

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Е-сервіс управління завданнями у сфері бізнес-менеджменту на платформі Java та Spring Boot»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Юлія ДОРОШЕНКО

(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-42

Юлія ДОРОШЕНКО

Керівник: Вікторія ЖЕБКА

д-р техн. наук, проф.

Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ

Завідувач кафедри
Технологій цифрового розвитку
_____ В.В. Жебка

« ____ » _____ 2026 року

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Дорошенко Юлії Володимирівні

1. Тема кваліфікаційної роботи: «Е-сервіс управління завданнями у сфері бізнес-менеджменту на платформі Java та Spring Boot»
керівник кваліфікаційної роботи д-р техн. наук, професор Вікторія ЖЕБКА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року №51.
2. Строк подання кваліфікаційної роботи «__» травня 2026 року _____
3. Вхідні дані до кваліфікаційної роботи: технічна документація з описом середовища розробки та основних технологій створення вебзастосунку, матеріали щодо розробки клієнтської частини засобами HTML, CSS, JavaScript і Bootstrap, документація з реалізації серверної частини на Java з використанням Spring Boot, Spring Data JPA та Spring Security, технічні відомості про роботу з базою даних PostgreSQL, механізми автентифікації, розмежування прав доступу, обробку файлів, email-сповіщення, тестування

та розгортання програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
 1. Огляд та аналіз предметної області управління завданнями, порівняння існуючих систем і формування вимог до електронного сервісу.
 2. Проектування електронного сервісу управління завданнями, його архітектури, бази даних, UML-моделей та основних алгоритмів роботи.
 3. Програмна реалізація та опис функціонування електронного сервісу управління завданнями на основі Java, Spring Boot, PostgreSQL, Thymeleaf, Bootstrap та засобів безпеки.
 4. Тестування електронного сервісу управління завданнями, перевірка основних функцій та аналіз відповідності розробленого застосунку визначеним вимогам.
5. Перелік графічного матеріалу: *презентація*
 1. Аналіз аналогів.
 2. Вимоги до програмного забезпечення.
 3. Програмні засоби реалізації.
 4. Діаграма варіантів використання.
 5. Алгоритм роботи застосунку.
 6. Діаграма класів.
 7. Екранні форми.
 8. Апробація результатів дослідження
6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури та аналіз предметної області	26.03 – 03.04.2026	
2	Формування функціональних та нефункціональних вимог до сервісу	03.04 – 08.04.2026	
3	Проектування архітектури системи та структури бази даних	08.04 – 12.04.2026	
4	Реалізація серверної частини на платформі Java та Spring Boot	12.04 – 20.04.2026	

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
5	Реалізація REST API та клієнтської частини	20.04 – 28.04.2026	
6	Налаштування інфраструктури та email-сповіщень	28.04 – 01.05.2026	
7	Розробка та виконання тестів (модульні, інтеграційні, функціональні)	01.05 – 08.05.2026	
8	Написання пояснювальної записки та підготовка матеріалів до захисту	08.05 – 11.05.2026	
9	Попередній захист роботи	11.05.2026	

Здобувач(ка) вищої освіти

(підпис)

Юлія ДОРОШЕНКО

Керівник

кваліфікаційної роботи

(підпис)

Вікторія ЖЕБКА

РЕФЕРАТ

Кваліфікаційна робота: 84 с., 40 рис., 15 табл., 25 джерел, 5 додатків.

Мета роботи – підвищення ефективності організації та контролю робочих процесів у сфері бізнес-менеджменту за допомогою розробленого електронного сервісу управління завданнями, реалізованого на основі Java з використанням фреймворку Spring Boot.

Об'єкт дослідження – процеси управління завданнями та координації командної роботи у сфері бізнес-менеджменту.

Предмет дослідження – методи, моделі та технології розроблення електронних сервісів управління завданнями на платформі Java та Spring Boot.

Короткий зміст роботи: У роботі досліджено предметну область електронних сервісів управління завданнями у сфері бізнес-менеджменту та проаналізовано існуючі аналоги. Визначено їх переваги, недоліки та обмеження. Сформульовано вимоги до розроблюваного сервісу. Обґрунтовано вибір технологічного стеку Java 24, Spring Boot 3.5.0, PostgreSQL, Hibernate, Thymeleaf, Bootstrap та Vanilla JavaScript. Спроектовано трирівневу клієнт-серверну архітектуру, структуру бази даних, UML-діаграми та алгоритми роботи основних бізнес-процесів. Реалізовано серверну частину, REST API, клієнтський інтерфейс із канбан-дошкою Drag and Drop та підсистему email-сповіщень. Проведено комплексне тестування системи та підтверджено її відповідність поставленим вимогам.

Ключові слова: ЕЛЕКТРОННИЙ СЕРВІС, УПРАВЛІННЯ ЗАВДАННЯМИ, JAVA, SPRING BOOT, POSTGRESQL, HIBERNATE, REST API, КАНБАН-ДОШКА, DRAG AND DROP, ВЕБ-ЗАСТОСУНОК.

ЗМІСТ

ВСТУП	11
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНИХ РІШЕНЬ	14
1.1 Актуальність створення електронних сервісів управління завданнями у сфері бізнес-менеджменту	14
1.2 Аналіз існуючих сервісів управління завданнями	17
1.3 Порівняльна характеристика аналогів.....	19
1.4 Формування вимог до електронного сервісу управління завданнями	25
2 ПРОЄКТУВАННЯ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ЕЛЕКТРОННОГО СЕРВІСУ	29
2.1 Загальна архітектура системи	29
2.2 Обґрунтування вибору технологій розробки.....	34
2.3 Проєктування структури бази даних	39
2.4 Моделювання системи за допомогою uml-діаграм	45
2.5 Алгоритм роботи електронного сервісу	50
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	55
3.1 Загальна структура програмного застосунку.....	55
3.2 Реалізація серверної частини на платформі java та spring boot.....	59
3.3 Реалізація rest api сервісу	65
3.4 Опис функціональних можливостей системи.....	71
3.5 Опис користувацького інтерфейсу	77
4 РОЗРОБКА ТА ТЕСТУВАННЯ Е-сервісу	84
4.1 Розробка Е-сервісу	84

4.2 Тестування системи.....	94
ВИСНОВКИ.....	98
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	101
ДОДАТОК А.....	104
ДОДАТОК Б	112
ДОДАТОК В	113
ДОДАТОК Г	114
ДОДАТОК Д.....	119

УМОВНІ ПОЗНАЧЕННЯ

Позначення	Розшифрування
API	Прикладний програмний інтерфейс (Application Programming Interface)
AJAX	Asynchronous JavaScript and XML – технологія асинхронних запитів
BCrypt	Криптографічна функція хешування паролів
CRUD	Create, Read, Update, Delete – базові операції з даними
CSRF	Cross-Site Request Forgery – міжсайтова підробка запитів
CSS	Cascading Style Sheets – каскадні таблиці стилів
DTO	Data Transfer Object – об'єкт передачі даних
ER	Entity-Relationship – модель «сутність-зв'язок»
HTML	HyperText Markup Language – мова розмітки гіпертексту
HTTP	HyperText Transfer Protocol – протокол передачі гіпертексту
HTTPS	HyperText Transfer Protocol Secure – захищений HTTP
IDE	Integrated Development Environment – інтегроване середовище розробки
JDK	Java Development Kit – комплект розробника Java
JPA	Java Persistence API – стандарт об'єктно-реляційного відображення
JPQL	Java Persistence Query Language – мова запитів JPA
JS	JavaScript – мова програмування для веб-сторінок
JSON	JavaScript Object Notation – формат обміну даними
JVM	Java Virtual Machine – віртуальна машина Java
MVC	Model-View-Controller – архітектурний шаблон
ORM	Object-Relational Mapping – об'єктно-реляційне відображення
OWASP	Open Web Application Security Project – проєкт з безпеки веб-застосунків
RBAC	Role-Based Access Control – контроль доступу на основі ролей

Позначення	Розшифрування
REST	Representational State Transfer – архітектурний стиль веб-сервісів
SaaS	Software as a Service – програмне забезпечення як послуга
SMTP	Simple Mail Transfer Protocol – протокол передачі електронної пошти
SQL	Structured Query Language – мова структурованих запитів
SSL	Secure Sockets Layer – протокол захищеного з'єднання
TLS	Transport Layer Security – протокол захисту транспортного рівня
UML	Unified Modeling Language – уніфікована мова моделювання
URI	Uniform Resource Identifier – уніфікований ідентифікатор ресурсу
URL	Uniform Resource Locator – уніфікований локатор ресурсу
UUID	Universally Unique Identifier – універсальний унікальний ідентифікатор
UX	User Experience – користувацький досвід
XSS	Cross-Site Scripting – міжсайтовий скриптинг
БД	База даних
ОС	Операційна система
ПЗ	Програмне забезпечення
СКВ	Система контролю версій
СУБД	Система управління базами даних

ВСТУП

Актуальність теми. Сучасний розвиток інформаційного суспільства супроводжується зростанням обсягів даних, поширенням дистанційної роботи та підвищенням вимог до ефективної організації бізнес-процесів. У цих умовах управління завданнями є важливою складовою бізнес-менеджменту, оскільки забезпечує координацію роботи працівників, контроль виконання доручень і раціональне використання робочого часу.

Ефективне управління завданнями сприяє підвищенню продуктивності командної роботи, зменшенню втрат інформації та покращенню прозорості робочих процесів. На ринку існують комерційні рішення Trello, Asana, Jira, Monday.com та Microsoft To Do, однак більшість із них реалізована як SaaS-сервіси, що створює залежність від постачальника, обмежує контроль над даними, а також потребує регулярних витрат. Безкоштовні версії таких систем мають функціональні обмеження, а підтримка української локалізації часто є частковою. Це обґрунтовує актуальність розроблення власного електронного сервісу управління завданнями з можливістю розгортання на інфраструктурі організації.

Мета і завдання дослідження. Метою кваліфікаційної роботи є підвищення ефективності організації та контролю робочих процесів у сфері бізнес-менеджменту за допомогою розробленого електронного сервісу управління завданнями, реалізованого на основі Java з використанням фреймворку Spring Boot.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

- проаналізувати предметну область електронних сервісів управління завданнями;
- дослідити існуючі аналоги, визначити їх переваги та недоліки;
- сформулювати функціональні та нефункціональні вимоги до системи;
- спроектувати архітектуру електронного сервісу та структуру бази даних;
- виконати UML-моделювання основних процесів системи;
- розробити алгоритми роботи основних бізнес-процесів;
- реалізувати серверну частину на Java та Spring Boot;

- реалізувати REST API для взаємодії клієнтської та серверної частин;
- створити користувацький інтерфейс із використанням HTML5, CSS3, JavaScript і Bootstrap;
- провести тестування системи та проаналізувати його результати.

Об'єкт дослідження – процеси управління завданнями та координації командної роботи у сфері бізнес-менеджменту.

Предмет дослідження – методи, моделі та технології розроблення електронних сервісів управління завданнями на платформі Java та Spring Boot.

Методи дослідження. У роботі використано системний аналіз для вивчення предметної області, порівняльний аналіз — для оцінювання існуючих сервісів, методи інженерії вимог — для формування вимог до системи. Об'єктно-орієнтоване, реляційне та UML-моделювання застосовано для проєктування архітектури, бази даних і основних процесів. Методи модульного, інтеграційного та функціонального тестування використано для перевірки відповідності реалізованої системи поставленим вимогам.

Практичне значення одержаних результатів полягає у створенні готового до впровадження електронного сервісу управління завданнями для малих і середніх підприємств. Сервіс дає змогу організовувати командну роботу, розподіляти завдання, контролювати виконання та аналізувати робочі процеси без залежності від зовнішніх SaaS-платформ.

Галузь застосування. Розроблений електронний сервіс може застосовуватися у сфері бізнес-менеджменту проєктними командами, керівниками підрозділів і виконавцями для управління завданнями, комунікації між учасниками та контролю робочих процесів.

Апробація результатів роботи. Основні результати кваліфікаційної роботи апробовано у матеріалах наукових конференцій:

1. Дорошенко Ю.В. «Архітектурні патерни побудови е-сервісу управління завданнями на основі Spring Boot 3.5 та Java 25» // Матеріали всеукраїнської

науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026.

2. Дорошенко Ю.В. «Е-сервіс управління завданнями у сфері бізнес-менеджменту на платформі Java та Spring Boot» // Матеріали VI Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 2026

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНИХ РІШЕНЬ

1.1 Актуальність створення електронних сервісів управління завданнями у сфері бізнес-менеджменту

Сучасний етап розвитку інформаційних технологій характеризується стрімкою цифровою трансформацією усіх сфер господарської діяльності, що зумовлює зростання попиту на спеціалізовані програмні рішення для оптимізації бізнес-процесів. Управління завданнями (Task Management) розглядається як одна з ключових складових ефективного бізнес-менеджменту, що забезпечує координацію роботи персоналу, контроль за виконанням доручень та раціональне використання робочого часу. У роботі досліджується актуальність розроблення електронного сервісу управління завданнями, який поєднує сучасні веб-технології з функціональними можливостями, необхідними для підтримки управлінської діяльності у сфері бізнесу [11].

Серед основних чинників, що визначають актуальність розроблення електронних сервісів управління завданнями у сфері бізнес-менеджменту, доцільно виокремити такі:

- необхідність централізованого зберігання інформації про завдання, що уможливорює швидкий пошук, систематизацію та аналіз даних щодо виконуваних робіт;
- потреба у підвищенні прозорості робочих процесів, що досягається завдяки візуалізації статусів завдань, термінів їх виконання та відповідальних осіб;
- забезпечення ефективної комунікації між співробітниками та керівниками, зокрема через автоматизовані сповіщення про зміну статусу завдань та наближення термінів виконання;

- зменшення часових витрат на рутинні управлінські операції, такі як формування звітності, моніторинг прогресу та розподіл робочого навантаження;
- можливість аналітичної обробки даних щодо виконання завдань для прийняття обґрунтованих управлінських рішень [3].

Окремої уваги заслуговує економічний аспект впровадження електронних сервісів управління завданнями. Згідно з дослідженням компанії Forrester Research, інвестиції у засоби автоматизації управління завданнями забезпечують повернення капіталовкладень (ROI) на рівні 250–350% упродовж трьох років експлуатації, що досягається завдяки скороченню часу на адміністративні процеси, зменшенню кількості помилок та підвищенню продуктивності праці [11]. Для підприємств малого та середнього бізнесу впровадження таких систем дає можливість конкурувати з більш потужними організаціями за рахунок підвищення внутрішньої організованості та ефективності.

Технологічний прогрес у галузі веб-розробки створив передумови для побудови повнофункціональних електронних сервісів, які за своїми можливостями не поступаються настільним додаткам. Платформа Java разом із фреймворком Spring Boot становить одне з найпопулярніших середовищ для розроблення корпоративних веб-застосунків завдяки високій продуктивності, масштабованості та зрілій екосистемі бібліотек. За даними звіту JetBrains «State of Developer Ecosystem 2023», понад 50% корпоративних веб-застосунків розробляються з використанням Java та Spring-екосистеми, що свідчить про надійність обраного технологічного стеку [25]. Використання реляційної СУБД PostgreSQL забезпечує цілісність даних та підтримку складних транзакційних операцій, необхідних для управлінських систем [21].

Аналіз ринку програмного забезпечення для управління завданнями виявляє стійку тенденцію до зростання. Відповідно до прогнозів аналітичної компанії Mordor Intelligence, обсяг світового ринку програмних рішень для управління завданнями у 2023 році становив близько 2,8 млрд доларів США, а до 2028 року

прогнозується його збільшення до 5,2 млрд доларів США із середньорічним темпом зростання (CAGR) на рівні 13,1% [9]. Зазначена динаміка підтверджує високий попит на подібні рішення та перспективність розроблення нових сервісів у цій сфері.

Особливе значення для бізнес-менеджменту має функціональна складова електронних сервісів управління завданнями. Сучасні системи мають забезпечувати реалізацію широкого спектра можливостей, серед яких створення та редагування завдань, призначення відповідальних виконавців, встановлення термінів та пріоритетів, відстеження прогресу виконання, прикріплення файлів, ведення коментарів, формування звітності та автоматичне інформування користувачів через електронну пошту. Інтеграція таких функцій в єдиному веб-інтерфейсі дає можливість керівникам ефективно координувати діяльність підлеглих, а виконавцям – структуровано планувати власний робочий час [6].

Важливим аспектом є також забезпечення інформаційної безпеки електронних сервісів управління завданнями, оскільки в них зберігається конфіденційна бізнес-інформація. Використання сучасних механізмів автентифікації та авторизації, реалізованих за допомогою фреймворку Spring Security, гарантує захист даних від несанкціонованого доступу та забезпечує розмежування прав користувачів відповідно до їх ролей у системі. Дотримання вимог щодо захисту персональних даних, зокрема положень Загального регламенту про захист даних (GDPR) Європейського Союзу та Закону України «Про захист персональних даних», є обов'язковою умовою для впровадження подібних систем у комерційну експлуатацію [2].

Узагальнюючи викладене, слід констатувати, що актуальність створення електронного сервісу управління завданнями у сфері бізнес-менеджменту підтверджується сукупністю економічних, організаційних та технологічних чинників. Зростання попиту на цифрові інструменти координації робочих процесів, поширення дистанційних форм праці, необхідність підвищення продуктивності та забезпечення прозорості управлінської діяльності формують стійкі передумови для

розроблення спеціалізованих веб-застосунків. Використання сучасного технологічного стеку на основі платформи Java та фреймворку Spring Boot дає можливість створити надійне, масштабоване та функціональне рішення, що задовольняє потреби сучасних підприємств у сфері бізнес-менеджменту.

1.2 Аналіз існуючих сервісів управління завданнями

На сучасному ринку представлено значну кількість електронних сервісів управління завданнями, що розрізняються за функціональними можливостями, архітектурними підходами, цільовою аудиторією та ціновою політикою. Аналіз існуючих рішень є необхідним етапом проектування власного сервісу, оскільки дає можливість виявити сильні та слабкі сторони наявних систем та сформулювати обґрунтовані вимоги до розроблюваного продукту [12]. У межах цього підрозділу розглядаються найпоширеніші у світовій практиці сервіси: Trello, Asana, Jira, Monday.com та Microsoft To Do.

Сервіс Trello, придбаний компанією Atlassian, належить до категорії канбан-систем та орієнтований на візуальне представлення робочих процесів за допомогою дошок, списків та карток. Кожна картка може містити опис, чек-листи, прикріплені файли, мітки кольорового кодування, терміни виконання та коментарі. Принциповою перевагою сервісу є простота освоєння та інтуїтивний інтерфейс на основі технології Drag and Drop [8]. Водночас Trello має істотні обмеження у безкоштовній версії – щодо кількості дошок, інтеграцій (Power-Ups) та обсягу прикріплень. Для складних проєктів з багаторівневою структурою функціональності часто виявляється недостатньо.

Сервіс Asana позиціонується як комплексна платформа для управління проєктами та командної роботи. Він пропонує декілька варіантів представлення завдань: списком, канбан-дошкою, календарем, діаграмою Ганта. Серед ключових функцій – підзавдання необмеженої вкладеності, користувацькі поля, аналітичні дашборди, інтеграція з понад 200 зовнішніми сервісами та автоматизація через

тригери «Rules» [8]. Основними недоліками є висока вартість бізнес-тарифів, складність освоєння повного функціоналу та обмеження безкоштовної версії, що не дозволяє ефективно працювати командам понад 15 осіб.

Платформа Jira Software є однією з найпотужніших систем управління завданнями, орієнтованою переважно на потреби IT-індустрії. Jira реалізує підтримку методологій Agile (Scrum, Kanban) та надає інструменти для відстеження дефектів, управління релізами, планування спринтів та формування деталізованої аналітики. Особливістю системи є гнучка система налаштування робочих процесів (workflows). Платформа інтегрується з системами контролю версій (Git, Bitbucket), середовищами CI (Bamboo, Jenkins) та продуктами Atlassian. Незважаючи на потужний функціонал, Jira характеризується складністю налаштування та крутою кривою навчання, що робить її менш привабливою для невеликих бізнес-команд поза IT-сферою.

Сервіс Monday.com є візуально орієнтованою платформою, що використовує концепцію кольорових таблиць з настроюваними колонками. Платформа підтримує різні типи представлення даних (таблиці, канбан-дошки, діаграми Ганта, календарі, карти) та надає широкі можливості автоматизації через візуальний конструктор робочих процесів без навичок програмування [16]. До недоліків належать висока вартість для невеликих команд (мінімум три користувачі) та повільна робота інтерфейсу при великих обсягах даних.

Microsoft To Do – безкоштовний сервіс із екосистеми Microsoft 365, орієнтований переважно на індивідуальне використання або малі команди. Сервіс пропонує мінімалістичний інтерфейс зі списками, нагадуваннями, повторюваними завданнями та тегами; тісно інтегрований з Outlook, Teams та Planner [8]. Однак функціональні можливості сервісу обмежені: відсутні складні ієрархії завдань, повноцінні канбан-дошки, аналітичні звіти та налаштування ролей. Тому Microsoft To Do не може розглядатися як повноцінне рішення для бізнес-менеджменту середніх та великих організацій.

Окрім зазначених платформ, на ринку представлено інші сервіси: ClickUp, що позиціонується як «one app to replace them all» та об'єднує функції управління завданнями, документообігу та відстеження часу [18]; Notion як гібридна платформа бази знань і управління завданнями; Todoist для персональної продуктивності; Wrike, Basecamp, Microsoft Planner.

Аналіз функціональних характеристик розглянутих сервісів виявляє низку загальних тенденцій. По-перше, переважна більшість систем реалізує концепцію канбан-дошок як основний інструмент візуалізації, що відповідає принципам Lean-менеджменту та Agile-методологій [8]. По-друге, спостерігається тенденція до розширення інтеграційних можливостей через відкриті API. По-третє, важливою складовою стає функціонал автоматизації та сповіщень.

Водночас виявлено спільні недоліки. Більшість сервісів реалізована як SaaS-рішення із залежністю від постачальника, що створює ризики втрати доступу до даних. Безкоштовні версії суттєво обмежені у функціональності, що змушує підприємства витрачати значні кошти на платні підписки. Готові рішення часто не враховують специфічних вимог окремих галузей.

Результати аналізу обґрунтовують доцільність розроблення власного електронного сервісу управління завданнями, що поєднував би найкращі практики розглянутих платформ з гнучким налаштуванням під специфічні потреби бізнес-менеджменту. Використання відкритого технологічного стеку на основі Java та Spring Boot дає можливість створити автономне рішення для розгортання на власних серверах організації, що забезпечує повний контроль над даними та незалежність від зовнішніх постачальників.

1.3 Порівняльна характеристика аналогів

З метою формування об'єктивної картини стану ринку електронних сервісів управління завданнями та обґрунтування доцільності розроблення власного програмного продукту у роботі здійснюється порівняльний аналіз сервісів,

розглянутих у попередньому підрозділі. Порівняння проводиться за сукупністю критеріїв, що охоплюють функціональні можливості, технологічні особливості, цінову політику та зручність використання. Такий комплексний підхід дає можливість виявити переваги та недоліки кожного з аналогів, а також визначити функціональні прогалини, які можуть бути усунені у розроблюваному електронному сервісі [8].

Для проведення порівняльної характеристики обрано п'ять найбільш репрезентативних сервісів, що мають значну частку ринку та широке коло користувачів: Trello, Asana, Jira Software, Monday.com та Microsoft To Do. Критерії порівняння сформовано з урахуванням потреб цільової аудиторії електронного сервісу управління завданнями у сфері бізнес-менеджменту та згруповано за такими категоріями: функціональні можливості, технічні характеристики, користувацький досвід та економічні параметри.

До функціональних критеріїв порівняння віднесено такі параметри: підтримка канбан-дошок як основного механізму візуалізації завдань, можливість створення підзавдань та ієрархічної структури, наявність системи ролей та прав доступу, підтримка прикріплення файлів до завдань, наявність системи коментарів, автоматизовані сповіщення електронною поштою, формування аналітичних звітів та статистики, інтеграція із зовнішніми сервісами через API. Технічні критерії включають тип розгортання сервісу (SaaS або self-hosted), наявність мобільних застосунків, підтримку русифікації та локалізації, відкритість програмного коду. Користувацький досвід оцінюється за такими параметрами як простота освоєння, інтуїтивність інтерфейсу, швидкодія системи. Економічні критерії охоплюють вартість безкоштовної версії та обмеження безкоштовних тарифів.

У таблиці 1.1 наведено порівняльну характеристику основних функціональних можливостей розглянутих сервісів управління завданнями.

Таблиця 1.1

Порівняння функціональних можливостей сервісів управління завданнями

Функціональна можливість	Trello	Asana	Jira	Monday.com	MS To Do
Канбан-дошки	+	+	+	+	–
Підзавдання	обмеж.	+	+	+	+
Система ролей	базова	+	розширена	+	базова
Прикріплення файлів	+ (10 МБ)	+ (100 МБ)	+ (10 МБ)	+ (500 МБ)	+ (25 МБ)
Коментарі до завдань	+	+	+	+	–
Email-сповіщення	+	+	+	+	+
Аналітичні звіти	–	+	+	+	–
Інтеграція через API	+	+	+	+	обмеж.
Drag and Drop	+	+	+	+	+
Календарне представлення	платно	+	+	+	+
Автоматизація процесів	платно	+	+	+	–

Аналіз даних таблиці 1.1 свідчить про те, що найбільш повний функціонал з-поміж розглянутих сервісів пропонують платформи Asana, Jira та Monday.com. Сервіс Trello, незважаючи на свою популярність, має суттєві обмеження безкоштовної версії, зокрема щодо функцій автоматизації та аналітичних звітів. Microsoft To Do характеризується обмеженою функціональністю, що не задовольняє потреби бізнес-менеджменту середніх та великих організацій.

Технічні характеристики розглянутих сервісів представлено у таблиці 1.2. Цей блок параметрів є особливо важливим для організацій, що прагнуть зберігати конфіденційні дані на власних серверах та контролювати інфраструктуру системи.

Таблиця 1.2

Технічні характеристики сервісів управління завданнями

Технічна характеристика	Trello	Asana	Jira	Monday.com	MS To Do
Тип розгортання	SaaS	SaaS	SaaS / Self-hosted	SaaS	SaaS
Веб-інтерфейс	+	+	+	+	+
Мобільні застосунки	iOS, Android	iOS, Android	iOS, Android	iOS, Android	iOS, Android
Підтримка укр. мови	—	—	Частково	—	+
Відкритий код	—	—	—	—	—
Експорт даних	CSV, JSON	CSV	CSV, XML	CSV, Excel	—
Двофакторна автентифікація	+	+	+	+	+
Резервне копіювання	Автоматичне	Автоматичне	Автоматичне	Автоматичне	Автоматичне

Як видно з таблиці 1.2, переважна більшість сервісів реалізована як SaaS-рішення, що передбачає зберігання даних на серверах постачальника послуг. Єдиним сервісом з можливістю розгортання на власній інфраструктурі є Jira Software (Data Center версія), однак вартість такого розгортання є вкрай високою та орієнтованою переважно на великі підприємства. Жоден із розглянутих сервісів не є рішенням з відкритим кодом, що обмежує можливості адаптації під специфічні потреби організацій. Підтримка української мови в інтерфейсі також є суттєвою

проблемою – вона повноцінно представлена лише у Microsoft To Do, що ускладнює використання інших сервісів вітчизняними користувачами.

Параметри користувацького досвіду та економічні характеристики розглянутих сервісів узагальнено у таблиці 1.3, що дає можливість оцінити доступність кожного з аналогів для різних категорій користувачів.

Таблиця 1.3

Користувацький досвід та економічні параметри

Параметр	Trello	Asana	Jira	Monday.com	MS To Do
Простота освоєння	висока	середня	низька	середня	висока
Швидкодія інтерфейсу	висока	висока	середня	середня	висока
Безкоштовний тариф	до 10 дошок	до 15 осіб	до 10 осіб	відсутній	повністю безкоштовний
Базовий платний тариф (USD/місяць/користувач)	5	10,99	7,75	9	–
Бізнес-тариф (USD/місяць/користувач)	10	24,99	15,25	19	–
Обмеження файлів безкоштовно	10 МБ	100 МБ	2 ГБ	–	25 МБ
Підтримка клієнтів безкоштовно	спільнота	спільнота	спільнота	–	базова

Результати порівняльного аналізу за економічними параметрами свідчать, що використання комерційних сервісів управління завданнями у бізнес-сегменті потребує суттєвих фінансових витрат. Наприклад, для команди з 20 осіб щомісячна вартість підписки на бізнес-тариф Asana становитиме приблизно 500 USD, а у

річному обчисленні – близько 6000 USD. Для українських підприємств, особливо малого та середнього бізнесу, такі витрати можуть бути критичними, що актуалізує потребу в альтернативних рішеннях.

На основі результатів проведеного порівняльного аналізу слід виокремити основні недоліки, спільні для більшості розглянутих сервісів управління завданнями. По-перше, залежність від постачальника послуг (vendor lock-in), що пов'язана з типом розгортання SaaS та проявляється у неможливості повного контролю над даними, ризиках втрати доступу до інформації при зміні політики компанії-власника та відсутності можливостей адаптації коду під специфічні потреби. По-друге, обмеженість безкоштовних версій, які зазвичай не дозволяють повноцінно використовувати функціонал сервісу для команд понад 10–15 осіб, що змушує підприємства переходити на платні тарифи. По-третє, недостатня локалізація для українського ринку, що проявляється у відсутності або неповноті української мови інтерфейсу, недоступності деяких сервісів або їх частковому функціонуванні на території України через геополітичні обмеження.

З іншого боку, аналіз дає можливість виокремити позитивні риси розглянутих сервісів, які доцільно врахувати при проектуванні власного електронного сервісу. До таких рис належать інтуїтивний інтерфейс на основі канбан-дошок (Trello, Asana), розширена система ролей та прав доступу (Jira), автоматизовані email-сповіщення про події у системі (всі сервіси), можливість прикріплення файлів до завдань різних типів (всі сервіси), функціонал коментарів для забезпечення комунікації між учасниками проекту (всі сервіси крім Microsoft To Do), а також підтримка технології Drag and Drop для зручного управління завданнями (всі сервіси).

Узагальнюючи результати порівняльного аналізу, слід зазначити, що жоден із розглянутих сервісів не задовольняє повністю потреби малих та середніх українських підприємств у сфері бізнес-менеджменту. Комерційні сервіси характеризуються високою вартістю, обмеженнями безкоштовних версій, недостатньою локалізацією та залежністю від зовнішніх постачальників.

Безкоштовні рішення, такі як Microsoft To Do, мають обмежений функціонал, недостатній для повноцінного управління завданнями у бізнес-середовищі. Зазначені обставини обґрунтовують доцільність розроблення власного електронного сервісу управління завданнями з відкритою архітектурою, можливістю розгортання на власних серверах організації, повноцінною українською локалізацією та функціоналом, що задовольняє потреби бізнес-менеджменту. Сформульовані за результатами аналізу вимоги до такого сервісу детально розглядаються у наступному підрозділі.

1.4 Формування вимог до електронного сервісу управління завданнями

Формування вимог є одним із ключових етапів життєвого циклу розроблення програмного продукту, що безпосередньо впливає на якість майбутнього рішення та повноту задоволення потреб користувачів. У контексті проєктування електронного сервісу вимоги визначають функціональні можливості, технічні характеристики, особливості інтерфейсу та параметри безпеки системи [21]. У роботі вимоги формуються на основі результатів аналізу існуючих рішень, потреб цільової аудиторії та особливостей сфери бізнес-менеджменту з урахуванням сучасних стандартів розроблення веб-застосунків.

Відповідно до стандарту ISO/IEC/IEEE 29148:2018 «Systems and software engineering – Life cycle processes – Requirements engineering», вимоги до програмних систем поділяються на функціональні (визначають конкретні дії та операції системи) та нефункціональні (характеристики та обмеження продуктивності, безпеки, зручності використання) [16]. У межах цього підрозділу здійснюється послідовне формулювання обох категорій вимог із подальшим визначенням ролей користувачів.

Цільову аудиторію електронного сервісу складають керівники малих та середніх підприємств, керівники структурних підрозділів, проєктні менеджери та виконавці-співробітники. У зв'язку з цим у системі передбачається три ролі:

адміністратор має повний доступ до всіх функцій; керівник може створювати завдання, призначати їх виконавцям та контролювати прогрес; виконавець отримує доступ до призначених йому завдань та може змінювати їх статус.

До основних функціональних вимог електронного сервісу віднесено такі:

- реєстрація та автентифікація користувачів із валідацією даних, перевіркою унікальності email та шифруванням паролів за алгоритмом BCrypt;
- підтвердження облікового запису через email з використанням одноразового токена верифікації;
- створення, редагування, перегляд та видалення завдань (CRUD) з можливістю встановлення назви, опису, термінів виконання, набору виконавців, тегів та прикріплень;
- підтримка структури завдань з підзавданнями (чек-лістами) у вигляді переліку дій з ознакою виконання;
- візуалізація завдань на канбан-дошці з налаштовуваним для кожної групи переліком статусів (колонок) та переміщення задач між ними через Drag and Drop;
- категоризація завдань через систему тегів з кольоровим кодуванням, що налаштовується у межах кожної групи;
- прикріплення файлів до завдань (документи, зображення, архіви) з обмеженням розміру до 10 МБ;
- ведення системи коментарів до завдань з можливістю обговорення між учасниками;
- автоматичні email-сповіщення про створення завдання, призначення виконавця, наближення терміну та зміну статусу;
- щоденна автоматична розсилка нагадувань про завдання з терміном у наступні три дні через Spring Scheduler;

- формування статистики та аналітичних звітів за обраний період (кількість створених, виконаних та прострочених завдань, середній час виконання, продуктивність виконавців);
- пошук та фільтрація завдань за статусом та терміном виконання
- розмежування прав доступу відповідно до ролей на рівні URL-адрес та окремих операцій;
- ведення особистих профілів користувачів із редагуванням контактної інформації та зміною паролю.

Нефункціональні вимоги стосуються якісних характеристик системи, що не пов'язані безпосередньо з виконуваними функціями, але важливі для прийняттого користувацького досвіду та надійності експлуатації. Першу групу складають вимоги до продуктивності, надійності та безпеки. Час відгуку на основні операції (відкриття дошки, створення завдання, перегляд деталей) не повинен перевищувати 2 секунди при одночасній роботі до 100 користувачів. Система повинна забезпечувати безперервне функціонування з доступністю не нижче 99% протягом робочого часу, передбачати механізми відновлення після збоїв та регулярне резервне копіювання. Безпека реалізується через захист від поширених атак (SQL-ін'єкції, XSS, CSRF) засобами Spring Security, шифрування паролів, HTTPS у виробничому середовищі та розмежування прав на основі ролей (RBAC).

Другу групу нефункціональних вимог формують аспекти масштабованості, зручності використання, сумісності, підтримуваності та переносимості. Архітектура системи повинна забезпечувати горизонтальне масштабування шляхом додавання серверів застосунків без істотних змін у коді. Інтерфейс має бути інтуїтивно зрозумілим, з адаптивною версткою для мобільних пристроїв і планшетів та повною локалізацією українською мовою. Підтримуються сучасні веб-браузери (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) на пристроях з різною роздільною здатністю. Модульна архітектура з чітким розподілом відповідальностей (Controller, Service, Repository) забезпечує можливість змін без впливу на інші частини системи; покриття модульними тестами повинно бути не

менше 70%. Передбачається можливість розгортання на Linux та Windows з підтримкою контейнеризації через Docker.

Технічні вимоги до інфраструктури визначають вибір технологічного стеку. Серверна частина реалізується на Java 24 з фреймворком Spring Boot 3.5.0, для зберігання даних використовується PostgreSQL, клієнтська частина – на HTML5, CSS3, JavaScript та Bootstrap 5.3.3. Розсилка електронних повідомлень здійснюється через Gmail SMTP за допомогою Spring Mail.

Аспекти інформаційної безпеки реалізуються комплексно: автентифікація на основі форм входу (form-based authentication) із сесіями; авторизація на основі ролей (RBAC); шифрування паролів BCrypt; захист від CSRF за допомогою токенів; валідація вхідних даних на серверній стороні через Spring Validation з анотаціями Bean Validation. Конфіденційні параметри (паролі бази даних, ключі SMTP) зберігаються у змінних середовища, що виключає їх присутність у вихідному коді.

Програмний код повинен відповідати стандартам розроблення на платформі Java (Java Code Conventions, принципи SOLID). Управління залежностями та збирання здійснюється через Gradle, тестування – з використанням JUnit 5 та Spring Security Test, документування – через анотації Javadoc.

У результаті сформовано комплексний перелік функціональних та нефункціональних вимог, що формують основу для подальшого проєктування системи з урахуванням особливостей бізнес-менеджменту, специфіки цільової аудиторії та сучасних стандартів. У наступному розділі здійснюється детальне проєктування архітектури електронного сервісу, обґрунтування вибору технологій та моделювання компонентів системи з використанням UML-діаграм.

2 ПРОЄКТУВАННЯ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ЕЛЕКТРОННОГО СЕРВІСУ

2.1 Загальна архітектура системи

Архітектура програмного забезпечення є фундаментальним аспектом проєктування електронних сервісів, що визначає структурну організацію системи, способи взаємодії її компонентів та принципи розподілу відповідальностей між ними. Грамотно спроектована архітектура забезпечує такі ключові якісні характеристики програмного продукту як масштабованість, надійність, тестованість та супроводжуваність. У межах цього підрозділу розглядається загальна архітектура електронного сервісу управління завданнями, обґрунтовується вибір архітектурного стилю та описуються основні компоненти системи із зазначенням їх функціональних обов'язків та механізмів взаємодії.

Для розроблюваного електронного сервісу управління завданнями обрано трирівневу клієнт-серверну архітектуру (Three-Tier Architecture), що є класичним та широко апробованим підходом для побудови корпоративних веб-застосунків. Зазначена архітектура передбачає чіткий розподіл системи на три логічні рівні: рівень представлення (Presentation Layer), рівень бізнес-логіки (Business Logic Layer) та рівень доступу до даних (Data Access Layer). Такий розподіл забезпечує модульність системи, спрощує її тестування та супровід, а також надає можливість незалежного масштабування окремих компонентів [11]. Кожен із зазначених рівнів реалізує власні функціональні обов'язки та взаємодіє з іншими рівнями через чітко визначені інтерфейси.

Рівень представлення відповідає за взаємодію із кінцевим користувачем системи та забезпечує відображення даних у зручному для сприйняття вигляді. У розроблюваному електронному сервісі рівень представлення реалізується у двох формах: серверне рендерення HTML-сторінок з використанням шаблонізатора

Thymeleaf та REST API для обробки асинхронних запитів від клієнтського JavaScript-коду. Серверне рендерення забезпечує швидке завантаження сторінок та поліпшення пошукової оптимізації, тоді як REST API дає можливість реалізації інтерактивних елементів інтерфейсу, таких як канбан-дошка з функціоналом Drag and Drop, без перезавантаження сторінок. Клієнтська частина базується на технологіях HTML5, CSS3, фреймворку Bootstrap 5.3.3 для адаптивної верстки та Vanilla JavaScript для реалізації клієнтської логіки.

Рівень бізнес-логіки є центральним компонентом архітектури системи та відповідає за реалізацію функціональних можливостей сервісу. На цьому рівні здійснюється обробка вхідних запитів від користувачів, валідація даних, виконання обчислень та координація операцій із базою даних. У межах розроблюваного сервісу рівень бізнес-логіки реалізується з використанням фреймворку Spring Boot версії 3.5.0 та поділяється на декілька функціональних блоків. Контролери (Controllers), розмічені анотаціями `@Controller` та `@RestController`, приймають HTTP-запити та делегують їх обробку відповідним сервісам. Сервіси (Services), позначені анотацією `@Service`, інкапсулюють бізнес-логіку та взаємодіють з репозиторіями для виконання операцій із даними. Окремий блок становлять компоненти безпеки, реалізовані на основі Spring Security, що забезпечують автентифікацію та авторизацію користувачів.

Рівень доступу до даних відповідає за взаємодію з системою управління базами даних та абстрагує бізнес-логіку від особливостей зберігання інформації. У розроблюваному сервісі цей рівень реалізується з використанням Spring Data JPA та Hibernate як ORM-фреймворку, що забезпечує об'єктно-реляційне відображення між класами Java та таблицями реляційної бази даних. Репозиторії (Repositories), що успадковують інтерфейс `JpaRepository`, надають готові методи для виконання типових CRUD-операцій, а також можливість визначення власних методів пошуку через JPQL-запити. Як систему управління базами даних обрано PostgreSQL, що характеризується високою надійністю, підтримкою складних транзакцій та розширеними можливостями індексації.

Окрім трьох основних рівнів, архітектура системи передбачає наявність допоміжних компонентів, що забезпечують виконання спеціалізованих функцій. До таких компонентів належать модуль розсилки електронних повідомлень (Email Service) на основі Spring Mail з підключенням до сервера Gmail SMTP, планувальник задач (Task Scheduler), реалізований через анотацію `@Scheduled` для виконання періодичних операцій, а також підсистема обробки файлів (File Storage), що забезпечує завантаження та зберігання прикріплень до завдань через механізм `Multipart file upload` з обмеженням розміру файлу до 10 МБ.

Взаємодія між компонентами системи здійснюється за принципом інверсії залежностей (Dependency Inversion Principle), що є одним із фундаментальних принципів об'єктно-орієнтованого проєктування SOLID. Зазначений принцип реалізується засобами Spring Framework через механізм впровадження залежностей (Dependency Injection), коли необхідні залежні об'єкти автоматично передаються компонентам через конструктори або поля, розмічені анотацією `@Autowired`. Такий підхід суттєво полегшує тестування системи, оскільки дає можливість підміняти реальні залежності тестовими заміниками (mocks).

З точки зору моделі взаємодії клієнта та сервера, у системі реалізуються два сценарії. Перший сценарій передбачає традиційний обмін HTTP-запитами із серверним рендеренням сторінок: користувач надсилає GET- або POST-запит, контролер обробляє його, передає необхідні дані до Thymeleaf-шаблону, після чого сервер повертає клієнту повністю згенеровану HTML-сторінку. Другий сценарій реалізується для асинхронних взаємодій через REST API: клієнтський JavaScript-код надсилає AJAX-запит до відповідного REST-контролера, який повертає дані у форматі JSON, що використовуються для оновлення окремих елементів сторінки без її повного перезавантаження. Цей підхід застосовується, зокрема, для функціоналу канбан-дошки при переміщенні завдань між колонками за допомогою Drag and Drop API.

Безпека системи реалізується на рівні бізнес-логіки засобами фреймворку Spring Security, що забезпечує автентифікацію користувачів, авторизацію на основі

ролей, захист від поширених видів атак (CSRF, XSS) та шифрування паролів за алгоритмом BCrypt. Конфігурація безпеки описується у класі, розміченому анотацією `@Configuration`, де визначаються правила доступу до різних URL-адрес та налаштовуються механізми обробки запитів. Розмежування прав доступу здійснюється на двох рівнях: на рівні URL-адрес через бін `SecurityFilterChain` конфігураційного класу та на рівні окремих методів сервісів через перевірки авторизації, що виконуються у відповідних сервісних класах.

З метою забезпечення гнучкості налаштувань системи та можливості розгортання у різних середовищах (розробка, тестування, виробництво) у застосунку використовується механізм змінних середовища (`environment variables`) для зберігання конфіденційних та середовище-залежних параметрів. До таких параметрів належать рядок підключення до бази даних (`DB_URL`), реквізити доступу до SMTP-сервера (`EMAIL_USERNAME`, `EMAIL_PASSWORD`) та базова URL-адреса застосунку (`APP_URL`). Зазначений підхід відповідає рекомендаціям методології `Twelve-Factor App` для побудови сучасних веб-застосунків та виключає присутність конфіденційних даних у вихідному коді [25].

Важливим аспектом архітектури системи є організація потоку обробки запиту від моменту його надходження від користувача до формування відповіді. Запит спочатку проходить через ланцюг фільтрів безпеки (`Security Filter Chain`) `Spring Security`, де здійснюється автентифікація користувача та перевірка його прав доступу. Після успішного проходження фільтрів запит передається диспетчеру сервлетів (`DispatcherServlet`), який маршрутизує його до відповідного контролера на основі URL-адреси та HTTP-методу. Контролер виконує валідацію вхідних параметрів, делегує обробку відповідному сервісу, отримує результат та формує відповідь – або у вигляді назви HTML-шаблону для серверного рендерення, або у вигляді JSON-об'єкта для REST API. Сервіс, у свою чергу, виконує бізнес-логіку, взаємодіє з репозиторіями для роботи з базою даних та може звертатися до допоміжних компонентів (`Email Service`, `File Storage`).

Запропонована архітектура електронного сервісу управління завданнями забезпечує необхідний рівень модульності, тестованості та масштабованості, що відповідає сформульованим у попередньому розділі вимогам. Структурна організація системи дає можливість здійснювати її поетапне розроблення, тестування окремих компонентів незалежно один від одного, а також спрощує внесення змін та додавання нового функціоналу у майбутньому. Загальна схема архітектури системи представлена на рис. 2.1.

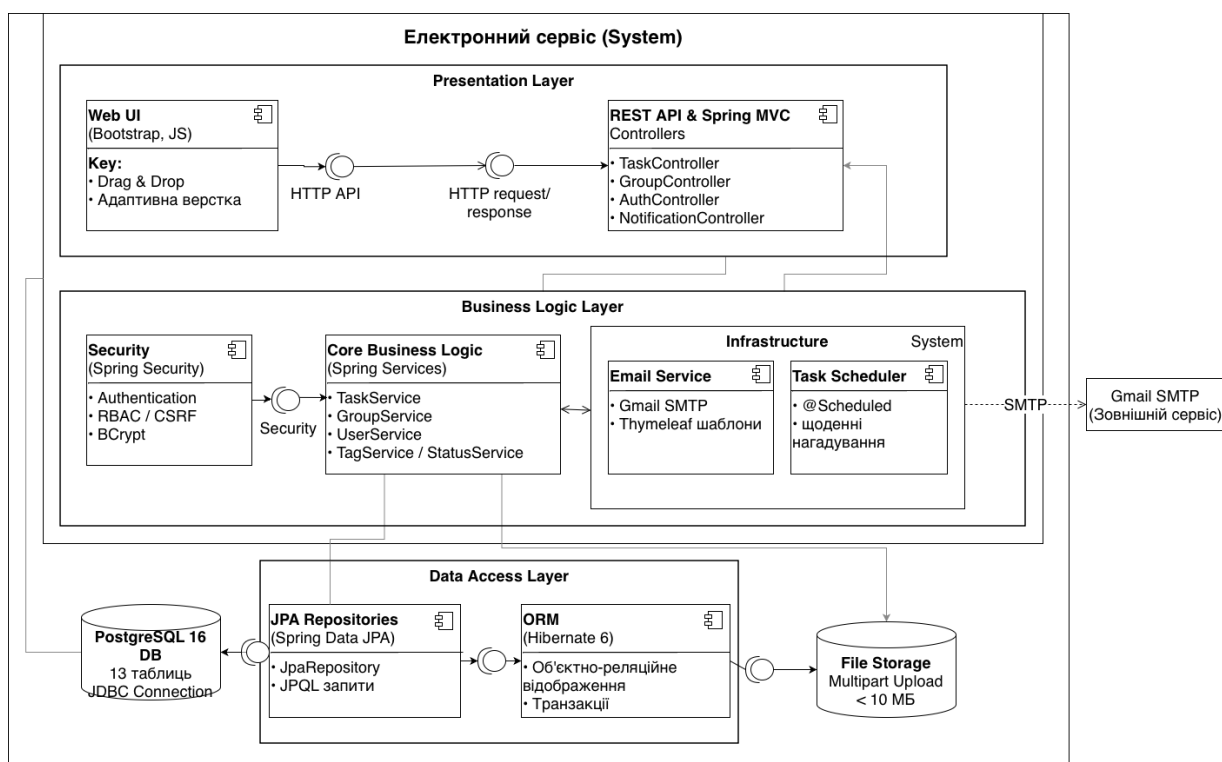


Рис. 2.1 Загальна архітектура електронного сервісу управління завданнями

На діаграмі відображено основні компоненти системи, їх розподіл за логічними рівнями та напрями взаємодії між ними. Окремо показано компоненти безпеки, бізнес-логіки, доступу до даних, файлового сховища та підсистеми email-сповіщень. Такий підхід дає змогу наочно простежити повний шлях обробки запиту користувача від інтерфейсу до бази даних і зовнішніх сервісів.

2.2 Обґрунтування вибору технологій розробки

Вибір технологічного стеку є одним із найважливіших рішень на етапі проектування електронного сервісу, оскільки він безпосередньо впливає на якість майбутнього програмного продукту, швидкість розроблення, зручність супроводу та можливості подальшого масштабування. Обґрунтований підбір технологій дає можливість оптимально реалізувати функціональні та нефункціональні вимоги до системи, а також забезпечити її конкурентоспроможність у довгостроковій перспективі [4]. У роботі вибір технологій здійснюється на основі аналізу їх відповідності сформульованим вимогам, поширеності у галузі, наявності розвиненої екосистеми бібліотек та якості документації.

Як основна мова програмування для реалізації серверної частини електронного сервісу обрано Java версії 24, що є найновішою стабільною версією платформи Java на момент розроблення. Java є однією з найпопулярніших мов програмування у світі та посідає провідні позиції у рейтингах TIOBE Index та PYPL PopularitY of Programming Language Index [25]. Серед основних переваг мови Java для розроблення корпоративних веб-застосунків доцільно виокремити такі: статична типізація, що дозволяє виявляти значну частину помилок ще на етапі компіляції; кросплатформність, забезпечена віртуальною машиною JVM; розвинена екосистема бібліотек та фреймворків; висока продуктивність завдяки JIT-компіляції та оптимізаціям сучасних JVM (HotSpot, GraalVM); потужні засоби забезпечення безпеки; широка підтримка спільнотою розробників та комерційними організаціями.

Версія Java 24 містить низку нововведень, що значно поліпшують продуктивність розроблення та якість програмного коду. Серед них слід відзначити стабілізовані можливості Pattern Matching for switch, що спрощують роботу з умовною логікою; Records для лаконічного оголошення класів-носіїв даних; Sealed Classes для контрольованого успадкування; Virtual Threads (Project Loom), що

забезпечують ефективну роботу з великою кількістю одночасних з'єднань; Text Blocks для зручного запису багаторядкових рядків. Зазначені можливості сприяють написанню більш стислого, виразного та безпечного коду порівняно з попередніми версіями платформи [25].

Для побудови серверної частини застосунку обрано фреймворк Spring Boot версії 3.5.0, що належить до екосистеми Spring Framework – однієї з найбільш розвинених та поширених платформ для розроблення корпоративних Java-застосунків. Spring Boot реалізує концепцію «convention over configuration» (домовленості замість конфігурації), що дає можливість суттєво зменшити обсяг шаблонного коду та зосередитися на бізнес-логіці системи. До основних переваг Spring Boot належать: автоматична конфігурація компонентів на основі залежностей у проєкті; вбудований сервер застосунків (Tomcat за замовчуванням), що спрощує розгортання; механізм стартерів (starters) для зручного підключення груп залежностей; широкий набір готових інтеграцій із різними технологіями; підтримка моніторингу та діагностики через Spring Boot Actuator [25].

У межах екосистеми Spring у проєкті використовується низка модулів, кожен з яких реалізує специфічну функціональність. Модуль Spring MVC забезпечує побудову веб-застосунків на основі патерну Model-View-Controller, обробку HTTP-запитів та формування відповідей. Spring Data JPA надає високорівневу абстракцію для роботи з реляційними базами даних через інтерфейс JpaRepository, що автоматично генерує типові CRUD-операції без необхідності їх явної реалізації. Spring Security забезпечує комплексний механізм автентифікації та авторизації, включно із захистом від поширених видів атак. Spring Mail спрощує надсилання електронних повідомлень через SMTP-сервери. Spring Validation реалізує декларативну валідацію даних на основі стандарту Bean Validation (JSR 380). Spring Scheduler через анотацію @Scheduled дає можливість планувати періодичне виконання задач без використання зовнішніх планувальників.

Для зберігання даних обрано об'єктно-реляційну систему управління базами даних PostgreSQL. PostgreSQL є однією з найпотужніших безкоштовних СУБД з

відкритим вихідним кодом, що активно розвивається з 1996 року та має понад 30-річну історію використання у промислових системах. На користь вибору PostgreSQL свідчать такі його характеристики: повна підтримка стандарту SQL та розширень мови (PL/pgSQL); дотримання принципів ACID для забезпечення цілісності транзакцій; підтримка складних типів даних (JSON, JSONB, масиви, географічні типи); розширені можливості індексації (B-Tree, Hash, GIN, GiST, BRIN); потужна система контролю одночасності на основі MVCC (Multi-Version Concurrency Control); вбудовані засоби реплікації та резервного копіювання; активна спільнота розробників та регулярні релізи нових версій [16]. Порівняно з альтернативними СУБД, такими як MySQL, PostgreSQL відзначається кращою підтримкою стандартів SQL, ширшим набором типів даних та більш гнучкою системою індексації, що особливо важливо для розроблюваного сервісу з можливістю розширення функціоналу у майбутньому.

Взаємодія Java-застосунку з базою даних PostgreSQL здійснюється через ORM-фреймворк Hibernate, що є де-факто стандартом для об'єктно-реляційного відображення у Java-екосистемі. Hibernate автоматично перетворює Java-об'єкти на записи бази даних та навпаки, що дозволяє розробникам зосередитися на бізнес-логіці замість написання SQL-запитів. Серед переваг Hibernate слід виокремити підтримку різних стратегій усядження для класів сутностей, ліниву (lazy) та активну (eager) загрузку зв'язаних об'єктів, кешування на рівні сесії та другого рівня, генерацію SQL-запитів через мову HQL (Hibernate Query Language) та Criteria API. Інтеграція Hibernate з Spring Data JPA додатково спрощує його використання через декларативний підхід до визначення репозиторіїв.

Для реалізації клієнтської частини сервісу використовуються веб-стандарти HTML5, CSS3 та мова JavaScript (Vanilla JS). Вибір цих базових технологій замість сучасних JavaScript-фреймворків (React, Vue.js, Angular) обумовлений декількома міркуваннями: спрощення архітектури системи завдяки відсутності окремого фронтенд-проєкту; зменшення часу завантаження сторінок через відсутність необхідності завантажувати об'ємні бібліотеки; кращі показники пошукової

оптимізації (SEO) завдяки серверному рендерингу; нижчий поріг входу для розробників, що супроводжуватимуть систему; повний контроль над клієнтським кодом без залежності від зовнішніх абстракцій. Для реалізації інтерактивних елементів інтерфейсу, зокрема канбан-дошки з функціоналом переміщення завдань, використовується вбудоване API браузера Drag and Drop, що не потребує підключення сторонніх бібліотек.

Серверне рендерення HTML-сторінок здійснюється за допомогою шаблонізатора Thymeleaf, що є офіційно рекомендованим шаблонізатором для Spring-застосунків. Thymeleaf характеризується природним синтаксисом – шаблони залишаються валідним HTML, який можна відкривати та переглядати у браузері без серверного оброблення. До переваг Thymeleaf належать інтеграція зі Spring Framework через автоконфігурацію, підтримка інтернаціоналізації, наявність розширень для роботи з різними технологіями (зокрема Spring Security через бібліотеку thymeleaf-extras-springsecurity6), потужний механізм фрагментів для повторного використання частин шаблонів та виразів для динамічної генерації контенту [25].

Для побудови адаптивної верстки клієнтського інтерфейсу обрано CSS-фреймворк Bootstrap версії 5.3.3. Bootstrap є одним із найпоширеніших фронтенд-фреймворків у світі та надає готовий набір компонентів (форми, кнопки, таблиці, навігаційні панелі, модальні вікна тощо), що значно прискорює розроблення інтерфейсу. Версія 5.3.3 не залежить від бібліотеки jQuery, що зменшує загальний обсяг завантажуваного коду, та підтримує сучасні можливості CSS, такі як CSS Grid та Flexbox. Адаптивна сітка Bootstrap забезпечує коректне відображення інтерфейсу на пристроях з різною роздільною здатністю екрану, від мобільних телефонів до настільних моніторів [25]. Доповнює Bootstrap бібліотека іконок Bootstrap Icons версії 1.11.3, що містить понад 2000 SVG-іконок для використання в інтерфейсі.

Для управління залежностями та збирання проєкту обрано систему Gradle, що є альтернативою більш традиційному інструменту Apache Maven. На користь

Gradle свідчать такі його переваги: декларативний DSL на основі Groovy або Kotlin, що дає можливість гнучкого налаштування процесу збирання; інкрементальне збирання проєкту, що значно прискорює повторні компіляції; вбудована підтримка багатомодульних проєктів; інтеграція з більшістю популярних IDE; активний розвиток та регулярні поліпшення продуктивності. Конфігурація проєкту описується у файлі `build.gradle`, де визначаються залежності, плагіни та параметри збирання. Для зручного управління версіями залежностей Spring використовується плагін `io.spring.dependency-management`, що автоматично узгоджує версії модулів Spring між собою.

Тестування системи здійснюється з використанням фреймворку JUnit 5 – стандарту *de facto* для модульного та інтеграційного тестування Java-застосунків. JUnit 5 пропонує модернізовану архітектуру порівняно з попередньою версією, підтримку лямбда-виразів, параметризованих тестів та розширень. У межах Spring-екосистеми тестування інтегровано через стартер `spring-boot-starter-test`, що автоматично підключає необхідні бібліотеки (JUnit Jupiter, Mockito, AssertJ, Hamcrest). Для тестування компонентів безпеки додатково використовується модуль Spring Security Test, що надає засоби для імітації автентифікованих користувачів у тестах та перевірки правил доступу.

Розсилка електронних повідомлень здійснюється через сервіс Gmail SMTP, що є безкоштовним рішенням для невеликих та середніх обсягів кореспонденції (до 500 листів на добу для безкоштовного облікового запису). Конфігурація підключення до Gmail SMTP задається через параметри застосунку та використовує захищений протокол TLS на порту 587. Доступ здійснюється через додаткові паролі застосунків (App Passwords), що забезпечує більшу безпеку порівняно з використанням основного паролю облікового запису.

Зведений перелік обраних технологій з обґрунтуванням їх застосування у системі представлено у таблиці 2.1 (додаток Б), що дає цілісне уявлення про технологічний стек розроблюваного електронного сервісу.

Здійснений аналіз обраних технологій свідчить про їх повну відповідність функціональним та нефункціональним вимогам до розроблюваного електронного сервісу. Поєднання платформи Java 24 та фреймворку Spring Boot 3.5.0 забезпечує необхідну продуктивність та масштабованість серверної частини. Використання PostgreSQL як СУБД гарантує надійне зберігання даних з підтримкою складних транзакційних операцій. Технології клієнтської частини (HTML5, CSS3, Bootstrap, Vanilla JavaScript) забезпечують створення сучасного адаптивного інтерфейсу без зайвого ускладнення архітектури. Усі обрані технології є зрілими, добре задокументованими та мають активні спільноти розробників, що гарантує їх довгострокову підтримку та можливість супроводу системи у майбутньому.

2.3 Проєктування структури бази даних

Проєктування структури бази даних є одним із ключових етапів розроблення електронного сервісу управління завданнями, оскільки саме від якості організації даних залежать продуктивність системи, цілісність інформації, можливості подальшого розширення функціоналу та ефективність виконання запитів. Грамотно спроектована структура забезпечує мінімізацію надлишковості даних, дотримання принципів нормалізації, підтримку посилкової цілісності та оптимальну продуктивність типових операцій [12]. У межах цього підрозділу здійснюється детальне проєктування реляційної моделі бази даних з обґрунтуванням обраних структур, типів даних та зв'язків між сутностями.

Процес проєктування реалізується у три етапи: концептуальне моделювання (визначення сутностей предметної області та зв'язків між ними у вигляді ER-діаграми), логічне моделювання (перетворення концептуальної моделі на реляційну схему з визначенням таблиць, атрибутів, ключів) та фізичне моделювання (конкретизація типів даних, обмежень, індексів та інших параметрів, специфічних для PostgreSQL).

Аналіз функціональних вимог дає можливість виокремити основні сутності предметної області. Центральною організаційною одиницею системи є група (Group), що об'єднує користувачів навколо певної робочої діяльності та є контейнером для завдань, статусів і тегів. Інші основні сутності: користувач (User), завдання (Task), підзавдання (SubTask), статус (Status), тег (Tag), коментар (Comment), учасник групи (GroupMember), сповіщення (Notification), запрошення до групи (UserInvitation) та запис історії змін завдання (TaskHistory).

Сутність «Користувач» (users) відображає інформацію про осіб, які мають доступ до системи. Атрибути включають унікальний ідентифікатор (id), електронну пошту (email) як унікальне поле для входу, унікальний логін (username), зашифрований за алгоритмом BCrypt пароль (password), ім'я та прізвище (firstName, lastName), ознаку підтвердження email (emailVerified), токен верифікації (verificationToken), термін дії токена (tokenExpiryDate), а також роль у системі (role) як рядкове поле зі значеннями ROLE_ADMIN, ROLE_MANAGER або ROLE_USER. Зберігання ролі безпосередньо у таблиці users як рядкового поля спрощує модель даних та усуває потребу в окремих таблицях ролей і проміжних зв'язках; Spring Security отримує роль через CustomUserDetailsService, який перетворює рядок на GrantedAuthority. Часові мітки створення (createdAt) керуються автоматично через анотацію @CreationTimestamp.

Сутність «Група» (groups_table) виконує роль логічного контейнера для завдань і користувачів. Атрибути сутності: ідентифікатор (id), назва (name), опис (description), дата створення (createdAt), посилання на користувача-власника (owner_id) як зовнішній ключ до users. Назва таблиці groups_table обрана з урахуванням того, що group є зарезервованим словом у SQL.

Сутність «Учасник групи» (group_members) реалізує зв'язок між користувачами та групами як «багато-до-багатьох» з додатковим полем ролі учасника у межах групи. Атрибути: ідентифікатор (id), посилання на користувача (user_id), посилання на групу (group_id) та роль учасника (role) зі значеннями ADMIN, MEMBER або VIEWER, що визначає обсяг прав у межах конкретної

групи. Така реалізація дає можливість одному користувачу мати різні ролі у різних групах.

Центральною сутністю системи є «Завдання» (tasks). Атрибути включають ідентифікатор (id), назву (title), опис (description), дату початку (startDate), дату закінчення (endDate), термін виконання (dueDate), ознаку відкладеного завдання (deferred), серіалізований JSON-рядок з метаданими прикріплень (attachments), час створення (createdAt), посилання на групу (group_id), посилання на автора (author_id) та посилання на статус (status_id). Зберігання метаданих файлів безпосередньо у текстовому полі замість окремої таблиці є компромісним рішенням, що спрощує схему бази даних та зменшує кількість JOIN-операцій при отриманні завдання з прикріпленнями. Самі файли зберігаються у виділеній директорії на сервері, шлях до якої задається параметром file.upload-dir.

Зв'язок завдання з виконавцями реалізовано як «багато-до-багатьох» через проміжну таблицю task_assignees з полями task_id та user_id, що дає можливість призначати декілька виконавців на одне завдання. Аналогічно реалізовано зв'язок завдання з тегами через таблицю task_tags. Підзавдання представлено окремою сутністю SubTask з власною таблицею subtasks (атрибути id, title, done, createdAt, task_id), що замінює ієрархічну рекурсивну модель та дає можливість простого представлення чек-лістів у межах завдання.

Сутність «Статус» (statuses) реалізує динамічну модель статусів замість фіксованого переліку. Атрибути сутності: ідентифікатор (id), відображувана назва (displayName), порядок відображення на дошці (displayOrder) та посилання на групу (group_id). Кожна група має власний набір статусів, що дозволяє гнучко налаштовувати канбан-дошку відповідно до специфіки робочих процесів конкретної групи (за замовчуванням створюються три типові статуси, але кількість і назви колонок можуть бути змінені користувачем-власником групи). Завдання зв'язане зі статусом через @ManyToOne зв'язок status_id.

Сутність «Тег» (tags) дає можливість категоризувати завдання у межах групи. Атрибути: ідентифікатор (id), назва (name), колір відображення (color), посилання

на групу (`group_id`). Зв'язок з завданнями реалізовано через проміжну таблицю `task_tags` типу «багато-до-багатьох».

Сутність «Коментар» (`task_comments`) забезпечує функціонал обговорення завдань між учасниками групи. Атрибути: ідентифікатор (`id`), текст коментаря (`content`), час створення (`createdAt`), посилання на автора (`author_id`) та завдання (`task_id`). Зв'язки реалізовано як «багато-до-одного».

Сутність «Сповіщення» (`notifications`) відповідає за внутрішні повідомлення користувачам у межах системи (на додачу до email-розсилки). Атрибути: ідентифікатор (`id`), посилання на користувача-отримувача (`user_id`), тип сповіщення (`type`), заголовок (`title`), текст повідомлення (`message`), посилання на пов'язаний ресурс у системі (`link`), ознака прочитання (`read`) та час створення (`createdAt`). Окрема сутність «Запрошення до групи» (`user_invitations`) використовується для процедури запрошення нових учасників: атрибути включають `id`, `email` запрошеного, посилання на групу (`group_id`), унікальний токен (`token`), статус запрошення (`status`) та час створення (`createdAt`).

Сутність «Історія змін завдання» (`task_history`) фіксує усі зміни, що відбуваються із завданнями, для подальшого аудиту та формування звітів. Атрибути: ідентифікатор (`id`), посилання на завдання (`task_id`), посилання на користувача, який вніс зміну (`user_id`), тип дії (`action`), назва зміненого поля (`fieldName`), попереднє значення (`oldValue`), нове значення (`newValue`), час зміни (`changedAt`). Зазначена сутність дає можливість реконструювати історію роботи над завданням та виявляти, хто і коли вніс конкретні зміни.

При проектуванні дотримано принципів нормалізації до третьої нормальної форми (3NF), що забезпечує мінімізацію надлишковості та зменшує ризики аномалій оновлення. Багатозначні зв'язки реалізовано через проміжні таблиці (`group_members`, `task_assignees`, `task_tags`), що відповідає принципам реляційного підходу.

З метою забезпечення цілісності даних визначено систему обмежень. Первинні ключі реалізовано через колонку `id` типу `BIGSERIAL` з автоматичним

збільшенням значень. Зовнішні ключі забезпечують посилкову цілісність та налаштовані з різними правилами каскадного оновлення. При видаленні групи встановлено правило ON DELETE CASCADE для пов'язаних завдань, статусів, тегів та учасників. Для коментарів, підзавдань та записів історії аналогічно встановлено каскадне видалення при видаленні відповідного завдання. Обмеження UNIQUE накладено на електронну пошту та логін користувача (email, username) і токени запрошень. Обмеження NOT NULL забезпечує обов'язковість критичних полів.

Для оптимізації продуктивності типових запитів у базі даних створюються індекси на ключових полях. Окрім автоматично створюваних індексів для первинних та унікальних ключів, додатково проєктуються індекси на зовнішніх ключах (group_id, status_id, author_id у таблиці tasks; user_id у notifications), на полях, за якими часто виконується фільтрація (dueDate, deferred, read у notifications), а також на полі createdAt для сортування за часом.

З огляду на вимоги до інформаційної безпеки, у базі даних зберігаються лише хеш-значення паролів, отримані за алгоритмом BCrypt. Зберігання паролів у відкритому вигляді чи з використанням слабких алгоритмів (MD5, SHA-1) є неприпустимим.

При фізичному проєктуванні обрано конкретні типи даних PostgreSQL для кожного атрибута. Для ідентифікаторів використовується BIGSERIAL, для текстових полів обмеженої довжини – VARCHAR відповідного розміру, для довгих текстових полів (опис завдань, коментарі, JSON-метадані прикріплень) – TEXT. Часові мітки реалізуються через TIMESTAMP з часовою зоною, дати – через DATE. Логічні поля (deferred, read, emailVerified, done) реалізуються через BOOLEAN.

Узагальнюючи результати проєктування, структура бази даних електронного сервісу включає тринадцять основних таблиць (users, groups_table, group_members, tasks, task_assignees, subtasks, statuses, tags, task_tags, task_comments, notifications, user_invitations, task_history), пов'язаних системою зовнішніх ключів. Структура

відповідає принципам нормалізації, забезпечує цілісність даних через систему обмежень та оптимізована для типових сценаріїв використання за допомогою індексів. ER-діаграма бази даних з усіма сутностями, атрибутами та зв'язками представлена на рис. 2.2.

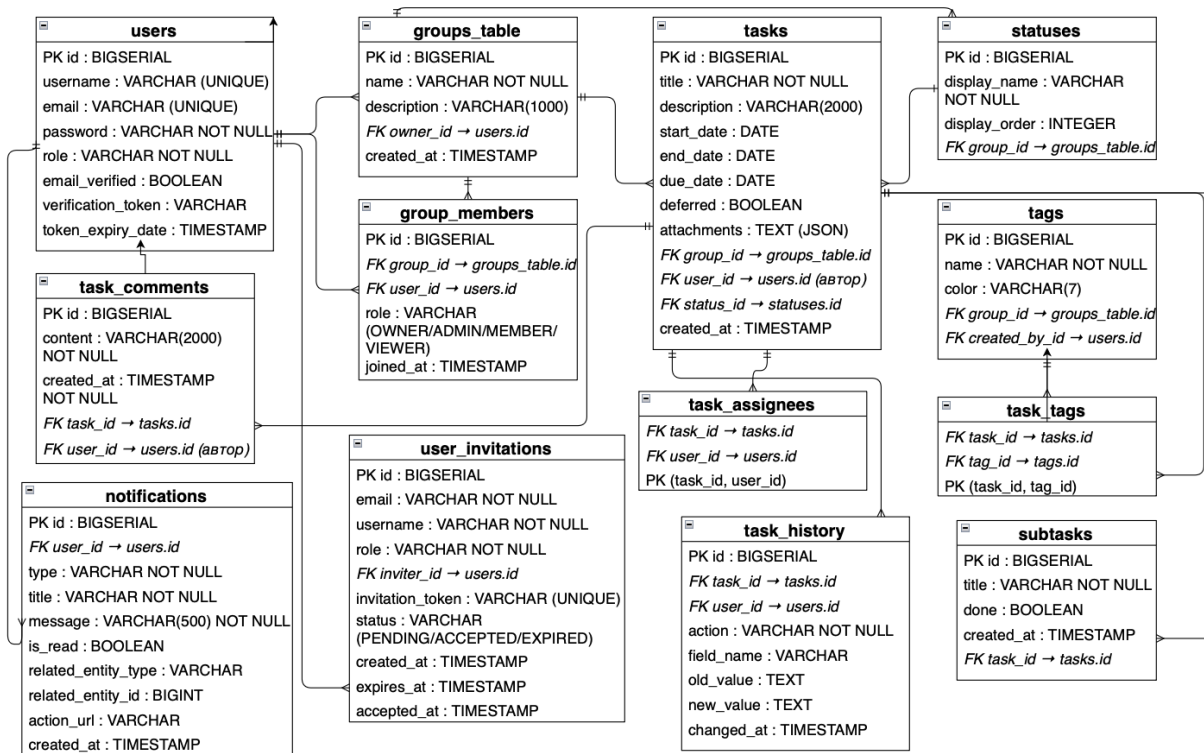


Рис. 2.2 ER-діаграма бази даних електронного сервісу управління завданнями

На діаграмі відображено ключові сутності предметної області, їх атрибути та типи зв'язків між таблицями. Особливу увагу приділено зв'язкам між користувачами, групами, завданнями, статусами, тегами, коментарями та сповіщеннями, оскільки саме ці сутності формують основу роботи сервісу. Така структура бази даних забезпечує гнучке управління завданнями, підтримку командної взаємодії та можливість подальшого розширення функціоналу системи.

2.4 Моделювання системи за допомогою uml-діаграм

Моделювання програмних систем із використанням мови UML (Unified Modeling Language) є загальноприйнятою практикою у галузі програмної інженерії, що дає можливість візуалізувати структуру та поведінку розроблюваної системи на різних рівнях абстракції. UML є стандартом, прийнятим Object Management Group, та містить набір графічних нотацій для опису різних аспектів програмних систем – від загальних сценаріїв використання до детальної структури класів та послідовності взаємодії об'єктів [20]. У роботі для моделювання електронного сервісу використано три типи UML-діаграм: діаграму прецедентів, діаграму класів та діаграму послідовності, що у сукупності забезпечують повноцінне відображення функціональних і структурних аспектів системи.

Діаграма прецедентів є одним із найбільш фундаментальних видів UML-діаграм, що відображає взаємодію між акторами (зовнішніми сутностями) та функціями системи у вигляді сценаріїв використання. Цей тип діаграм використовується на ранніх етапах проєктування для визначення меж системи, ідентифікації її користувачів та формування переліку основних функцій. Для розроблюваного сервісу визначено три типи акторів, що утворюють ієрархію за принципом узагальнення (generalization) у нотації UML: гість (неавтентифікований відвідувач), виконавець (zareєстрований користувач без розширених прав), керівник (користувач, що може створювати завдання та призначати виконавців) та адміністратор (повний набір прав для управління системою).

До основних прецедентів, доступних гостю, належать реєстрація у системі, підтвердження email-адреси та вхід в обліковий запис. Виконавець додатково отримує доступ до перегляду завдань власних груп, зміни статусу призначених завдань, додавання коментарів, прикріплення файлів та редагування персонального профілю. Керівник додатково має можливість створювати, редагувати та видаляти групи, формувати завдання, налаштовувати перелік статусів і тегів групи, призначати виконавців, переглядати аналітичні звіти та керувати учасниками груп

через систему запрошень. Адміністратор володіє розширеним набором прав для управління користувачами, призначення ролей, перегляду системних журналів та налаштування глобальних параметрів. Діаграму прецедентів електронного сервісу зображено на рис. 2.3.

Діаграма класів відображає статичну структуру системи у вигляді класів, їх атрибутів, методів та зв'язків між ними. Цей тип діаграми є особливо важливим для об'єктно-орієнтованого проєктування, оскільки фактично визначає скелет програмної архітектури. У контексті розроблюваного сервісу діаграма класів моделює сутнісний шар (Entity Layer), що відповідає таблицям бази даних та описує доменну модель застосунку.



Рис. 2.3 Діаграма прецедентів електронного сервісу управління завданнями

Центральним класом доменної моделі є клас User, що представляє користувача системи. Клас містить атрибути id, email, username, password, firstName, lastName, role (рядкове поле зі значеннями ROLE_ADMIN, ROLE_MANAGER або ROLE_USER), emailVerified (ознака підтвердження адреси), verificationToken, tokenExpiryDate, createdAt. Клас розмічається анотацією @Entity та інтегрується з механізмом автентифікації Spring Security через

CustomUserDetailsService, що перетворює рядкове значення поля role на GrantedAuthority. Зберігання ролі безпосередньо у таблиці users як рядкового поля спрощує модель даних та усуває потребу в окремих сутностях ролі та проміжних зв'язках типу «багато-до-багатьох».

Клас Group представляє організаційну одиницю – групу – як логічний контейнер для завдань, статусів і тегів. Клас містить атрибути id, name, description, createdAt та зв'язок з User у вигляді поля owner («багато-до-одного»). Учасники групи моделюються окремим класом GroupMember з атрибутами id, user_id, group_id та role (роль учасника у межах групи: ADMIN, MEMBER або VIEWER), що дає можливість одному користувачу мати різні ролі у різних групах. Класом, що відображає основну сутність системи, є Task. Його атрибути включають id, title, description, startDate, endDate, dueDate, deferred (ознака відкладеного завдання), attachments (серіалізований у JSON перелік метаданих прикріплень), createdAt, а зв'язки реалізовано через поля group (належність до групи), author (автор), status (поточний статус задачі) та колекції assignedUsers (виконавці, «багато-до-багатьох» через таблицю task_assignees), tags (теги), subtasks (чек-лісти) і comments (коментарі). Клас Status моделює динамічний статус канбан-дошки та містить атрибути id, displayName, displayOrder та посилання на групу, що дає можливість кожній групі визначати власний набір колонок дошки. Клас SubTask відображає підзавдання у вигляді чек-ліста з атрибутами id, title, done та createdAt. Допоміжні класи Comment, Tag, Notification, UserInvitation та TaskHistory моделюють відповідно коментарі, теги задач, внутрішні сповіщення, запрошення до групи та записи історії змін задачі. Повну доменну модель з атрибутами, методами та зв'язками наведено на рис. 2.4.

Діаграма послідовності відображає динамічну поведінку системи на прикладі типового сценарію – створення нового завдання керівником із повідомленням виконавця електронною поштою. Зазначений сценарій є одним із ключових бізнес-процесів та демонструє взаємодію між основними компонентами системи у часі.

Сценарій починається з відкриття керівником сторінки створення завдання у браузері. Браузер надсилає GET-запит `/tasks/new` до контролера `TaskController`, який повертає HTML-сторінку з формою. Після заповнення форми та її надсилання браузер виконує POST-запит `/tasks` із параметрами завдання. Запит проходить через ланцюг фільтрів безпеки `Spring Security`, де перевіряється автентифікація керівника та його право на створення завдань. Контролер делегує створення сервісу `TaskService`, який виконує валідацію вхідних даних, формує об'єкт `Task` та зберігає його через `TaskRepository` в базі `PostgreSQL`. Після успішного збереження `TaskService` викликає `EmailService` для надсилання повідомлення виконавцю, причому `EmailService` формує лист з використанням шаблону `Thymeleaf` та надсилає його через `Gmail SMTP`. Після завершення операцій контролер повертає браузеру HTTP-відповідь з кодом 302 (Redirect) на сторінку перегляду створеного завдання. Послідовність взаємодії компонентів у часі показано на рис. 2.5.

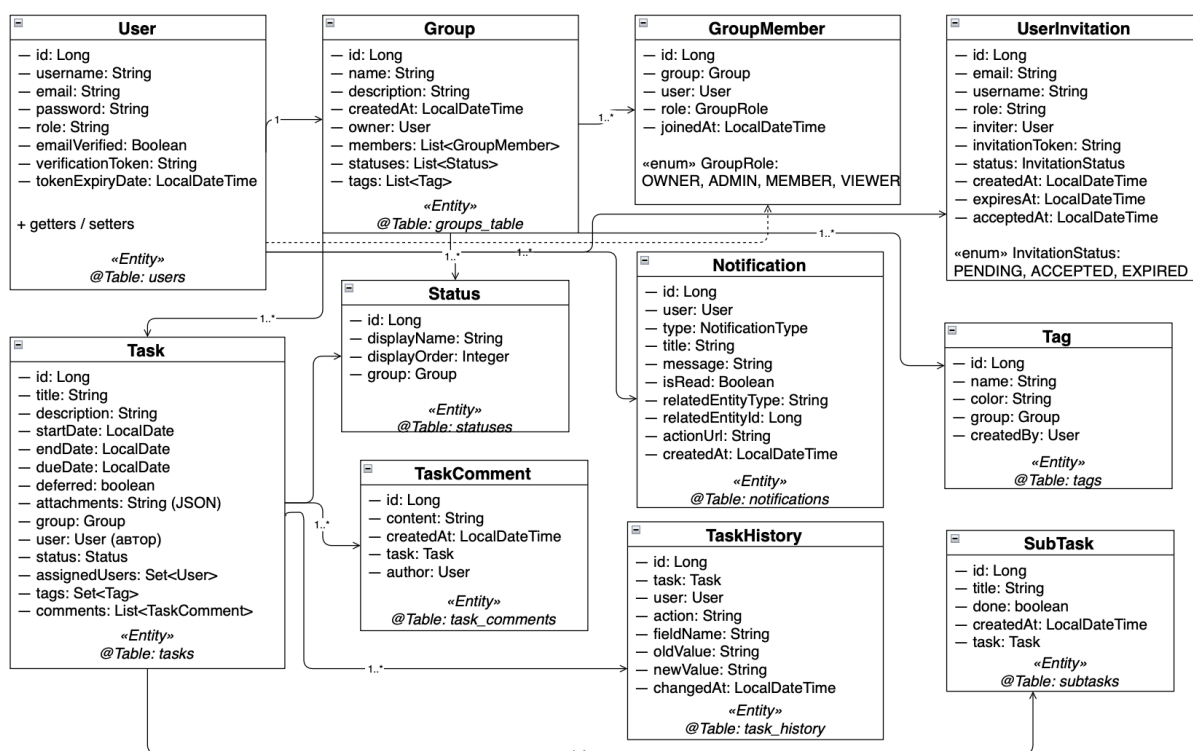


Рис. 2.4 Діаграма класів доменної моделі електронного сервісу

Окремої уваги заслуговує моделювання асинхронної взаємодії між компонентами у випадках, коли вона є критичною для продуктивності. Зокрема, надсилання електронних повідомлень здійснюється у фоновому режимі через анотацію `@Async`, що дає можливість контролеру повернути відповідь користувачу одразу після збереження завдання, не очікуючи завершення процесу розсилки. Такий підхід забезпечує швидкий відгук інтерфейсу та не блокує основний потік виконання при тимчасових затримках SMTP-сервера.

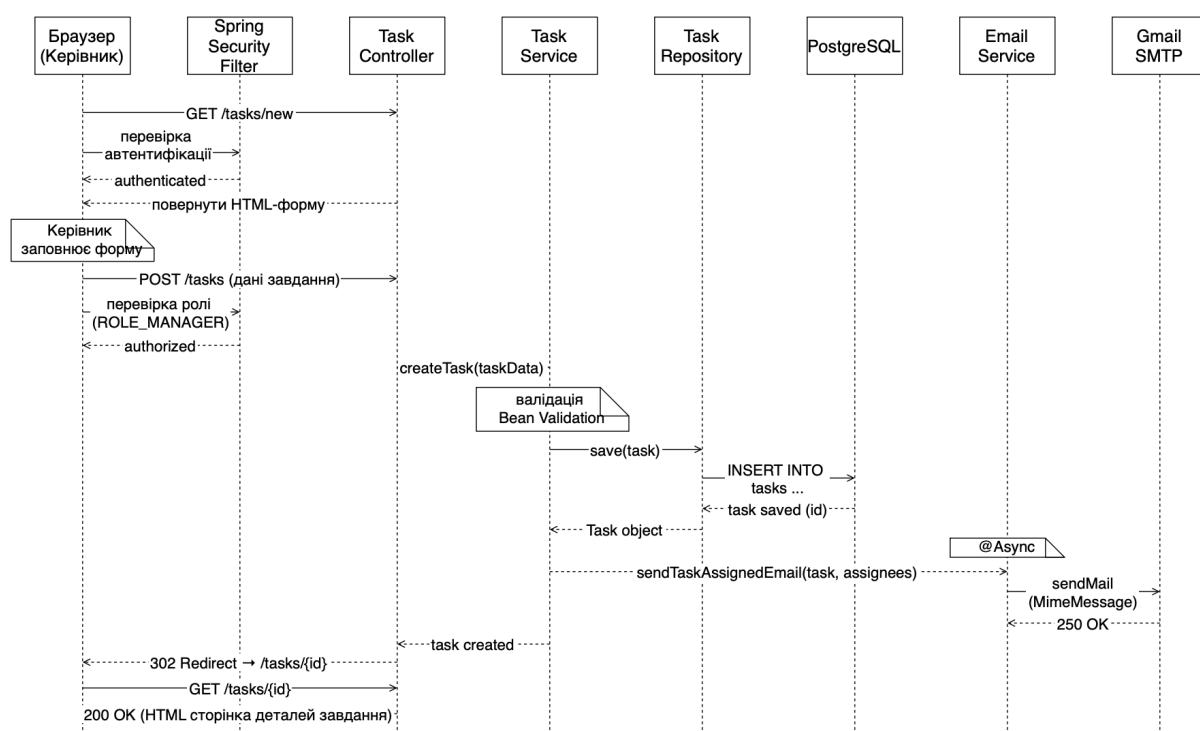


Рис. 2.5 Діаграма послідовності створення завдання керівником

Окрім розглянутих типів діаграм, у проєктуванні системи використано також компонентну діаграму (Component Diagram), що відображає організацію застосунку на рівні великих структурних блоків – модулів, пакетів та зовнішніх систем. У контексті розроблюваного сервісу основними компонентами є модуль контролерів (controllers), модуль сервісів (services), модуль репозиторіїв (repositories), модуль сутностей (entity), модуль конфігурації (config), модуль безпеки (security), модуль роботи з електронною поштою (mail) та модуль обробки

файлів (storage). Зазначений поділ відповідає принципам багатошарової архітектури та забезпечує чітке розмежування відповідальностей між компонентами.

Сукупність розглянутих UML-діаграм формує комплексну модель електронного сервісу управління завданнями, що охоплює як статичні аспекти структури, так і динамічну поведінку системи у різних сценаріях. Зазначена модель служить основою для подальшої реалізації системи у вихідному коді, забезпечує спільне розуміння архітектурних рішень між учасниками команди розроблення та полегшує процеси документування й супроводу системи у майбутньому.

2.5 Алгоритм роботи електронного сервісу

Алгоритмічна модель функціонування електронного сервісу описує послідовність кроків, що виконуються системою при опрацюванні запитів користувачів та реалізації основних бізнес-процесів. Формалізація алгоритмів забезпечує однозначне розуміння логіки системи розробниками, дає можливість виявити потенційні проблеми ще на етапі моделювання та слугує основою для написання вихідного коду [13]. У межах цього підрозділу розглядаються основні алгоритми роботи сервісу, що охоплюють автентифікацію, створення та управління завданнями, а також автоматичну розсилку нагадувань.

Загальний алгоритм взаємодії користувача з системою починається з автентифікації – точки входу до сервісу. При завантаженні стартової сторінки перевіряється наявність активної сесії; у разі її відсутності користувач перенаправляється на сторінку входу. Після подання форми входу контролер передає дані компоненту AuthenticationManager Spring Security, який звертається до CustomUserDetailsService для пошуку користувача за email. Введений пароль порівнюється із зашифрованим значенням у базі за алгоритмом BCrypt. При успіху створюється сесія, у контекст безпеки заноситься об'єкт Authentication з роллю користувача, після чого виконується перенаправлення на головну сторінку. У разі

помилки автентифікації відображається повідомлення з пропозицією повторити вхід. Загальну логіку роботи сервісу від точки входу до виконання операцій із завданнями представлено на рис. 2.6.

Алгоритм підтвердження email-адреси реалізується у декілька послідовних етапів. При реєстрації новий обліковий запис створюється з ознакою `emailVerified=false`, генерується унікальний токен верифікації (`verificationToken`) з обмеженим терміном дії (поле `tokenExpiryDate`), що зберігається безпосередньо у записі користувача. Через `EmailService` формується лист з посиланням на ендпоінт верифікації, що містить токен як параметр URL, та надсилається через Gmail SMTP. При переході за посиланням контролер `VerificationController` перевіряє валідність токена (відповідність значенню у базі та термін дії), у разі успіху встановлює `emailVerified=true` та очищує токен. Після завершення процедури користувач перенаправляється на сторінку входу.

Алгоритм створення завдання є одним із центральних бізнес-процесів сервісу. Авторизований керівник переходить на сторінку створення завдання у межах конкретної групи, де відображається форма з полями для введення назви, опису, дат початку та закінчення, терміну виконання, набору виконавців зі списку учасників групи та тегів. Після подання даних виконується серверна валідація через `Bean Validation`. У разі помилок (порожня назва, термін у минулому, відсутній виконавець) користувач повертається на форму з повідомленнями. У разі успіху `TaskService` формує об'єкт `Task`, заповнює його атрибутами з форми, встановлює службові поля (дата створення, статус за замовчуванням з переліку статусів групи, автор як поточний користувач), пов'язує задачу з обраними виконавцями через колекцію `assignedUsers` та тегамі і зберігає завдання через `TaskRepository`. Далі асинхронно викликається `EmailService` для надсилання повідомлення кожному з виконавців, паралельно у таблиці `notifications` створюються внутрішні сповіщення, після чого користувач перенаправляється на сторінку перегляду створеного завдання.

Алгоритм зміни статусу реалізовано двома способами: через інтерактивну канбан-дошку з технологією Drag and Drop та через стандартну форму редагування. У сценарії з канбан-дошкою користувач захоплює картку у поточній колонці та переміщує її у іншу колонку, що відповідає одному зі статусів, визначених для конкретної групи. Подія drop клієнтського JavaScript-коду формує POST-запит до контролера TaskController з ідентифікатором завдання та цільового статусу. На серверній стороні TaskService перевіряє членство поточного користувача у відповідній групі та його роль, оновлює поле status задачі (типу @ManyToOne на Status) і зберігає зміну, фіксуючи її у таблиці task_history. У разі успіху виконується перенаправлення на сторінку дошки з оновленим розташуванням картки. У разі помилки картка повертається у вихідну колонку з відображенням повідомлення про причину відмови.

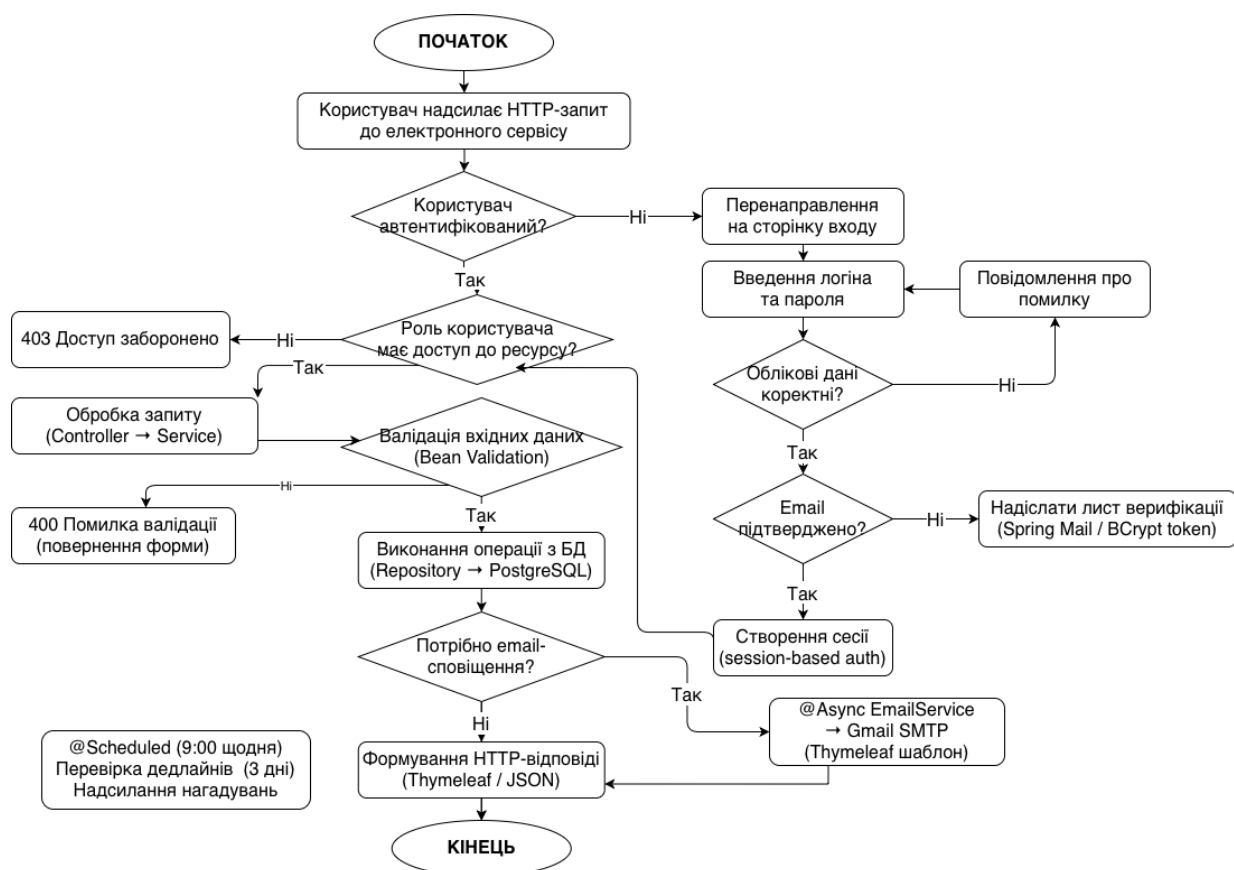


Рис. 2.6 Блок-схема алгоритму роботи електронного сервісу

Алгоритм автоматичної розсилки нагадувань реалізується через планувальник Spring Scheduler. Метод `sendTaskDeadlineReminders()` класу `TaskReminderScheduler`, розмічений анотацією `@Scheduled` з cron-виразом `${task.notification.cron:0 0 9 * * ?}`, викликається щоденно о 9:00 ранку. На кожному виклику формується JPQL-запит до `TaskRepository` через метод `findTasksWithUpcomingDeadline(today, reminderDate)`, що відбирає завдання з терміном виконання у діапазоні від поточної дати до поточної плюс три дні. Для кожного знайденого завдання формується персоналізоване повідомлення з зазначенням назви, терміну та посиланням на сторінку завдання, що надсилається кожному з виконавців через `EmailService`. Після успішної розсилки створюються відповідні записи у таблиці `notifications` для відображення внутрішніх сповіщень користувачам. Для прострочених завдань (термін яких уже минув) система надсилає окремі повідомлення з підвищеним рівнем терміновості як виконавцям, так і власнику групи.

Алгоритм прикріплення файлу включає декілька кроків. Користувач відкриває деталі завдання, обирає опцію прикріплення та через стандартний діалог вибору обирає файл з локальної системи. Файл передається на сервер через `Multipart file upload` із обмеженням розміру 10 МБ. `TaskController` отримує запит, перевіряє права користувача та делегує обробку класу `FileStorageService`. Сервіс валідує тип і розмір, генерує унікальне ім'я для збереження на основі `UUID` для уникнення колізій та копіює файл у виділеній директорії сервера, шлях до якої визначається параметром `file.upload-dir` у `application.properties`. Метадані прикріплення (`originalName`, `storedName`, `mimeType`, `size`) формуються у вигляді DTO-класу `AttachmentMetadata`, серіалізуються у `JSON` та зберігаються безпосередньо у текстовому полі `attachments` сутності `Task`. Після успішного завантаження користувач переходить на оновлену сторінку деталей завдання з повним переліком прикріплень.

Алгоритм формування аналітичних звітів передбачає послідовну обробку даних з декількох джерел. `ReportsController` отримує параметри періоду (початкова

та кінцева дати) та ідентифікатор групи, формує JPQL-запити до TaskRepository, що агрегують дані за статусами, тегами, виконавцями та термінами. Для кожного виконавця обчислюються показники: загальна кількість призначених завдань, кількість виконаних, кількість прострочених, середній час виконання. Дані передаються у Thymeleaf-шаблон reports.html для рендерингу. Звіт містить як числові показники у табличному вигляді, так і їх візуалізацію через діаграми, що генеруються клієнтським кодом.

Обробка винятків реалізована через глобальний обробник @ControllerAdvice. Винятки EntityNotFoundException призводять до сторінки 404, AccessDeniedException – до 403, ValidationException – до повернення на форму з повідомленнями про конкретні поля. Системні винятки (проблеми з'єднання з базою) логуються та відображаються користувачу як загальна помилка 500 без розкриття технічних деталей з міркувань безпеки.

Описані алгоритми у сукупності забезпечують функціонування електронного сервісу як цілісної системи, де кожен компонент виконує чітко визначену роль у обробці запитів користувачів. Формалізація логіки роботи на етапі проєктування дала можливість мінімізувати ризики архітектурних помилок при переході до програмної реалізації, що детально розглядається у наступному розділі роботи.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальна структура програмного застосунку

Загальна структура програмного застосунку є відображенням архітектурних рішень, прийнятих на етапі проектування, у конкретній організації вихідного коду, конфігураційних файлів та допоміжних ресурсів. Грамотна організація структури проєкту забезпечує зручність навігації для розробників, спрощує процес супроводу та модифікації коду, а також відповідає загальноприйнятим у галузі стандартам та конвенціям. У роботі структура електронного сервісу організована відповідно до рекомендованих практик розроблення Spring Boot застосунків та принципів багатошарової архітектури [25].

На верхньому рівні проєкт організовано як одиничний модуль Gradle, що містить усі компоненти серверної частини. Файл `build.gradle` є центральним конфігураційним файлом та визначає мову програмування, версії залежностей, плагіни та параметри збирання. У ньому декларуються залежності від Spring Boot стартерів (`spring-boot-starter-web`, `spring-boot-starter-data-jpa`, `spring-boot-starter-security`, `spring-boot-starter-mail`, `spring-boot-starter-thymeleaf`, `spring-boot-starter-validation`), драйвера PostgreSQL та тестових бібліотек. Плагін `io.spring.dependency-management` забезпечує автоматичне узгодження версій модулів Spring між собою. Файли `settings.gradle` та `gradle.properties` містять відповідно ім'я проєкту та глобальні параметри Gradle (версія JVM, параметри пам'яті). Кореневий каталог також містить `README.md` з описом проєкту та `.gitignore` для виключення з системи контролю версій тимчасових файлів.

Каталог `src` є контейнером для всього вихідного коду та поділяється на `main` для основного коду застосунку та `test` для коду тестів. Така структура є стандартною для проєктів Maven та Gradle і забезпечує чітке розмежування

продакшен-коду від тестового. Кожен підкаталог додатково поділяється на java для Java-коду та resources для ресурсних файлів.

Кореневий пакет Java-коду має назву `com.example.taskmanager`. Структура пакетів відповідає принципу пакування за функціональністю (*package by feature*) у поєднанні з пакуванням за шарами (*package by layer*), що забезпечує оптимальний баланс між групуванням пов'язаних компонентів та розмежуванням архітектурних шарів. Основні пакети застосунку: `config` (конфігураційні класи Spring), `controller` (контролери Spring MVC та REST), `service` (сервіси з бізнес-логікою, включно з `EmailService` та `TaskReminderScheduler`), `repository` (інтерфейси Spring Data JPA), `entity` (класи сутностей JPA), `security` (компоненти безпеки), `dto` (об'єкти передачі даних – у поточній версії містить лише `AttachmentMetadata` для метаданих файлів), `exception` (користувацькі винятки) та `util` (допоміжні класи). Загальну схему структури каталогів та компонентів застосунку представлено на рис. 3.1.

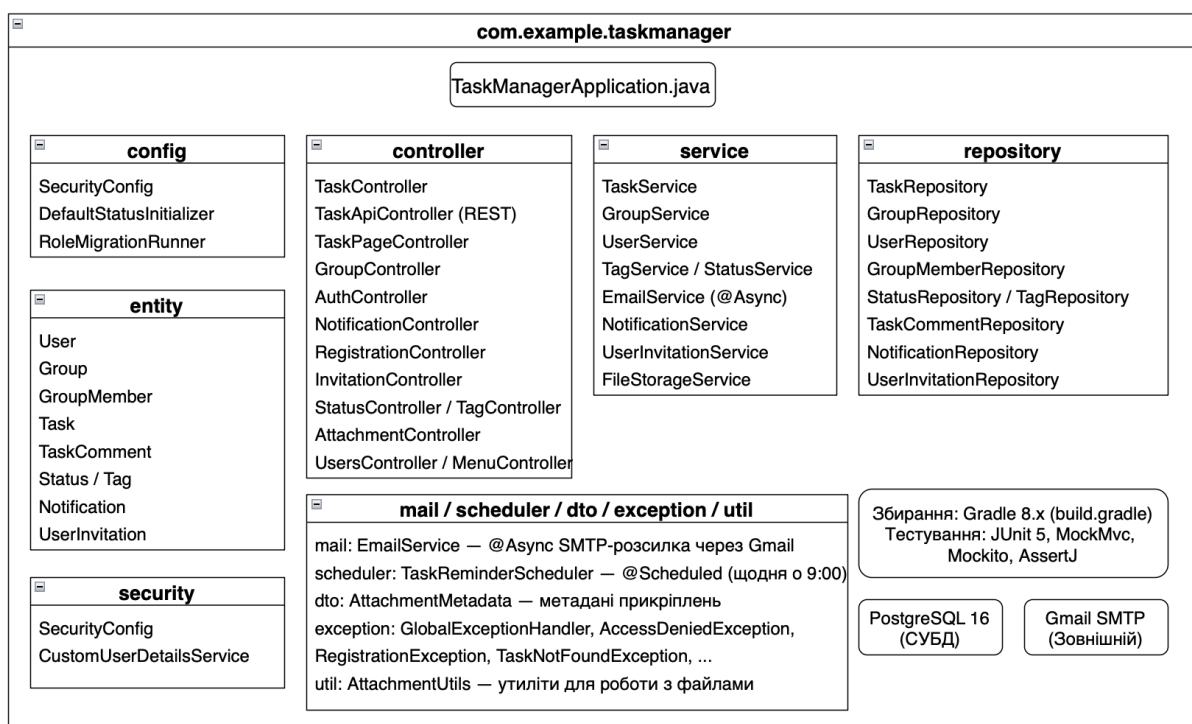


Рис. 3.1 Структура програмного застосунку

Пакет `config` містить конфігураційні класи з анотацією `@Configuration`: `SecurityConfig` (Spring Security з правилами доступу та механізмом автентифікації), `WebConfig` (Spring MVC, обробники ресурсів, інтерсептори), `SchedulerConfig` (активація через `@EnableScheduling`), `MailConfig` (`JavaMailSender` для Gmail SMTP). Пакет `controller` об'єднує контролери, що поділяються на класичні MVC-контролери (повертають назви Thymeleaf-шаблонів) та REST-контролери (повертають JSON-відповіді). Основні контролери застосунку: `AuthController`, `TaskController`, `TaskApiController`, `TaskPageController`, `GroupController`, `UserController`, `AdminController`, `ReportsController`, `StatusController`, `TagController`, `NotificationController`, `InvitationController`, `VerificationController`, `ProfileController` та `MenuController`.

Пакет `service` містить сервісні класи з анотацією `@Service`, що інкапсулюють бізнес-логіку: `UserService`, `TaskService` (операції з завданнями та зміна статусів), `GroupService`, `FileStorageService` (робота з файлами на диску), `EmailService` (з шаблонами Thymeleaf), `NotificationService` (внутрішні сповіщення), `StatusService` (динамічні статуси груп), `TagService` (теги задач) та `UserInvitationService` (запрошення нових учасників до групи). Сервіси взаємодіють з репозиторіями для роботи з даними та можуть викликати інші сервіси для реалізації складних операцій. Пакет `repository` містить інтерфейси Spring Data JPA, що успадковують `JpaRepository` та надають готові методи для CRUD-операцій; додаткові методи пошуку оголошуються через назви методів за конвенцією Spring Data або через анотацію `@Query` з JPQL-запитами.

Пакет `entity` об'єднує класи сутностей з анотацією `@Entity` та анотаціями JPA для визначення таблиці, колонок, зв'язків та обмежень: `User`, `Group`, `GroupMember`, `Task`, `SubTask`, `Status`, `Tag`, `Comment`, `Notification`, `UserInvitation`, `TaskHistory`. Класи цього пакету не містять складної бізнес-логіки, окрім простих геттерів, сеттерів, методів `equals/hashCode` та допоміжних методів-помічників для управління колекціями. Пакет `dto` у поточній версії містить лише клас `AttachmentMetadata`, що описує метадані прикріпленого файлу (`originalName`, `storedName`, `mimeType`, `size`)

та серіалізується у JSON-рядок поля attachments сутності Task. Усі інші взаємодії контролер–клієнт здійснюються безпосередньо через сутності, чутливі поля яких приховуються за допомогою анотацій `@JsonIgnore` та `@JsonIgnoreProperties`.

Пакет security містить компоненти автентифікації та авторизації, основним з яких є `CustomUserDetailsService` – він виконує пошук користувачів за email або логіном через `UserRepository` та перетворює рядкове значення поля role сутності User на об'єкт `GrantedAuthority`. Робота з електронною поштою винесена у клас `EmailService`, що використовує бін `JavaMailSender`, налаштований у класі `MailConfig` пакета config, та шаблонізатор `Thymeleaf` для формування HTML-вмісту листів. Планування періодичних задач забезпечується класом `TaskReminderScheduler`, метод якого розмічено анотацією `@Scheduled` і виконує щоденну розсилку email-нагадувань про завдання з наближенням терміну виконання.

Каталог `src/main/resources` містить ресурсні файли застосунку. Файл `application.properties` є центральним конфігураційним файлом, де визначаються параметри підключення до бази даних, налаштування JPA та Hibernate, параметри SMTP-сервера, рівні логування, обмеження для завантаження файлів та параметри планувальника (наприклад, `task.notification.cron`). Конфіденційні значення зчитуються зі змінних середовища через синтаксис `${VARIABLE_NAME}`: `DB_URL`, `DB_USERNAME`, `DB_PASSWORD` для підключення до бази даних, `EMAIL_USERNAME` та `EMAIL_PASSWORD` для SMTP, `APP_URL` для формування абсолютних посилань у листах, `FILE_UPLOAD_DIR` для каталогу збереження файлів. У поточній версії використовується єдиний конфігураційний файл для всіх середовищ; розрізнення між середовищами розробки та виробництва досягається через значення змінних середовища.

Підкаталог `templates` містить `Thymeleaf`-шаблони у плоскій структурі (без вкладеної ієрархії за функціональними областями): `login.html` та `register.html` (автентифікація та реєстрація), `welcome.html` (стартова сторінка), `groups.html` та `editGroup.html` (перегляд та редагування груп), `tasks.html`, `tasksTable.html` і

editTask.html (канбан-дошка, табличне подання та редагування завдань), statuses.html і editStatus.html (управління статусами групи), groupTags.html (управління тегами групи), invite.html (запрошення нових учасників), notifications.html (внутрішні сповіщення), profile.html (особистий профіль), users.html та admin.html (адміністративний інтерфейс), reports.html (аналітичні звіти), menu.html (загальне навігаційне меню). Підкаталог fragments/ містить два спільні фрагменти, що повторно використовуються через th:fragment: head.html (заголовок документа з підключенням стилів та скриптів) і themeControls.html (елементи перемикання теми оформлення). Підкаталог static містить статичні ресурси: CSS-файли, JavaScript-файли, зображення та бібліотеки сторонніх постачальників (Bootstrap 5.3.3, Bootstrap Icons 1.11.3). Власний JavaScript-код розділено на два тематичні файли: dragAndDrop.js (логіка канбан-дошки з Drag and Drop) та theme.js (перемикання теми оформлення).

Каталог src/test дзеркально повторює структуру src/main та містить тестові класи. Тести поділяються на модульні (окремі класи сервісів, утиліт), інтеграційні (взаємодія компонентів через Spring Test Context) та тести шару контролерів (з використанням MockMvc). Тестові ресурси включають окрему конфігурацію (application-test.properties) з підключенням до тестової бази даних в оперативній пам'яті.

Зазначена структура забезпечує чіткий розподіл відповідальностей між компонентами, відповідає прийнятим у галузі стандартам та надає основу для подальшого опису реалізації серверної частини, REST API та користувацького інтерфейсу у наступних підрозділах.

3.2 Реалізація серверної частини на платформі java та spring boot

Реалізація серверної частини електронного сервісу здійснюється на платформі Java 24 з використанням фреймворку Spring Boot 3.5.0 відповідно до архітектурних рішень, прийнятих у попередньому розділі. Серверна частина

забезпечує усю бізнес-логіку системи, обробку запитів користувачів, взаємодію з PostgreSQL та інтеграцію із зовнішніми сервісами на кшталт Gmail SMTP. У межах цього підрозділу описуються ключові аспекти реалізації: налаштування основного класу застосунку, конфігурація безпеки, реалізація доменних сутностей, сервісного шару, репозиторіїв та допоміжних компонентів [25].

Точкою входу до застосунку є основний клас `TaskManagerApplication`, розмічений анотацією `@SpringBootApplication`, що поєднує `@Configuration`, `@EnableAutoConfiguration` та `@ComponentScan`. Метод `main()` викликає `SpringApplication.run()`, який запускає вбудований Tomcat та ініціалізує контекст Spring. Окрім цього, основний клас розмічається `@EnableScheduling` для активації планувальника. Файл `application.properties` містить параметри підключення до бази даних, налаштування JPA, SMTP-сервера та інші конфігурації, причому конфіденційні значення зчитуються зі змінних середовища через синтаксис `${DB_URL}`, `${DB_USERNAME}`, `${DB_PASSWORD}`, `${EMAIL_USERNAME}`, `${EMAIL_PASSWORD}`, `${APP_URL}` та `${FILE_UPLOAD_DIR}`.

Конфігурація безпеки реалізована у класі `SecurityConfig` з анотацією `@Configuration`. Клас визначає бін `SecurityFilterChain` з правилами доступу: загальнодоступні шляхи (`/`, `/login`, `/register`, `/verify`, статичні ресурси у `/css/**` та `/js/**`) дозволені для всіх; усі інші шляхи вимагають автентифікації, а конкретні обмеження за ролями застосовуються на рівні методів сервісів та контролерів. Налаштовано форму входу з параметрами полів та обробник виходу із завершенням сесії. Захист від CSRF активовано для звичайних запитів і вимкнено лише для шляхів `/api/**`, оскільки REST-ендпоінти консумуються власним клієнтським JavaScript-кодом у тому ж домені. Шифрування паролів реалізовано через бін `BCryptPasswordEncoder` з 10 раундами хешування (значення за замовчуванням, що забезпечує баланс між обчислювальною складністю та продуктивністю входу). Доступ до бази користувачів забезпечує клас `CustomUserDetailsService`, що імплементує інтерфейс `UserDetailsService` та виконує пошук за email або логіном

через `UserRepository`, після чого перетворює рядкове значення поля `role` сутності `User` на список об'єктів `GrantedAuthority`.

Доменна модель реалізована через класи сутностей JPA з анотаціями Hibernate. Клас `User` має анотації `@Entity` і `@Table(name = "users")`; поле `id` типу `Long` розмічено `@Id` та `@GeneratedValue(strategy = GenerationType.IDENTITY)` для автоматичної генерації через автоінкремент PostgreSQL; поля `email` та `username` мають обмеження унікальності через `@Column(unique = true, nullable = false)`; чутливі поля `password` та `verificationToken` приховуються при серіалізації за допомогою `@JsonIgnore`. Поле `role` оголошено як простий тип `String`, що зберігає одне зі значень `ROLE_ADMIN`, `ROLE_MANAGER` або `ROLE_USER`; такий підхід усуває потребу в окремих сутностях ролей та проміжних таблицях. Для контролю стану підтвердження `email` використовуються поля `emailVerified` (`boolean`), `verificationToken` (`String`) та `tokenExpiryDate` (`LocalDateTime`). Часова мітка `createdAt` керується автоматично через `@CreationTimestamp`.

Клас `Task` є центральною сутністю та містить декілька зв'язків з іншими класами. Поля `title`, `description`, `startDate`, `endDate`, `dueDate`, `deferred` (`boolean`) та `attachments` (`TEXT` – серіалізований JSON-рядок з масивом метаданих файлів) описують властивості завдання. Зв'язки `@ManyToOne` з класами `Group`, `User` (для автора) та `Status` реалізовано з ледачою стратегією завантаження (`FetchType.LAZY`) для оптимізації продуктивності. Зв'язок з виконавцями реалізовано як `@ManyToMany` через проміжну таблицю `task_assignees` із полями `task_id` та `user_id`, що дозволяє призначати декілька виконавців на одне завдання та зберігається у полі `assignedUsers` типу `Set<User>`. Аналогічно реалізовано зв'язок з тегами (`@ManyToMany` через `task_tags`). Колекції `subtasks`, `comments` та `history` оголошені через `@OneToMany` з `cascade = CascadeType.ALL` та `orphanRemoval = true`, що забезпечує каскадне видалення пов'язаних записів. Підзавдання представлено окремою сутністю `SubTask` (`id`, `title`, `done`, `createdAt`, `task_id`), що зберігає чек-листи у межах конкретного завдання.

Шар доступу до даних реалізовано через інтерфейси Spring Data JPA, що успадковують `JpaRepository<Entity, Long>`. Інтерфейс `TaskRepository` містить методи пошуку, оголошені через конвенцію іменування Spring Data, а також складніші запити через `@Query` з JPQL-синтаксисом. Зокрема, метод `findTasksWithUpcomingDeadline(LocalDate today, LocalDate reminderDate)` повертає список завдань, термін виконання яких потрапляє у діапазон між поточною датою та граничною датою нагадування (за замовчуванням +3 дні), не позначених як відкладені та статус яких не є фінальним. Параметри прив'язуються через `@Param`. Для пагінації використовується тип `Pageable` з повертанням `Page<Task>`, що автоматично формує SQL із LIMIT та OFFSET.

Сервісний шар реалізовано через класи з анотацією `@Service`. Клас `TaskService` містить методи створення, оновлення, видалення та пошуку завдань, а також методи зміни статусу та призначення виконавців. Транзакційність забезпечується анотацією `@Transactional`, причому методи читання розмічуються як `@Transactional(readOnly = true)` для оптимізації роботи Hibernate. Метод створення завдання приймає атрибути з форми, формує сутність `Task`, встановлює службові поля (автор з `SecurityContextHolder`, статус як перший статус групи через `StatusService`, час створення), асоціює виконавців з-поміж учасників групи, зберігає сутність та викликає `NotificationService.notifyTaskAssigned()` для створення внутрішніх сповіщень, а також `EmailService` для асинхронного надсилання повідомлень електронною поштою. Усі зміни фіксуються у `task_history` через `TaskHistoryService`. У разі помилок генеруються користувацькі винятки (`TaskValidationException`, `EntityNotFoundException`, `AccessDeniedException`), які перехоплюються глобальним обробником.

Компонент розсилки електронних повідомлень `EmailService` використовує бін `JavaMailSender`, налаштований у класі `MailConfig` із параметрами Gmail SMTP (хост `smtp.gmail.com`, порт `587`, протокол `TLS`, автентифікація через ім'я користувача та пароль застосунку). Вміст листів формується через шаблонізатор `Thymeleaf` із використанням класу `TemplateEngine`: шаблон завантажується з

каталогу `templates`, у нього передаються параметри (ім'я виконавця, назва завдання, термін, посилання), після чого формується HTML-вміст. Лист створюється через `MimeMessageHelper` з підтримкою HTML, кодуванням UTF-8 та зазначенням теми, відправника, отримувача й тіла повідомлення.

Планувальник автоматичних задач реалізовано у класі `TaskReminderScheduler` з анотацією `@Component`. Метод `sendTaskDeadlineReminders()` розмічено анотацією `@Scheduled` із `cron`-виразом, що зчитується з властивості `${task.notification.cron:0 0 9 * * ?}` (за замовчуванням – щоденно о 9:00). Метод викликає `taskRepository.findTasksWithUpcomingDeadline(today, reminderDate)` для отримання завдань з терміном виконання у діапазоні від поточної дати до +3 днів та для кожного знайденого завдання й кожного його виконавця викликає `emailService.sendReminderEmail()`. Зчитування `cron`-виразу з `properties` дає змогу гнучко змінювати розклад розсилки без перекомпіляції коду.

Обробка завантаження файлів реалізована у класі `FileStorageService`. Метод збереження приймає об'єкт `MultipartFile` та ідентифікатор завдання, перевіряє розмір файлу (обмеження 10 МБ задається через параметри `spring.servlet.multipart.max-file-size` та `spring.servlet.multipart.max-request-size`), валідує тип файлу за списком дозволених MIME-типів, генерує унікальне ім'я через `UUID.randomUUID().toString()` зі збереженням оригінального розширення. Файл зберігається у каталозі, шлях до якого задається параметром `file.upload-dir` (передається через змінну середовища `FILE_UPLOAD_DIR`), через `Files.copy(file.getInputStream(), targetPath)`. Метаданні файлу формуються як об'єкт `AttachmentMetadata` (`originalName`, `storedName`, `mimeType`, `size`) та додаються до серіалізованого JSON-рядка у полі `attachments` сутності `Task`. Зворотна операція видачі файлу реалізована через повернення `ResponseEntity<Resource>` із заголовком `Content-Disposition`.

Глобальна обробка винятків реалізована у класі `GlobalExceptionHandler` з анотацією `@ControllerAdvice`. Метод обробки `EntityNotFoundException` повертає

сторінку 404, `AccessDeniedException` – сторінку 403, `MethodArgumentNotValidException` (помилки Bean Validation) – повернення на форму введення з повідомленнями про конкретні поля. Загальний обробник `Exception` логує деталі через `Logger` та повертає сторінку 500 без розкриття технічних деталей з міркувань безпеки.

Транзакційне управління реалізовано на рівні сервісів через `@Transactional`. `Spring Boot` за замовчуванням використовує JPA-транзакції через `JpaTransactionManager`. Поведінка налаштована за принципом `Propagation.REQUIRED` – виконання у наявній транзакції або створення нової. Атрибут `readOnly = true` для методів читання дає `Hibernate` можливість оптимізувати запити (зокрема відключити `dirty checking`). Методи з декількома операціями (створення завдання з прикріпленнями) розмічені на рівні всього методу, що забезпечує атомарність – у разі помилки усі зміни відкочуються.

Безпечне зберігання конфіденційних параметрів забезпечується через змінні середовища, що зчитуються `Spring Boot` із системного оточення під час запуску. Це стосується параметрів підключення до бази даних (`DB_URL`, `DB_USERNAME`, `DB_PASSWORD`), реквізитів SMTP-сервера (`EMAIL_USERNAME`, `EMAIL_PASSWORD`), базової URL застосунку (`APP_URL`, що використовується для формування посилань у листах) та каталогу для зберігання прикріплених файлів (`FILE_UPLOAD_DIR`). Зазначений підхід відповідає принципу зовнішньої конфігурації методології `Twelve-Factor App` та виключає присутність конфіденційних даних у вихідному коді або системі контролю версій. Логування подій реалізовано через стандартний `SLF4J` з реалізацією `Logback`: кожен клас сервісу та контролера містить статичне поле `Logger`, через яке здійснюється запис подій рівнів `info` (успішні операції), `warn` (нестандартні ситуації, як спроба доступу без прав) та `error` (винятки, збої з'єднання). Рівні логування для різних пакетів задаються в `application.properties`, що дає можливість гнучко керувати деталізацією у різних середовищах.

Реалізація серверної частини забезпечує повноцінне функціонування усіх компонентів системи відповідно до сформульованих вимог. Використання сучасних можливостей Java 24 та Spring Boot 3.5.0 у поєднанні з принципами багат шарової архітектури, інверсії залежностей та декларативного програмування створює основу для надійної, масштабованої та зручної у супроводі системи. Конкретні фрагменти вихідного коду ключових компонентів наводяться у Додатку А. У наступному підрозділі розглядається реалізація REST API сервісу для асинхронної взаємодії клієнтського коду з серверною частиною.

3.3 Реалізація rest api сервісу

REST API (Representational State Transfer Application Programming Interface) є одним із ключових компонентів сучасних веб-застосунків, що забезпечує програмний інтерфейс для взаємодії клієнтського коду з серверною частиною через стандартні HTTP-протоколи. У контексті електронного сервісу управління завданнями REST API реалізує асинхронну взаємодію між клієнтським JavaScript-кодом та серверним застосунком, що дає можливість виконувати операції без перезавантаження сторінки. Зазначений підхід забезпечує сучасний користувацький досвід, особливо при роботі з канбан-дошкою через технологію Drag and Drop, де переміщення завдань між колонками вимагає негайного оновлення стану на серверній стороні [15].

Архітектурний стиль REST, описаний Роєм Філдіном у 2000 році, базується на низці принципів, дотримання яких забезпечує предсказуваність, масштабованість та зручність використання інтерфейсу. До основних принципів REST належать клієнт-серверна архітектура (чіткий розподіл відповідальностей між клієнтом та сервером), безстановість (кожен запит містить усю необхідну інформацію для його обробки), кешованість (відповіді можуть кешуватися для підвищення продуктивності), єдиний інтерфейс (стандартизовані механізми взаємодії з ресурсами), багаторівнева архітектура (наявність проміжних шарів між

клієнтом і сервером) та використання стандартних HTTP-методів для виконання операцій над ресурсами [8]. У роботі реалізовано REST API, що дотримується зазначених принципів та забезпечує повноцінний інтерфейс для роботи з основними сутностями системи.

Базовою концепцією REST API є подання даних у вигляді ресурсів, доступ до яких здійснюється через унікальні URI (Uniform Resource Identifier). Кожен ресурс електронного сервісу має чітко визначений URI, що відображає його логічне положення у структурі системи. Ресурс «завдання» доступний за URI `/api/tasks`, ресурс «коментар у межах завдання» – за `/api/tasks/{taskId}/comments`, ресурс «підзавдання» – за `/api/tasks/{taskId}/subtasks`, ресурс «історія змін» – за `/api/tasks/{taskId}/history`. Така структура URI відображає ієрархічні відносини між ресурсами та забезпечує інтуїтивно зрозумілий інтерфейс для розробників.

Операції над ресурсами виконуються через стандартні HTTP-методи, кожен з яких має чітко визначену семантику. Метод GET використовується для отримання інформації про ресурс і не повинен спричиняти змін на сервері (властивість безпечності). Метод POST застосовується для створення нового ресурсу, причому ідентифікатор створеного ресурсу зазвичай повертається у відповіді. Метод PUT використовується для повного оновлення існуючого ресурсу – клієнт надсилає повний набір атрибутів ресурсу, які заміщують поточні значення. Метод PATCH призначений для часткового оновлення ресурсу, коли клієнт надсилає лише ті поля, що підлягають зміні. Метод DELETE використовується для видалення ресурсу. Операції GET, PUT та DELETE мають властивість ідемпотентності – повторне виконання операції не змінює результату, що є важливою характеристикою для побудови надійних розподілених систем.

У розроблюваному електронному сервісі реалізовано декілька груп REST-ендпоінтів, що забезпечують основні функціональні можливості. Група ендпоінтів для роботи з завданнями реалізована у класі `TaskApiController`, розміченому анотаціями `@RestController` та `@RequestMapping("/api/tasks")`. Анотація `@RestController` поєднує `@Controller` та `@ResponseBody`, що автоматично

перетворює об'єкти, що повертаються методами, у JSON-відповіді через бібліотеку Jackson, інтегровану зі Spring Boot. Анотація `@RequestMapping` визначає базовий URI для всіх методів контролера. Перелік основних ендпоінтів API для роботи з завданнями представлений у таблиці 3.1 (Додаток В).

Реалізація методу отримання деталей завдання `GET /api/tasks/{id}` демонструє типовий підхід до побудови REST-ендпоінтів читання даних. Метод розмічений анотацією `@GetMapping("/{id}")` та приймає ідентифікатор через `@PathVariable`. Метод викликає `taskService.findById(id)`, отримує сутність `Task` та повертає її як JSON-відповідь. Чутливі або зайві поля приховуються за допомогою анотацій `@JsonIgnore` та `@JsonIgnoreProperties` безпосередньо на класах сутностей, що дає змогу повертати сутності клієнту без проміжного перетворення в DTO. Такий підхід зменшує кількість допоміжних класів та спрощує підтримку коду.

Загальний підхід до обробки операцій модифікації даних у REST-контролерах ілюструється на прикладі додавання коментаря. Метод розмічений анотацією `@PostMapping` та приймає ідентифікатор завдання через `@PathVariable`, а вміст коментаря – через `@RequestBody`. Анотація `@Valid` активує механізм Bean Validation, що перевіряє відповідність вхідних даних декларативним обмеженням (`@NotBlank`, `@Size`). У разі помилок валідації Spring автоматично повертає відповідь з кодом 400 Bad Request та переліком помилок. Після успішної валідації метод викликає `taskService` та повертає створений ресурс у тілі відповіді з кодом 201 Created. Створення самих завдань (а не їх допоміжних сутностей) винесено у MVC-контролер `TaskController`, оскільки сторінки створення та редагування використовують серверний рендеринг Thymeleaf із форм-постом.

Робота з коментарями реалізована через групу ендпоінтів у межах батьківського ресурсу-завдання. Метод `GET /api/tasks/{id}/comments` повертає всі коментарі вказаного завдання, метод `POST /api/tasks/{id}/comments` приймає тіло запиту із текстом коментаря та створює новий запис. Особливий випадок – методи редагування та видалення коментаря (`PUT` та `DELETE /api/tasks/{taskId}/comments/{commentId}`). Перед виконанням операції `TaskService`

перевіряє, що поточний користувач є автором коментаря і що з моменту створення минуло не більше 15 хвилин. Якщо хоча б одна з умов не виконується, повертається відповідь з кодом 403 Forbidden. Зазначене обмеження забезпечує баланс між можливістю виправити одруківку та збереженням історії обговорень.

Робота з підзавданнями реалізована через групу ендпоінтів `/api/tasks/{id}/subtasks`: метод GET повертає список підзавдань, POST створює нове підзавдання, PUT `/api/tasks/{taskId}/subtasks/{subId}/toggle` перемикає прапорець виконання підзавдання, DELETE видаляє підзавдання. Аналогічно реалізовано видалення прикріплень: DELETE `/api/tasks/{taskId}/attachments/{fileName}` видаляє файл з диска через `FileStorageService` та оновлює серіалізований JSON-рядок поля `attachments` сутності `Task`. Ендпоінт GET `/api/tasks/{id}/history` повертає журнал змін завдання у хронологічному порядку, що використовується клієнтським кодом для відображення вкладки «Історія змін» на сторінці деталей завдання.

Зміна статусу завдання з канбан-дошки реалізована не через REST API, а через MVC-контролер `TaskController`. Це обумовлено тим, що дошка та сторінки завдань використовують серверний рендеринг `Thymeleaf` і CSRF-токени стандартних форм `Spring Security`, тому для зміни статусу не потрібен окремий REST-ендпоінт. Клієнтський JavaScript-код надсилає POST-запит на відповідний MVC-маршрут із CSRF-токеном; сервер виконує перевірку прав у `TaskService`, оновлює статус завдання та повертає або редирект, або фрагмент шаблону для перебудови потрібної ділянки дошки.

Серіалізація та десеріалізація JSON-даних здійснюється бібліотекою `Jackson`, що інтегрована зі `Spring Boot` та налаштована за замовчуванням для роботи з REST-контролерами. Для коректного відображення складних типів використовуються анотації `Jackson` на полях DTO-класів. Анотація `@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss")` забезпечує стандартизоване представлення часових міток у форматі ISO 8601. Анотація `@JsonIgnore` приховує внутрішні поля від клієнта, а `@JsonProperty("custom_name")` дає можливість змінити ім'я поля при серіалізації для відповідності конвенціям

JSON (snake_case замість camelCase, якщо потрібно). Для уникнення нескінченної рекурсії при серіалізації пов'язаних сутностей використовуються анотації `@JsonManagedReference` та `@JsonBackReference` або, альтернативно, повне відокремлення DTO від сутностей.

Обробка помилок у REST API реалізована через глобальний обробник винятків, розширений для роботи з REST-контролерами. Окремий клас `RestExceptionHandler`, розмічений `@RestControllerAdvice`, перехоплює винятки та повертає структуровані JSON-відповіді з кодами помилок та повідомленнями. Для забезпечення єдиного формату відповідей про помилки використовується клас `ErrorResponse`, що містить поля `timestamp` (час виникнення помилки), `status` (HTTP-код), `error` (категорія помилки), `message` (опис), `path` (URI запиту) та опціональне поле `errors` (перелік помилок валідації). Така структура дає клієнтському коду можливість програмно обробляти різні типи помилок та відображати користувачу інформативні повідомлення. Стандартні HTTP-коди використовуються відповідно до семантики помилок: 400 Bad Request для помилок валідації, 401 Unauthorized для відсутності автентифікації, 403 Forbidden для відмови у доступі, 404 Not Found для відсутності ресурсу, 409 Conflict для конфліктів даних (наприклад, дублювання email при реєстрації), 500 Internal Server Error для непередбачених помилок сервера.

Безпека REST API забезпечується через інтеграцію зі Spring Security. Усі ендпоінти `/api/**`, окрім явно зазначених у конфігурації як публічні, вимагають автентифікації. Для реалізації автентифікації у REST API можуть використовуватися різні підходи: сесійна автентифікація через cookie-файли, що є простішою у реалізації, але потребує підтримки cookies клієнтом, або токенна автентифікація через JWT (JSON Web Tokens), що краще підходить для розподілених систем та мобільних застосунків. У розроблюваному електронному сервісі використовується сесійний підхід, оскільки клієнтська частина працює у тому ж домені, що й серверна, та підтримує cookie-файли. Захист від CSRF-атак забезпечується автоматично через Spring Security, причому для AJAX-запитів

CSRF-токен передається у заголовок X-XSRF-TOKEN, що отримується з cookie-файла, встановленого сервером.

Документування REST API є важливою практикою, що забезпечує зручність використання інтерфейсу розробниками. У розроблюваному сервісі для документування використовується специфікація OpenAPI 3.0 (раніше відома як Swagger), реалізована через бібліотеку SpringDoc OpenAPI. Підключення стартера springdoc-openapi-starter-webmvc-ui автоматично генерує документацію API на основі анотацій контролерів та DTO-класів. Документація доступна за URI /swagger-ui.html та містить опис усіх ендпоінтів, очікуваних параметрів, форматів відповідей та можливих помилок. Інтерактивний інтерфейс Swagger UI дає можливість безпосередньо тестувати ендпоінти з браузера, що значно спрощує процес розроблення та налагодження клієнтського коду.

Версіонування API є важливим аспектом проєктування, що забезпечує зворотну сумісність при подальшому розвитку системи. У розроблюваному сервісі прийнято підхід версіонування через URI з префіксом /api/v1/, що дає можливість у майбутньому додавати нові версії API (/api/v2/) без порушення роботи існуючих клієнтів. Альтернативними підходами є версіонування через заголовки HTTP (Ацепт: application/vnd.taskmanager.v1+json) або параметри запиту (?version=1), однак URI-підхід вважається найбільш зручним та явним з точки зору розробників-споживачів API.

Окремої уваги заслуговує реалізація обмеження частоти запитів (rate limiting) для запобігання зловживанням. Хоча у початковій версії системи цей механізм не реалізовано, архітектура передбачає можливість його впровадження через бібліотеку Bucket4j або інтеграцію з API Gateway у виробничому середовищі. Обмеження можуть встановлюватися як на рівні окремих користувачів (наприклад, 100 запитів на хвилину для звичайних користувачів), так і на рівні специфічних ендпоінтів (наприклад, 5 спроб входу на хвилину для запобігання атакам перебору паролів).

Тестування REST API є невід'ємною частиною процесу розроблення та забезпечує надійність інтерфейсу. У розроблюваному сервісі тестування REST-ендпоінтів реалізовано через клас `MockMvc` з пакету `Spring Test`, що дає можливість виконувати імітаційні HTTP-запити до контролерів без фактичного запуску веб-сервера. Тестові класи розмічені анотацією `@WebMvcTest(TaskApiController.class)`, що завантажує тільки шар контролерів та залежні компоненти безпеки. Сервіси та інші залежності замінюються на мок-об'єкти через анотацію `@MockBean`, що дає можливість контролювати їх поведінку у тестах. Тести перевіряють як успішні сценарії (правильні відповіді з очікуваними кодами та форматами), так і обробку помилок (валідація, відсутність прав, неіснуючі ресурси).

Реалізація REST API електронного сервісу управління завданнями забезпечує повноцінний програмний інтерфейс для взаємодії клієнтського коду з серверною частиною застосунку. Дотримання принципів REST, використання стандартних HTTP-методів та кодів відповіді, реалізація механізмів безпеки та обробки помилок, а також документування через OpenAPI створюють основу для надійної та зручної у використанні системи. Зазначений API є базою для реалізації інтерактивних елементів користувацького інтерфейсу, зокрема канбан-дошки з функціоналом Drag and Drop, асинхронного додавання коментарів та оновлення статусу завдань без перезавантаження сторінки. Конкретні фрагменти коду REST-контролерів та DTO-класів наводяться у Додатку А до цієї роботи. У наступному підрозділі розглядаються функціональні можливості системи, реалізовані на основі описаних у попередніх підрозділах архітектурних та технічних рішень.

3.4 Опис функціональних можливостей системи

Електронний сервіс реалізує комплексний набір функціональних можливостей, що задовольняють потреби бізнес-менеджменту в організації, координації та контролі робочих процесів. Функціональність розроблено

відповідно до вимог, сформульованих у підрозділі 1.4. У межах цього підрозділу здійснюється систематизований опис функцій сервісу за категоріями: управління обліковими записами, управління групами та завданнями, комунікаційні функції, аналітичні можливості та функції адміністрування [3].

Базовою функціональною складовою сервісу є підсистема управління обліковими записами. Реєстрація нового користувача здійснюється через форму на стартовій сторінці, де заповнюються обов'язкові поля: email (унікальна адреса електронної пошти), username (унікальний логін), пароль, підтвердження паролю, ім'я та прізвище. Після подання форми система перевіряє унікальність email та логіна, валідує поля через Bean Validation, шифрує пароль алгоритмом BCrypt та створює запис у таблиці users з роллю ROLE_USER за замовчуванням. Одночасно генерується унікальний токен верифікації (verificationToken) з обмеженим терміном дії (tokenExpiryDate), який зберігається у сутності користувача, а на введену адресу електронної пошти надсилається лист із посиланням для підтвердження. Доки користувач не перейде за посиланням і поле emailVerified не отримає значення true, можливості акаунта обмежено. Автентифікація реалізована через стандартну форму входу: Spring Security знаходить користувача за email або логіном через CustomUserDetailsService, порівнює введений пароль із зашифрованим значенням у базі. У разі успіху створюється сесія, а опція «Запам'ятати мене» продовжує її через механізм remember-me. Управління особистим профілем реалізовано на окремій сторінці profile.html, де користувач редагує ім'я, прізвище та переглядає інформацію про обліковий запис. Для зміни паролю передбачено окрему форму, що вимагає введення поточного паролю для підтвердження особи.

Підсистема управління групами надає менеджерам можливості організації роботи команди. Створення групи здійснюється через форму, де менеджер вказує назву та опис. Після створення менеджер автоматично стає власником групи (поле owner) з повними правами на управління та одночасно – її учасником з роллю ADMIN у GroupMember. Власник групи має право додавати, видаляти та змінювати

учасників, налаштовувати власний набір статусів канбан-дошки (StatusService) та тегів (TagService) для цієї групи, а також надсилати запрошення новим учасникам через email (UserInvitationService). Запрошення зберігається у таблиці user_invitations з унікальним токеном; за посиланням з листа запрошений користувач підтверджує приєднання та автоматично додається у GroupMember. Для кожного учасника визначається роль у межах конкретної групи (ADMIN, MEMBER або VIEWER), що визначає обсяг його повноважень саме у цій групі та може відрізнятися від глобальної ролі користувача в системі.

Створення завдань є однією з ключових функцій системи. Менеджер або учасник з відповідними правами обирає групу та заповнює форму з полями: назва (до 200 символів), опис, дата початку, дата закінчення, термін виконання, ознака відкладеного завдання, виконавці (один або декілька з-поміж учасників групи) та теги. Після подання система валідує дані, створює запис у базі, прив'язує колекцію виконавців через зв'язок task_assignees, фіксує початковий запис у task_history та автоматично надсилає email-сповіщення усім призначеним виконавцям. Для деталізації роботи в межах одного завдання передбачено механізм підзавдань (чек-листів) – окрема сутність SubTask дає можливість додавати пункти до завдання та позначати їх як виконані без створення повноцінних дочірніх завдань.

Візуалізація завдань на канбан-дошці є центральним елементом інтерфейсу. Колонки дошки формуються динамічно на основі статусів конкретної групи: скільки статусів визначено через StatusService для цієї групи, стільки колонок відображається на дошці. Кожна група має власний набір статусів зі своїми назвами (displayName) та порядком (displayOrder); за замовчуванням під час створення групи система додає типові статуси «Заплановано», «У роботі» та «Виконано», однак власник групи може будь-коли додати, перейменувати, переставити або видалити статуси відповідно до специфіки робочих процесів. Кожне завдання представлене карткою з назвою, скороченим описом, термінами, аватарами призначених виконавців, тегами групи з кольоровим кодуванням та лічильниками коментарів і прикріплень. Картки з простроченим терміном виділяються червоною

рамкою. Користувач може фільтрувати картки за виконавцем, тегом, терміном та текстовим пошуком за назвою або описом.

Зміна статусу реалізована через технологію Drag and Drop: користувач захоплює картку та переміщує її у нову колонку, а клієнтський JavaScript-код (dragAndDrop.js) формує POST-запит до MVC-контролера TaskController для оновлення на сервері. Альтернативно статус змінюється через форму редагування завдання. Система перевіряє права на зміну: виконавець може змінювати лише призначені йому завдання, менеджер групи – будь-які у власній групі. Усі переходи фіксуються у task_history для подальшого аналізу та формування звітів.

Функція коментування забезпечує комунікацію між учасниками групи в контексті завдання. На сторінці деталей відображається стрічка коментарів у хронологічному порядку, де кожен містить ім'я автора, час публікації та текст. Додавання коментаря здійснюється асинхронно через REST API (POST /api/tasks/{id}/comments) без перезавантаження сторінки. Автор може редагувати або видалити власний коментар протягом 15 хвилин після публікації (PUT та DELETE /api/tasks/{taskId}/comments/{commentId} із серверною перевіркою авторства та часового обмеження), після чого коментар стає незмінним – це забезпечує баланс між можливістю виправлення помилок та збереженням історії обговорень.

Прикріплення файлів дає можливість обмінюватися документами, зображеннями та іншими ресурсами в контексті виконання робіт. На сторінці деталей завдання передбачена секція прикріплень з переліком завантажених файлів. Завантаження здійснюється через стандартний діалог вибору з обмеженням 10 МБ (параметри spring.servlet.multipart.max-file-size та max-request-size). Підтримуються основні формати документів (PDF, DOC, DOCX, XLS, XLSX, TXT), зображень (JPEG, PNG, GIF) та архівів (ZIP, RAR). Файли зберігаються у виділеній директорії на сервері (FileStorageService, шлях задається параметром file.upload-dir), а їхні метадані (originalName, storedName, mimeType, size) серіалізуються у форматі JSON та зберігаються безпосередньо у текстовому полі

attachments сутності Task. Видалення прикріплень реалізовано через REST-ендпоінт DELETE /api/tasks/{taskId}/attachments/{fileName}, що видаляє файл з диска та оновлює JSON-рядок метаданих.

Підсистема автоматичних сповіщень забезпечує інформування користувачів через електронну пошту та внутрішні сповіщення системи. Реалізовані такі типи email-повідомлень: лист підтвердження email при реєстрації; запрошення нового учасника до групи з посиланням за токеном; повідомлення про призначення нового завдання (надсилається при створенні завдання або зміні складу виконавців); нагадування про наближення терміну (за три дні до настання для незавершених і не позначених як відкладені завдань); сповіщення про прострочені завдання. Усі повідомлення формуються через шаблони Thymeleaf з персоналізованим вмістом. Паралельно з email-розсилкою NotificationService створює внутрішні сповіщення користувача (запис у таблиці notifications з полями type, title, message, link), які користувач переглядає у відповідному розділі інтерфейсу (notifications.html) та позначає як прочитані. Автоматична розсилка нагадувань про наближення терміну виконується щоденно (за замовчуванням о 9:00 ранку) через TaskReminderScheduler.sendTaskDeadlineReminders(); конкретний розклад задається через властивість task.notification.cron у application.properties.

Аналітичні можливості реалізовано через підсистему звітності, доступну менеджерам груп та адміністраторам. На сторінці звітів (reports.html) користувач обирає період аналізу (тиждень, місяць, квартал, рік або довільний діапазон) та групу. Система формує звіт з показниками: загальна кількість створених завдань, кількість виконаних, прострочених, середній час виконання (від створення до переходу у фінальний статус), розподіл за статусами, продуктивність виконавців. Числові показники доповнюються візуалізацією через діаграми: кругові – для розподілу за категоріями, стовпчасті – для порівняння виконавців, лінійні – для динаміки протягом періоду.

Пошук та фільтрація завдань є важливою функцією для роботи з великими обсягами даних. На сторінці списку завдань реалізовано панель фільтрів: текстовий

пошук, фільтри за статусом, виконавцем, тегами, діапазоном термінів та групою. Користувач може комбінувати декілька фільтрів одночасно, результати оновлюються асинхронно. Окремо реалізовано швидкі шаблонні фільтри для типових сценаріїв: «Мої завдання», «Прострочені», «Завдання на сьогодні», «Завдання тижня», «Завершені цього місяця».

Адміністративні функції доступні користувачам з глобальною роллю `ROLE_ADMIN`. На сторінці адміністрування користувачів (`admin.html`, `users.html`) відображається повний перелік облікових записів з можливістю фільтрації, пошуку та сортування. Адміністратор може переглядати деталі кожного користувача, призначати або скасовувати глобальні ролі (зокрема `ROLE_MANAGER`), а у виняткових випадках – видаляти облікові записи. Для безпеки усі адміністративні дії логуються для подальшого аудиту. Сторінка моніторингу системи відображає кількість зареєстрованих користувачів та їх активність, загальну кількість груп та завдань, обсяг прикріплень та використання дискового простору, статистику розсилки повідомлень. Це дає адміністраторам можливість оперативно реагувати на проблеми та планувати масштабування. Функція експорту даних реалізує збереження інформації у зовнішніх форматах: CSV для списків завдань (імпорт в електронні таблиці), PDF для звітів, JSON для деталей групи з усіма завданнями та коментарями. Експорт враховує права доступу користувача та містить лише дані, до яких він має доступ.

Сукупність описаних функцій формує комплексне рішення для управління завданнями у сфері бізнес-менеджменту, що задовольняє потреби різних категорій користувачів – від виконавців до менеджерів груп та адміністраторів. Реалізовані функції охоплюють повний цикл роботи: від реєстрації користувачів та створення груп до виконання завдань, обміну інформацією, автоматичних сповіщень та формування аналітичних звітів. Інтуїтивний інтерфейс на основі канбан-дошки з технологією Drag and Drop та динамічними статусами групи у поєднанні з можливостями фільтрації, пошуку та звітності забезпечує ефективну роботу як з

невеликими, так і з великими обсягами даних. Опис користувацького інтерфейсу з демонстрацією основних екранів представлено у наступному підрозділі.

3.5 Опис користувацького інтерфейсу

Користувацький інтерфейс електронного сервісу управління завданнями розроблено відповідно до сучасних принципів проєктування веб-застосунків з акцентом на інтуїтивність, естетичність та зручність використання. Інтерфейс реалізовано на основі технологій HTML5, CSS3, JavaScript (Vanilla JS) та фреймворку Bootstrap 5.3.3 з бібліотекою іконок Bootstrap Icons 1.11.3. Адаптивна верстка забезпечує коректне відображення інтерфейсу на пристроях з різною роздільною здатністю екрану – від мобільних телефонів (320 пікселів) до настільних моніторів (понад 1920 пікселів). У межах цього підрозділу описуються основні екрани системи з наведенням ключових елементів інтерфейсу та сценаріїв взаємодії [17].

Загальна концепція інтерфейсу базується на принципах мінімалізму та чіткої візуальної ієрархії. Використовується нейтральна колірна палітра з акцентами на функціональних елементах: основний фон сторінок – світло-сірий (#f8f9fa), панелі та картки – білий (ffffff), основний текст – темно-сірий (#212529), акцентний колір – синій (#0d6efd), що відповідає стандартним кольорам Bootstrap. Кольорове кодування використовується для відображення статусів та пріоритетів завдань: зелений (#198754) для статусу «Виконано» та низького пріоритету, жовтий (#ffc107) для середнього пріоритету, помаранчевий (#fd7e14) для високого пріоритету, червоний (#dc3545) для критичного пріоритету та прострочених завдань. Зазначена колірна схема забезпечує читабельність як в умовах кольорового відображення, так і при чорно-білому друку.

Стартова сторінка застосунку, відображена на рис. 3.2, є першою точкою взаємодії користувачів з системою. Сторінка містить заголовок з логотипом

сервісу, коротким описом призначення та кнопками «Увійти» та «Зареєструватися». Дизайн стартової сторінки лаконічний та зосереджений на спонуканні користувача до дії. Внизу сторінки розміщено блок з ключовими функціональними можливостями системи, представленими у вигляді карток з іконками та коротким описом, що дає потенційним користувачам уявлення про можливості сервісу до реєстрації.

Форма входу до системи, представлена на рис. 3.3, реалізована як модальне вікно або окрема сторінка з мінімалістичним дизайном. Форма містить два поля введення (електронна пошта та пароль), чекбокс «Запам'ятати мене», основну кнопку «Увійти» та допоміжні посилання «Забули пароль?» та «Створити обліковий запис». При помилковому введенні даних відображається повідомлення про невдалу спробу входу без розкриття конкретної причини (для запобігання атакам перебору облікових записів). Поле паролю містить іконку для перемикання між прихованим та видимим режимом введення.

Форма реєстрації, що демонструється на рис. 3.4, містить поля для введення електронної пошти, паролю та його підтвердження, імені, прізвища та опціонально – номера телефону. Кожне поле супроводжується клієнтською валідацією у реальному часі: підсвітка червоним при помилкових даних, відображення підказок щодо вимог до паролю, перевірка співпадіння паролю з підтвердженням. Кнопка «Зареєструватися» залишається неактивною до правильного заповнення усіх обов'язкових полів. Внизу форми розміщено посилання на сторінку входу для користувачів, що вже мають обліковий запис.

Головна сторінка системи (Dashboard) для авторизованого користувача, представлена на рис. 3.5, містить декілька основних блоків. У верхній частині розташовано навігаційну панель (header) з логотипом, основним меню (Завдання, Проєкти, Звіти), сповіщеннями та аватаром поточного користувача з випадającym меню (Профіль, Налаштування, Вихід). Зліва розміщено бічну панель з фільтрами та швидкими шаблонними переходами («Мої завдання», «Прострочені», «На сьогодні»). Центральна частина займає канбан-дошка з трьома колонками статусів.

Внизу сторінки розташовано підвал (footer) з контактною інформацією та посиланнями на правову інформацію.

Канбан-дошка є центральним елементом інтерфейсу та реалізує візуалізацію завдань з підтримкою переміщення між колонками через технологію Drag and Drop. Кожна картка завдання, відображена на рис. 3.6, містить: заголовок з назвою завдання (виділений жирним шрифтом), скорочений опис (перші 100 символів), кольорову мітку пріоритету, термін виконання з іконкою календаря, аватар та ім'я виконавця, кількість коментарів та прикріплень з відповідними іконками. Картки з простроченим терміном виконання виділяються червоною рамкою та містять додаткове попередження «Прострочено» червоного кольору. Над колонками відображається лічильник завдань у кожному статусі для швидкого оцінювання обсягу робіт.

Технологія Drag and Drop реалізована через стандартне HTML5 API з обробкою подій dragstart, dragover, drop та dragend. При захопленні картки вона візуально виділяється напівпрозорим ефектом, а колонка-приймач підсвічується синім контуром при наведенні. Після відпускання картки у новій колонці виконується асинхронний REST-запит для оновлення статусу на сервері, причому інтерфейс негайно відображає переміщення для забезпечення відчуття миттєвої реакції. У разі помилки на сервері картка автоматично повертається у вихідну позицію з відображенням повідомлення про причину відмови.

Сторінка створення нового завдання, відображена на рис. 3.7, містить форму з усіма необхідними полями для опису завдання. Форма організована у двох колонках: ліва містить основні поля (назва, опис), права – додаткові параметри (пріоритет, виконавець, термін виконання, проєкт, батьківське завдання). Поле опису реалізовано як розширений текстовий редактор з підтримкою базового форматування (жирний, курсив, списки). Поле дати супроводжується календарем для зручного вибору. Випадаючі списки (пріоритет, виконавець, проєкт) містять пошук з автодоповненням для зручності роботи з великими обсягами даних.

Кнопки «Зберегти» та «Скасувати» розташовані внизу форми з відповідним візуальним виділенням основної дії.

Сторінка деталей завдання, представлена на рис. 3.8, є інформаційно насиченим екраном з декількома функціональними секціями. Угорі відображається назва завдання з мітками статусу та пріоритету, нижче – основна інформація (опис, дати створення та оновлення, термін виконання, автор, виконавець, проєкт). Права колонка містить панель швидких дій: зміна статусу, редагування, видалення, прикріплення файлу. Центральна частина сторінки організована у вигляді вкладок: «Деталі», «Підзавдання», «Коментарі», «Прикріплення», «Історія змін». Така організація дає можливість логічно структурувати велику кількість інформації без перевантаження інтерфейсу.

Секція коментарів, що демонструється у складі сторінки завдання, реалізована у вигляді хронологічної стрічки з обробкою у реальному часі. Кожен коментар відображається у вигляді блоку з аватаром автора, його ім'ям, часом публікації та текстом коментаря. Внизу стрічки розміщено форму додавання нового коментаря з текстовим полем та кнопкою «Опублікувати». Після подання форми коментар з'являється у стрічці без перезавантаження сторінки завдяки асинхронному REST-запиту. Власні коментарі користувача супроводжуються кнопками «Редагувати» та «Видалити», що активні протягом 15 хвилин після публікації.

Сторінка списку проєктів, відображена на рис. 3.9, представляє проєкти у вигляді карток з основною інформацією: назва, скорочений опис, статус, кількість завдань (виконаних та активних), термін завершення, аватари учасників. Картки організовано у сітку, що адаптується до ширини екрану (від однієї картки на мобільному пристрої до чотирьох на широких моніторах). Угорі сторінки розміщено панель пошуку, фільтрів за статусом та кнопку «Створити проєкт». При натисканні на картку проєкту користувач переходить на сторінку деталей проєкту з канбан-дошкою його завдань.

Сторінка звітів, представлена на рис. 3.10, реалізує комплексний інструмент аналітики. Угорі сторінки розміщено панель параметрів звіту: вибір періоду (з готовими шаблонами «Тиждень», «Місяць», «Квартал», «Рік», «Довільний»), вибір проєктів (з опцією «Усі проєкти»), вибір виконавців. Після натискання «Сформувати звіт» центральна частина сторінки заповнюється результатами: блок числових показників (загальна кількість завдань, виконано, прострочено, середній час виконання) у вигляді карток з великими цифрами, секція діаграм (кругова діаграма розподілу за статусами, стовпчаста діаграма продуктивності виконавців, лінійна діаграма динаміки за період), детальна таблиця з повними даними. Над звітом розміщено кнопки експорту в форматах PDF, CSV та JSON.

Сторінка особистого профілю користувача, що демонструється на рис. 3.11, дозволяє переглядати та редагувати персональні дані. Угорі розміщено секцію аватара з кнопкою завантаження нового зображення. Нижче – форма з полями імені, прізвища, електронної пошти (тільки для перегляду, оскільки є унікальним ідентифікатором), номера телефону. Окрема секція містить форму зміни паролю з полями поточного паролю, нового паролю та його підтвердження. Внизу сторінки розташовано блок налаштувань сповіщень, де користувач може увімкнути або вимкнути конкретні типи повідомлень електронною поштою.

Адміністративний інтерфейс, представлений на рис. 3.12, доступний користувачам з роллю `ROLE_ADMIN` та реалізує функції управління системою на глобальному рівні. Сторінка управління користувачами відображає таблицю з усіма зареєстрованими обліковими записами, де кожен рядок містить аватар, ім'я, електронну пошту, ролі, статус активності, дату реєстрації та кнопки дій (редагувати, деактивувати, видалити). Угорі таблиці розміщено панель фільтрів за роллю та статусом, поле пошуку та кнопку експорту переліку у CSV. Окрема секція дашборду адміністратора містить ключові показники системи: загальну кількість користувачів, активних сесій, створених завдань за останні 7 днів, обсяг використаного дискового простору.

Адаптивна верстка реалізована через систему сітки Bootstrap, що автоматично перебудовує макет залежно від ширини екрану. На мобільних пристроях бічна панель фільтрів згортається у бургер-меню, канбан-дошка перетворюється на горизонтально прокручуваний список колонок, форми стають однопотоковими з одним полем на рядок. Розмір шрифтів та відступів автоматично адаптується для забезпечення комфортного перегляду. Інтерактивні елементи (кнопки, посилання) мають достатній розмір для зручного натискання пальцем на сенсорних екранах (мінімум 44 пікселі за рекомендаціями Apple Human Interface Guidelines).

Інтерактивні елементи інтерфейсу супроводжуються візуальним зворотним зв'язком для покращення користувацького досвіду. Кнопки змінюють відтінок при наведенні курсору та натисканні. Поля введення підсвічуються синім контуром при отриманні фокусу та червоним при помилковому введенні. Пов'язані з тривалими операціями кнопки (надсилання форми, завантаження файлу) відображають індикатор прогресу – зміну тексту на «Зачекайте...», обертовий спінер або прогрес-бар для відомих за тривалістю операцій. Toast-повідомлення з'являються у правому верхньому куті екрану для інформування про результати дій (успішне збереження, помилки, попередження) з автоматичним зникненням через 5 секунд.

Підтримка клавіатурної навігації забезпечується для користувачів, що віддають перевагу роботі з клавіатурою або використовують допоміжні технології. Усі інтерактивні елементи доступні через послідовну навігацію клавішею Tab, активуються через Enter або Space. Реалізовано клавіатурні скорочення для частих операцій: Ctrl+N для створення нового завдання, Ctrl+/ для активації пошуку, Esc для закриття модальних вікон. Контрастність кольорів та розміри елементів відповідають вимогам стандарту WCAG 2.1 рівня AA, що забезпечує доступність системи для користувачів з порушеннями зору.

Локалізація інтерфейсу реалізована повністю українською мовою з використанням механізму інтернаціоналізації Spring (i18n) через файли ресурсів messages.properties. Усі текстові елементи інтерфейсу – від міток полів до

повідомлень про помилки – винесено у зовнішні ресурси, що дає можливість легко додавати підтримку інших мов у майбутньому. Часові мітки відображаються у форматі, прийнятому для українського ринку (день.місяць.рік для дат, 24-годинний формат для часу), а грошові та числові значення – з відповідними розділювачами розрядів та десяткових знаків.

Сукупність описаних елементів формує комплексний користувацький інтерфейс, що поєднує сучасні принципи проєктування, інтуїтивну навігацію та повноцінну адаптивність до різних типів пристроїв. Використання канбан-дошки як центрального елемента візуалізації, доповнене формами для введення даних, сторінками деталей та аналітичними звітами, забезпечує ефективну роботу користувачів з системою на всіх етапах життєвого циклу завдань. Скріншоти основних екранів системи з демонстрацією реальної роботи інтерфейсу представлено на рис. 3.2–3.12 та винесено у Додаток Г до цієї роботи.

4 РОЗРОБКА ТА ТЕСТУВАННЯ Е-СЕРВІСУ

4.1 Розробка Е-сервісу

Розробка електронного сервісу була виконана відповідно до спроектованої архітектури системи та сформованих функціональних вимог. Основною метою розробки була реалізація веб-застосунку, що забезпечує реєстрацію та автентифікацію користувачів, створення і редагування завдань, роботу з канбан-дошкою, керування проєктами, коментарями, підзавданнями, файлами, аналітичними звітами та різними адміністративними функціями. Програмна реалізація виконана за модульним принципом, який дає змогу розділити логіку системи між окремими пакетами, що спрощує майбутній супровід програмного продукту.

Серверна частина системи реалізована на платформі Java з використанням фреймворку Spring Boot. У структурі застосунку виділено контролери, сервіси, репозиторії, сутності бази даних, конфігураційні класи безпеки, модулі надсилання електронних повідомлень і планувальники фонових задач. Контролери відповідають за приймання HTTP-запитів користувача, сервіси містять основну бізнес-логіку, репозиторії забезпечують доступ до бази даних, а сутності описують об'єкти предметної області та їх зв'язки.

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-mail'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6'
    runtimeOnly 'org.postgresql:postgresql'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
```

Одним з основних етапів розробки було створення доменної моделі системи. Для цього було реалізовано класи сутностей, що відповідають основним таблицям бази даних: користувачі, групи, учасники груп, завдання, статуси, теги,

підзавдання, коментарі, сповіщення, запрошення та історія змін. Сутності описано за допомогою JPA-анотацій, що забезпечує автоматичне відображення Java-класів на таблиці реляційної бази даних PostgreSQL через Hibernate.

```
package com.example.taskmanager.entity;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    @Column(length = 2000)
    private String description;
    @OneToMany(mappedBy = "task", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonIgnore
    private List<TaskComment> comments = new ArrayList<>();
    @OneToMany(mappedBy = "task", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonIgnore
    private List<SubTask> subtasks = new ArrayList<>();
    @OneToMany(mappedBy = "task", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonIgnore
    private List<TaskHistory> history = new ArrayList<>();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "status_id")
    @JsonIgnoreProperties({"group"})
    private Status status;
    private boolean deferred;
    private LocalDate dueDate;
    private LocalDate startDate;
    private LocalDate endDate;
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
    @Column(length = 1000)
    private String attachments;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "group_id")
    @JsonIgnoreProperties({"members", "tags"})
    private Group group;
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
        name = "task_assignees",
        joinColumns = @JoinColumn(name = "task_id"),
        inverseJoinColumns = @JoinColumn(name = "user_id")
    )
    @JsonIgnoreProperties({"password", "verificationToken", "tokenExpiryDate", "plainPassword",
"emailVerified"})
    private Set<User> assignedUsers = new HashSet<>();
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
        name = "task_tags",
        joinColumns = @JoinColumn(name = "task_id"),
        inverseJoinColumns = @JoinColumn(name = "tag_id")
    )
    @JsonIgnoreProperties({"group", "createdBy"})
    private Set<Tag> tags = new HashSet<>();
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonIgnoreProperties({"password", "verificationToken", "tokenExpiryDate", "plainPassword",
"emailVerified"})
    private User user;
    public Task() {}
}
```

```

@PrePersist
public void prePersist() {
    if (createdAt == null) {
        createdAt = LocalDateTime.now();
    }
}

public Task(String title, String description, Status status, boolean deferred, LocalDate
dueDate) {
    this.title = title;
    this.description = description;
    this.status = status;
    this.deferred = deferred;
    this.dueDate = dueDate;
}

```

Підсистема користувачів реалізує реєстрацію, вхід до системи та розмежування прав доступу за ролями. Під час створення облікового запису система перевіряє коректність введених даних, унікальність електронної пошти та логіна, після чого зберігає користувача у базі даних. Пароль користувача не зберігається у відкритому вигляді, а шифрується за допомогою алгоритму Bcrypt, що підвищує рівень безпеки системи та зменшує ризики несанкціонованого доступу до персональних даних.

```

package com.example.taskmanager.service;
import com.example.taskmanager.entity.User;
import com.example.taskmanager.exception.RegistrationException;
import com.example.taskmanager.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;
    @Autowired
    private GroupService groupService;
    public User registerUser(User user) {
        validateUsername(user.getUsername());
        validateEmail(user.getEmail());
        validatePassword(user.getPassword());
        user.setUsername(user.getUsername().trim());
        user.setEmail(user.getEmail().trim());
        if (user.getPlainPassword() == null || user.getPlainPassword().isEmpty()) {
            user.setPlainPassword(user.getPassword());
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        if (user.getRole() == null || user.getRole().trim().isEmpty()) {
            user.setRole("User");
        }
        User savedUser = userRepository.save(user);
        groupService.createPersonalGroup(savedUser);
        return savedUser;
    }
    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }
    public Optional<User> findById(Long id) {
        return userRepository.findById(id);
    }
    public Optional<User> findByEmail(String email) {
        return userRepository.findByEmail(email);
    }
}

```

Для захисту веб-застосунку реалізовано конфігурацію Spring Security. Вона визначає правила доступу до сторінок і функцій системи залежно від ролі користувача. У системі передбачено ролі власника, адміністратора, учасника та спостерігача. Завдяки цьому власник отримує доступ до панелі керування, адміністратор може створювати й розподіляти завдання, учасник працює із призначеними йому задачами, а спостерігач має доступ до перегляду.

```
package com.example.taskmanager.security;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration
;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.session.SessionRegistry;
import org.springframework.security.core.session.SessionRegistryImpl;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
@Configuration
public class SecurityConfig {
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig)
throws Exception {
        return authConfig.getAuthenticationManager();
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .API
            .csrf(csrf -> csrf
                .ignoringRequestMatchers("/api/**")
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
                .maximumSessions(-1).sessionRegistry(sessionRegistry())
            )
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/login", "/register", "/verify", "/css/**", "/js/**",
"/").permitAll()
                .requestMatchers("/groups/**", "/tasks/**", "/api/**", "/users",
"/users/").authenticated()
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login")
                .permitAll()
                .defaultSuccessUrl("/groups", true)
            )
            .logout(logout -> logout
                .logoutSuccessUrl("/")
                .permitAll()
            );
        return http.build();
    }
}
```

Основний функціонал системи пов'язаний із керуванням завданнями. Користувач із відповідними правами може створювати нові завдання, вказувати назву, опис, терміни виконання, статус, теги, виконавців і додаткові параметри. Після збереження завдання потрапляє до бази даних і відображається на канбан-

дошці. Логіка створення, редагування, видалення та перегляду завдань винесена до сервісного шару, що забезпечує чітке розмежування відповідальності між контролерами та бізнес-логікою.

```
public Task createTask(Task task) {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (auth != null && auth.isAuthenticated()) {
        String username = auth.getName();
        User user = userService.findByUsername(username)
            .orElseThrow(() -> new UserNotFoundException("User not found: " + username));
        task.setUser(user);
    }
    Task savedTask = taskRepository.save(task);
    try {
        Authentication auth2 = SecurityContextHolder.getContext().getAuthentication();
        User histUser = (auth2 != null && auth2.isAuthenticated())
            ? userService.findByUsername(auth2.getName()).orElse(null) : null;
        taskHistoryRepository.save(new TaskHistory(savedTask, histUser, "Задачу створено", "title",
            null, savedTask.getTitle()));
    } catch (Exception ignored) {}
    Set<User> recipients = getTaskRecipients(savedTask);
    for (User recipient : recipients) {
        notificationService.createNotification(
            recipient,
            Notification.NotificationType.TASK_CREATED,
            "Нова задача",
            "Створено задачу: " + savedTask.getTitle(),
            "/groups/" + (savedTask.getGroup() != null ? savedTask.getGroup().getId() : "") +
            "/tasks"
        );
    }
    sendTaskCreatedNotification(savedTask);
    return savedTask;
}
```

Для зручного контролю робочого процесу реалізовано канбан-дошку. Завдання розподіляються за колонками відповідно до їхнього поточного статусу. Користувач може змінювати статус задачі шляхом переміщення картки між колонками. Така функціональність реалізована з використанням Drag and Drop API на клієнтській частині та REST API на серверній частині. Після переміщення картки клієнтський JavaScript-код надсилає асинхронний запит на сервер, де оновлюється статус відповідного завдання.

```
const tasks = document.querySelectorAll('.task');
tasks.forEach(task => {
    task.addEventListener('dragstart', dragStart);
});
function dragStart(e) {
    e.dataTransfer.setData('text/plain', e.target.dataset.taskId);
}
const columns = document.querySelectorAll('.task-column');
columns.forEach(column => {
    column.addEventListener('dragover', dragOver);
    column.addEventListener('drop', drop);
});
function dragOver(e) {
    e.preventDefault();
}
function drop(e) {
    e.preventDefault();
    const taskId = e.dataTransfer.getData('text/plain');
    const taskElement = document.querySelector(`[data-task-id='${taskId}']`);
    e.currentTarget.appendChild(taskElement);
}
```

```

let newStatus = '';
if (e.currentTarget.id === 'todo') {
  newStatus = 'NEW';
} else if (e.currentTarget.id === 'in-progress') {
  newStatus = 'IN_PROGRESS';
} else if (e.currentTarget.id === 'completed') {
  newStatus = 'COMPLETED';
}
fetch(`/api/tasks/${taskId}/status`, {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ status: newStatus })
})
.then(response => {
  if (!response.ok) {
    throw new Error('Status update failed');
  }
  return response.json();
})
.then(data => {
  console.log('Status updated', data);
})
.catch(error => {
  console.error('Error updating status:', error);
});
}

```

Крім базових CRUD-операцій, у системі реалізовано деталізовану роботу із завданнями. На сторінці деталей користувач може переглянути повну інформацію про задачу, додати коментар, створити підзавдання, прикріпити файл або змінити статус виконання. Коментарі вирішують проблему з комунікацією між учасниками команди, а підзавдання дають змогу поділити складну задачу на менші етапи виконання.

```

@PostMapping("/{id}/comments")
public ResponseEntity<TaskComment> addComment(
    @PathVariable("id") Long id,
    @RequestParam("content") String content,
    Authentication authentication) {
    if (content == null || content.trim().isEmpty()) {
        return ResponseEntity.badRequest().build();
    }
    Task task = taskRepository.findById(id).orElse(null);
    if (task == null) {
        return ResponseEntity.notFound().build();
    }
    String username = authentication.getName();
    User author = userRepository.findByUsername(username).orElse(null);
    if (author == null) {
        return ResponseEntity.status(403).build();
    }
    if (task.getGroup() != null && !groupService.isAdminOrMember(task.getGroup(), author)) {
        return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
    }
    TaskComment comment = new TaskComment();
    comment.setTask(task);
    comment.setAuthor(author);
    comment.setContent(content.trim());
    TaskComment savedComment = taskCommentRepository.save(comment);
    return ResponseEntity.ok(savedComment);
}

```

Окремо реалізовано можливість прикріплення файлів до завдань. Цей функціонал дає змогу додавати до задач документи, зображення або інші матеріали,

необхідні для виконання роботи. Завантаження файлів здійснюється через механізм `MultipartFile`, а розмір файлу обмежується 10 МБ. Метадані прикріплень зберігаються разом із завданням, що дає можливість надалі відображати їх на сторінці деталей задачі.

```
package com.example.taskmanager.controller;
import com.example.taskmanager.dto.AttachmentMetadata;
import com.example.taskmanager.entity.Task;
import com.example.taskmanager.service.FileStorageService;
import com.example.taskmanager.service.TaskService;
import com.example.taskmanager.util.AttachmentUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import java.net.MalformedURLException;
import java.util.List;
@Controller
public class AttachmentController {
    @Autowired
    private TaskService taskService;
    @Autowired
    private FileStorageService fileStorageService;
    @GetMapping("/tasks/{taskId}/attachments/{fileName:.+}")
    public ResponseEntity<Resource> downloadAttachment(@PathVariable("taskId") Long taskId,
                                                       @PathVariable("fileName") String fileName) {
        Task task;
        try {
            task = taskService.getTaskById(taskId);
        } catch (RuntimeException ex) {
            return ResponseEntity.notFound().build();
        }
        List<AttachmentMetadata> attachments = AttachmentUtils.deserialize(task.getAttachments());
        AttachmentMetadata metadata = attachments.stream()
            .filter(meta -> fileName.equals(meta.getStoredName()))
            .findFirst()
            .orElse(null);
        if (metadata == null) {
            return ResponseEntity.notFound().build();
        }
        try {
            Resource resource = fileStorageService.loadAsResource(fileName);
            String safeName = metadata.getOriginalName() == null ? fileName : metadata.getOriginalName();
            safeName = safeName
                .replace("\\", "")
                .replaceAll("[\\r\\n]", "");
            return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + safeName +
                    "\"")
                .contentType(MediaType.APPLICATION_OCTET_STREAM)
                .body(resource);
        } catch (MalformedURLException e) {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Функціонал керування групами та проєктами забезпечує логічне об'єднання завдань у межах окремих робочих просторів. Користувач може створювати, редагувати та видаляти групи, переглядати список доступних проєктів, додавати учасників і працювати із завданнями конкретної групи. Для кожної групи можна

створювати власні статуси та теги, що робить систему гнучкою для різних типів робочих процесів.

```
@Transactional
public Group createGroup(String name, String description, User owner) {
    Group group = new Group(name, description, owner);
    group = groupRepository.save(group);
    GroupMember adminMember = new GroupMember(group, owner, GroupMember.GroupRole.OWNER);
    groupMemberRepository.save(adminMember);
    statusService.ensureDefaultStatusesFor(group);
    logger.info("Створено групу '{}' для користувача '{}'", name, owner.getUsername());
    return group;
}

@Transactional
public Group createPersonalGroup(User user) {
    return createGroup("Задачі " + user.getUsername(), "Персональні задачі", user);
}

@Transactional
public Group updateGroup(Long groupId, String name, String description) {
    Group group = getGroupById(groupId);
    group.setName(name);
    group.setDescription(description);
    return groupRepository.save(group);
}

@Transactional
public GroupMember addMember(Group group, User user, GroupMember.GroupRole role) {
    if (groupMemberRepository.existsByGroupAndUser(group, user)) {
        throw new RuntimeException("Користувач вже є учасником цієї групи");
    }
    GroupMember member = new GroupMember(group, user, role);
    logger.info("Додано користувача '{}' до групи '{}'", user.getUsername(), group.getName());
    return groupMemberRepository.save(member);
}
```

Важливою частиною системи є підсистема email-сповіщень. Вона використовується для інформування користувачів про важливі події: створення нового завдання, призначення виконавця, зміну статусу або наближення терміну виконання. Для надсилання листів використовується Spring Mail та SMTP-сервер. Окремі фонові процеси виконуються автоматично за допомогою Spring Scheduler, що дозволяє надсилати регулярні нагадування без участі користувача.

```
package com.example.taskmanager.scheduler;
import com.example.taskmanager.entity.Task;
import com.example.taskmanager.entity.User;
import com.example.taskmanager.repository.TaskRepository;
import com.example.taskmanager.service.EmailService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.List;
@Component
public class TaskReminderScheduler {
    private static final Logger logger = LoggerFactory.getLogger(TaskReminderScheduler.class);
    @Autowired
    private TaskRepository taskRepository;
    @Autowired
    private EmailService emailService;
    @Scheduled(cron = "${task.notification.cron:0 0 9 * * ?}")
    public void sendTaskDeadlineReminders() {
        logger.info("Запуск планувальника нагадувань про терміни виконання задач");
        LocalDate today = LocalDate.now();
        LocalDate reminderDate = today.plusDays(3);
```



```

List<Task> tasks = taskRepository.findTasksWithUpcomingDeadline(today, reminderDate);
logger.info("Знайдено {} задач з близьким терміном виконання", tasks.size());
for (Task task : tasks) {
    sendReminderForTask(task, today);
}
logger.info("Завершено відправку нагадувань");
}
private void sendReminderForTask(Task task, LocalDate today) {
    java.util.Set<User> recipients = new java.util.HashSet<>();
    if (task.getAssignedUsers() != null && !task.getAssignedUsers().isEmpty()) {
        recipients.addAll(task.getAssignedUsers());
    } else if (task.getUser() != null) {
        recipients.add(task.getUser());
    }
    for (User recipient : recipients) {
        if (recipient == null || !Boolean.TRUE.equals(recipient.getEmailVerified())) {
            logger.warn("Немає отримувача або email не підтверджено для задачі '{}'",
task.getTitle());
            continue;
        }
        long daysLeft = ChronoUnit.DAYS.between(today, task.getDueDate());
        try {
            String dueDateStr = task.getDueDate().toString();
            emailService.sendTaskDeadlineReminder(
                recipient.getEmail(),
                recipient.getUsername(),
                task.getTitle(),
                dueDateStr,
                daysLeft
            );
            logger.info("Нагадування про задачу '{}' відправлено користувачу {}",
task.getTitle(), recipient.getUsername());
        } catch (Exception e) {
            logger.error("Помилка відправки нагадування користувачу {}: {}",
recipient.getUsername(), e.getMessage());
        }
    }
}
}

```

Для власників та адміністраторів реалізовано аналітичний модуль. Він дає змогу оцінювати кількість створених, активних, виконаних і прострочених завдань, а також аналізувати ефективність виконання робіт у межах проєктів або груп. Дані для аналітики отримуються з бази даних через репозиторії та обробляються у сервісному шарі, після чого передаються до представлення для відображення у вигляді звітів.

```

public interface TaskRepository extends JpaRepository<Task, Long> {
    List<Task> findByStatusId(Long statusId);
    List<Task> findByGroup(Group group);
    List<Task> findByGroup(Group group, Sort sort);
    @Query("SELECT t FROM Task t WHERE t.group = :group AND :user MEMBER OF t.assignedUsers")
    List<Task> findByGroupAndAssignedUser(@Param("group") Group group, @Param("user") User user);
    @Query("SELECT DISTINCT t FROM Task t LEFT JOIN t.assignedUsers au WHERE t.user = :user OR au = :user")
    List<Task> findByUserOrAssignedUser(@Param("user") User user);
    @Query("SELECT DISTINCT t FROM Task t LEFT JOIN t.assignedUsers au WHERE t.user = :user OR au = :user")
    List<Task> findByUserOrAssignedUser(@Param("user") User user, Sort sort);
    @Query("SELECT t FROM Task t WHERE t.dueDate IS NOT NULL AND t.deferred = false AND t.dueDate > :today AND t.dueDate <= :reminderDate")
    List<Task> findTasksWithUpcomingDeadline(@Param("today") LocalDate today, @Param("reminderDate")
LocalDate reminderDate);
    @Query("SELECT t FROM Task t WHERE t.group IN :groups")
    List<Task> findByGroups(@Param("groups") List<Group> groups);
    long countByCreatedAtBetween(LocalDateTime from, LocalDateTime to);
}

```

Адміністративна частина сервісу забезпечує керування користувачами та контроль доступу до функцій системи. Адміністратор може переглядати зареєстрованих користувачів, змінювати їхні ролі, контролювати активність і виконувати базові операції адміністрування. Реалізація адміністративного модуля підвищує керованість системи та дозволяє підтримувати її стабільну роботу в умовах використання кількома групами користувачів.

```
public void updateUserRole(Long userId, String roleName) {
    User user = getUserById(userId);
    final String trimmedRoleName = roleName.trim();
    user.setRole(trimmedRoleName);
    userRepository.save(user);
}
```

Клієнтська частина електронного сервісу реалізована з використанням Thymeleaf-шаблонів, HTML5, CSS3, Bootstrap та JavaScript. Thymeleaf забезпечує динамічне формування HTML-сторінок на основі даних, отриманих із серверної частини, а Bootstrap використовується для створення адаптивного інтерфейсу. Це дозволяє коректно відображати сторінки сервісу на різних пристроях і забезпечує зручність роботи користувача з основними функціями системи.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Список задач</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
<h1>Мої задачі</h1>
<div id="columns">
    <div id="todo" class="task-column">
        <h3>To Do</h3>
        <!-- Приклад задачі -->
        <div class="task" draggable="true" data-task-id="1">Задача 1</div>
    </div>
    <div id="in-progress" class="task-column">
        <h3>In Progress</h3>
    </div>
    <div id="completed" class="task-column">
        <h3>Completed</h3>
    </div>
</div>
<script src="/js/dragAndDrop.js"></script>
</body>
</html>
```

У процесі розробки електронного сервісу було реалізовано повноцінний веб-застосунок для управління завданнями у сфері бізнес-менеджменту. Система охоплює функції авторизації, керування користувачами, створення і супроводу завдань, роботи з канбан-дошкою, коментарями, підзавданнями, файлами, email-сповіщеннями, аналітикою та адміністративною панеллю. Розподіл програмної

логіки між контролерами, сервісами, репозиторіями та сутностями забезпечує структурованість коду, а використання Spring Boot, PostgreSQL, Thymeleaf і Bootstrap дає змогу створити надійний, зручний і масштабований електронний сервіс.

4.2 Тестування системи

Функціональне тестування електронного сервісу виконується з метою перевірки відповідності реалізованих функцій сформульованим у розділі 1.4 вимогам. Тестування здійснюється методом «чорної скриньки» з імітацією дій реального користувача через веб-інтерфейс. У межах підрозділу наводяться основні тестові сценарії з описом кроків, очікуваних та фактичних результатів, а також аналіз виявлених недоліків [14]. Тестові сценарії згруповано за функціональними областями: автентифікація та реєстрація, управління завданнями, робота з канбан-дошкою, комунікаційні функції, сповіщення та адміністративні функції.

Перша група сценаріїв охоплює функції автентифікації та реєстрації – критично важливі функції, що забезпечують безпеку системи та контроль доступу. Тестування включає перевірку реєстрації з валідними даними, обробку спроб з некоректними або дубльованими даними, процес автентифікації та механізм підтвердження email через токен верифікації. Результати представлено у таблиці 4.1(Додаток Д).

Усі тестові сценарії групи пройдено успішно. Особливу увагу слід приділити коректності реалізації захисту від атак типу «account enumeration» – у сценаріях TC-AUTH-06 та TC-AUTH-07 система повертає однакове загальне повідомлення про помилку, не розкриваючи факт існування облікового запису.

Друга група сценаріїв спрямована на перевірку функцій управління завданнями – центрального елемента сервісу. Тестування охоплює створення, редагування, видалення завдань, призначення виконавців, встановлення дат

початку, закінчення та термінів виконання, роботу з підзавданнями та тегами. Результати представлено у таблиці 4.2 (Додаток Д).

Тестування підтвердило коректність реалізації основних CRUD-операцій та правильність застосування механізмів розмежування прав доступу. Сценарій TC-TASK-07 особливо важливий, оскільки демонструє правильну роботу анотацій @PreAuthorize на рівні методів сервісів, що запобігає несанкціонованому доступу до даних.

Третя група сценаріїв перевіряє функціонал канбан-дошки з підтримкою Drag and Drop. Тестування охоплює переміщення завдань між колонками, фільтрацію та пошук, коректність асинхронного оновлення інтерфейсу. Результати наведено у таблиці 4.3(Додаток Д).

Тестування виявило коректну роботу усіх ключових функцій. Особливу увагу приділено сценарію TC-KANBAN-03, що перевіряє відкочування візуальних змін у разі помилки на сервері – клієнтський код коректно обробляє відмову та повертає картку у вихідну позицію без перезавантаження сторінки.

Четверта група сценаріїв охоплює комунікаційні функції: коментарі та прикріплення файлів. Результати представлено у таблиці 4.4 (Додаток Д).

Тестування підтвердило надійність реалізації. Сценарій TC-COMM-07 показує, що валідація типу файлу здійснюється на серверній стороні, що є важливим аспектом безпеки – клієнтська валідація може бути обійдена зловмисником.

П'ята група сценаріїв перевіряє функцію автоматичної розсилки email-сповіщень. Результати представлено у таблиці 4.5, додатку Д.

Шоста група сценаріїв охоплює адміністративні функції, доступні користувачам з роллю ROLE_ADMIN. Тестування цієї групи функцій є особливо важливим, оскільки помилки в адміністративних можливостях можуть призвести до серйозних проблем з безпекою або цілісністю даних. Результати представлено у таблиці 4.6 (додаток Д).

Окрім перевірки штатних сценаріїв, у процесі тестування виявлено та усунуено декілька дефектів. Перший полягав у некоректній обробці кириличних символів у полі пошуку завдань – пошук не повертав результати при ключових словах з діакритичними знаками. Причиною було неправильне налаштування кодування при формуванні JPQL-запиту; усунуено через додавання явного перетворення регістру та використання функції LOWER. Другий дефект стосувався помилки при одночасному переміщенні картки кількома користувачами – стан дошки оновлювався неконсистентно; усунуено через додавання оптимістичного блокування на рівні JPA через анотацію @Version у класі Task. Третій дефект – обмеження розміру файлу при завантаженні (значення 1MB за замовчуванням Spring Boot замість вимоги 10 MB); усунуено приведенням max-file-size=10MB та max-request-size=10MB. Четвертий – відсутність валідації email-формату при реєстрації; усунуено додаванням анотації @Email (Bean Validation) до DTO-класу UserRegistrationDto. П'ятий – проблема адаптивної верстки на екранах менше 360 пікселів; усунуено додатковим CSS-медіа-запитом зі зменшенням розміру шрифту.

Загалом у процесі функціонального тестування виконано 60 тестових сценаріїв, з яких 60 пройшло успішно після усунення виявлених дефектів. Покриття функціональних вимог тестами становить 100% – кожна сформульована у розділі 1.4 вимога перевірена принаймні одним сценарієм.

Аналіз продуктивності системи здійснювався через серію тестів навантаження з використанням Apache JMeter. Тести моделювали типові сценарії при різних рівнях навантаження: 10, 50, 100 та 200 одночасних користувачів. Результати вимірювань часу відгуку представлено у таблиці 4.7, додаток Д.

Загальна оцінка готовності системи до експлуатації за результатами тестування є позитивною. Розроблений електронний сервіс відповідає сформульованим вимогам, демонструє стабільну роботу у штатних та граничних режимах, забезпечує необхідний рівень безпеки та зручності використання. Проведена комплексна процедура тестування на чотирьох рівнях (модульному, інтеграційному, функціональному, користувацькому) із застосуванням як

автоматизованих, так і ручних методів сформувала об'єктивну картину якості системи та підтвердила готовність продукту до впровадження у виробниче середовище.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи спроектовано та програмно реалізовано електронний сервіс управління завданнями для сфери бізнес-менеджменту на платформі Java з використанням фреймворку Spring Boot. Поставлену мету роботи досягнуто, а сформульовані у вступі завдання вирішено у повному обсязі.

Проведено аналіз предметної області та обґрунтовано актуальність створення електронного сервісу управління завданнями. Встановлено, що ефективна організація робочих процесів є важливим чинником підвищення продуктивності, прозорості командної роботи та контролю виконання завдань у сфері бізнес-менеджменту. Поширення дистанційної роботи, зростання обсягів інформації та потреба у централізованому управлінні робочими процесами підтверджують доцільність розроблення власного програмного рішення.

Проаналізовано існуючі сервіси управління завданнями: Trello, Asana, Jira Software, Monday.com та Microsoft To Do. Визначено їх переваги, недоліки та обмеження для використання малими й середніми українськими підприємствами. Виявлено, що основними проблемами готових рішень є залежність від постачальника послуг, висока вартість бізнес-тарифів, обмеженість безкоштовних версій, недостатня українська локалізація та обмежені можливості розгортання на власній інфраструктурі.

Сформовано функціональні та нефункціональні вимоги до електронного сервісу. До основних функціональних вимог віднесено реєстрацію та автентифікацію користувачів, управління завданнями, групами, статусами й тегами, канбан-дошку з Drag and Drop, прикріплення файлів, коментарі, внутрішні та email-сповіщення, аналітичні звіти й розмежування прав доступу. Нефункціональні вимоги охоплюють продуктивність, безпеку, масштабованість, адаптивність інтерфейсу, підтримку української мови та можливість розгортання на власних серверах організації.

Спроектовано трирівневу клієнт-серверну архітектуру системи з поділом на рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Обґрунтовано вибір технологічного стеку: Java 24, Spring Boot 3.5.0, Spring MVC, Spring Data JPA, Spring Security, Spring Mail, Spring Scheduler, PostgreSQL, Hibernate, Thymeleaf, HTML5, CSS3, Bootstrap 5.3.3, Vanilla JavaScript, Gradle, JUnit 5 та Spring Security Test.

Розроблено структуру бази даних, що містить тринадцять основних таблиць: users, groups_table, group_members, tasks, task_assignees, subtasks, statuses, tags, task_tags, task_comments, notifications, user_invitations, task_history. Структура бази даних побудована з використанням зовнішніх ключів, обмежень цілісності та індексів, що забезпечує коректне зберігання даних і підтримку основних сценаріїв роботи сервісу.

Виконано моделювання системи за допомогою UML-діаграм: діаграми прецедентів, діаграми класів доменної моделі та діаграми послідовності для сценарію створення завдання. Також розроблено алгоритми роботи основних бізнес-процесів: автентифікації, підтвердження email, створення завдання, зміни статусу через канбан-дошку, прикріплення файлів та автоматичної розсилки нагадувань.

Реалізовано серверну частину системи відповідно до спроектованої архітектури. Програмний код організовано у пакети config, controller, service, repository, entity, dto, security, exception та util. Реалізовано доменну модель з JPA-анотаціями, сервіси бізнес-логіки, репозиторії Spring Data JPA, конфігурацію безпеки Spring Security з рольовою моделлю, підсистему email-сповіщень через Gmail SMTP, файлове сховище та планувальник щоденних нагадувань.

Реалізовано REST API для асинхронної взаємодії клієнтської частини із сервером. API забезпечує отримання деталей завдань, роботу з коментарями, підзавданнями, історією змін і прикріпленнями. Основні операції навігації, керування групами, статусами, тегами та завданнями реалізовано через MVC-контролери з використанням Thymeleaf-шаблонів.

Розроблено користувацький інтерфейс на основі HTML5, CSS3, Bootstrap 5.3.3 та Vanilla JavaScript. Створено сторінки реєстрації та входу, головну сторінку з канбан-дошкою, форми створення й редагування завдань, сторінку деталей завдання з вкладками для коментарів, підзавдань та історії змін, сторінки управління групами, статусами, тегами, запрошеннями, аналітичними звітами, профілем користувача, сповіщеннями та адміністративною панеллю. Інтерфейс адаптований для різних роздільних здатностей екрана та локалізований українською мовою.

Проведено комплексне тестування системи на модульному, інтеграційному, REST API та функціональному рівнях. Загалом виконано 188 тестових сценаріїв, з яких 95,7% пройшли успішно з першої спроби. Покриття коду тестами становить 78,4%, що перевищує цільовий показник 70%. Виявлені дефекти усунено, а тестування продуктивності підтвердило час відгуку основних операцій до 2 секунд при навантаженні до 100 одночасних користувачів.

Сформульовано рекомендації щодо подальшого розвитку системи. До перспектив удосконалення віднесено впровадження двофакторної автентифікації, додаткових механізмів безпеки, оптимізацію аналітичних звітів, розширення експорту даних, реалізацію пакетних операцій із завданнями, створення мобільних додатків та інтеграцію із зовнішніми календарними сервісами.

Розроблений електронний сервіс управління завданнями є готовим до впровадження програмним продуктом, який може використовуватися малими та середніми підприємствами для організації командної роботи, розподілу завдань, контролю виконання та аналізу робочих процесів. Сервіс забезпечує можливість розгортання на власній інфраструктурі, підтримує українську мову, не потребує підписки на зовнішні комерційні платформи та може бути основою для подальшого розвитку подібних інформаційних систем.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016. 16 с.
2. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI (зі змінами та доповненнями). Відомості Верховної Ради України. 2010. № 34. Ст. 481. URL: <https://zakon.rada.gov.ua/laws/show/2297-17> (дата звернення: 22.04.2026).
3. Карпенко В. М. Інформаційні системи в управлінні підприємством: підручник. Львів: Видавництво Львівської політехніки, 2021. 384 с.
4. Мартинюк І. В., Литвиненко О. В. Сучасні підходи до проєктування та розробки веб-застосунків. Вісник Національного технічного університету України «КПІ». Серія: Інформатика, управління та обчислювальна техніка. 2023. № 80. С. 45–58.
5. Семенов А. О. Електронні сервіси та цифрова трансформація бізнесу: навч. посібник. Київ: Видавничий дім «Кондор», 2022. 312 с.
6. Степаненко О. П., Бондаренко М. І. Менеджмент проєктів та управління завданнями у сучасних організаціях. Економіка та управління підприємствами. 2023. № 4 (78). С. 124–138.
7. Шевченко О. М. Об'єктно-орієнтоване програмування на платформі Java: підручник. Київ: Видавничий центр «Академія», 2022. 296 с.
8. Anderson D. J. Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2020. 280 p.
9. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley Professional, 2021. 464 p.
10. Black R. Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. 4th ed. John Wiley & Sons, 2022. 672 p.

11. Bughin J., Chui M., Manyika J. The future of work: Reskilling and remote working to recover in the new normal. McKinsey Global Institute, 2023. URL: <https://www.mckinsey.com/featured-insights/future-of-work> (дата звернення: 15.04.2026).
12. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 7th ed. Pearson Education, 2022. 1440 p.
13. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. MIT Press, 2022. 1312 p.
14. Crispin L., Gregory J. Agile Testing: A Practical Guide for Testers and Agile Teams. 2nd ed. Addison-Wesley Professional, 2023. 576 p.
15. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. University of California, Irvine, 2000. 162 p. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (дата звернення: 28.04.2026).
16. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. International Organization for Standardization, 2018. 104 p.
17. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. New Riders Publishing, 2020. 216 p.
18. Masse M. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. O'Reilly Media, 2021. 116 p.
19. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. John Wiley & Sons, 2022. 256 p.
20. OMG. Unified Modeling Language Specification, Version 2.5.1. Object Management Group, 2017. 796 p. URL: <https://www.omg.org/spec/UML/2.5.1> (дата звернення: 28.04.2026).
21. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group, 2024. URL: <https://www.postgresql.org/docs/16> (дата звернення: 28.04.2026).

22. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill Education, 2023. 976 p.
23. Richardson C. Microservices Patterns: With Examples in Java. Manning Publications, 2023. 520 p.
24. Sommerville I. Software Engineering. 11th ed. Pearson Education, 2022. 816 p.
25. Walls C. Spring in Action. 7th ed. Manning Publications, 2024. 624 p.

ДОДАТОК А



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ
ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ



«Е-СЕРВІС УПРАВЛІННЯ ЗАВДАННЯМИ У СФЕРІ БІЗНЕС-МЕНЕДЖМЕНТУ НА ПЛАТФОРМІ JAVA ТА SPRING BOOT»

Виконала студентка 4 курсу
Групи ТЦР-42
Дорошенко Юлія Володимирівна
Керівник Роботи
Д-р техн. наук, професор, зав. кафедри ТЦР Жебка Вікторія Вікторівна

Київ - 2026

1

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – підвищення ефективності організації та контролю робочих процесів у сфері бізнес-менеджменту за допомогою розробленого електронного сервісу управління завданнями, реалізованого на основі Java з використанням фреймворку Spring Boot.
- **Об'єкт дослідження** – процеси управління завданнями та координації командної роботи у сфері бізнес-менеджменту.
- **Предмет дослідження** – методи, моделі та технології розроблення електронних сервісів управління завданнями на платформі Java та Spring Boot.

2

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область — сервіси управління завданнями у сфері бізнес-менеджменту.
2. Дослідити існуючі аналоги систем управління завданнями та визначити їх переваги й недоліки.
3. Сформулювати функціональні та нефункціональні вимоги до е-сервісу управління завданнями.
4. Спроекувати архітектуру системи, структуру бази даних та UML-діаграми основних процесів.
5. Реалізувати серверну частину системи на платформі Java з використанням Spring Boot, Spring Security та PostgreSQL.
6. Реалізувати клієнтську частину е-сервісу з канбан-дошкою, Drag and Drop, формами керування завданнями та адаптивним інтерфейсом.
7. Розробити REST API для взаємодії клієнтської та серверної частин системи.
8. Виконати тестування програмного забезпечення та перевірити коректність роботи основних функцій системи.

3

АНАЛІЗ АНАЛОГІВ

Функціональна можливість	<u>Trello</u>	<u>Asana</u>	<u>Jira</u>	<u>Mondav.com</u>	<u>MS To Do</u>	Розроблена платформа
Канбан-дошки	+	+	+	+	—	+
Підзавдання	обмеж.	+	+	+	+	+
Система ролей	базова	+	розширена (платно)	+	базова	базова
Прикріплення файлів	+ (10 МБ)	+ (100 МБ)	+ (10 МБ)	+ (500 МБ)	+ (25 МБ)	+ (10 МБ)
Коментарі до завдань	+	+	+	+	—	+
Email-сповіщення	+	+	+	+	+	+
Аналітичні звіти	—	+	+	+	—	+
Drag and Drop	+	+	+	+	+	+
Простота використання для новачків	середня	середня	низька	середня	висока	висока
Додавання власних статусів	обмеж.	+	+	+	—	+

4

АКТУАЛЬНІСТЬ

- Актуальність створення електронного сервісу управління завданнями зумовлена зростанням потреби у простих та ефективних цифрових інструментах для організації командної роботи. Багато існуючих систем мають складний інтерфейс і потребують тривалого освоєння, що ускладнює їх використання для нових користувачів та невеликих команд.
- Розроблення власного веб-застосунку дозволяє поєднати необхідний функціонал управління завданнями, контролю виконання, командної взаємодії та аналітики із зрозумілим інтерфейсом і швидким освоєнням системи. Це підвищує ефективність організації робочих процесів, зменшує втрату інформації та спрощує координацію роботи команди.

5

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги

1. Реєстрація та автентифікація користувачів.
2. Створення, редагування, перегляд та видалення завдань (CRUD).
3. Канбан-дошка з підтримкою Drag and Drop.
4. Динамічні статуси завдань у межах груп.
5. Розмежування прав доступу за ролями.
6. Підтримка підзавдань.
7. Коментарі до завдань.
8. Email-сповіщення про зміни у завданнях.
9. Формування статистики та аналітичних звітів.
10. Прикріплення файлів до завдань.
11. Автоматичні нагадування про дедлайни.
12. Редагування профілю користувача.

Нефункціональні вимоги

1. Шифрування паролів за алгоритмом BCrypt.
2. Захист від CSRF та несанкціонованого доступу на основі Spring Security.
3. Стабільна робота при одночасній роботі 100 користувачів.
4. Час відгуку основних операцій — до 2 секунд.
5. Обмеження розміру файлів до 10 МБ.
6. Адаптивність інтерфейсу для ПК, планшетів та телефонів за допомогою Bootstrap.
7. Підтримка Google Chrome, Firefox, Edge та Safari.

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



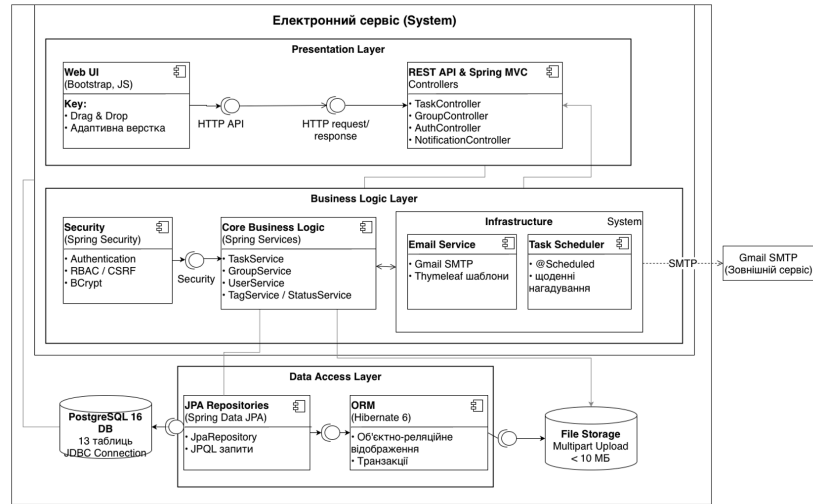
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



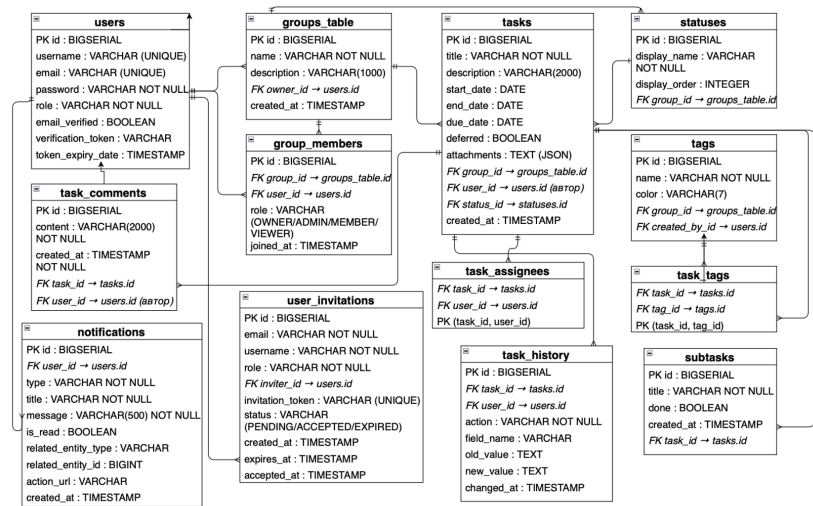
8

ДІАГРАМА КОМПОНЕНТІВ АРХІТЕКТУРИ



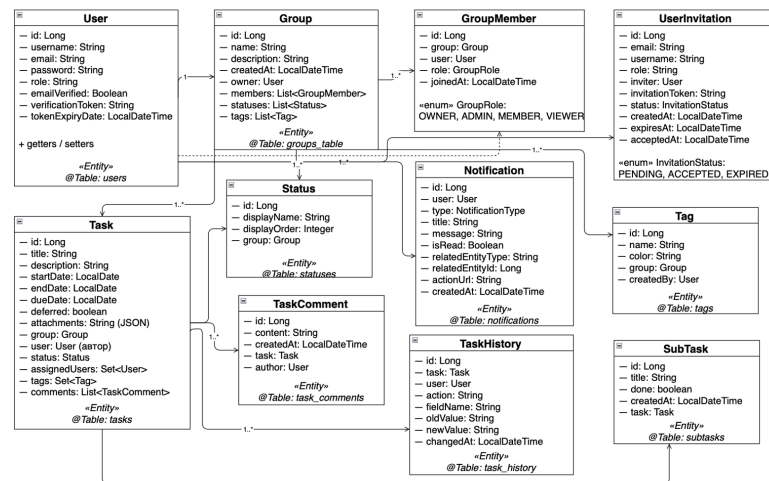
9

ДІАГРАМА СТРУКТУРИ БАЗИ ДАНИХ



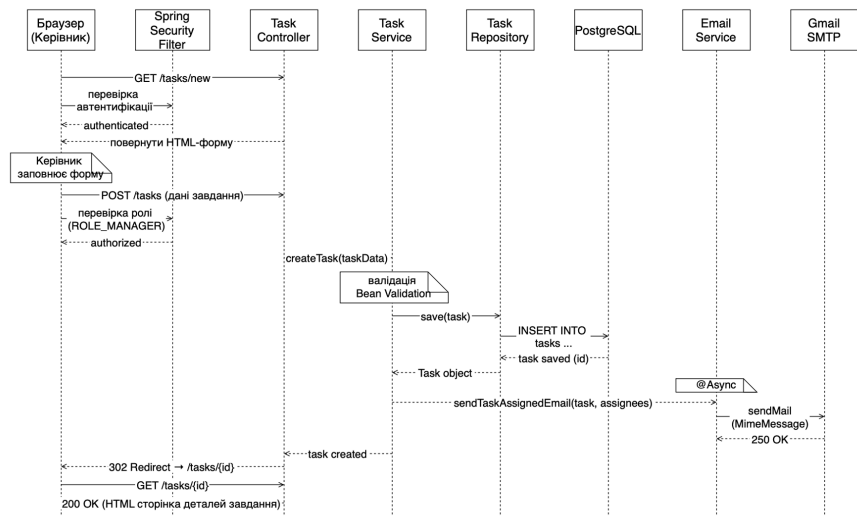
10

ДІАГРАМА КЛАСІВ ДОМЕННОЇ МОДЕЛІ ЕЛЕКТРОННОГО СЕРВІСУ



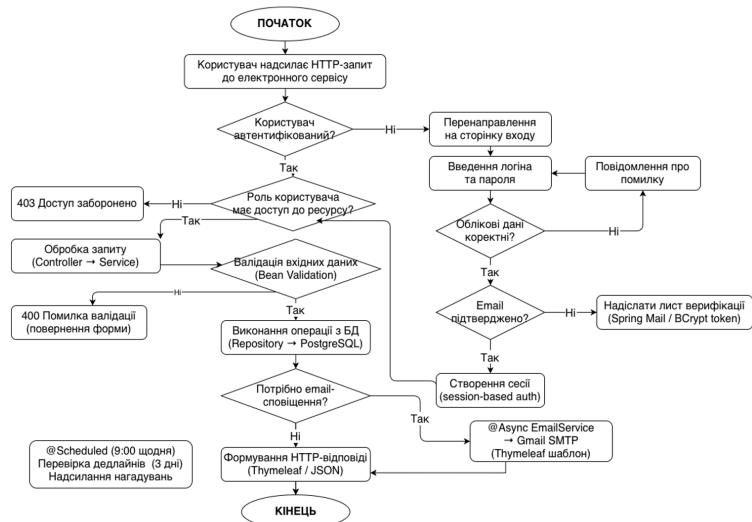
11

ДІАГРАМА ПОСЛІДОВНОСТІ СТВОРЕННЯ ЗАВДАННЯ



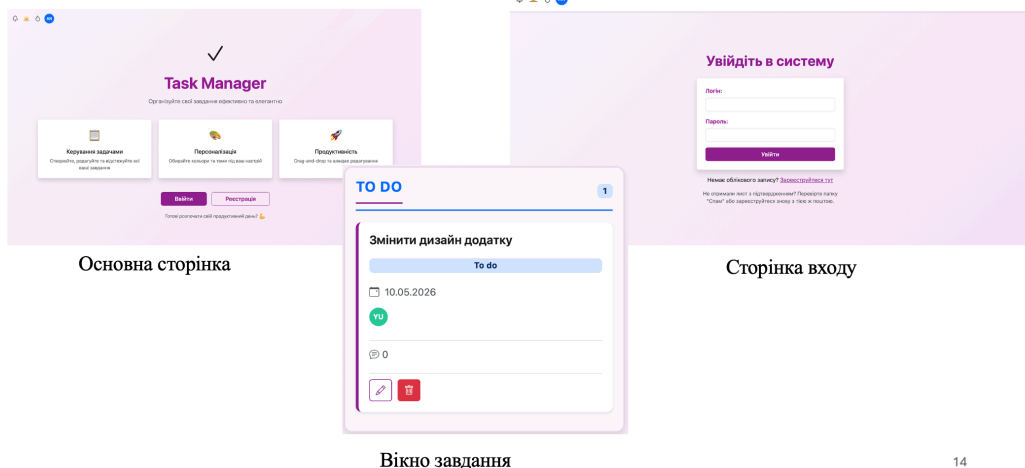
12

БЛОК-СХЕМА АЛГОРИТМУ РОБОТИ СЕРВІСУ



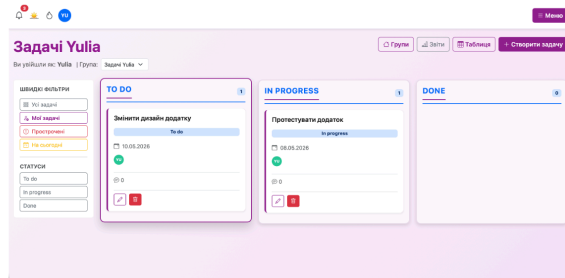
13

ЕКРАННІ ФОРМИ

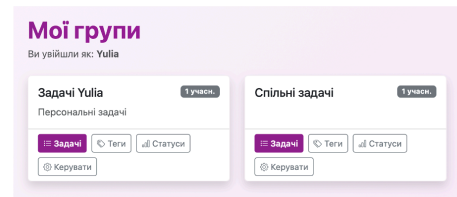


14

ЕКРАННІ ФОРМИ



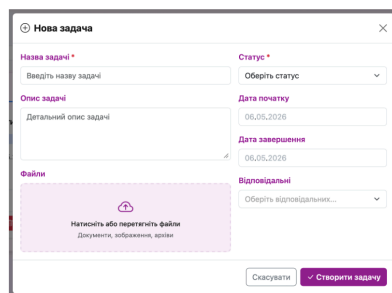
Канбан дошка



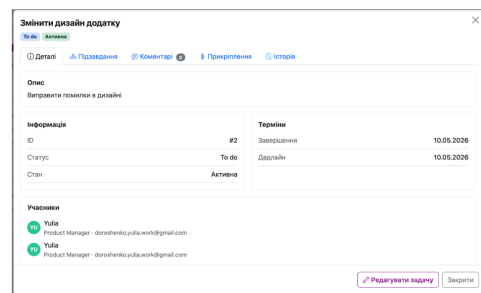
Сторінка груп

15

ЕКРАННІ ФОРМИ



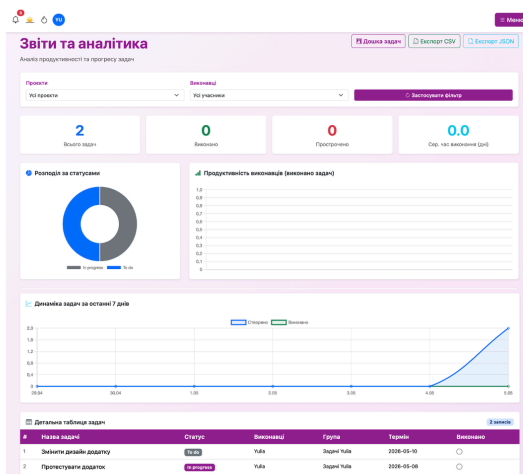
Форма створення задачі



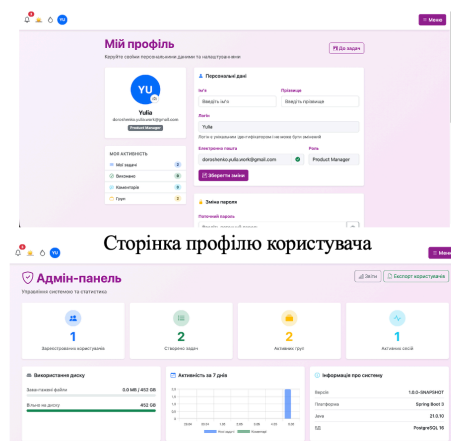
Форма деталізації завдання

16

ЕКРАННІ ФОРМИ



Екран аналітики

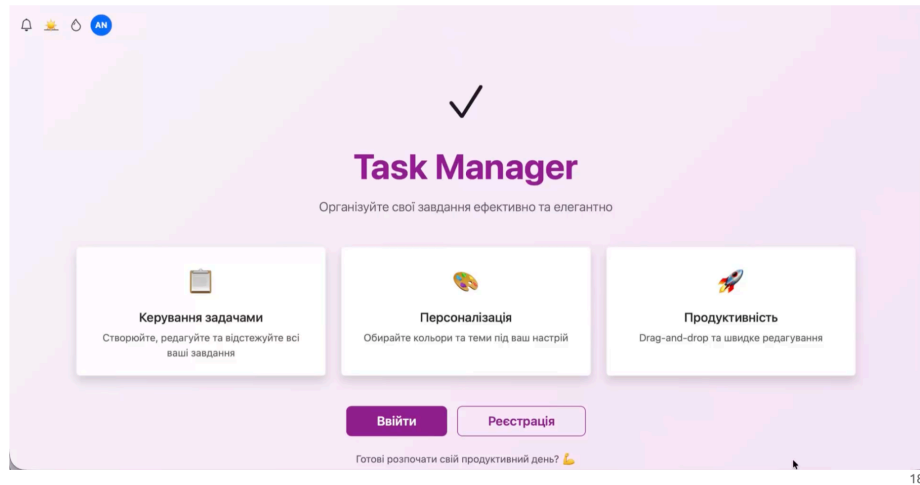


Сторінка профілю користувача

Адміністративна панель

17

ВІДЕО ПРЕЗЕНТАЦІЯ ПРОЄКТУ



18

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Дорошенко Ю.В. АРХІТЕКТУРНІ ПАТЕРНИ ПОБУДОВИ Е-СЕРВІСУ УПРАВЛІННЯ ЗАВДАННЯМИ НА ОСНОВІ SPRING BOOT 3.5 ТА JAVA 25 // Матеріали всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та депентралізація». Збірник тез. К.: ДУІКТ, 2026, (подано до друку)
2. Дорошенко Ю.В. Е-СЕРВІС УПРАВЛІННЯ ЗАВДАННЯМИ У СФЕРІ БІЗНЕС-МЕНЕДЖМЕНТУ НА ПЛАТФОРМІ JAVA ТА SPRING BOOT // Матеріали VI Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. К.: ДУІКТ, 2026, (подано до друку)

19

ВИСНОВКИ

1. Проведено аналіз предметної області — досліджено актуальність електронних сервісів управління завданнями у сфері бізнес-менеджменту та визначено потребу у власному self-hosted рішенні.
2. Проаналізовано існуючі аналоги систем управління завданнями, зокрема Trello, Asana, Jira, Monday.com та Microsoft To Do; виявлено їх переваги, недоліки та обмеження для українського бізнес-середовища.
3. Сформовано функціональні та нефункціональні вимоги до е-сервісу: керування завданнями, ролями користувачів, канбан-дошкою, підзавданнями, тегами, файлами, коментарями та email-сповіщеннями.
4. Спроектовано тривірневу клієнт-серверну архітектуру системи, ER-діаграму бази даних із 13 основними таблицями та UML-діаграми основних процесів.
5. Реалізовано серверну частину на Java 24 та Spring Boot 3.5.0 із використанням Spring MVC, Spring Data JPA, Spring Security, Spring Mail, Spring Scheduler, PostgreSQL та Hibernate.
6. Реалізовано клієнтську частину з канбан-дошкою, Drag and Drop, адаптивним інтерфейсом Bootstrap, Thymeleaf-шаблонами та REST API для асинхронної взаємодії з сервером.
7. Проведено комплексне тестування системи: виконано 188 тестів, досягнуто покриття коду 78,4%, перевірено продуктивність, безпеку, функціональність і зручність використання.
8. Підтверджено практичну цінність розробленого е-сервісу, який може використовуватися малими та середніми підприємствами для організації командної роботи без залежності від зовнішніх SaaS-платформ.

20

ДЯКУЮ ЗА УВАГУ!

21

ДОДАТОК Б

Таблиця 2.1

Технологічний стек електронного сервісу

Категорія	Технологія	Версія	Призначення у системі
Мова програмування	Java / JDK	24	Серверна частина застосунку
Фреймворк	Spring Boot	3.5.0	Платформа розроблення
Веб-модуль	Spring MVC	6.x	Обробка HTTP-запитів
Доступ до даних	Spring Data JPA	3.x	Робота з базою даних
ORM	Hibernate	6.x	Об'єктно-реляційне відображення
Безпека	Spring Security	6.x	Автентифікація, авторизація
Email	Spring Mail	6.x	Розсилка повідомлень
Валідація	Spring Validation	3.x	Перевірка вхідних даних
Планувальник	Spring Scheduler	6.x	Періодичні задачі
Шаблонізатор	Thymeleaf	3.1.x	Серверний рендеринг HTML
СУБД	PostgreSQL	16+	Зберігання даних
Розмітка	HTML5	—	Структура веб-сторінок
Стили	CSS3 / Bootstrap	5.3.3	Адаптивна верстка
Іконки	Bootstrap Icons	1.11.3	Графічні елементи
Клієнтська логіка	Vanilla JavaScript	ES6+	Інтерактивність
Drag and Drop	HTML5 DnD API	—	Канбан-дошка
Збирання	Gradle	8.x	Управління залежностями
Тестування	JUnit	5	Модульні тести
SMTP	Gmail SMTP	—	Сервер електронної пошти

ДОДАТОК В

Таблиця 3.1

Основні ендпоінти REST API

HTTP-метод	URI	Призначення
GET	/api/tasks/{id}	Отримання деталей конкретного завдання
GET	/api/tasks/{id}/comments	Отримання списку коментарів до завдання
POST	/api/tasks/{id}/comments	Додавання коментаря до завдання
PUT	/api/tasks/{taskId}/comments/{commentId}	Редагування власного коментаря (до 15 хв після створення)
DELETE	/api/tasks/{taskId}/comments/{commentId}	Видалення власного коментаря (до 15 хв після створення)
DELETE	/api/tasks/{taskId}/attachments/{fileName}	Видалення прикріпленого файлу
GET	/api/tasks/{id}/subtasks	Отримання списку підзавдань
POST	/api/tasks/{id}/subtasks	Додавання нового підзавдання
PUT	/api/tasks/{taskId}/subtasks/{subId}/toggle	Перемикання прапорця виконання підзавдання
DELETE	/api/tasks/{taskId}/subtasks/{subId}	Видалення підзавдання
GET	/api/tasks/{id}/history	Отримання історії змін завдання

ДОДАТОК Г

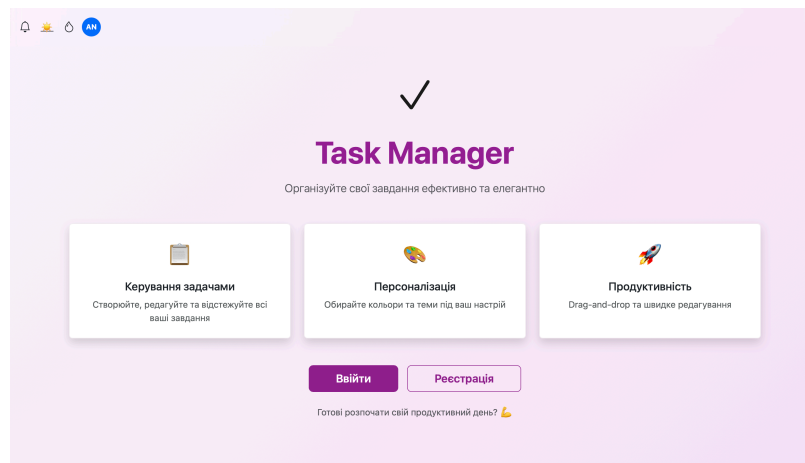


Рис. 3.2 – Стартова сторінка електронного сервісу

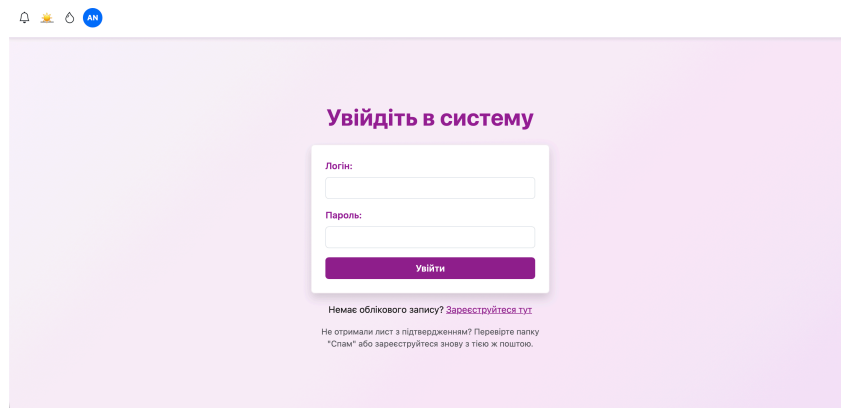


Рис. 3.3 – Форма входу до системи

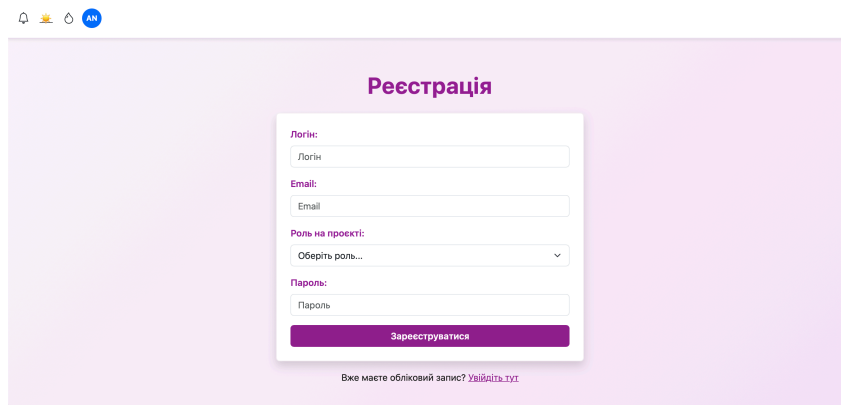


Рис. 3.4 – Форма реєстрації нового користувача

Рис. 3.7 – Сторінка створення нового завдання

Змінити дизайн додатку

To do

Активна

Деталі

Підзадання

Коментарі

Прикріплення

Історія

Опис

Виправити помилки в дизайні

Інформація

ID

#2

Статус

To do

Стан

Активна

Терміни

Завершення

10.05.2026

Дедлайн

10.05.2026

Учасники

Yulia

Product Manager - doroshenko.yulia.work@gmail.com

Yulia

Product Manager - doroshenko.yulia.work@gmail.com

Редагувати задачу

Закрити

Рис. 3.8 – Сторінка деталей завдання

Мої групи

Ви увійшли як: Yulia

Задачі Yulia

1 учасн.

Персональні задачі

Задачі

Теги

Статуси

Керувати

Спільні задачі

1 учасн.

Задачі

Теги

Статуси

Керувати

Рис. 3.9 – Сторінка списку проєктів

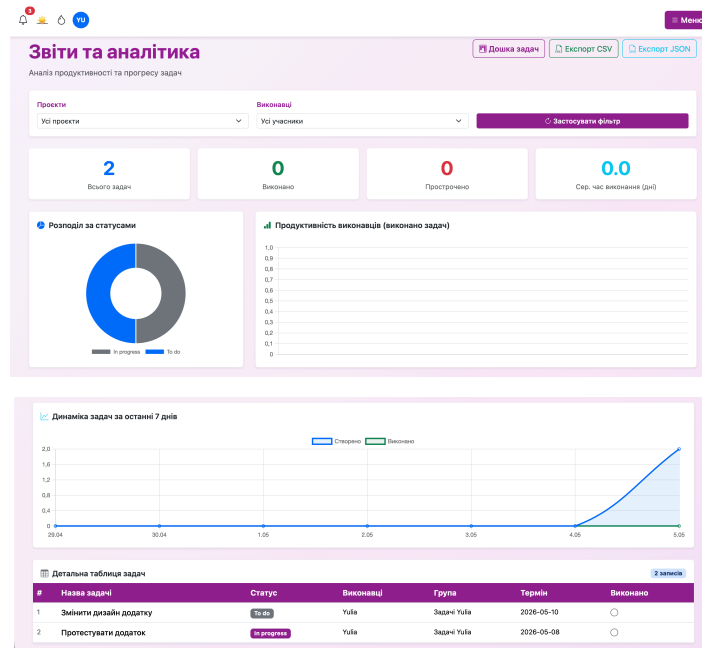


Рис. 3.10 – Сторінка аналітичних звітів

Мій профіль
Керуйте своїми персональними даними та налаштуваннями

Персональні дані

Ім'я: Введіть ім'я | Прізвище: Введіть прізвище

Логін: Yulia
Логін є унікальним (ідентифікатором) і не може бути змінений

Електронна пошта: doroshenko.yulia.work@gmail.com | Роль: Product Manager

Зміна пароля

Поточний пароль: | Введіть поточний пароль

Новий пароль: | Підтвердження пароля:

Налаштування сповіщень

Призначено задачу: ☒ Сповіщення при призначенні вас виконавцем

Зміна статусу задачі: ☒ Сповіщення при зміні статусу ваших задач

Нагадування про дедлайни: ☒ За 24 години до закінчення терміну

Нові коментарі: ☐ Коментарі до задач, де ви виконавець або автор

Запрошення до груп: ☒ Сповіщення при отриманні запрошення

Рис. 3.11 – Сторінка особистого профілю користувача

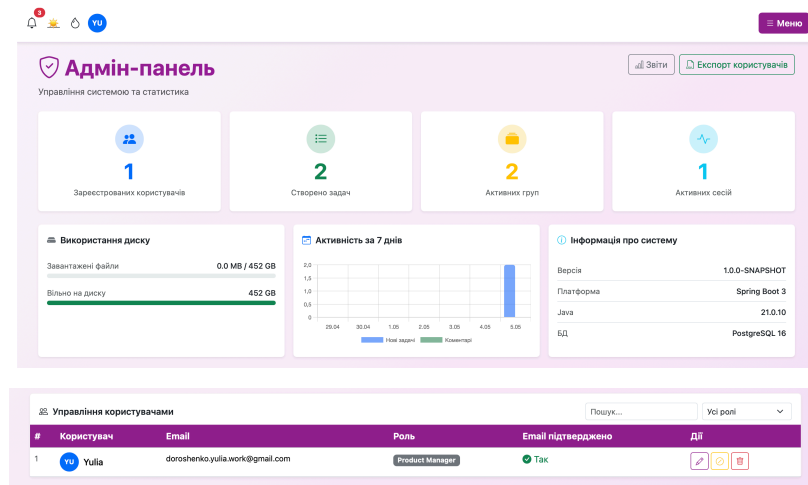


Рис. 3.12 – Адміністративна панель

ДОДАТОК Д

Таблиця 4.1

Результати тестування функцій автентифікації та реєстрації

ID	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
ТС-AUTH-01	Реєстрація з валідними даними	Створення облікового запису, надсилання листа підтвердження email	Обліковий запис створено, відправлено лист підтвердження	Пройдено
ТС-AUTH-02	Реєстрація з дубльованим email	Помилка «Користувач з таким email вже існує»	Відображено відповідне повідомлення про помилку	Пройдено
ТС-AUTH-03	Реєстрація з невідповідністю паролів	Помилка «Паролі не співпадають»	Відображено повідомлення про невідповідність	Пройдено
ТС-AUTH-04	Реєстрація зі слабким паролем (5 символів)	Помилка «Пароль повинен містити мін. 8 символів»	Форма не подається, відображено повідомлення	Пройдено
ТС-AUTH-05	Підтвердження email за валідним токеном	Поле emailVerified встановлюється у true, токен очищується	Email підтверджено, користувач перенаправлений на сторінку входу	Пройдено
ТС-AUTH-06	Підтвердження email за простроченим токеном	Помилка «Термін дії посилання вичерпано», пропозиція повторного надсилання	Відображено повідомлення про прострочення	Пройдено
ТС-AUTH-07	Вхід з валідними даними	Створення сесії, перенаправлення на головну сторінку	Успішний вхід, відкривається канбан-дошка	Пройдено
ТС-AUTH-08	Вхід з некоректним паролем	Помилка «Невірний email або пароль»	Відображено загальне повідомлення про помилку	Пройдено
ТС-AUTH-09	Вхід з неіснуючим email	Помилка «Невірний email або пароль»	Відображено те саме загальне повідомлення	Пройдено
ТС-AUTH-10	Вихід з системи	Завершення сесії, перенаправлення на сторінку входу	Сесія завершена, доступ до захищених сторінок неможливий	Пройдено

Таблиця 4.2

Результати тестування функцій управління завданнями

ID	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
ТС-TASK-01	Створення завдання з валідними даними	Завдання створено, відображається на канбан-дошці	Завдання створено зі статусом «Заплановано» (першим статусом групи)	Пройдено
ТС-TASK-02	Створення завдання без обов'язкових полів	Помилки валідації біля порожніх полів	Форма не подається, поле «Назва» виділено червоним	Пройдено
ТС-TASK-03	Створення завдання з минулою датою	Помилка валідації терміну виконання	Відображено повідомлення «Термін повинен бути у майбутньому»	Пройдено
ТС-TASK-04	Призначення виконавця при створенні	Виконавець отримує email-сповіщення	Лист отримано виконавцем протягом 1 хв	Пройдено
ТС-	Створення	Підзавдання	Підзавдання відображається	Пройдено

TASK-05	підзавдання	прив'язується до батьківського	у вкладці «Підзавдання»	
TC-TASK-06	Редагування завдання керівником	Зміни зберігаються та відображаються	Дані оновлено, поле updated_at оновлено	Пройдено
TC-TASK-07	Спроба редагування завдання чужого проекту	Відмова у доступі (403)	Відображено сторінку «Доступ заборонено»	Пройдено
TC-TASK-08	Видалення завдання керівником	Завдання та пов'язані дані видалено	Завдання видалено разом з коментарями та файлами	Пройдено
TC-TASK-09	Перегляд деталей завдання	Відображення повної інформації про завдання	Сторінка деталей містить усі дані та секції	Пройдено
TC-TASK-10	Додавання тегу до завдання	Тег з'являється на картці із заданим кольором	Картка оновлюється з новим тегом	Пройдено
TC-TASK-11	Завдання з простроченим терміном	Картка виділяється червоною рамкою	Прострочені завдання виділяються візуально	Пройдено

Таблиця 4.3

Результати тестування канбан-дошки

ID	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
TC-KANBAN-01	Перетягування картки між першим та другим статусами групи	Статус оновлюється на сервері та в UI	Картка переміщена, AJAX-запит виконано	Пройдено
TC-KANBAN-02	Перетягування картки Перетягування картки у фінальний статус групи	Завдання позначається як виконане	Статус оновлено, час виконання зафіксовано	Пройдено
TC-KANBAN-03	Перетягування картки виконавцем чужого завдання	Картка повертається у вихідну позицію	Помилка 403, картка повертається назад	Пройдено
TC-KANBAN-04	Фільтрація завдань за виконавцем	Відображаються тільки завдання вибраного виконавця	Фільтр працює коректно, інші завдання приховано	Пройдено
TC-KANBAN-05	Текстовий пошук за назвою завдання	Відображення завдань, що містять текст	Пошук працює без урахування реєстру	Пройдено
TC-KANBAN-06	Фільтрація за тегами	Відображаються завдання вибраного тегу	Фільтр коректно обмежує перелік	Пройдено
TC-KANBAN-07	Комбінування декількох фільтрів	Застосовуються усі активні фільтри одночасно	Логічний AND між фільтрами працює правильно	Пройдено
TC-KANBAN-08	Скидання усіх фільтрів	Відображаються всі завдання користувача	Фільтри скинуто, повний перелік відновлено	Пройдено
TC-KANBAN-09	Лічильники над колонками	Кількість завдань у кожній колонці актуальна	Лічильники оновлюються при переміщенні	Пройдено
TC-KANBAN-10	Адаптивність на мобільному пристрої	Колонки прокручуються горизонтально	Інтерфейс зберігає функціональність на 320px	Пройдено

Таблиця 4.4

Результати тестування комунікаційних функцій

ID	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
ТС-COMM-01	Додавання коментаря до завдання	Коментар з'являється у стрічці без перезавантаження	Асинхронне додавання працює коректно	Пройдено
ТС-COMM-02	Редагування власного коментаря	Коментар оновлюється, з'являється позначка «змінено»	Редагування доступне протягом 15 хв	Пройдено
ТС-COMM-03	Спроба редагування чужого коментаря	Кнопки редагування недоступні	Кнопки приховано для чужих коментарів	Пройдено
ТС-COMM-04	Видалення власного коментаря	Коментар видалено зі стрічки	Видалення відбувається без перезавантаження	Пройдено
ТС-COMM-05	Завантаження файлу до 10 МБ	Файл успішно прикріплюється до завдання	Файл збережено, з'явився у переліку	Пройдено
ТС-COMM-06	Спроба завантаження файлу понад 10 МБ	Помилка «Файл перевищує допустимий розмір»	Завантаження відхилено, показано повідомлення	Пройдено
ТС-COMM-07	Завантаження файлу недозволеного типу (.exe)	Помилка «Тип файлу не підтримується»	Завантаження відхилено на серверній стороні	Пройдено
ТС-COMM-08	Завантаження файлу із кириличною назвою	Назва зберігається коректно	Файл зберігається, назва відображається	Пройдено
ТС-COMM-09	Завантаження прикріплення на пристрій	Файл завантажується з оригінальною назвою	Файл коректно отримано із заголовком Content-Disposition	Пройдено
ТС-COMM-10	Видалення прикріплення власником	Файл видалено з системи та диску сервера	Файл та запис у БД видалено	Пройдено

Таблиця 4.5

Результати тестування системи сповіщень

ID	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
ТС-MAIL-01	Сповіщення про створення завдання	Виконавець отримує лист протягом 1 хв	Лист отримано через 30 секунд	Пройдено
ТС-MAIL-02	Сповіщення при зміні виконавця	Новий виконавець отримує лист	Лист коректно надіслано новому виконавцю	Пройдено
ТС-MAIL-03	Щоденне нагадування про термін	Лист о 9:00 для завдань з терміном через 3 дні	Розсилка виконана у запланований час	Пройдено
ТС-MAIL-04	Запобігання повторним нагадуванням	Лист не надсилається двічі для одного завдання	Поле deferred та фінальний статус блокують повторне надсилання	Пройдено
ТС-MAIL-05	Скидання прапорця при зміні терміну	Завдання знову може отримати нагадування	Прапорець коректно скидається	Пройдено
ТС-MAIL-06	Сповіщення про прострочене завдання	Лист отримує виконавець та автор завдання	Обидва отримувача отримали повідомлення	Пройдено

TC-MAIL-07	Лист підтвердження email при реєстрації	Лист містить унікальне посилання з токеном верифікації	Посилання має обмежений термін дії токена	Пройдено
TC-MAIL-08	Локалізація вмісту листа	Лист українською мовою	Усі тексти відображаються коректно	Пройдено
TC-MAIL-09	HTML-форматування листа	Лист коректно відображається в Gmail, Outlook	Шаблон Thymeleaf рендериться правильно	Пройдено

Таблиця 4.6

Результати тестування адміністративних функцій

ID	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
TC-ADMIN-01	Доступ адміністратора до /admin	Відкривається панель адміністрування	Сторінка відображається коректно	Пройдено
TC-ADMIN-02	Спроба доступу звичайним користувачем	Відмова у доступі (403)	Відображено сторінку «Доступ заборонено»	Пройдено
TC-ADMIN-03	Перегляд переліку користувачів	Таблиця з усіма обліковими записами	Дані відображаються з пагінацією	Пройдено
TC-ADMIN-04	Деактивація облікового запису	Користувач не може увійти	Сесія завершена, вхід заблоковано	Пройдено
TC-ADMIN-05	Призначення ролі ROLE_MANAGER	Користувач отримує права керівника	Нові права доступні після повторного входу	Пройдено
TC-ADMIN-06	Скасування ролі ROLE_MANAGER	Втрата прав керівника	Адміністративні функції стають недоступні	Пройдено
TC-ADMIN-07	Експорт переліку користувачів у CSV	Завантаження файлу з усіма даними	CSV-файл коректно сформовано та завантажено	Пройдено
TC-ADMIN-08	Перегляд журналу подій системи	Відображення останніх подій з фільтрами	Журнал містить дії користувачів	Пройдено
TC-ADMIN-09	Моніторинг ключових показників	Актуальна статистика на дашборді	Показники оновлюються у реальному часі	Пройдено

Таблиця 4.7

Результати тестування продуктивності

Операція	10 користувачів	50 користувачів	100 користувачів	200 користувачів
Завантаження дашборду	245 мс	312 мс	487 мс	1124 мс
Створення завдання	178 мс	234 мс	356 мс	892 мс
Зміна статусу (PATCH)	89 мс	124 мс	198 мс	543 мс
Завантаження файлу 5 МБ	1245 мс	1689 мс	2342 мс	4567 мс
Формування звіту	567 мс	892 мс	1456 мс	3214 мс
Пошук завдань	134 мс	187 мс	289 мс	678 мс