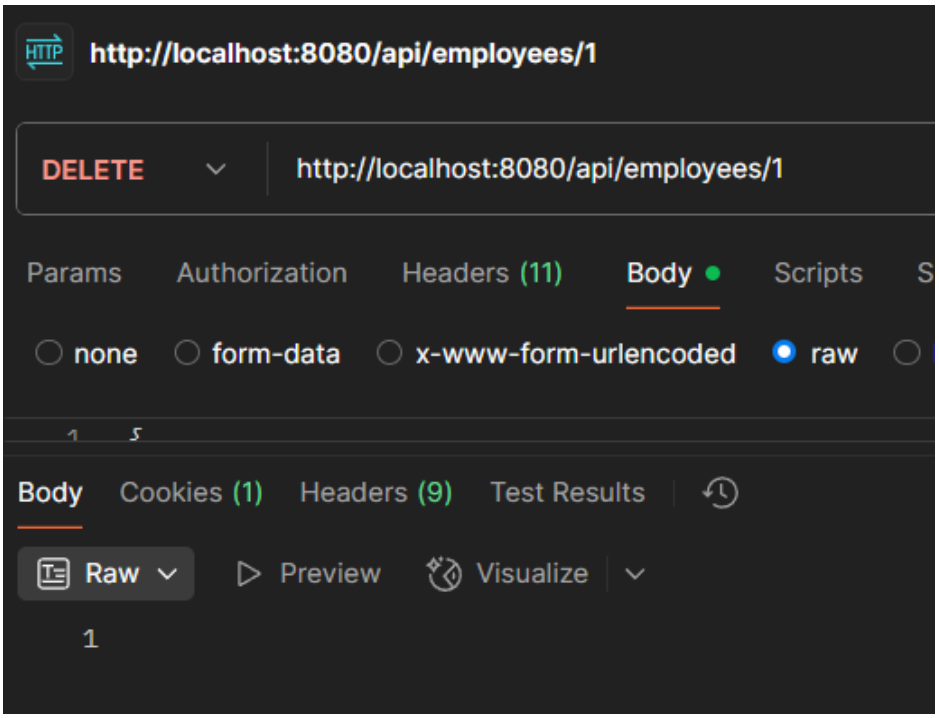

Рисунок 7 – Успішне видалення співробітника

Створення нової задачі

Метод: POST

URL: http://localhost:8080/api/tasks

Очікування: Створюється нове завдання з вказаними параметрами (назва, опис, ID проєкту, ID виконавця, статус). У відповідь сервер повертає створений об'єкт з автоматично згенерованим id, а також з полями createdAt і completedAt, які наразі дорівнюють null.

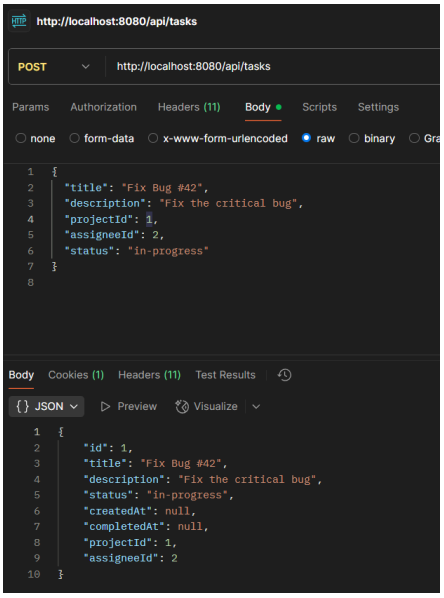

Рисунок 8 – Успішне створення завдання

Оновлення задачі

Метод: PUT

URL: http://localhost:8080/api/tasks/1

Очікування: Оновлюється завдання з ідентифікатором 1. Нові дані (змінені назва, опис, статус) зберігаються у базі даних. У відповіді сервер повертає оновлений об'єкт із збереженим id та іншими полями, включаючи createdAt і completedAt, які залишаються null.

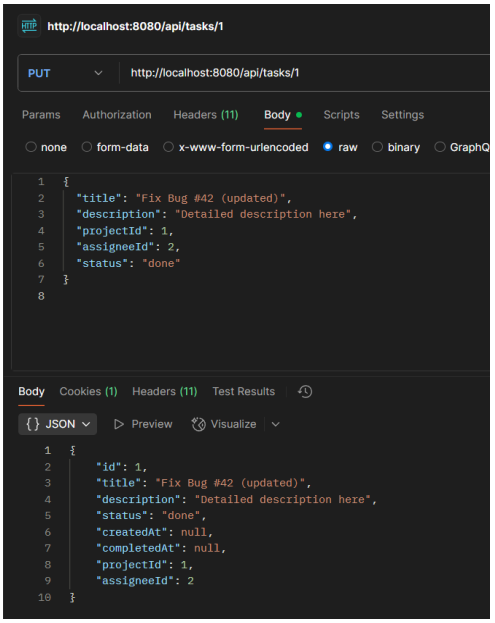

Рисунок 9 – Успішне оновлення завдання

Створення команди

Метод: POST

URL: http://localhost:8080/api/teams

Очікування: Створюється нова команда з вказаною назвою. У відповідь сервер повертає створений об'єкт із згенерованим id, назвою команди та порожнім списком учасників.

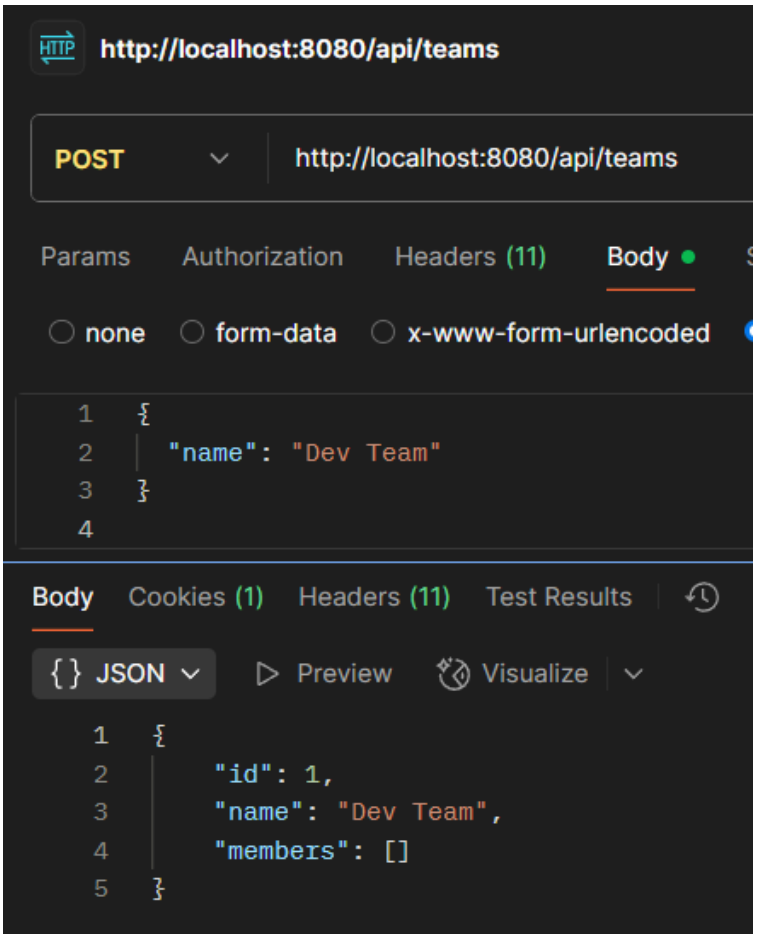

Рисунок 10 – Успішне створення команди

Додавання учасника до команди
Метод: POST

URL: http://localhost:8080/api/teams/1/members/3

Очікування: До команди з ідентифікатором 1 додається учасник з ідентифікатором 3. У відповіді повертається оновлений об’єкт команди, що містить нового члена з відповідними полями: ім’я, посада, email.

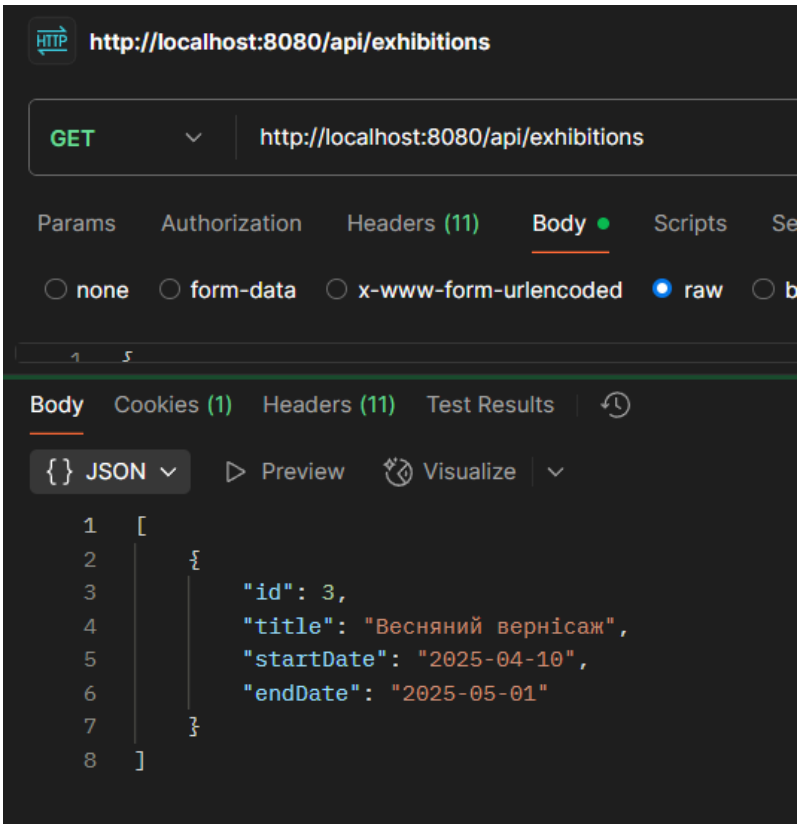

Рисунок 11 – Додавання учасника до команди

Висновок

У результаті виконання курсової роботи було розроблено інформаційну систему для управління проєктами в компанії, яка автоматизує основні бізнес-процеси у сфері планування, виконання та контролю проєктної діяльності. Реалізований застосунок дозволяє ефективно керувати даними про проєкти, задачі, працівників, команди та призначення, забезпечуючи прозорість і контроль усіх етапів життєвого циклу проєкту.

Система підтримує повний набір CRUD-операцій для основних сутностей, реалізує пошук, фільтрацію та аналітику за різними параметрами, а також надає можливість формування команд і призначення ролей учасникам. Впроваджено механізми статистики та звітності, що дозволяє аналізувати продуктивність працівників, стан виконання задач і ефективність проєктів.

Архітектура системи побудована за трирівневим принципом із чітким розділенням на контролери, сервіси та репозиторії, що реалізовано з використанням Spring Boot, Spring Data JPA та Spring Security. Для безпечного обміну даними між клієнтом і сервером застосовано DTO, а для захисту API — сучасні механізми автентифікації та авторизації, включаючи JWT та OAuth2. Особлива увага приділена валідації вхідних даних і централізованій обробці помилок, що підвищує надійність і зручність використання системи.

Функціональність програмного продукту була протестована за допомогою сучасних інструментів для тестування REST API, що підтвердило стабільність, коректність та відповідність заявленим вимогам.

Отже, поставлені завдання були успішно виконані: створено гнучку, масштабовану та безпечну систему управління проєктами, яка підвищує ефективність колективної роботи, покращує взаємодію між учасниками, забезпечує прозорість і контроль усіх процесів у компанії. Отримані під час розробки знання та практичні навички стануть основою для подальшого професійного розвитку у сфері розробки сучасних інформаційних систем.

Список використаних джерел

1. Spring Security Reference Documentation. OAuth 2.0 Resource Server JWT URL: https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html (дата звернення: 12.06.2025)
2. Habr. JWT-аутентификация при помощи Spring Boot 3 и Spring Security 6 URL: https://habr.com/ru/articles/784508/ (дата звернення: 13.06.2025)
3. DevOps Blog. Spring Boot 3 + Spring Security 6 With JWT Authentication and Authorization [New 2024] URL: https://blog.devops.dev/spring-boot-3-spring-security-6-with-jwt-authentication-and-authorization-new-2024-989323f29b84 (дата звернення: 11.06.2025)
4. Dzen. Java 1305. Как работает аутентификация и авторизация в Spring Security с использованием JWT токена? URL: https://dzen.ru/a/ZRp9AI9FaRPAV9cE (дата звернення: 14.06.2025)
5. YouTube – SPRING SECURITY 6 with JWT Authentication: Secure Your App in MINUTES! URL: https://www.youtube.com/watch?v=B1SUyu98HvQ (дата звернення: 13.06.2025)
6. YouTube – JWT-аутентификация для нативных приложений - Spring Security URL: https://www.youtube.com/watch?v=iURuqbGeRNc (дата звернення: 14.06.2025)