Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))
Кафедра інформаційних систем і комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення системи контролю виконання завдань "DutyDesk" із використанням Spring Boot та Angular»

Виконав: студент *IV* курсу групи *ІСТ-41*
спеціальності 126 "Інформаційні системи
і технології"
(шифр і назва напряму підготовки, спеціальності)

Цяцяк Орест Володимирович
(прізвище та ініціали)

Керівник *Павлюк У.В.*
(прізвище та ініціали)

Рецензент Процик Ю. С.
(прізвище та ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІСКМ

_____ Сторожук О.Л.
„15" 11 2024 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Цяцяку Оресту Володимировичу
(прізвище, ім'я, по батькові)

1. Тема бакалаврської роботи: *"Розроблення системи контролю виконання завдань "DutyDesk" із використанням Spring Boot та Angular"*
керівник роботи *Павлюк Уляна Володимирівна, к.е.н.,*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від *«15» листопада 2024 року № С-884*
2. Термін подання студентом роботи 12.06.2025 р.
3. Вихідні дані до проекту (роботи) *Ознайомитися із проблемною областю; розробити сервер та односторінковий сайт для зручного контролю завдань; провести тестування роботи розробленого проекту.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
Стан предметної області
Інформаційне забезпечення
Програмне та технічне забезпечення
Висновки
Список використаних джерел
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація із висвітленням ключових моментів розробки системи в розрізі розділів пояснювальної записки (див. п.4)
6. Дата видачі завдання *16 листопада 2024 року.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Планування та архітектура	18.11.2024 р.	*Виконано*
2	Налаштування проєкту	21.11.2024 р.	*Виконано*
3	Створення структури бази даних	23.11.2024 р.	*Виконано*
6	Реалізація серверної частини	31.12.2024 р.	*Виконано*
7	Реалізація вебінтерфейсу	17.02.2025 р.	*Виконано*
8	Докеризація проєктів сервера та інтерфейсу	25.02.2025 р.	*Виконано*
9	Деплой з використанням GCP	05.03.2025 р.	*Виконано*
10	Підготовка розділу 1	20.05.2025 р.	*Виконано*
11	Підготовка розділу 2	05.06.2025 р.	*Виконано*
12	Підготовка розділу 3 та висновків	10.06.2025 р.	*Виконано*
13	Перевірка пояснювальної записки на наявність плагіату	19.06.2025 р.	*Виконано*
14	Рецензування пояснювальної записки	20.06.2025 р.	*Виконано*
15	Підготовка презентаційних матеріалів та доповіді	20.06.2025 р.	*Виконано*

Студент _____ (підпис) *Цяцяк О.В.* (прізвище та ініціали)

Керівник роботи _____ (підпис) *Павлюк У.В.* (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить *48* сторінок пояснювальної записки, *28* рисунків, *2* додатки та *15* використаних джерел.

Ця робота включає розробку браузерної системи контролю виконання завдань “DutyDesk”. У проєкті були використані сучасні вебтехнології: Java, Spring Boot, Spring Data JPA, Spring Web, Spring Security, RESTful APIs, OpenAPI/Swagger, PostgreSQL, Maven, OOP, Git, CSS, HTML, JavaScript, Typescript, Angular, Angular Material, SCSS, Google Cloud Platform (GCP), Docker, Nginx. Мета системи – контроль виконання службових обов'язків та доручень.

Ключові слова*: Angular, інформаційна система, вебзастосунок, RESRful APIs, Spring Boot.*

ABSTRACT

The diploma thesis contains *48* pages of explanatory notes, *28* figures, *2* appendices and *15* sources.

This work includes the development of a browser-based task management system “DutyDesk”. This project used modern web technologies: Java, Spring Boot, Spring Data JPA, Spring Web, Spring Security, RESTful APIs, OpenAPI/Swagger, PostgreSQL, Maven, OOP, Git, CSS, HTML, JavaScript, Typescript, Angular, Angular Material, SCSS, Google Cloud Platform (GCP), Docker, Nginx. The purpose of the system is to monitor the performance of official duties and assignments.

Keywords*: Angular, information system, web application, RESRful APIs, Spring Boot.*

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити систему контролю виконання завдань “DutyDesk” з можливістю перегляду, додавання, вилучення, підтвердження та створення по шаблону фактів виконання обов’язків. Мета системи – контроль виконання службових обов'язків та доручень.

Для забезпечення мінімально необхідного функціоналу реалізувати наступне:

1. *Сторінка виконавців*
   - 1.1. Перегляд виконавців
   - 1.2. Перегляд трьох останніх непідтверджених завдання інших користувачів
   - 1.3. Перегляд трьох останніх незакінчених власних завдань
2. *Сторінка завдань*
   - 2.1. Детальний перегляд завдань інших користувачів або власних
   - 2.2. Фіксація виконання власних завдань
   - 2.3. Видалення запису про виконання завдання
     - 2.3.1. Неможливість видалити підтверджений запис про виконання завдання
     - 2.3.2. Можливість видалення власного не підтвердженого запису про виконання для звичайного користувача
     - 2.3.3. Можливість видалення любого не підтвердженого запису про виконання для адміністратора
   - 2.4. Зміна статусу власного завдання
   - 2.5. Перегляд найвикористовуваніших шаблонів
   - 2.6. Зручний пошук шаблонів
3. *Сторінка шаблонів для завдань*
   - 3.1. Перегляд шаблонів для всіх користувачів
   - 3.2. Створення шаблонів лише для адміністратора
   - 3.3. Модифікація шаблонів лише для адміністратора
   - 3.4. Видалення шаблонів лише для адміністратора
4. *Сторінка логіну та реєстрації*
   - 4.1. Реєструвати може лише адміністратор
5. *Поп-ап зміни власного паролю*
6. *Анонімний користувач*
   - 6.1. Вхід відбувається через особливе посилання
   - 6.2. Всі чутливі дані інших користувачів мають бути конфіденціалізуватися.

В проекті рекомендовано використати такі технології як: Java, Spring Boot, Spring Data JPA, Spring Web, Spring Security, RESTful APIs, OpenAPI/Swagger, SQL, PostgreSQL, Maven, OOP, Git, CSS, HTML, JavaScript, Typescript, Angular, Angular Material, SCSS, Google Cloud Platform (GCP), Docker, Nginx.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

AGILE – гнучка методологія

CI/CD – автоматизоване розгортання

CLI – консольний інтерфейс

JSON – текстовий формат

REST – архітектурний стиль

RESTFUL – стандартний інтерфейс

SCRUM – командний фреймворк

SPA – односторінковий застосунок

UI – візуальний інтерфейс

UML – модель проєктування

UX – користувацький досвід

WORKFLOW – процес роботи

XML – розмітка даних

БД – база даних

БЕКЕНД – серверна логіка

ЕНДПОІНТИ – точки доступу

СПРИНТИ – короткі інтервали

ФРОНТЕНД – інтерфейс користувача

ВСТУП

Сучасні тенденції цифровізації управлінських процесів у різних сферах діяльності висувають нові вимоги до інструментів контролю, планування та моніторингу виконання службових обов'язків. В умовах зростання обсягів інформації, динаміки змін у завданнях та високих вимог до відповідальності працівників, важливим є створення ефективних засобів контролю виконання доручень.

Більшість організацій сьогодні використовують загальні системи управління проєктами, які не завжди адаптовані під потреби внутрішнього контролю службових завдань та обов'язків. Відсутність можливості швидкого створення типових завдань, контролю їх виконання, підтвердження фактів виконання або своєчасного реагування на затримки може призводити до зниження ефективності управління, дублювання дій або втрати відповідальності.

Актуальність теми дослідження зумовлена потребою в розробці спеціалізованого інструменту для управління та контролю виконання службових завдань – системи "DutyDesk", яка забезпечить прозорий процес відстеження виконання обов'язків та створення типових шаблонів завдань.

Об'єктом дослідження є процес організації та контролю виконання службових завдань у межах цифрового середовища.

Предметом дослідження виступають методи та засоби розробки інформаційних систем для управління обов'язками: моделі представлення задач, інтерфейси для користувацької взаємодії та механізми фіксації фактів виконання.

Метою роботи є розробка функціональної системи "DutyDesk" для контролю виконання завдань, яка дозволяє здійснювати їх додавання, редагування, підтвердження виконання, а також формування завдань за шаблоном.

Для досягнення мети дослідження передбачено виконання таких ***завдань***:

— проаналізувати існуючі програмні рішення у сфері управління завданнями
— розробити загальну архітектуру системи DutyDesk
— реалізувати механізм створення, редагування та видалення службових завдань
— забезпечити фіксацію та підтвердження фактів виконання завдань
— створити систему шаблонів для формування типових завдань
— впровадити зручний користувацький інтерфейс з урахуванням UX/UI практик.

Практичне значення роботи полягає у створенні універсального інструменту внутрішнього контролю, який може бути впроваджений у малих підприємствах або організаціях для покращення прозорості процесів, підвищення відповідальності працівників та спрощення управління службовими обов'язками.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис проблемної області

У невеликих компаніях ефективне управління внутрішніми процесами часто залежить не стільки від складних корпоративних систем, як від чіткої організації виконання щоденних завдань, прозорого делегування обов'язків та відповідального контролю. У таких колективах зазвичай відсутні спеціалізовані IT-рішення для моніторингу виконання службових доручень, а замість цього використовуються неформальні інструменти - наприклад, комунікація через месенджери, електронну пошту або таблиці. Це призводить до втрати контролю і до невиконання важливих обов'язків.

Ще однією проблемою є відсутність систематичного підходу до фіксації та підтвердження виконання поставлених задач. В умовах обмежених ресурсів та багатофункціональності співробітників, керівники не завжди мають змогу оперативно перевіряти стан виконання кожного завдання вручну. Це створює ризики перенесення строків, зниження відповідальності, а також ускладнює аналіз ефективності працівників.

Більшість існуючих систем управління проєктами (наприклад, Asana, Trello, Jira) орієнтовані на широкі функціональні можливості, що часто є надмірними для малого бізнесу. Їх впровадження потребує часу, додаткового навчання персоналу та фінансових витрат. Для невеликих компаній, де важливою є простота, швидкість адаптації та чіткість у відображенні завдань, такі інструменти не завжди є виправданими.

Таким чином, існує потреба у простій, інтуїтивній системі для контролю службових обов'язків, яка дозволить:

— фіксувати факт виконання завдання;

— працювати зі стандартними шаблонами завдань;

— отримувати загальну картину виконання без складних звітів;

— адаптуватися під структуру та потреби конкретного колективу без зайвого навантаження.

Розробка такої системи – DutyDesk – дозволить малим підприємствам автоматизувати рутинні аспекти контролю, покращити внутрішню дисципліну та зменшити втрати, пов'язані з людським фактором, без необхідності впровадження важких корпоративних рішень.

1.2 Існуючі аналоги

Trello – це легкий і зручний вебсервіс, який використовує методологію канбан для візуального управління завданнями. Його основа – дошки, які складаються зі списків (наприклад, "План", "У процесі", "Готово") та карток – конкретних завдань [12]. Користувачі можуть перетягувати картки між списками, тим самим змінюючи статуси виконання (рис. 1.1).

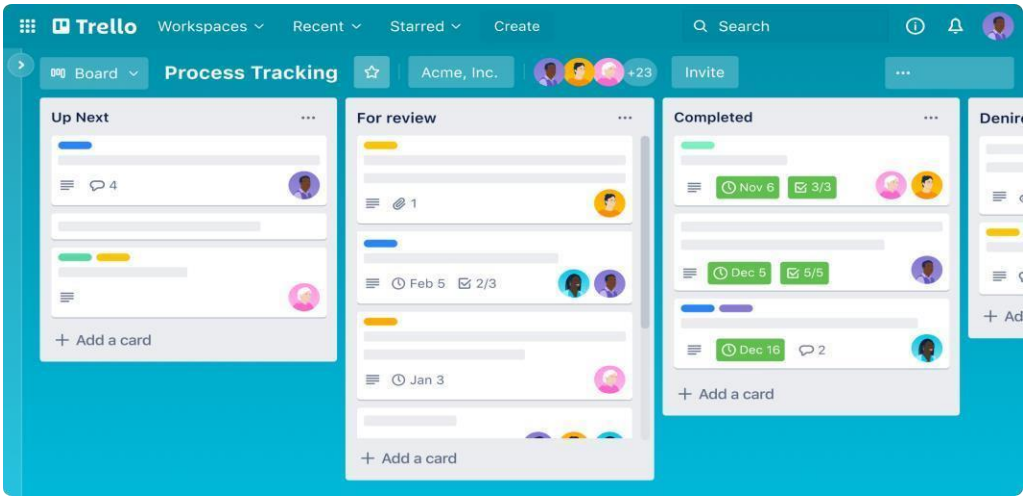

Рисунок 1.1 – Інтерфейс Trello

Основні функції:

— простий візуальний інтерфейс;

— призначення відповідальних за завдання;

— додавання тегів, дедлайнів, вкладень;

— коментарі, чеклісти, нагадування;

— інтеграції через Power-Ups (Slack, Google Drive тощо).

Переваги:

— ідеальний для невеликих команд;

— швидкий у навчанні;

— гнучкий у налаштуванні завдяки шаблонам;

— безкоштовний базовий тариф.

Недоліки:

— відсутність розгалуженої ієрархії задач;

— мінімальні можливості для аналітики;

— обмеженість безкоштовної версії у кількості розширень.

Trello підходить для невеликих організацій, які шукають простий і візуально зручний інструмент управління завданнями без складних технічних налаштувань.

Jira – це професійний інструмент для управління проєктами, який переважно використовується в IT-сфері, особливо для Agile- та Scrum-процесів. Розроблений компанією Atlassian, він дозволяє відстежувати помилки, вести беклог, спринти, задачі з чіткими зв'язками між ними (рис. 1.2) [11].

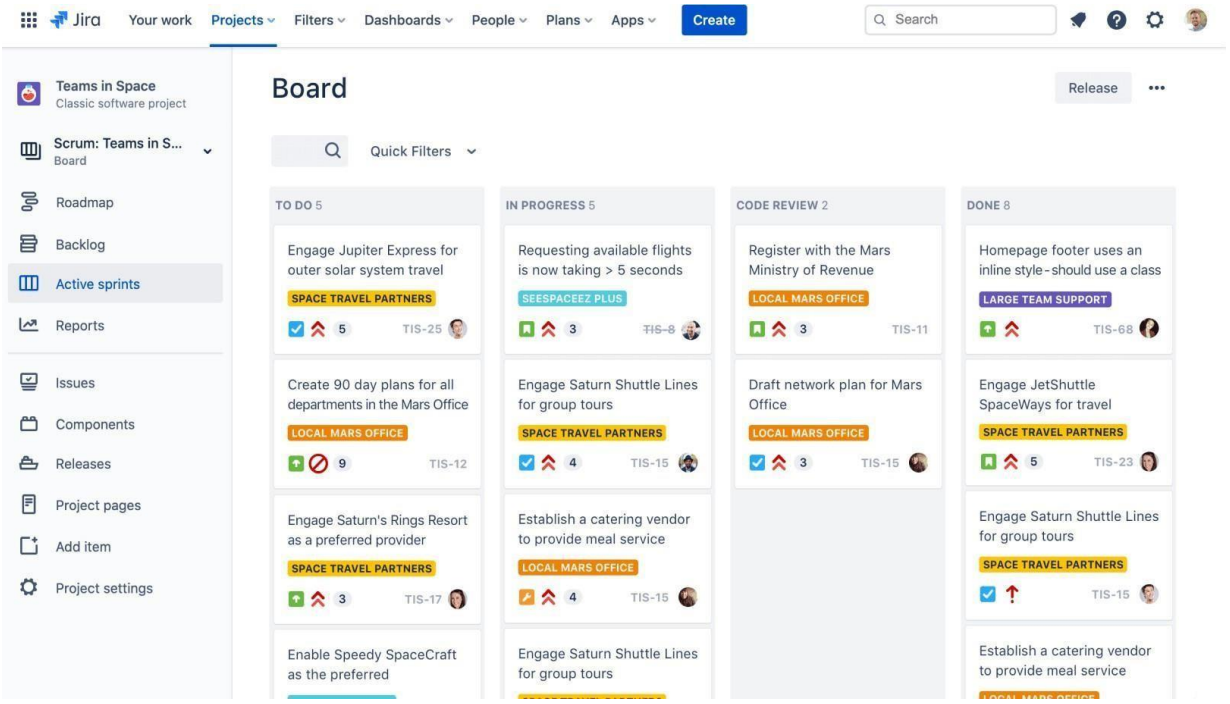

Рисунок 1.2 – Інтерфейс Jira

Основні функції:

— підтримка багаторівневої ієрархії завдань (епіки, задачі, підзадачі);

— гнучке налаштування робочих процесів (workflow);

— візуалізація проєктів за допомогою kanban та scrum-дощок;

— розширена система звітності та діаграм (burndown charts, velocity);

— інтеграція з Bitbucket, Confluence, Slack, GitHub тощо.

Переваги:

— підходить для команд розробки та складних технічних проєктів;

— повна кастомізація процесів;

— потужна аналітика.

Недоліки:

— висока складність у використанні для нетехнічних користувачів;

— вимагає налаштувань та адаптації;

— платна підписка (безкоштовно - до 10 користувачів).

Jira – це вибір для команд, де потрібно чітко контролювати великий обсяг задач з детальними зв'язками, ролями та звітністю. Для малих компаній, які не мають потреби в складному процесному підході, система може бути занадто громіздкою.

Notion – це універсальний інструмент для нотаток, баз даних, таблиць, візуальних структур та управління контентом [13]. Його можна адаптувати під будь- які внутрішні процеси компанії, включаючи управління завданнями (рис.1.3).

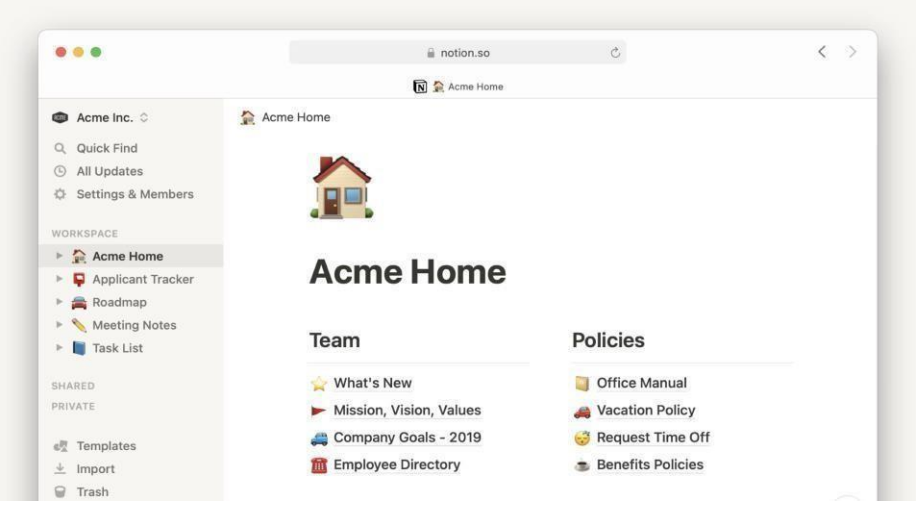

Рисунок 1.3 – Інтерфейс Notion

Основні функції:

— створення сторінок із вбудованими блоками (текст, таблиці, дошки, календарі);

— підтримка шаблонів та баз даних;

— колаборація в реальному часі;

— можливість імпорту з інших систем (Trello, Asana);

— сильна кастомізація під власні потреби.

Переваги:

— надзвичайна гнучкість;

— поєднує функції органайзера, вікі, менеджера завдань;

— ідеальний для ведення шаблонних службових обов'язків або чеклістів;

— добре працює для документообігу та контент-менеджменту.

Недоліки:

— не має "готової" логіки управління задачами - усе налаштовується вручну;

— обмежені можливості для класичного проєктного менеджменту;

— потребує часу на створення структур.

Notion – це платформа, що найкраще підійде командам, які хочуть об'єднати документацію, шаблони та систему задач в одному середовищі. У контексті контролю службових обов'язків дає широкі можливості для шаблонізації та фіксації, однак не має вбудованої системи сповіщень чи статусів виконання без додаткових налаштувань.

З огляду на наявні рішення, виникає потреба розроблення універсального інструменту, який би поєднував у собі кращі характеристики наявних аналогів

та водночас хоча б частково вирішував поширені проблеми у користуванні ними.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Основні аспекти веброзробки проєкту

Теоретичні основи розробки вебсайту охоплюють значний обсяг знань, включаючи технологічні аспекти, архітектуру системи, вибір найкращих інструментів і забезпечення належного рівня безпеки. У сучасній веброзробці особливо поширеними є архітектури з чітким розділенням фронтенду і бекенду, де обидві частини можуть розвиватися незалежно одна від одної.

Бекенд (Backend) відповідає за обробку логіки додатку, взаємодію з базою даних, забезпечення авторизації та автентифікації, а також за обробку HTTP-запитів від клієнта [14]. У цьому проєкті реалізація серверної частини здійснена за допомогою Java та Spring Boot, що дозволяє створити гнучкий і масштабований RESTful вебсервіс.

Основні компоненти:

— *Серверна логіка*: реалізується як набір REST-контролерів, які відповідають на запити типу GET, POST, PUT, DELETE. Вони обробляють запити, виконують бізнес-логіку і повертають відповіді у форматі JSON.

— *Бази даних:* для зберігання даних використовується реляційна система PostgreSQL, а для взаємодії з нею - Spring Data JPA, що забезпечує зручний доступ до даних за допомогою репозиторіїв.

— *Безпека:* реалізована через Spring Security, з урахуванням розмежування ролей (користувач/адміністратор), захисту REST-ендпоінтів та обмеження доступу до ресурсів.

— *Технології:* Java, Spring Boot, Spring Web, Spring Security, Spring Data JPA, RESTful API, Maven, Docker, GCP.

Основні переваги **Spring Boot**:

— *Швидкий старт:* мінімальна конфігурація для запуску програми завдяки вбудованим налаштуванням за замовчуванням.

— *Автоматична конфігурація:* Spring Boot автоматично налаштовує компоненти залежно від бібліотек у проєкті.

— *Вбудований сервер:* немає потреби окремо розгортати сервер (наприклад, Tomcat), оскільки він інтегрований.

— *Легка інтеграція з базами даних:* через Spring Data JPA та підтримку популярних СУБД.

— *Розширюваність:* можливість додати потрібні модулі (безпека, пошта, WebSocket тощо) як залежності.

— *Підтримка REST:* зручне створення RESTful API через анотації @RestController, @RequestMapping тощо.

— *Документація та спільнота:* велика кількість ресурсів та активна спільнота розробників.

Spring Boot дозволяє зосередитися на бізнес-логіці, зменшуючи рутинну роботу й конфігурацію.

Фронтенд (Frontend) – це частина системи, з якою безпосередньо взаємодіє користувач. У цьому проєкті він реалізований як Single Page Application (SPA) за допомогою фреймворку Angular. Це забезпечує високу швидкодію, мінімальну кількість перезавантажень сторінки та плавний користувацький досвід.

— *Інтерфейс користувача (UI)*: оформлення елементів відбувається за допомогою Angular Material, що гарантує сучасний та адаптивний дизайн. Вся логіка інтерфейсу реалізована на TypeScript.

— *Зручність використання (UX):* SPA-архітектура дозволяє динамічно оновлювати вміст сторінки без повного перезавантаження, забезпечуючи високу інтерактивність. Наприклад, оновлення списку завдань чи підтвердження виконання відбувається миттєво завдяки асинхронній взаємодії з бекендом через HTTP-запити.

— *Архітектура компонентів:* Angular використовує модульну структуру, що дозволяє розділити інтерфейс на незалежні компоненти (сторінки, діалоги, списки, таблиці тощо), полегшуючи розробку і тестування.

Технології: HTML, SCSS, TypeScript, Angular, Angular Material, RESTful API, Git.

Основні переваги **Angular** [15]:

— *Структурованість та модульність*: чітка архітектура з розділенням коду на компоненти, сервіси та модулі полегшує масштабування й підтримку проєкту.

— *TypeScript*: використання мови, яка додає типізацію та покращує читабельність і безпеку коду.

— *Двостороннє зв'язування даних (two-way data binding)*: автоматична синхронізація між моделлю та представленням спрощує роботу з формами та UI.

— *Інструменти для розробки:* CLI (Angular Command Line Interface) дозволяє швидко створювати, тестувати та збирати проєкти.

— *Вбудована підтримка RxJS*: зручна робота з асинхронністю, потоками даних і реактивним програмуванням.

— *Angular Material:* набір готових, адаптивних і естетичних UI-компонентів для швидкої розробки інтерфейсу.

— *Підтримка SPA:* забезпечує швидку роботу додатка без повного перезавантаження сторінки, що покращує користувацький досвід.

Взаємодія фронтенду та бекенду. Фронтенд і бекенд взаємодіють через REST API, що є стандартом для передачі даних між клієнтом і сервером у форматі JSON. Основні принципи цієї взаємодії включають:

— використання HTTP-методів (GET для отримання, POST для створення, PUT/PATCH для оновлення, DELETE для видалення);

— аутентифікацію запитів за допомогою токенів або сесій;

— стандартизацію структури відповідей і помилок;

— документацію API за допомогою OpenAPI (Swagger), що полегшує підтримку і розробку клієнтської частини.

Таке розділення між клієнтом і сервером дозволяє досягти високої гнучкості, масштабованості та розширюваності вебзастосунку.

Основні переваги взаємодії через **REST API**:

— *стандартизованість*: REST базується на стандартних HTTP-методах (GET, POST, PUT, DELETE), що робить API зрозумілим і передбачуваним;

— незалежність клієнта і сервера: фронтенд і бекенд можуть розроблятись окремо, змінюватись незалежно один від одного та навіть бути написаними на різних технологіях;

— *масштабованість*: REST дозволяє ефективно обслуговувати багато клієнтів і добре підходить для великих розподілених систем;

— *формат даних*: обмін інформацією відбувається у форматі JSON або XML, що легко обробляється на клієнтській стороні;

— *простота інтеграції:* REST API легко інтегрується з іншими сервісами, мобільними додатками або сторонніми клієнтами;

— *кешування*: HTTP дозволяє використовувати кешування відповідей, що зменшує навантаження на сервер і підвищує швидкодію;

— *документованість*: REST API легко описується та тестується за допомогою інструментів на кшталт OpenAPI (Swagger).

2.2 Побудова дерева проблем та дерева цілей

Ефективне інформаційно-алгоритмічне забезпечення вебсистеми контролю виконання службових обов’язків потребує всебічного підходу, що охоплює організаційні, технічні, безпекові, якісні та UX-аспекти. Лише системна робота над кожним із вище перелічених пунктів гарантує надійність, прозорість і прийнятність рішення для кінцевих користувачів.

Для початку розглянемо основні труднощі, які можуть виникати при впровадженні та експлуатації вебсистеми електронного контролю виконання службових обов’язків та доручень.

1. Організаційні бар’єри

1.1. Нечітке визначення процесів і ролей:

— відсутність стандартизованих регламентів робочих процесів;

— невизначеність повноважень та відповідальностей користувачів;

— розпорошеність інформації між відділами.

1.2. Опір змінам та низька мотивація

— недовіра до нових IT-інструментів;

— брак зацікавленості в прозорості контролю;

— нестача кваліфікованих «промоутерів» із середини організації.

1.3. Проблеми управління змінами

— відсутність плану навчання та підтримки користувачів;

— нечітке комунікування етапів впровадження;

— нерегулярне оновлення процедур відповідно до зворотнього зв’язку.

2. Технічні виклики

2.1. Інтеграція з існуючими системами

— різнорідні формати даних і API;

— ускладнена синхронізація довідкових довідників (співробітники, підрозділи тощо);

— неузгодженість часових поясів і робочих графіків.

2.2. Забезпечення надійності й відмовостійкості

— балансування навантаження при пікових зверненнях;

— механізми автоматичного резервного копіювання та відновлення;

— тестування «коротких» і «довгих» сценаріїв виконання завдань.

2.3. Масштабованість і продуктивність

— швидкість відгуку UI при великій кількості відкритих доручень;

— оптимізація запитів до бази даних;

— горизонтальне масштабування компонентів системи.

3. Безпека та захист даних

3.1. Аутентифікація та авторизація

— впровадження єдиної системи SSO/LDAP;

— розмежування прав доступу за ролями й контекстами (проєкти, підрозділи);

— багатофакторна аутентифікація для критичних операцій.

3.2. Конфіденційність і цілісність інформації

— шифрування даних «на льоту» (TLS) та «в спокої» (AES);

— контроль версій документів та логування змін;

— захист від SQL-ін’єкцій, XSS і CSRF-атак.

3.3. Аудит і звітність

— генерація детальних логів подій (хто, коли, що зробив);

— аналітика аномальних дій користувачів;

— зберігання історії виконання завдань за зобов’язаний термін.

4. Якість даних і процесів

4.1. Уніфікація та валідація введених даних

— єдина форма й шаблон доручень з обов’язковими полями;

— перевірка коректності термінів, груп отримувачів, статусів;

— автоматичні тригери для нагадувань про орієнтовні строки.

4.2. Відстеження статусів і SLA

— чіткий статус-lifecycle: “Новий” → “У роботі” → “Затримано” → “Виконано”;

— визначення показників продуктивності (кількість завершених доручень/день);

— налаштування оповіщень про прострочення.

4.3. Зворотний зв’язок і обробка виключень

— механізм коментування/запитів на уточнення по дорученню;

— процедура ескалації «складних» чи «спірних» завдань;

— аналітика причин затримок і відмов.

5. UX/UI та підтримка користувачів

5.1. Зручність інтерфейсу

— інтуїтивні дашборди зі статусами та KPI;

— фільтри, сортування й пошук за різними параметрами;

— мобільна адаптивність для роботи на планшетах/смартфонах.

5.2. Документація та навчальні матеріали

— онлайн-посібник із відео-уроками;

— вбудована контекстна допомога ("tooltips");

— регулярні вебінари та “office hours” для вирішення питань.

5.3. Підтримка та супровід

— служба технічної підтримки із SLA-угодою;

— розподіл запитів за пріоритетом і категоріями;

— регулярні оновлення й виправлення помилок.

На основі детального аналізу труднощів впровадження традиційних вебсистем електронного контролю службових обов’язків і доручень сформовано **головну стратегічну ціль** – розробка та забезпечення ефективного функціонування вебсистеми контролю службових обов’язків на базі наступних **рішень**:

1. Організаційне вдосконалення (забезпечується організаційними процедурами всередині організації та опосередковано стосується розроблюваного проєкту):

— формалізація ролей і процедур;

— розробка регламентів і інструкцій;

— стимулювання персоналу до використання системи.

2. Технічна надійність та масштабованість системи:

— інтеграція з існуючими IT-системами;

— модульність та розширюваність архітектури;

— висока швидкодія та стабільність.

3. Якість і цілісність даних

— автоматична перевірка введених даних;

— уніфікація форм і шаблонів;

— зберігання історії та версій доручень.

4. Зручність та підтримка користувачів

— інтуїтивний інтерфейс;

— мобільна доступність;

— наявність інструкцій, довідки і навчальних матеріалів.

5. Інформаційна безпека та контроль

— розмежування прав доступу;

— аудит та логування дій користувачів;

— захист переданих і збережених даних.

Реалізація такого дерева цілей забезпечить створення ефективної, зручної, безпечної та масштабованої вебсистеми контролю службових обов’язків та доручень. Це дозволить:

— значно підвищити прозорість управлінських процесів;

— забезпечити своєчасне і якісне виконання завдань;

— зменшити кількість помилок та втрат інформації;

— створити позитивний досвід користування системою для всіх ролей (керівників, виконавців, адміністраторів);

— зміцнити цифрову дисципліну в організації;

— мати достовірну аналітику для прийняття управлінських рішень.

У результаті організація отримає потужний інструмент для планування, моніторингу та оцінки ефективності службової діяльності.

Розроблена на основі таких рішень вебсистема контролю службових обов’язків та доручень може мати низку переваг над популярними інструментами на зразок Jira, Trello чи Notion, за умови, що вона буде розроблена під конкретні потреби організації, забезпечуючи високу адаптивність до структури управління; локальну інтеграцію з документообігом; повний контроль безпеки та даних; відповідність службовим регламентам та державним вимогам.

Потенційні переваги такої вебсистеми:

— повна відповідність структурам, статусам, регламентам, обліку;

— гнучке налаштування під посадові обов'язки, погодження;

— повна локалізація та відповідність нормативам (наприклад, україномовність);

— можна глибоко інтегрувати з внутрішніми IT-системами (наприклад, документообіг, кадрова система);

— орієнтована на посадові інструкції та службові доручення (простота для державного/муніципального сектора);

— повний контроль над хостингом, БД, журналами;

— без прив’язки до закордонних платформ.

2.3 Характеристики проєкту “DutyDesk”

Опис ідеї проєкту. Головною ідеєю проєкту є розробка платформи для контролю виконання службових обов'язків та завдань - "DutyDesk", яка спеціально орієнтована на потреби малих підприємств, приватних компаній та невеликих організацій. Система призначена для спрощення процесу моніторингу, підтвердження та аналізу виконання щоденних або періодичних завдань співробітників.

Основна мета полягає в тому, щоб надати керівникам і працівникам зручний інструмент для прозорого обліку виконання обов'язків без складних корпоративних систем, які часто є перевантаженими для малих структур. Завдяки простому та інтуїтивно зрозумілому інтерфейсу навіть користувачі без технічної підготовки зможуть ефективно працювати з системою.

Система дозволяє:

— формувати та редагувати шаблони завдань (доступно лише для адміністратора);

— створювати записи про виконання обов'язків на основі шаблонів;

— підтверджувати або видаляти записи з урахуванням ролі користувача (звичайного або адміністратора);

— вести облік виконаних робіт із зручною фільтрацією та пошуком;

— контролювати останні дії як власні, так і колег, забезпечуючи взаємний контроль.

Для адміністратора доступні функції реєстрації нових користувачів, модифікації та видалення шаблонів, а також можливість керувати записами інших користувачів. Звичайний працівник може працювати лише зі своїми даними та непідтвердженими записами.

Додатково підтримується можливість анонімного доступу через спеціальне запрошення, що може бути корисним для тимчасових працівників або зовнішніх виконавців.

Основною перевагою системи є її легкість, доступність та адаптація під реальні потреби малих компаній, які не мають ресурсів для впровадження важких ERP-систем або складних CRM-рішень. "DutyDesk" дозволяє зробити процес контролю обов'язків гнучким, прозорим та простим, не навантажуючи користувача зайвими функціями.

Для забезпечення безпеки та конфіденційності реалізовано систему доступів згідно з ролями та обмеженням видимості чутливих даних інших користувачів.

У перспективі платформа може бути розширена для роботи на мобільних застосунках, що дозволить працювати із завданнями в будь-якому місці та в зручний час.

Аналіз технологічних можливостей реалізації проєкту. Оскільки платформа "DutyDesk" орієнтована на використання переважно у малих компаніях, для її розробки було обрано сучасні, перевірені та безкоштовні технології, що забезпечують надійність, гнучкість та масштабованість рішення без значних фінансових витрат на початкових етапах.

Для створення серверної частини використовується стек технологій Java Spring Boot із застосуванням таких компонентів, як Spring Data JPA, Spring Web

та Spring Security. Це дозволяє легко та безпечно реалізувати RESTful API для обміну даними між клієнтською частиною та сервером, а також забезпечує необхідний рівень захисту даних та управління доступом згідно з ролями користувачів (адміністратор, звичайний користувач, анонімний користувач). Інтеграція з PostgreSQL забезпечує надійне збереження структурованих даних про користувачів, завдання та шаблони.

Клієнтська частина платформи реалізована за допомогою Angular, TypeScript та Angular Material, що дозволяє створити адаптивний, легкий та зручний для користувача вебінтерфейс. Завдяки використанню цього фреймворку забезпечується висока продуктивність, інтерактивність та можливість розгортання системи на будь-яких пристроях через браузер без необхідності встановлення додаткового програмного забезпечення.

В якості додаткових інструментів використовуються:

— Docker та Nginx для контейнеризації та розгортання застосунку на хмарних сервісах, таких як Google Cloud Platform (GCP);

— OpenAPI/Swagger для документування API та полегшення процесу розробки та тестування;

— Maven для управління залежностями та збором проєкту;

Вибраний технологічний стек забезпечує:

— мінімальні витрати на розробку та підтримку проєкту;

— можливість легкого масштабування та інтеграції нових функцій у майбутньому (наприклад, мобільної версії);

— високу доступність і гнучкість як для кінцевих користувачів (через веб), так і для розробників завдяки відкритим інструментам і популярним бібліотекам;

— простоту підтримки системи невеликою командою розробників, що критично важливо для продукту, орієнтованого на потреби малих компаній.

Завдяки використанню переважно безкоштовних технологій, а також відсутності залежності від дорогих ліцензій або платних компонентів, проєкт

може бути запущений та підтримуватись із мінімальними інвестиціями, що робить його ідеальним рішенням для малого бізнесу.

Аналіз ринкових можливостей стартап-проєкту на базі «DutyDesk». У сучасних умовах значна частина малих підприємств стикається з потребою ефективно організовувати внутрішні робочі процеси, зокрема контроль виконання службових завдань та доручень. На відміну від великих корпорацій, які часто використовують складні та дорогі системи управління завданнями (такі як Jira, Asana або Monday.com), малий бізнес потребує доступного, простого у використанні та гнучкого рішення, яке не вимагатиме значних фінансових та людських ресурсів для впровадження та обслуговування.

Саме ці потреби і задовольняє платформа "DutyDesk", яка розроблена спеціально для малого бізнесу, враховуючи такі особливості:

— простота встановлення та використання навіть без IT-відділу;

— можливість швидкого розгортання на локальному сервері або у хмарі (GCP);

— відсутність прихованих витрат або складних ліцензійних умов;

— гнучка рольова модель доступу (адміністратор, користувач, анонімний користувач), яка дозволяє забезпечити базові вимоги безпеки та конфіденційності.

На відміну від конкурентів, які орієнтовані на великі організації або корпоративний сектор, "DutyDesk" заповнює нішу малого бізнесу, де подібні рішення або відсутні, або мають зайву функціональність, яка ускладнює роботу невеликих команд.

Ще однією перевагою платформи є можливість її подальшої кастомізації під конкретні потреби окремих клієнтів, що для малого бізнесу є особливо цінним.

Ринкові виклики полягають, в першу чергу, у донесенні інформації до потенційних користувачів, оскільки малі компанії рідше слідкують за IT-новинками або інноваційними продуктами. Для подолання цієї проблеми буде

доцільним застосування прямого маркетингу, співпраці з локальними бізнес-об'єднаннями та створення демонстраційних версій продукту.

Також важливою перевагою є можливість масштабування системи – спочатку для малого бізнесу, а в майбутньому – для середнього або для віддалених команд (фрілансерів).

Таким чином, попри наявність відомих корпоративних рішень на ринку систем управління завданнями, сегмент малого бізнесу залишається недостатньо заповненим, що відкриває хороші перспективи для успішного запуску та розвитку продукту "DutyDesk" при наявності грамотної стратегії просування.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

На початковому етапі розробки була поставлена мета створити просту, зручну та ефективну систему для контролю виконання службових обов'язків та доручень у межах організаційної структури. Ідея виникла з потреби вирішити проблему неформального або неструктурованого підходу до фіксації виконання завдань у робочих середовищах, де важливо мати прозору й надійну історію виконання обов'язків.

Назва "DutyDesk" була обрана як поєднання слів "duty" (обов'язок) і "desk" (робоче місце), що символізує робочий інструмент для обліку й контролю обов'язків. Назва коротка, зрозуміла та відображає суть системи - забезпечити зручне "робоче місце" для керування дорученнями.

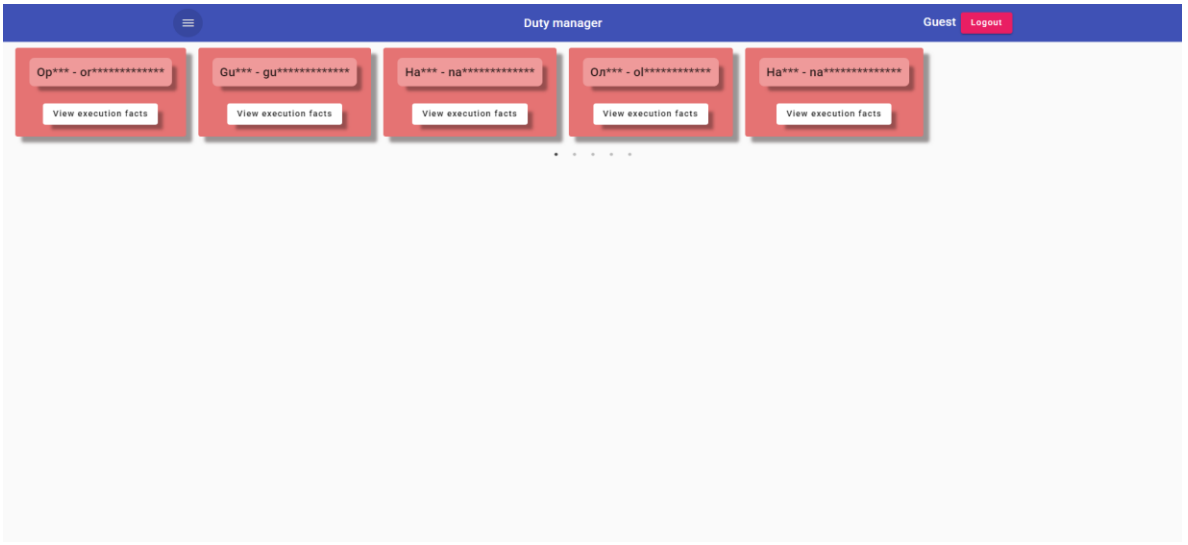

Рисунок 3.1 – Головна сторінка у гостьовому режимі

Основна ***функціональність системи*** включає:

— перегляд списку обов'язків і фактів їх виконання;

— додавання нових фактів виконання;

— підтвердження виконання обов'язку (іншим користувачем або адміністратором);

— створення фактів за шаблоном;

— редагування та видалення шаблонів;

— збереження всієї інформації в базі даних.

Система орієнтована на адміністративне управління обов'язками з можливістю централізованого контролю. Однією з ключових переваг над аналогічними рішеннями є простота, мінімалістичний підхід та орієнтація на конкретну потребу – фіксацію і підтвердження виконання конкретних обов'язків, а не загальне управління проєктами.

Таким чином, система "DutyDesk" має потенціал стати корисним внутрішнім інструментом для організацій, що потребують фіксації та контролю виконання рутинних або критичних завдань у зрозумілому й керованому форматі.

3.1 Опис архітектури проєкту

Проєкт "**DutyDesk**" побудований за принципами клієнт-серверної архітектури з використанням RESTful сервісу на стороні сервера та односторінкового застосунку (SPA) на клієнті.

- **Back-end** реалізований на базі Java з використанням Spring Boot, Spring Security, Spring Data JPA та REST API. Сервер відповідає за обробку бізнес-логіки, управління користувачами, завданнями, шаблонами, а також за аутентифікацію та авторизацію.
- **Front-end** розроблений з використанням Angular та Angular Material, що дозволяє створити динамічний та інтерактивний користувацький інтерфейс у вигляді SPA. Комунікація між клієнтом і сервером здійснюється через HTTP-запити до REST API.
- **База даних - PostgreSQL**, яка використовується для зберігання всіх структурованих даних системи: користувачів, завдань, шаблонів, фактів виконання тощо. Доступ до БД реалізовано через ORM-інструмент Spring Data JPA.
- Для розгортання проєкту використано Docker і Nginx як реверс-проксі сервер, що забезпечує маршрутизацію запитів та захист доступу.

- Система розгортається в хмарному середовищі Google Cloud Platform (GCP), що дозволяє забезпечити масштабованість, надійність та зручне управління ресурсами.

Ця архітектура забезпечує модульність, розширюваність та ефективну взаємодію між компонентами системи.

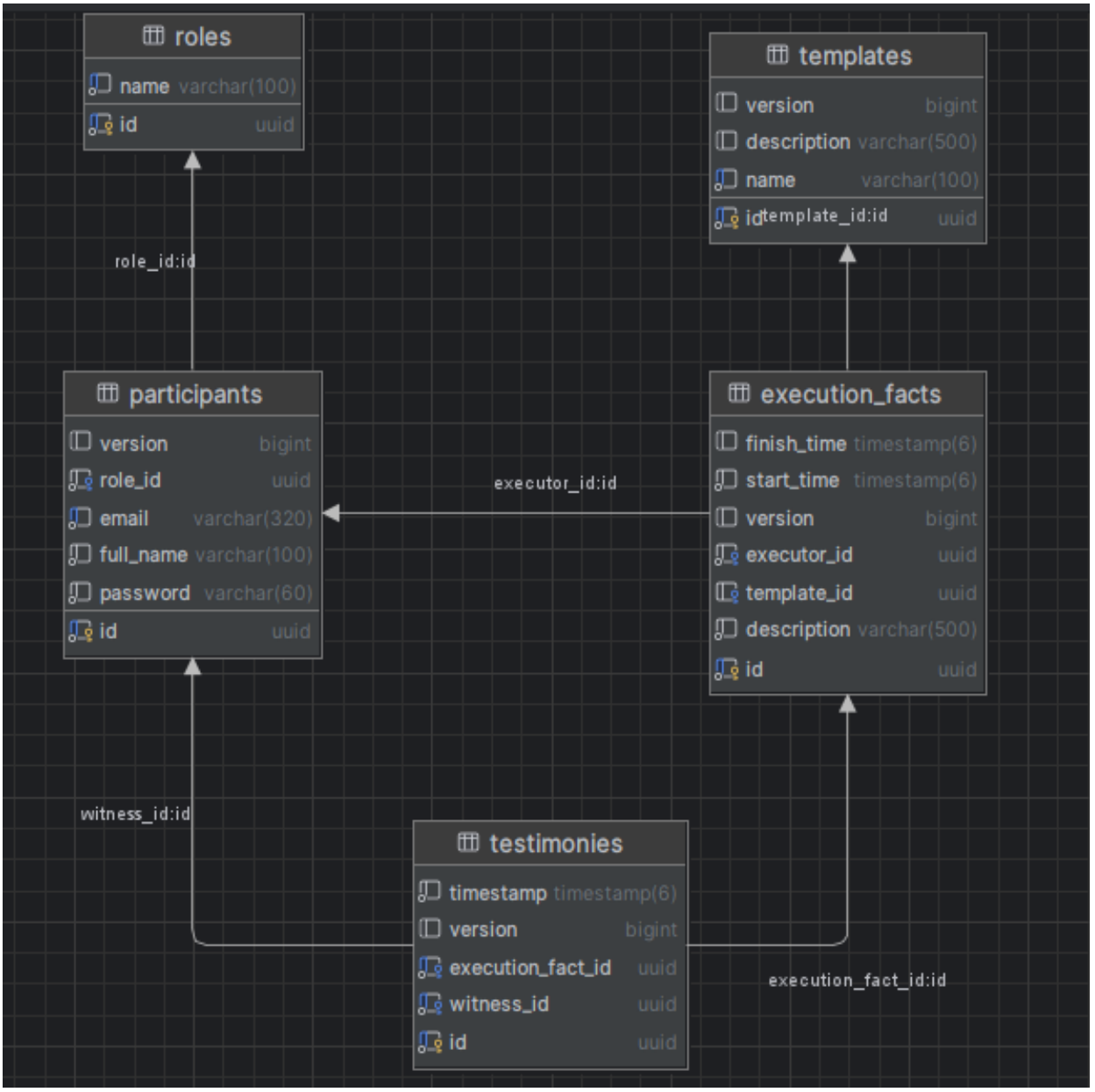

Рисунок 3.2 – UML діаграма структури БД

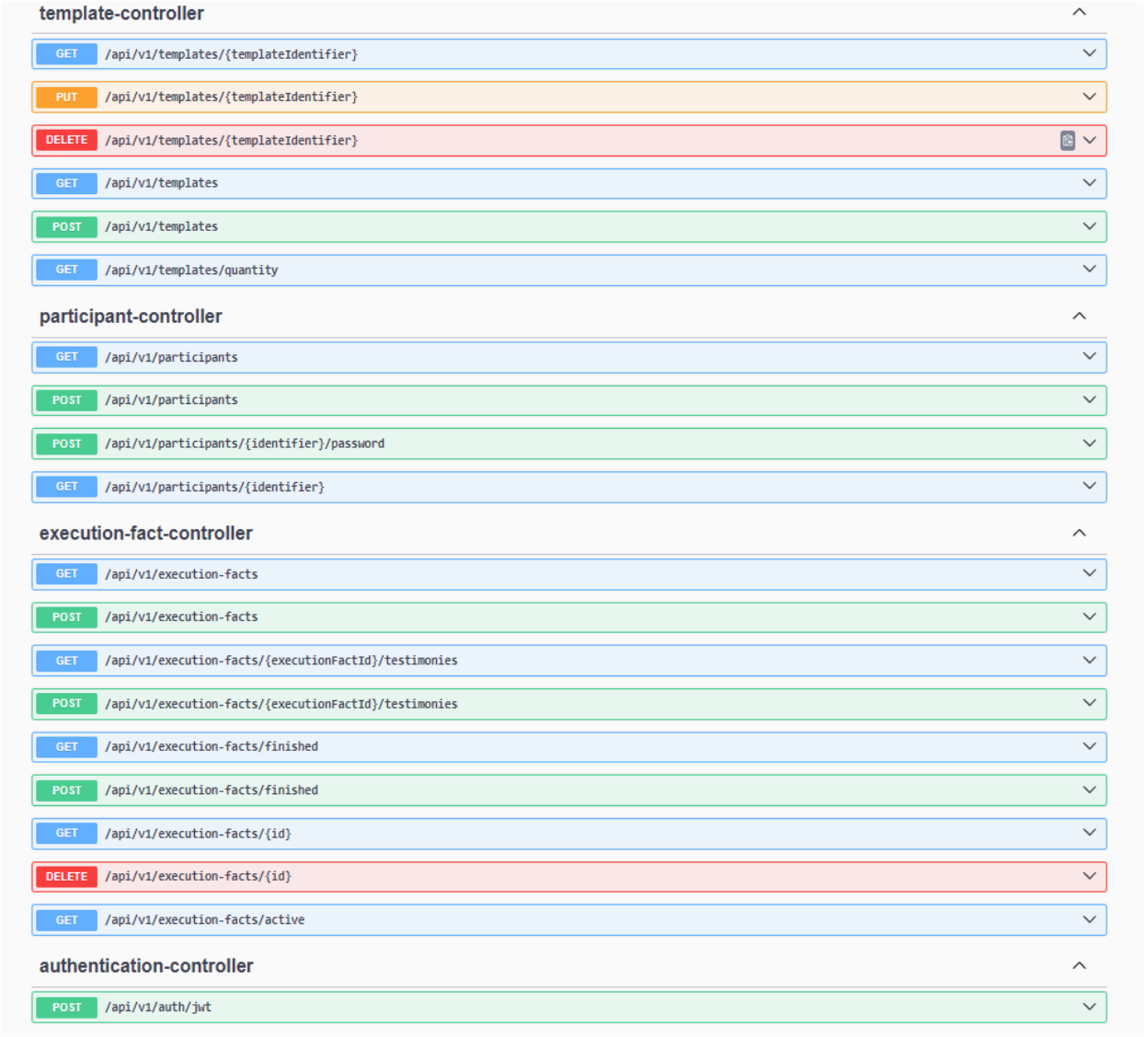

Рисунок 3.3 – Перелік ендпоінтів рестфул сервіса

3.2 Опис ролей та їх можливостей

Ролі та безпека імплементовані за допомогою Spring Security, ось основний код його налаштування:

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
{
        http
                .cors(httpSecurityCorsConfigurer ->
httpSecurityCorsConfigurer.configurationSource(corsConfigurationSource))
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers(HttpMethod.POST,
"/api/v1/auth/jwt").permitAll()
                        .requestMatchers(HttpMethod.GET, "/swagger-ui/**",
"/v3/api-docs/**").permitAll()
```

```
                        .requestMatchers(HttpMethod.POST,
"/api/v1/participants").hasRole(Role.ADMIN.getName())
                        .requestMatchers(HttpMethod.POST,
ANY_API_V_1).hasAnyRole(Role.ADMIN.getName(),  Role.PARTICIPANT.getName())
                        .requestMatchers(HttpMethod.PUT,
ANY_API_V_1).hasAnyRole(Role.ADMIN.getName(),  Role.PARTICIPANT.getName())
                        .requestMatchers(HttpMethod.DELETE,
ANY_API_V_1).hasAnyRole(Role.ADMIN.getName(),  Role.PARTICIPANT.getName())
                        .requestMatchers(HttpMethod.DELETE,
API_V_1_TEMPLATES).hasRole(Role.ADMIN.getName())
                        .requestMatchers(HttpMethod.PUT,
API_V_1_TEMPLATES).hasRole(Role.ADMIN.getName())
                        .requestMatchers(HttpMethod.POST,
API_V_1_TEMPLATES).hasRole(Role.ADMIN.getName())
                        .anyRequest().authenticated()
                )
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                )
                .exceptionHandling(exConf -> exConf.authenticationEntryPoint(
                        new RestResponseAuthenticationEntryPoint(objectMapper,
timeService)
                ))
                .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
                .addFilterBefore(jwtExceptionFilter,
JwtAuthenticationFilter.class);
        return http.build();
    }
```

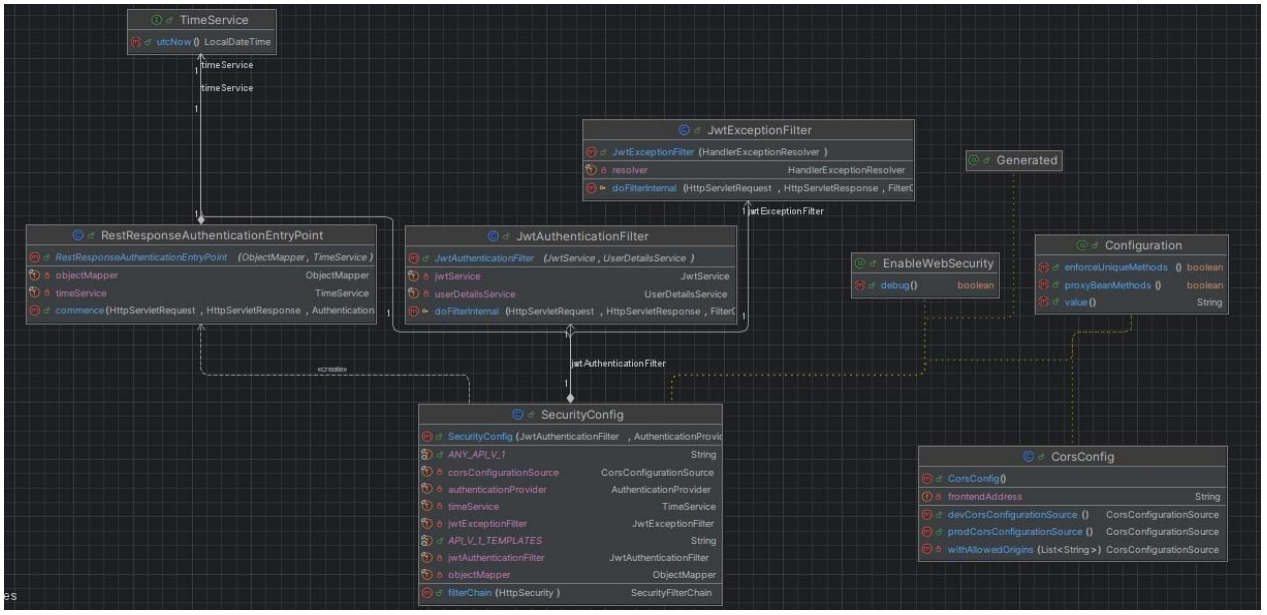

Рисунок 3.4 – UML діаграма імплементації безпеки

Guest – має можливість обмеженого перегляду сторінок в зашифрованому вигляді. Не може нічого додавати чи редагувати (рис. 3.1, 3.6, 3.7).

Рисунок 3.5 – Швидкий доступ до демонстрації роботи проекту (QR-код)[1]

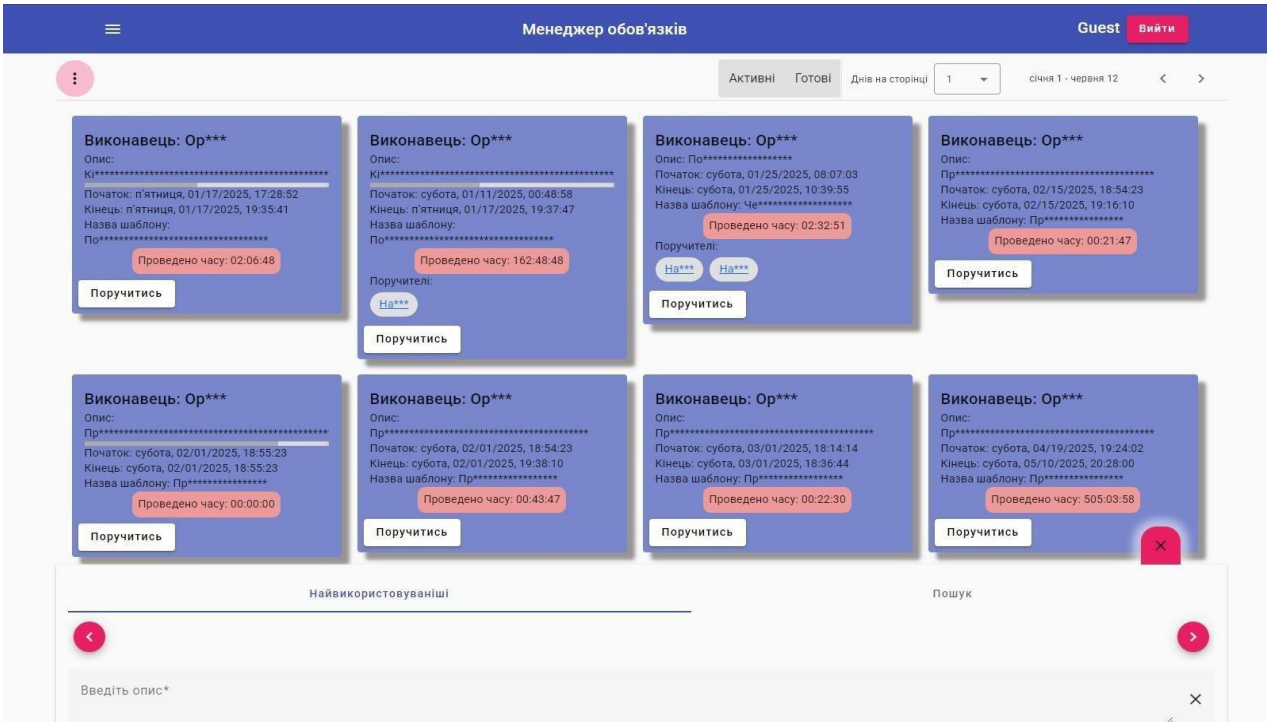

Рисунок 3.6 – Сторінка перегляду фактів виконання у гостьовому режимі

[1] https://duty-manager-frontend-uqlunybkrq-ew.a.run.app/uk-UA/?guest=true

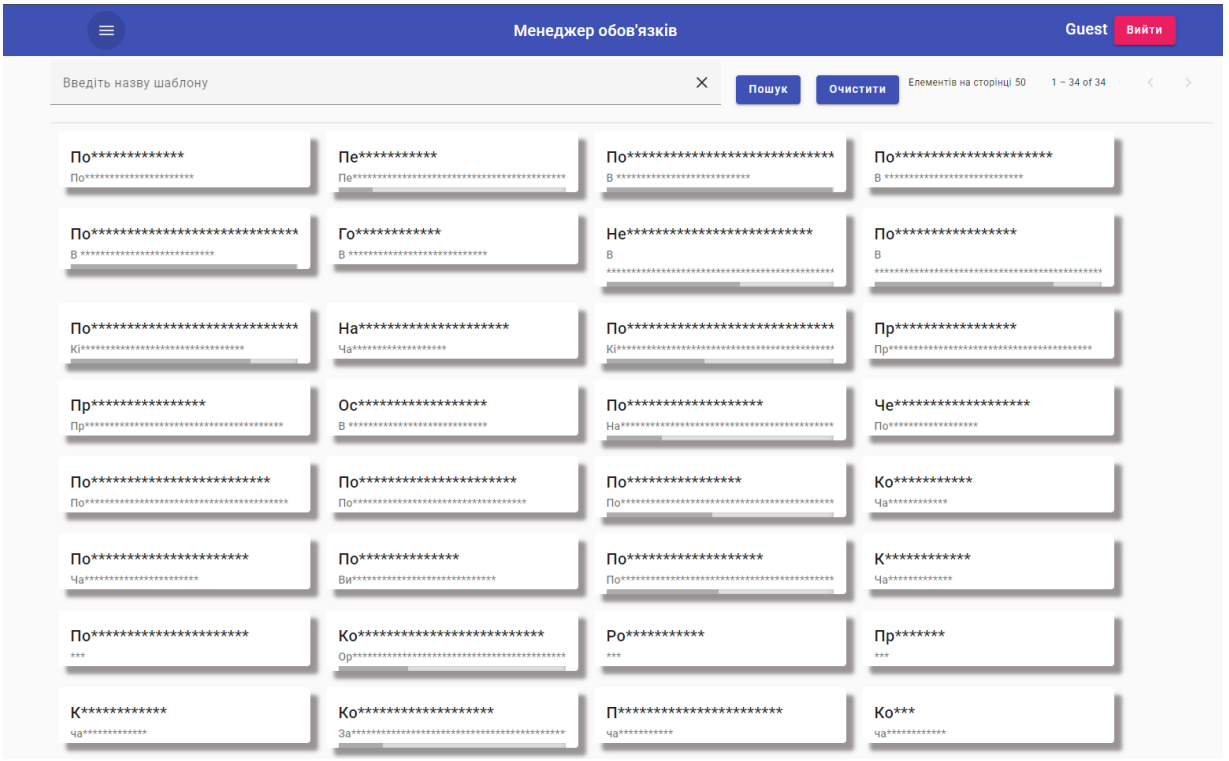

Рисунок 3.7 – Сторінка перегляду шаблонів у гостьовому режимі

User – має можливість перегляду фактів виконання будь кого. Може підтверджувати факт виконання якщо цей факт належить до іншого користувача. Може видаляти свої не підтверджені факти виконання. Може створювати свої факти виконання або факти виконання за шаблоном. Також може змінювати власний пароль (рис. 3.8, 3.9, 3.10, 3.11).

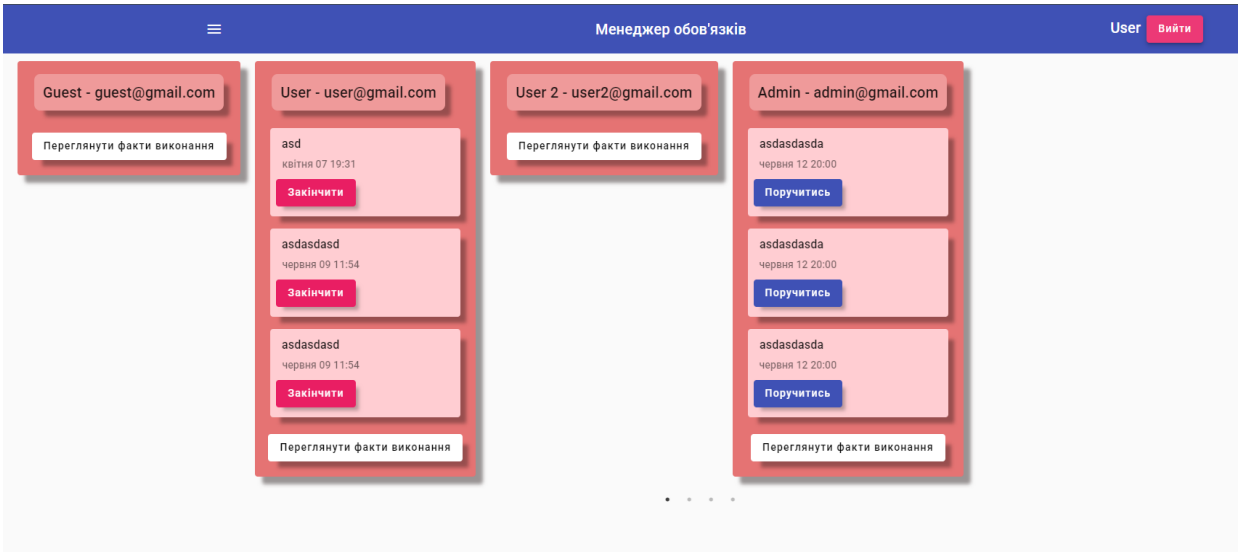

Рисунок 3.8 – Головна сторінка

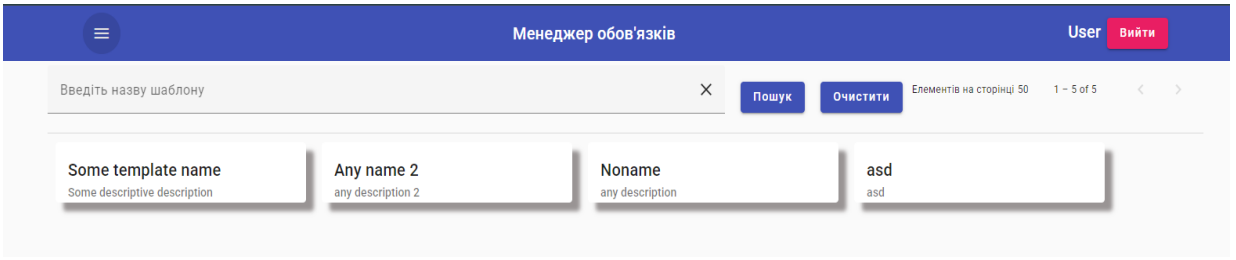

Рисунок 3.9 – Сторінка перегляду шаблонів

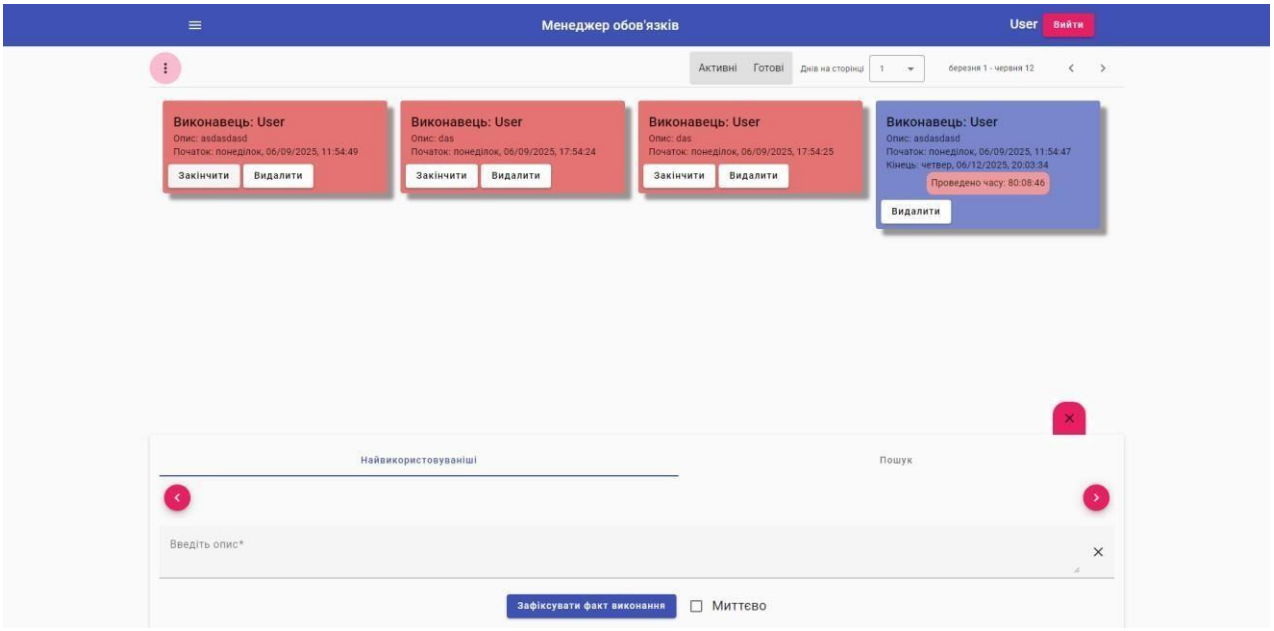

Рисунок 3.10 – Сторінка перегляду фактів виконання

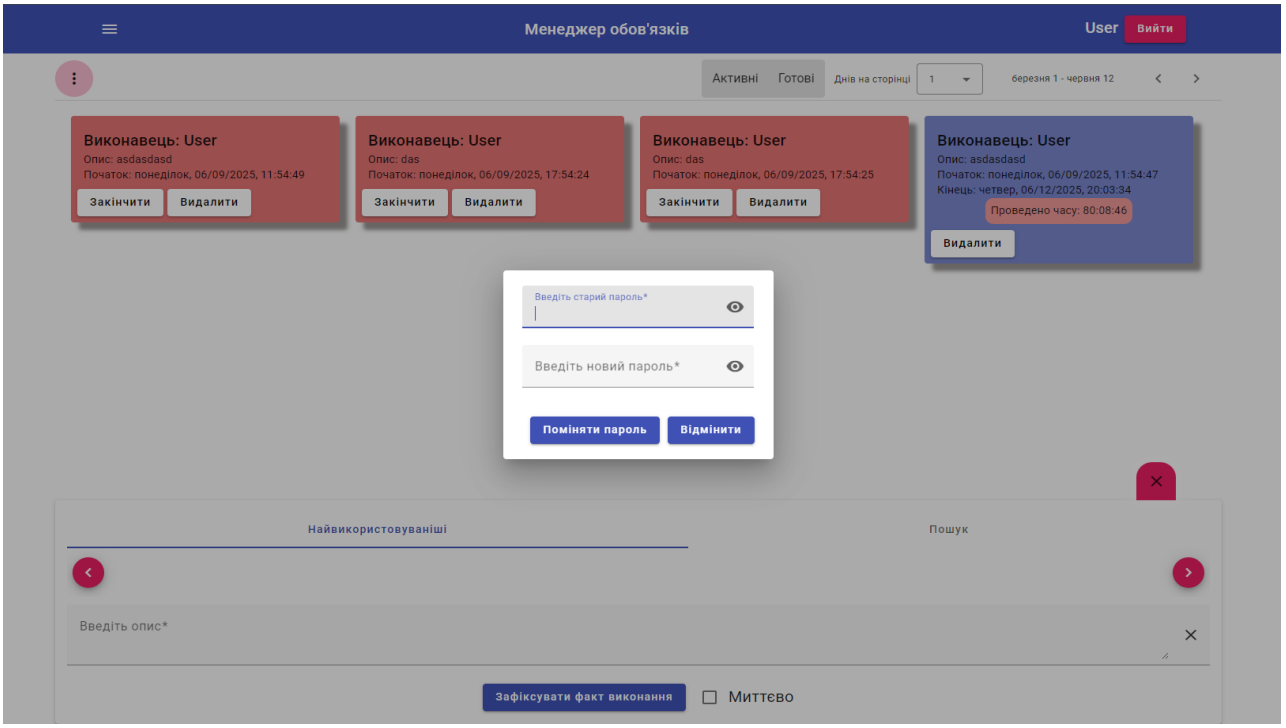

Рисунок 3.11 – Поп-ап зміни паролю

Admin – має всі можливості що й User. Також може додавати/редагувати/видаляти шаблони. Також може видаляти факти виконання інших користувачів за умови, що даний факт виконання не є підтвердженим. Також може реєструвати нових користувачів (рис. 3.12-3.14).

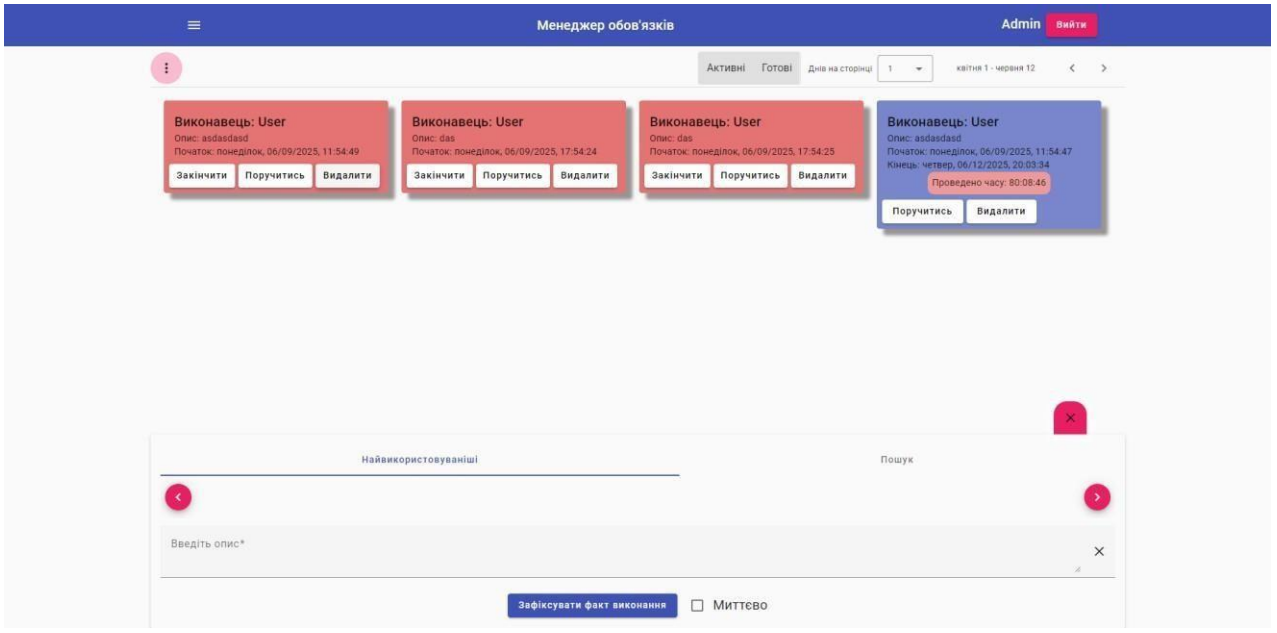

Рисунок 3.12 – Сторінка перегляду фактів виконання у режимі адміністратора

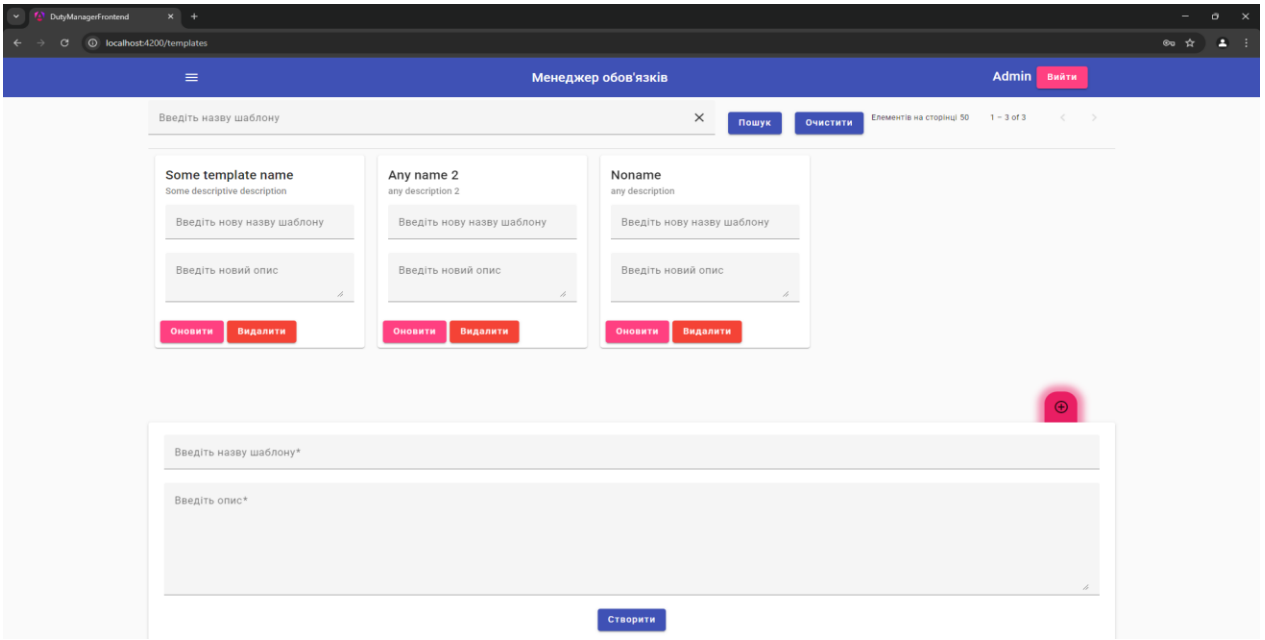

Рисунок 3.13 – Сторінка перегляду шаблонів у режимі адміністратора

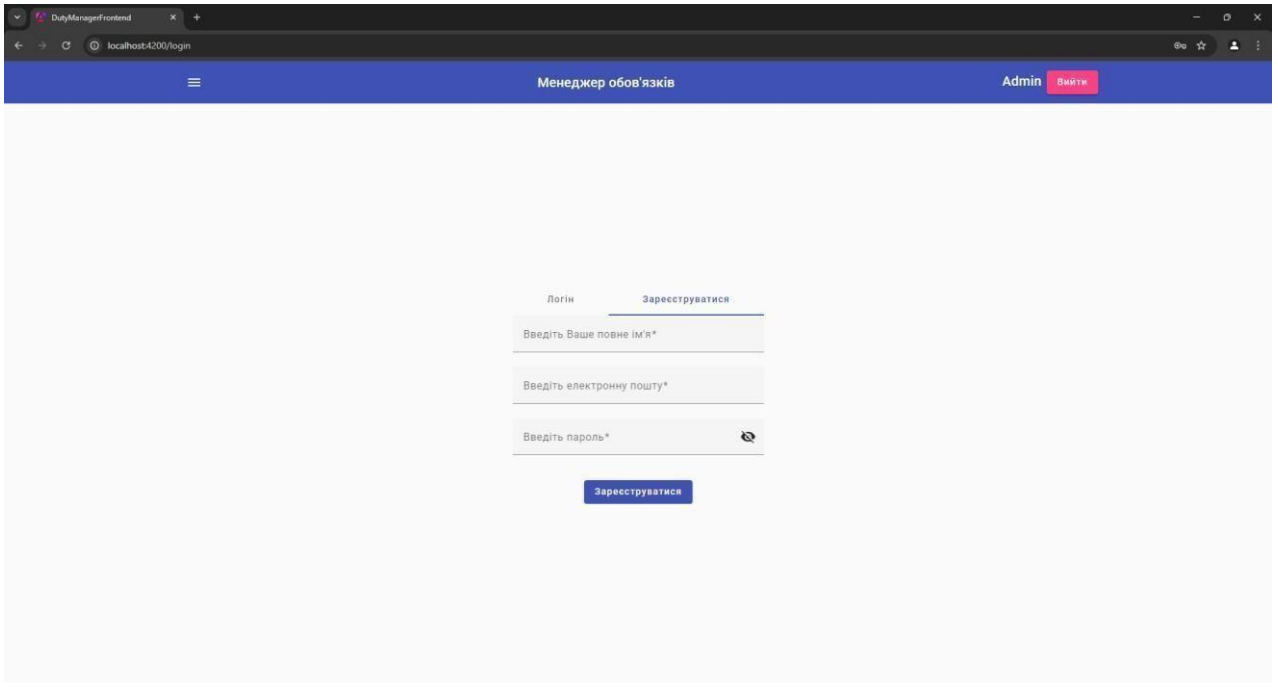

Рисунок 3.14 – Сторінка реєстрації

3.3 Опис сторінок

Сторінка логіну, також є поп-ап з логіном. Поп-ап з'являється за умови переходу до захищеної сторінки без попередньої авторизації. Також ця сторінка буде містити розділ реєстрації користувачів, за умови її перегляду адміністратором (рис. 3.14, 3.15, 3.16).

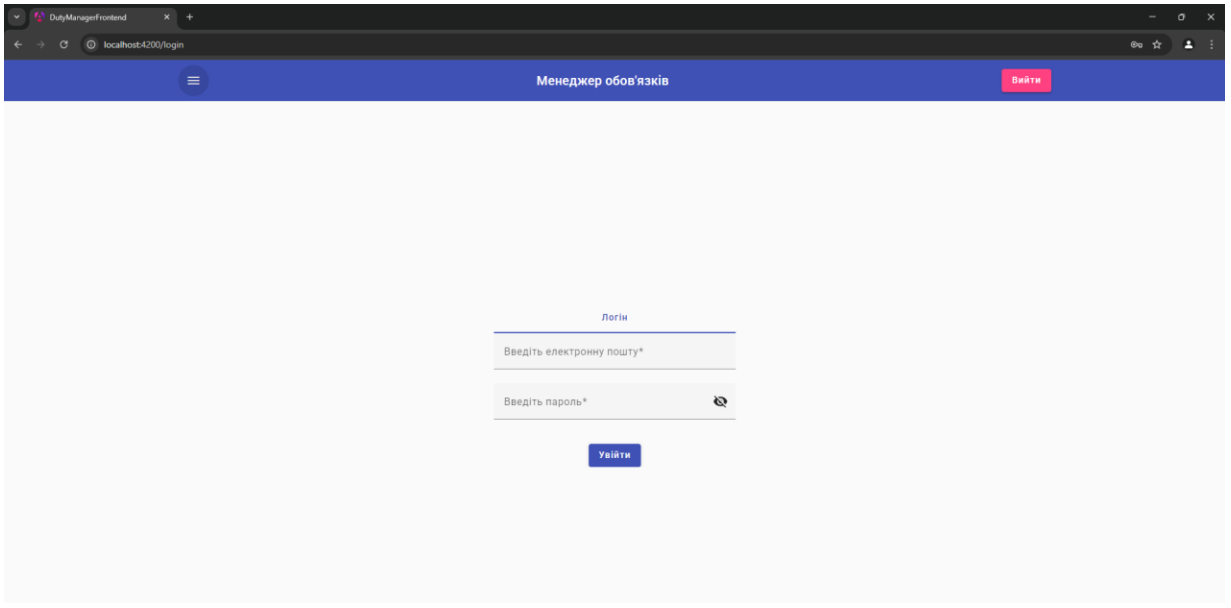

Рисунок 3.15 - Сторінка логіну

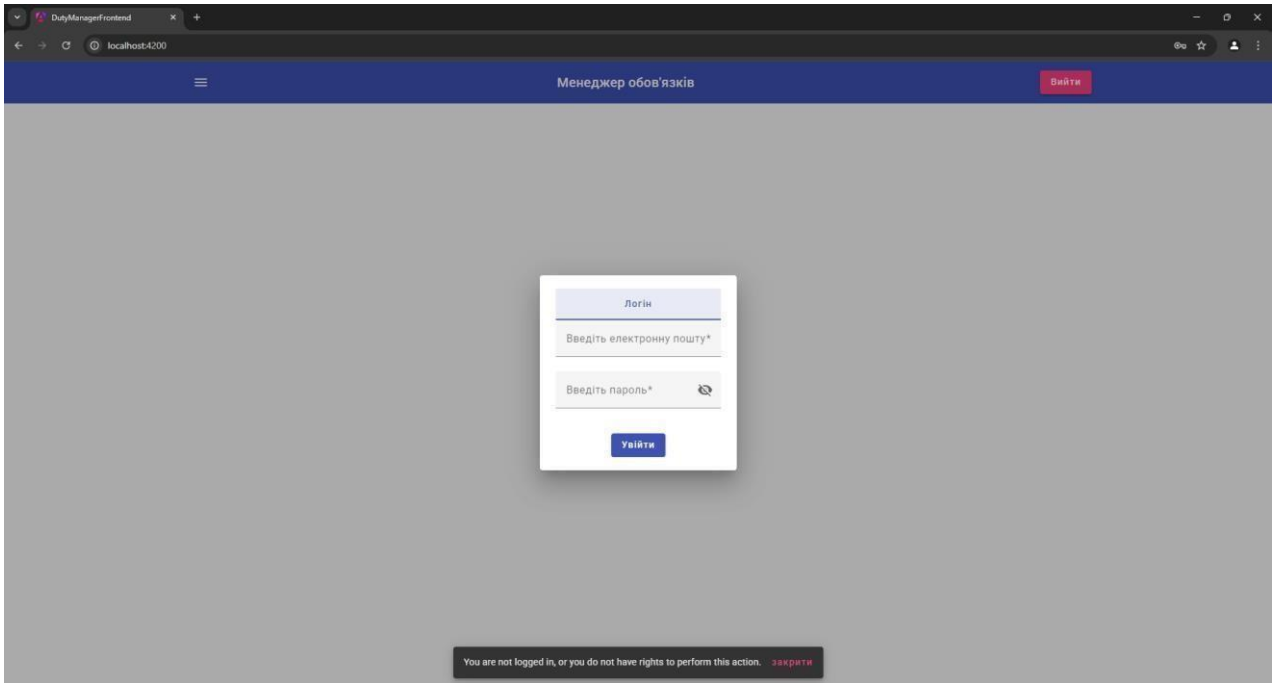

Рисунок 3.16 – Поп-ап логіну

Поп-ап з логіном було реалізовано за допомогою компонента з Angular material [2] та бібліотеки Axios [9].

```
        axios.interceptors.response.use(null, (error: AxiosError) => {
          const notAuthenticated = error.response?.status === 401;
          const loginIsCurrentPage = router.url === '/login';
          if (notAuthenticated && !loginIsCurrentPage && !this.loginFormOpened)
{
            const dialogRef = this.dialog.open(AuthenticationComponent);
            this.loginFormOpened = true;
            dialogRef.afterClosed().subscribe(() => {
              this.loginFormOpened = false;
            });
          }
          if (notAuthenticated && loginIsCurrentPage) {
            snackBar.open($localize`You are not logged in.`, $localize`close`,
{
              duration: 2000,
            });
          }
          if (error.config!.url === 'auth/jwt') {
            snackBar.open(
              $localize`Login or password are incorrect!`,
              $localize`close`,
              {
                duration: 2000,
              }
            );
          }
          return Promise.reject(error);
        });
```

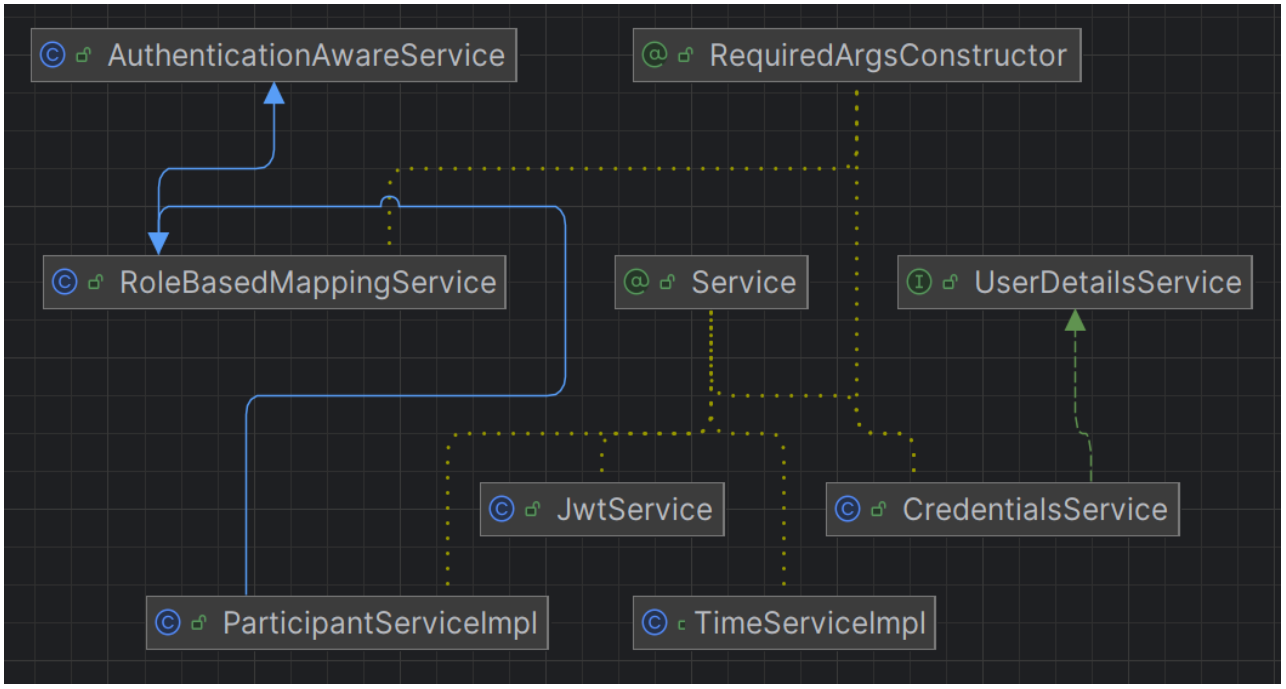

Рисунок 3.17 – UML-діаграма класів задіяних в бізнес-логіці для авторизації та автентифікації

Домашня сторінка. На ній можна переглянути свої незакінчені факти виконання обов'язків або переглянути непідтверджені вами факти виконання інших користувачів (рис. 3.8).

Сторінка перегляду фактів виконання. Тут можна створити свій факт виконання, переглядати детальніше свої факти виконання або факти виконання іншого користувача. Також тут реалізований функціонал фільтрації, розумного пошуку шаблонів та збереження нещодавно використаних шаблонів для швидкого доступу (рис. 3.10, 3.19, 3.20).

Для функціоналу розумного пошуку було використано pg_trgm модуль [5] з PostgreSql. Пропоную глянути на sql запит, який дозволяє отримати таку поведінку для пошуку.

```
WITH similarity_query AS (
   SELECT *, word_similarity(name, :name) AS name_similarity
   FROM templates
)
SELECT id, name, description, version
FROM similarity_query
WHERE name_similarity >= 0.1
ORDER BY name_similarity desc
LIMIT 50;
```

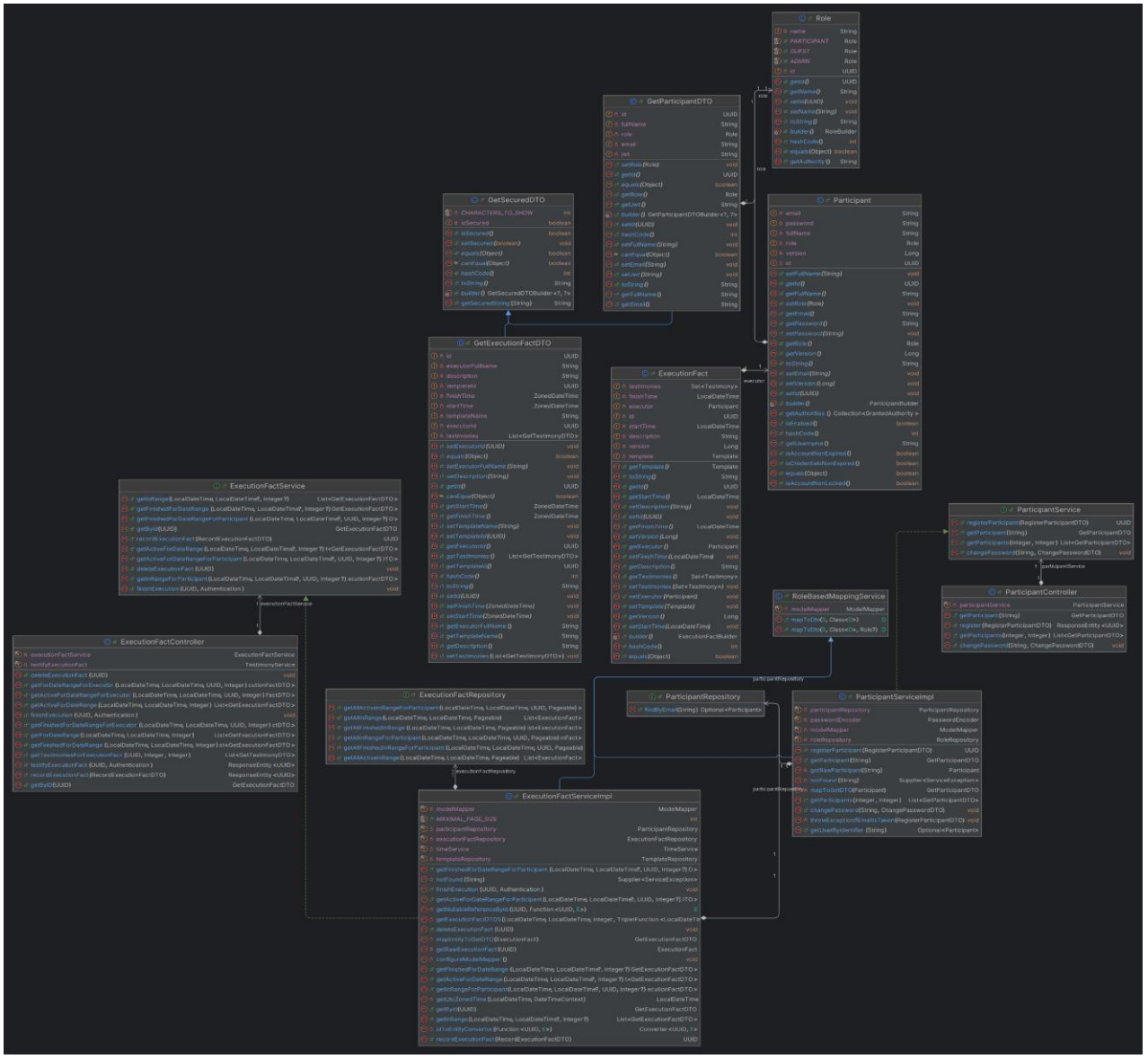

Рисунок 3.17 – UML-діаграма класів, які задіяні в бізнес-логіці на сервері для домашньої сторінки

Також тут доволі заплутана логіка відображення кнопок в залежності від стану сутності або ролей користувача, тому пропоную глянути на код, який це все реалізує.

```
        @if( (executionFact.finishTime === null && loggedInUserViewingHisFacts)
||
          (executionFact.finishTime === null && userIsAdmin) ||
          (!loggedInUserViewingHisFacts && userHaveNotTestified) ||
          (executionFact.testimonies.length === 0 && loggedInUserViewingHisFacts)
||
          (executionFact.testimonies.length === 0 && userIsAdmin)
          ) {
        <mat-card-actions>
          @if(executionFact.finishTime === null && (loggedInUserViewingHisFacts
||
```

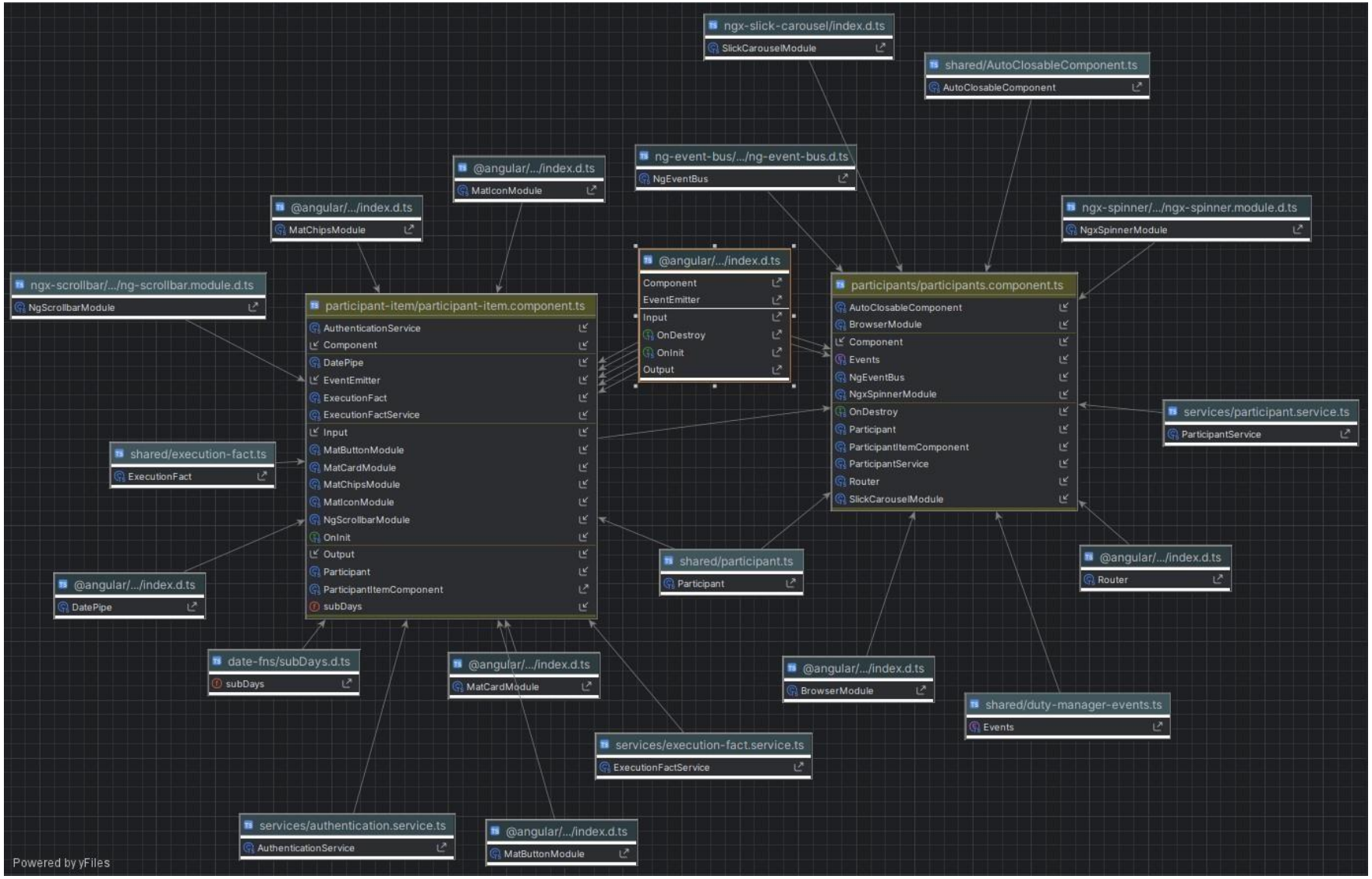

Рисунок 3.18 – UML-діаграма класів, які задіяні в бізнес-логіці на UI для домашньої сторінки

```
        (userIsAdmin && !loggedInUserViewingHisFacts))) {
        <button      type="button"      mat-raised-button      (click)="finish()"
i18n>Finish</button>
        } @if(!loggedInUserViewingHisFacts && userHaveNotTestified) {
        <button      type="button"      mat-raised-button      (click)="testify()"
i18n>Testify</button>
        } @if((loggedInUserViewingHisFacts || userIsAdmin) &&
        executionFact.testimonies.length === 0) {
        <button    type="button"    mat-raised-button    (click)="deleteFact()"
i18n>Delete</button>
        }
```

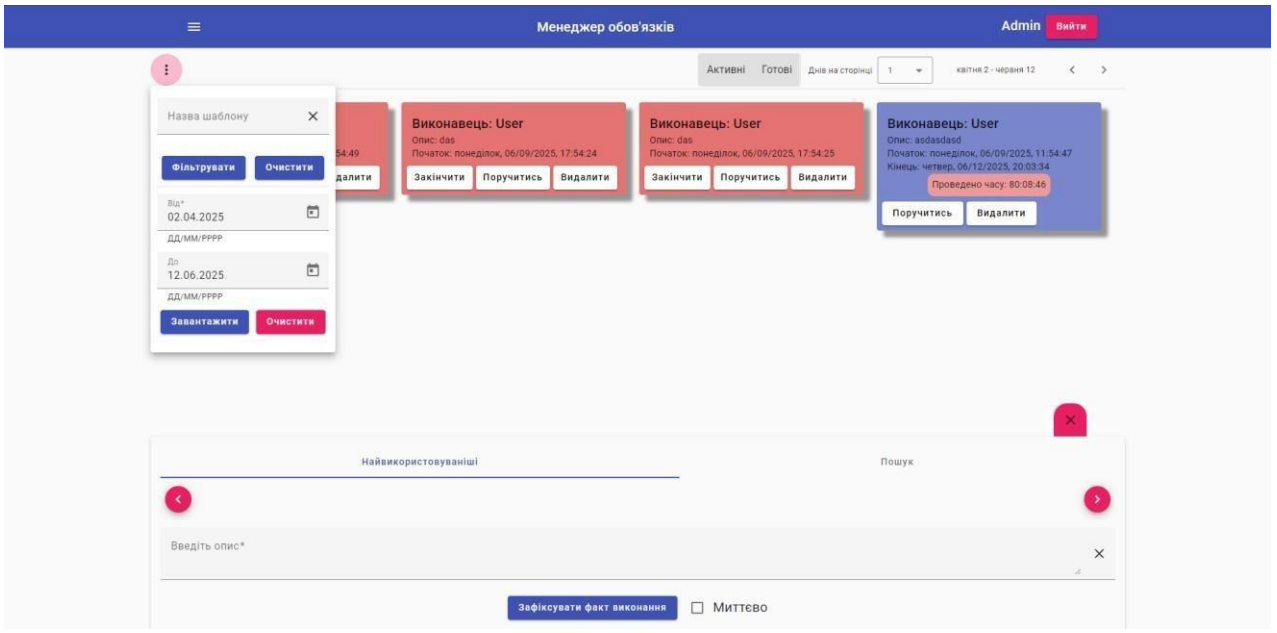

Рисунок 3.19 – Сторінка перегляду фактів виконання, демонстрація фільтрів

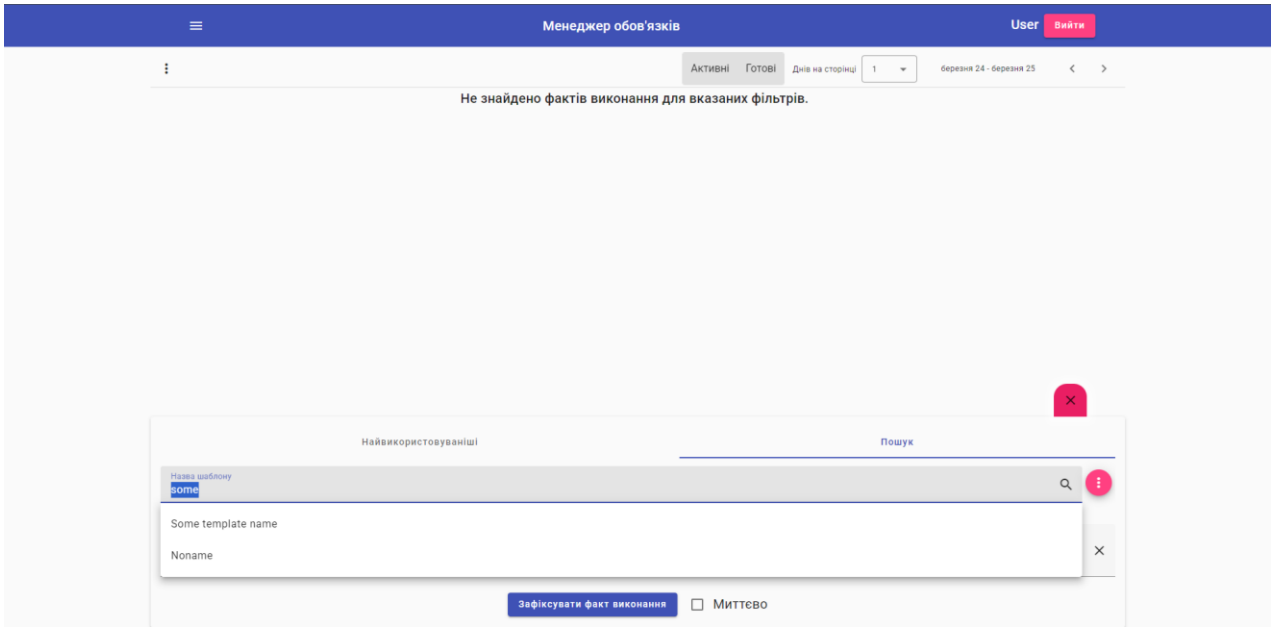

Рисунок 3.20 – Сторінка перегляду фактів виконання, демонстрація розумного пошуку

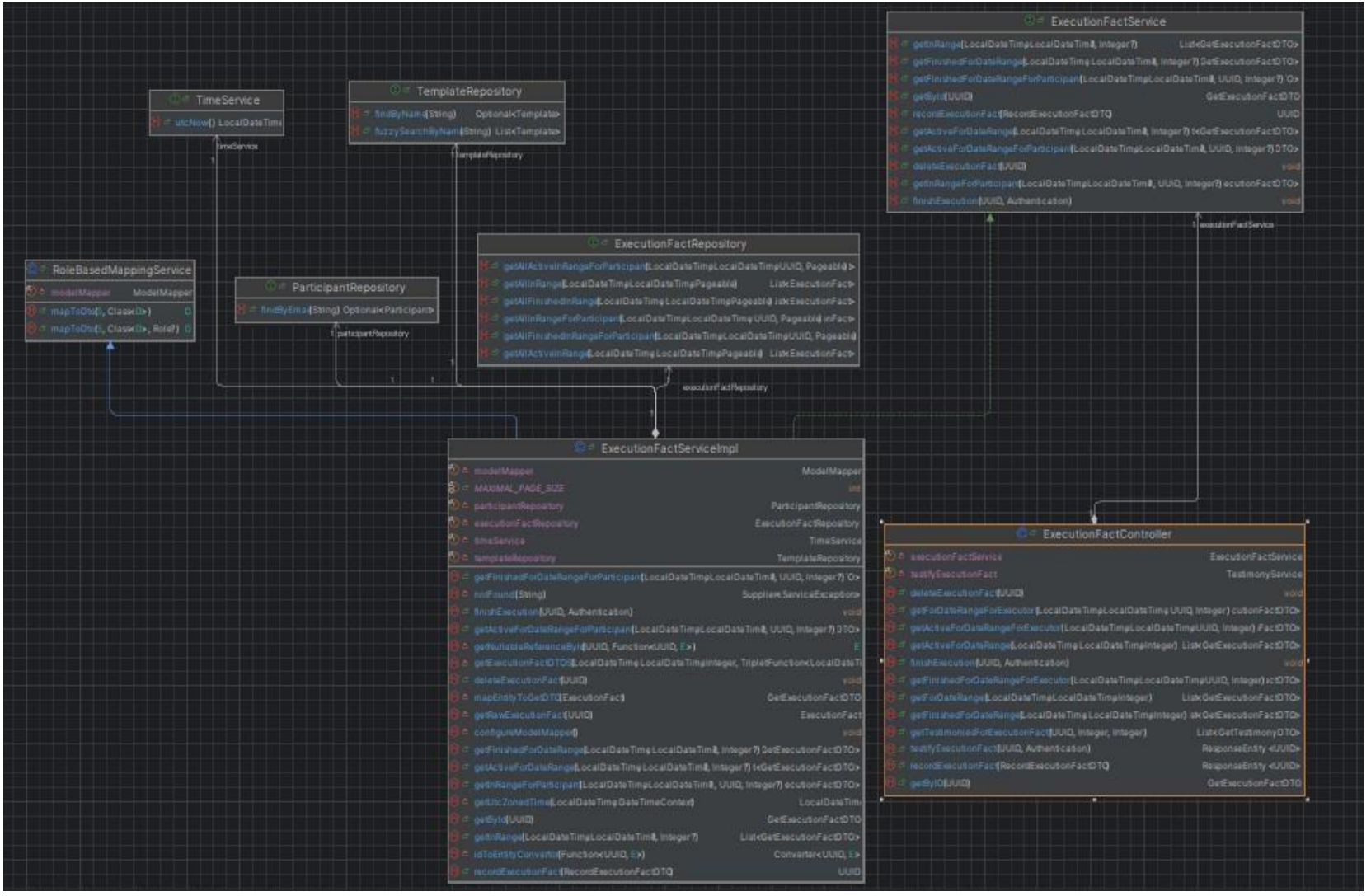

Рисунок 3.21 – UML-діаграма класів, які задіяні в бізнес-логіці на сервері для сторінки перегляду фактів виконання

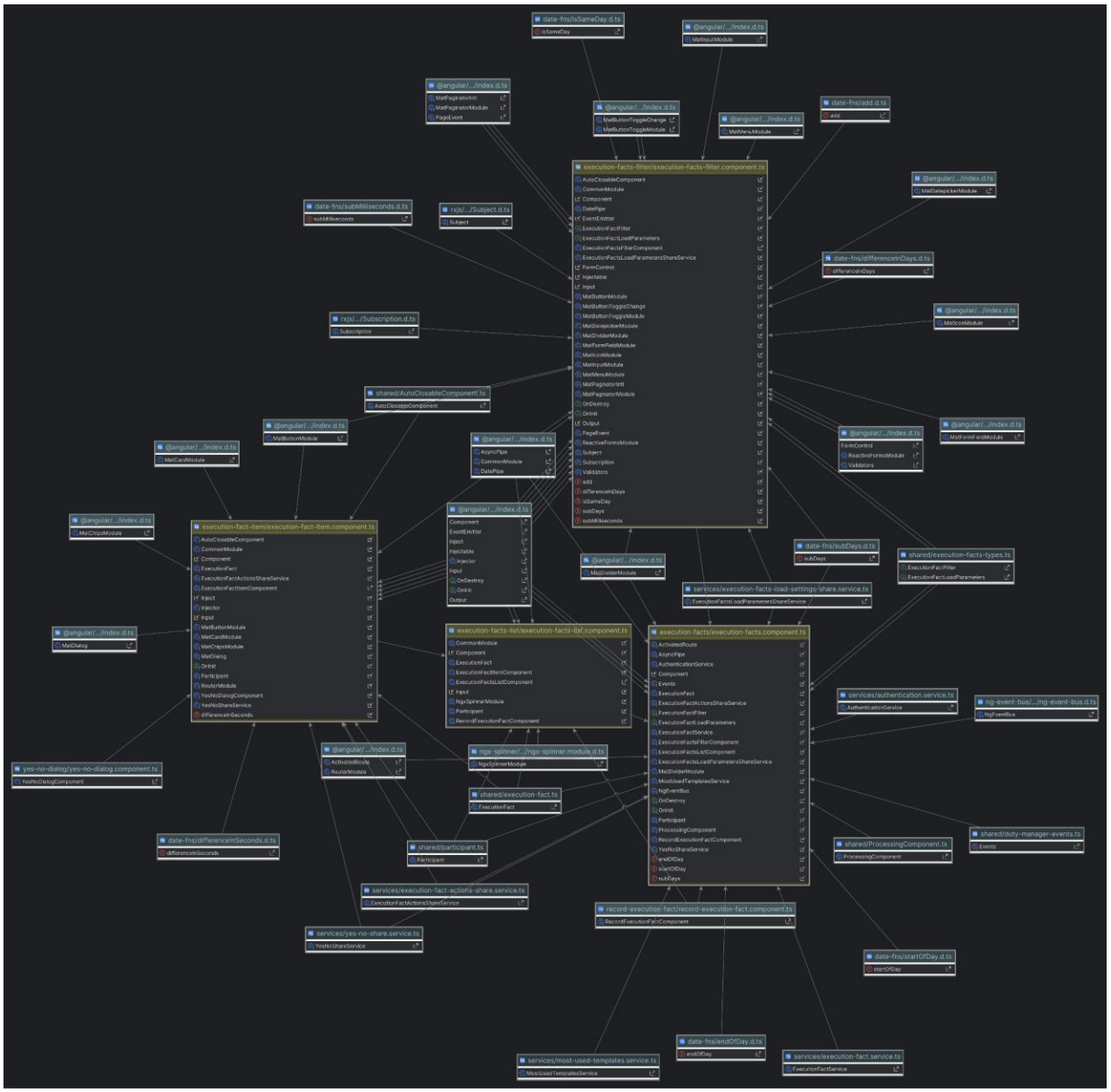

Рисунок 3.22 – UML-діаграма класів, які задіяні в бізнес-логіці на UI для сторінки перегляду фактів виконання

Сторінка перегляду шаблонів. Тут можна переглядати, створювати, редагувати або видаляти шаблони. Також тут є зручний фільтр для пошуку шаблонів (рис. 3.9, 3.11).

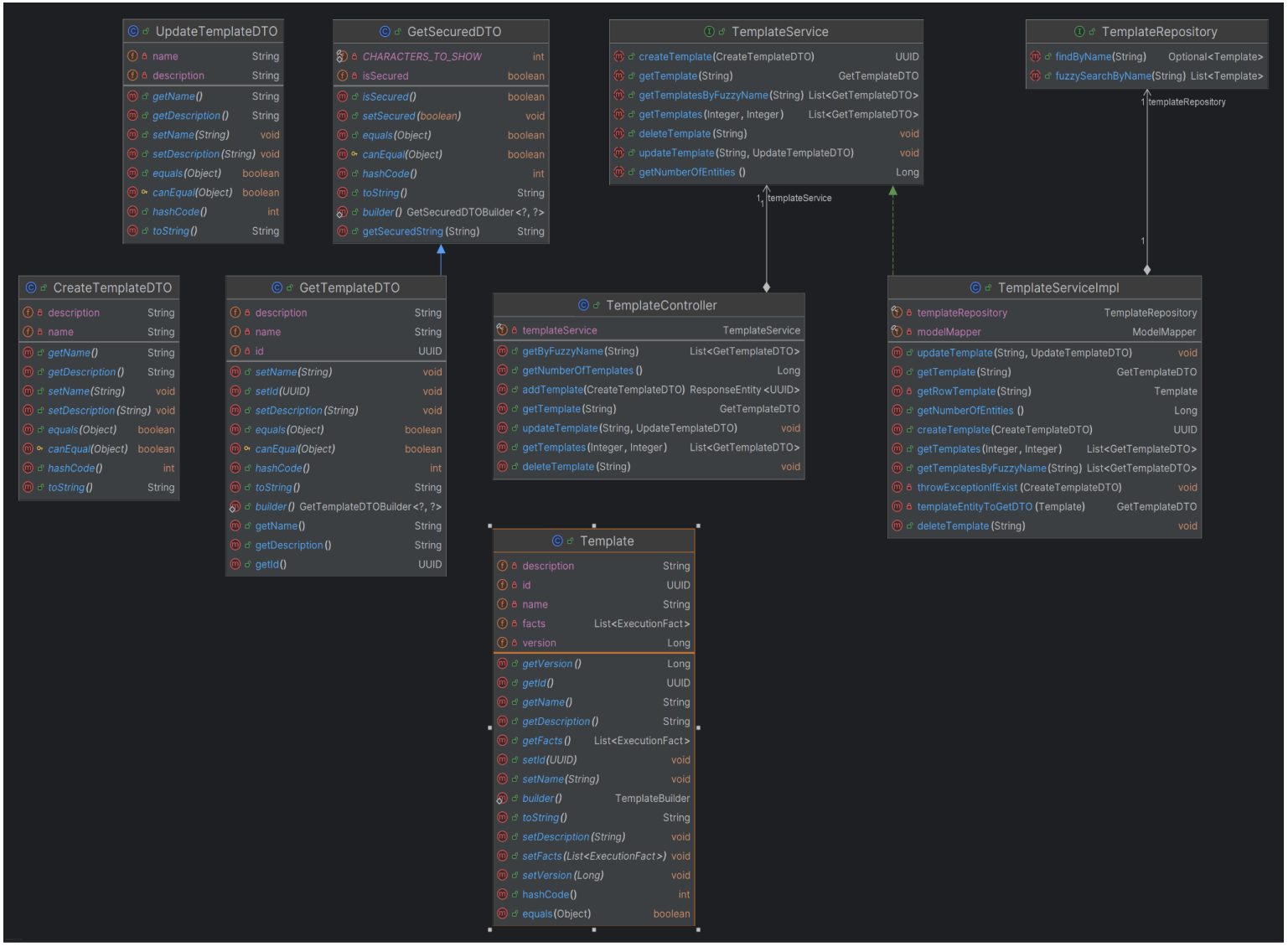

Рисунок 3.23 – UML-діаграма класів, які задіяні в бізнес-логіці на сервері для сторінки перегляду шаблонів

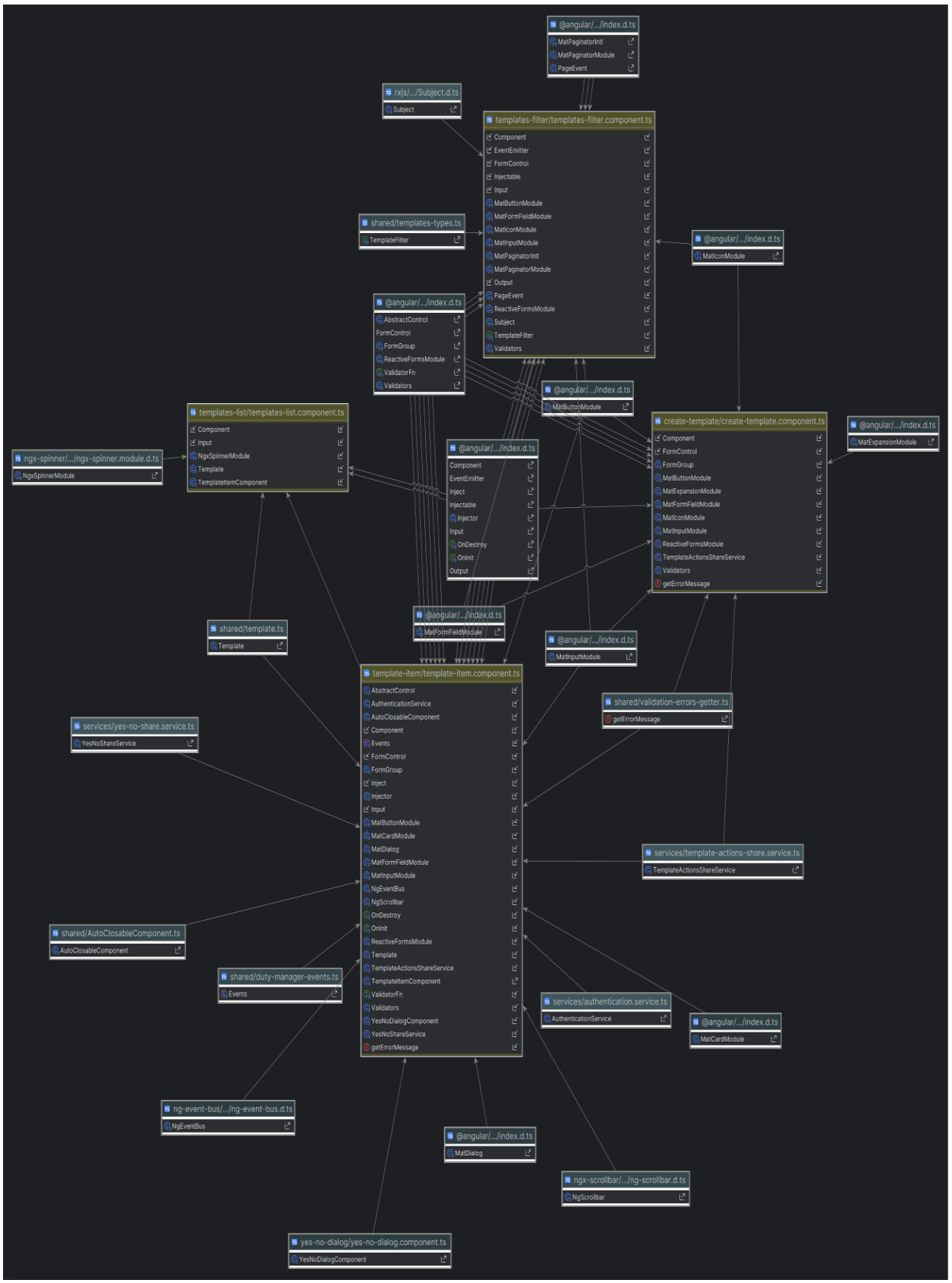

Рисунок 3.24 – UML-діаграма класів, які задіяні в бізнес-логіці на UI для сторінки перегляду шаблонів

Навігація у розробленому менеджері обов’язків представлена на рисунку 3.25.

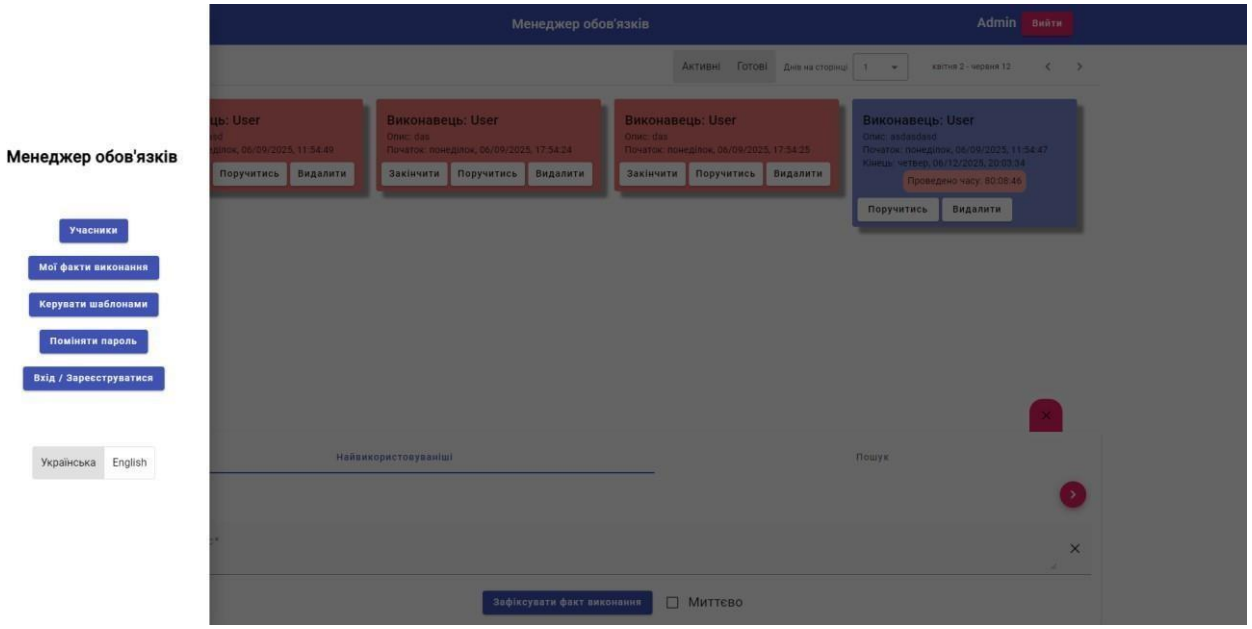

Рисунок 3.25 – Навігація

Ефективна навігація вебсистеми менеджера службових обов’язків “DutyDesk” забезпечує швидкий доступ до ключових функцій, мінімізує час пошуку інформації та підвищує продуктивність користувачів. Завдяки логічній структурі меню, що охоплює факти виконання, управління шаблонами, керування учасниками та налаштування системи, користувач легко орієнтується в інтерфейсі та може оперативно виконувати свої завдання не відволікаючись на зайві опції. Така мінімалістична побудова сприяє зменшенню навантаження на користувача, знижує ймовірність помилок і створює позитивний досвід взаємодії з системою.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено вебсистему "DutyDesk" для контролю виконання службових обов'язків і доручень працівників. Система забезпечує ключовий функціонал для ефективного управління завданнями: перегляд, створення, редагування, підтвердження, вилучення фактів виконання обов'язків, а також можливість використовувати шаблони завдань.

Розробка включала реалізацію таких важливих компонентів, як:

- сторінка виконавців із відображенням завдань;
- механізм контролю виконання завдань з урахуванням статусу підтвердження та ролі користувача;
- функціонал роботи з шаблонами;
- модуль аутентифікації та авторизації, який забезпечує доступ до системи лише уповноваженим особам, включаючи можливість конфіденційного входу через спеціальне посилання;
- захист персональних даних користувачів та обмеження доступу до чутливої інформації;

Для реалізації системи були використані сучасні технології Java, Spring Boot, Spring Security, RESTful API, PostgreSQL, Docker, Angular, GCP та інші інструменти, які забезпечили масштабованість, безпеку та зручність у користуванні.

Таким чином, поставлену мету дипломної роботи було досягнуто. Система "DutyDesk" може бути впроваджена в організаційне середовище для покращення внутрішнього контролю за виконанням обов'язків та підвищення відповідальності персоналу. У майбутньому система може бути доповнена мобільним додатком, розширеними аналітичними можливостями та інтеграцією з іншими корпоративними сервісами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Angular home [Електронний ресурс]. – Режим доступу: https://angular.dev/ – Дата звернення: 09.06.2025.
2. Angular Material UI component library [Електронний ресурс]. – Режим доступу: https://material.angular.io/ – Дата звернення: 09.06.2025.
3. Spring | Projects [Електронний ресурс]. – Режим доступу: https://spring.io/projects – Дата звернення: 09.06.2025.
4. Baeldung [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/ – Дата звернення: 09.06.2025.
5. PostgreSQL: Documentation: 17: F.33. pg_trgm — support for similarity of text using trigram matching [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/current/pgtrgm.html – Дата звернення: 09.06.2025.
6. TypeScript: Handbook – The TypeScript Handbook [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/handbook/intro.html – Дата звернення: 09.06.2025.
7. Google Cloud Documentation [Електронний ресурс]. – Режим доступу: https://cloud.google.com/docs – Дата звернення: 09.06.2025.
8. Nginx documentation [Електронний ресурс]. – Режим доступу: https://nginx.org/en/docs/ – Дата звернення: 09.06.2025.
9. Getting Started | Axios Docs [Електронний ресурс]. – Режим доступу: https://axios-http.com/docs/intro – Дата звернення: 09.06.2025.
10. Restful APIs [Електронний ресурс]. – Режим доступу: https://restfulapi.net/ – Дата звернення: 09.06.2025.
11. Що таке Jira і як з нею працювати [Електронний ресурс]. – Режим доступу: https://iampm.club/ua/blog/shho-take-jira-i-yak-z-neyu-praczyuvati/ Дата звернення: 09.06.2025.
12. Trello: що це таке та як ним користуватися [Електронний ресурс]. – Режим доступу: https://hostiq.ua/blog/ukr/what-is-trello-2/ Дата звернення: 09.06.2025.

13. Notion: основні функції та оновлення [Електронний ресурс]. – Режим доступу: https://www.theantmedia.com/post/notion-osnovni-funkciyi-ta-onovlennya Дата звернення: 09.06.2025.
14. Повний гід по бекенд розробці: теорія та практика [Електронний ресурс]. – Режим доступу: https://galaktica.io/blog/backend/ Дата звернення: 09.06.2025.
15. Що краще — React чи Angular? [Електронний ресурс]. – Режим доступу: https://artjoker.ua/blog/chto-luchshe-react-ili-angular/ Дата звернення: 09.06.2025.

ДОДАТКИ

Додаток А

Код основних сервісів та компонентів на фронтенді

```
import { Component, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs';

@Component({ template: '<h1>Auto Closable component</h1>' })
export class AutoClosableComponent implements OnDestroy {
private subscriptions: Subscription[] = [];

  public manage(subscription: Subscription) {
    this.subscriptions.push(subscription);
  }

  public ngOnDestroy(): void {
    this.subscriptions.forEach((sub) => sub.unsubscribe());
  }
}


import { NgEventBus } from 'ng-event-bus';
import { AutoClosableComponent } from './AutoClosableComponent';
import { Events } from './duty-manager-events';

export class ProcessingComponent<TData> extends AutoClosableComponent {
  private _data: TData[] = [];

  constructor(
    eventBus: NgEventBus,
    onLoginDataSupplier: () => Promise<TData[]>,
    private afterDataChangedCallBack?: (data: TData[]) => void
  ) {
    super();
    super.manage(
      eventBus.on(Events.LOGGED_IN).subscribe(() =>
        onLoginDataSupplier().then((data) => {
        this.replaceData(data);
        })
      )
    );
  }

  protected async replaceDataPiece(
    searchingCallback: (piece: TData) => boolean,
    supplyingCallback: () => Promise<TData | null>
  ) {
    const supply = await supplyingCallback();
    if (supply) {
      this._data.splice(this._data.findIndex(searchingCallback), 1, supply);
```

```
    } else {
      this._data.splice(this._data.findIndex(searchingCallback), 1);
    }
    if(this.afterDataChangedCallBack) {
      this.afterDataChangedCallBack(this._data);
    }
  }

  protected async addDataPiece(supplyingCallback: () => Promise<TData>) {
    this._data.push(await supplyingCallback());
    if(this.afterDataChangedCallBack) {
    this.afterDataChangedCallBack(this._data);
    }
  }

  protected replaceData(data: TData[]) {
    this._data = data;
    if(this.afterDataChangedCallBack) {
    this.afterDataChangedCallBack(data);
    }
  }

  get data() {
    return this._data;
  }
}


import { FormGroup, ValidationErrors } from '@angular/forms';

export function getErrorMessage(
  formControlName: string,
  form: FormGroup
): string {
  const formControl = form.get(formControlName);
  if (formControl?.hasError('required')) {
    return $localize`This field is required`;
  } else if (formControl?.hasError('email')) {
    return $localize`Invalid email`;
  } else if (formControl?.hasError('minlength')) {
    return $localize`Password should be at least 8 symbols`;
  } else if (formControl?.hasError('maxlength')) {
    return $localize`Max allowed length is ${formControl.getError('maxlength').requiredLength}`;
  }
  return errorsToText(formControl?.errors);
}

function errorsToText(errors: ValidationErrors | null | undefined): string {
  if (!errors) {
    return '';
  }
  let message = '';
  for (const prop in errors) {
    message += errors[prop] + ' ';
  }
```

```
  return message;
}
import { inject } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivateFn, Routes } from '@angular/router';
import { GuestService } from './services/guest.service';
import { PageMode } from '../shared/page-modes';
import { PageModeShareService } from './services/page-mode-share.service';

const guestActivate: CanActivateFn = (route: ActivatedRouteSnapshot) => {
  return inject(GuestService).checkForGuest(route);
};

const pageModeActivate = (mode: PageMode): CanActivateFn => {
  return (route: ActivatedRouteSnapshot) => {
  inject(PageModeShareService).nextLoadParameters(mode);
    return true;
  };
};

export const routes: Routes = [
  {
    path: 'login',
    loadComponent: () =>
      import('./authentication/authentication.component').then(
        (m) => m.AuthenticationComponent
      ),
    canActivate: [pageModeActivate(PageMode.STANDARD)],
  },
  {
    path: 'execution-facts',
    loadComponent: () =>
      import('./execution-facts/execution-facts.component').then(
        (m) => m.ExecutionFactsComponent
      ),
    canActivate: [pageModeActivate(PageMode.STANDARD)],
  },
  {
    path: 'templates',
    loadComponent: () =>
      import('./templates/templates.component').then(
        (m) => m.TemplatesComponent
      ),
    canActivate: [pageModeActivate(PageMode.STANDARD)],
  },
  {
    path: '',
    loadComponent: () =>
      import('./participants/participants.component').then(
        (m) => m.ParticipantsComponent
      ),
    canActivate: [guestActivate, pageModeActivate(PageMode.FULL_WIDTH)],
  },
];
```

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { MatSidenavModule } from '@angular/material/sidenav';
import { RouterOutlet } from '@angular/router';
import { NgEventBus } from 'ng-event-bus';
import { Events } from '../shared/duty-manager-events';
import { Role } from '../shared/participant';
import { HeaderComponent } from './header/header.component';
import { AuthenticationService } from './services/authentication.service';
import { ExecutionFactService } from './services/execution-fact.service';
import { ParticipantService } from './services/participant.service';
import { TemplateService } from './services/template.service';
import { SideNavComponent } from './side-nav/side-nav.component';
import { NgxSpinnerModule } from 'ngx-spinner';
import { PageModeShareService } from './services/page-mode-share.service';
import { Subscription } from 'rxjs';
import { PageMode } from '../shared/page-modes';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [
  RouterOutlet,
  HeaderComponent,
  MatSidenavModule,
  SideNavComponent,
  NgxSpinnerModule,
  ],
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss',
  providers: [ExecutionFactService, TemplateService, ParticipantService],
})
export class AppComponent implements OnInit, OnDestroy {
  private _userRole?: Role;
  private pageModeSubscription: Subscription;
  currentPageMode = PageMode.STANDARD;

  constructor(
    private authenticationService: AuthenticationService,
    eventBus: NgEventBus,
    pageModeShareService: PageModeShareService
  ) {
    this._userRole = this.getUserRole();
    eventBus
      .on(Events.LOGGED_IN)
      .subscribe(() => (this._userRole = this.getUserRole()));
    eventBus
      .on(Events.LOGGED_OUT)
      .subscribe(() => (this._userRole = undefined));
    this.pageModeSubscription = pageModeShareService.onLoadParameters(
    (mode) => (this.currentPageMode = mode)
    );
  }

  ngOnInit(): void {
    this.authenticationService.checkIfAlreadyLoggedIn();
```

```
  }

  ngOnDestroy(): void {
    this.pageModeSubscription.unsubscribe();
  }

  private getUserRole() {
    return this.authenticationService.getParticipant()?.role;
  }

  get userRole() {
    return this._userRole;
  }
}


import { Component, Inject } from '@angular/core';
import { MatButtonModule } from '@angular/material/button';
import {
  MAT_DIALOG_DATA,
  MatDialogActions,
  MatDialogClose,
  MatDialogContent,
  MatDialogTitle,
} from '@angular/material/dialog';
import { YesNoShareService } from '../services/yes-no-share.service';

@Component({
  selector: 'app-yes-no-dialog',
  standalone: true,
  imports: [
    MatButtonModule,
    MatDialogActions,
    MatDialogClose,
    MatDialogTitle,
    MatDialogContent,
  ],
  templateUrl: './yes-no-dialog.component.html',
  styleUrl: './yes-no-dialog.component.scss',
})
export class YesNoDialogComponent<TIssuer> {
  constructor(
    @Inject(MAT_DIALOG_DATA)
    public data: { title: string; message: string; issuer: TIssuer },
    public yesNoShareService: YesNoShareService<TIssuer>
  ) {}

  nextDecision(decision: boolean) {
    this.yesNoShareService.nextDecision({
    issuer: this.data.issuer,
      decision: decision,
    });
  }
}
```

```
import {
  Component,
  EventEmitter,
  Injectable,
  Input,
  Output,
} from '@angular/core';
import { FormControl, ReactiveFormsModule, Validators } from '@angular/forms';
import { MatButtonModule } from '@angular/material/button';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
import { MatInputModule } from '@angular/material/input';
import {
  MatPaginatorIntl,
  MatPaginatorModule,
  PageEvent,
} from '@angular/material/paginator';
import { TemplateFilter } from '../../shared/templates-types';
import { Subject } from 'rxjs';

@Injectable()
export class ExecutionFactPaginatorLabels extends MatPaginatorIntl {
  override itemsPerPageLabel = $localize`Items per page`;
  override nextPageLabel = $localize`Next page`;
  override previousPageLabel = $localize`Previous page`;
}

@Component({
  selector: 'templates-filter',
  standalone: true,
  imports: [
    MatButtonModule,
    ReactiveFormsModule,
    MatInputModule,
    MatFormFieldModule,
    MatPaginatorModule,
    MatIconModule,
  ],
  providers: [
    { provide: MatPaginatorIntl, useClass: ExecutionFactPaginatorLabels },
  ],
  templateUrl: './templates-filter.component.html',
  styleUrl: './templates-filter.component.scss',
})
export class TemplatesFilterComponent {
  senderTemplateName = new FormControl<string>('', [Validators.maxLength(100)]);
  activeFilters: TemplateFilter[] = [];
  @Input() defaultPage = { pageIndex: 0, pageSize: 50 };
  @Input() numberOfTemplates = 0;
  @Output() filtersChange = new EventEmitter<TemplateFilter[]>();
  @Output() pageChange = new EventEmitter<PageEvent>();

  updateFilters() {
    this.activeFilters = new Array<TemplateFilter>();
```

```
    const templateName = this.senderTemplateName.getRawValue();
    if (templateName && this.senderTemplateName.valid) {
    this.activeFilters.push((template) => {
        return template.name == templateName;
      });
    }
  }

  emitFilters() {
    this.filtersChange.emit(this.activeFilters);
  }

  clearFilters() {
    this.activeFilters = [];
  }
}


import { Component, OnDestroy, OnInit } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { MatDivider } from '@angular/material/divider';
import { PageEvent } from '@angular/material/paginator';
import { NgEventBus } from 'ng-event-bus';
import { ProcessingComponent } from '../../shared/ProcessingComponent';
import { Template } from '../../shared/template';
import { TemplateFilter } from '../../shared/templates-types';
import { CreateTemplateComponent } from '../create-template/create-template.component';
import { TemplateActionsShareService } from '../services/template-actions-share.service';
import { TemplateService } from '../services/template.service';
import { YesNoShareService } from '../services/yes-no-share.service';
import { TemplatesFilterComponent } from '../templates-filter/templates-filter.component';
import { TemplatesListComponent } from '../templates-list/templates-list.component';
import { AuthenticationService } from '../services/authentication.service';
import { Events } from '../../shared/duty-manager-events';

@Component({
  selector: 'templates',
  standalone: true,
  imports: [
    TemplatesListComponent,
    CreateTemplateComponent,
    TemplatesFilterComponent,
    MatDivider,
  ],
  providers: [
    MatDialog,
    {
      provide: 'templateYesNoShare',
      useValue: new YesNoShareService<Template>(),
    },
  ],
  templateUrl: './templates.component.html',
  styleUrl: './templates.component.scss',
})
export class TemplatesComponent
```

```
  extends ProcessingComponent<Template>
  implements OnInit, OnDestroy
{
  private _filters: TemplateFilter[] = [];
  private _page = { pageIndex: 0, pageSize: 50 };
  private _numberOfTemplates = 0;
  userIsAdmin: boolean = false;

  constructor(
    private templateService: TemplateService,
    eventBus: NgEventBus,
    templateActionsShare: TemplateActionsShareService,
    private authService: AuthenticationService
  ) {
    super(eventBus, () => this.loadTemplates());
    super.manage(
    templateActionsShare.onCreate((newTemplate) =>
       templateService.createTemplate(newTemplate).then((id) => {
         super.addDataPiece(() => templateService.fetchTemplate(id));
       })
     )
    );
    super.manage(
      templateActionsShare.onUpdate((updatedTemplate) => {
      templateService.updateTemplate(updatedTemplate).then(() =>
          super.replaceDataPiece(
            (template) => template.id === updatedTemplate.id,
            () => templateService.fetchTemplate(updatedTemplate.id)
          )
        );
      })
    );
    super.manage(
      templateActionsShare.onDelete((id) => {
      templateService.deleteTemplate(id).then(() =>
          super.replaceDataPiece(
            (template) => template.id === id,
            () => Promise.resolve(null)
          )
        );
      })
    );
    this.setIsUserAdmin();
    super.manage(
      eventBus.on(Events.LOGGED_IN).subscribe(() => this.setIsUserAdmin())
    );
  }

  ngOnInit(): void {
    this.loadTemplates().then((templates) => super.replaceData(templates));
    this.loadNumberOfTemplates();
  }

  private setIsUserAdmin() {
    this.userIsAdmin = this.authService.getParticipant()?.role.name === 'ADMIN';
```

```
  }

  private async loadTemplates() {
    return this.templateService.fetchTemplates(
      this._page.pageIndex,
      this._page.pageSize
    );
  }

  private async loadNumberOfTemplates() {
    this._numberOfTemplates = await this.templateService.getNumberOfTemplates();
  }

  async loadNewPage(pageEvent: PageEvent) {
    this._page = pageEvent;
    super.replaceData(await this.loadTemplates());
  }

  get templates() {
    return super.data.filter((template) =>
      this._filters.every((filter) => filter(template))
    );
  }

  get numberOfTemplates() {
    return this._numberOfTemplates;
  }

  set filters(filters: TemplateFilter[]) {
    this._filters = filters;
  }

  get page() {
    return this._page;
  }
}


import {
  Component,
  Inject,
  Injector,
  Input,
  OnDestroy,
  OnInit,
} from '@angular/core';
import {
AbstractControl,
FormControl,
FormGroup,
ReactiveFormsModule,
ValidatorFn,
Validators,
} from '@angular/forms';
import { MatButtonModule } from '@angular/material/button';
```

```
import { MatCardModule } from '@angular/material/card';
import { MatDialog } from '@angular/material/dialog';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { NgScrollbar } from 'ngx-scrollbar';
import { AutoClosableComponent } from '../../shared/AutoClosableComponent';
import { Template } from '../../shared/template';
import { getErrorMessage } from '../../shared/validation-errors-getter';
import { TemplateActionsShareService } from '../services/template-actions-share.service';
import { YesNoShareService } from '../services/yes-no-share.service';
import { YesNoDialogComponent } from '../yes-no-dialog/yes-no-dialog.component';
import { AuthenticationService } from '../services/authentication.service';
import { NgEventBus } from 'ng-event-bus';
import { Events } from '../../shared/duty-manager-events';

@Component({
  selector: 'template-item',
  standalone: true,
  imports: [
  MatCardModule,
  NgScrollbar,
  MatButtonModule,
  ReactiveFormsModule,
  MatInputModule,
  MatFormFieldModule,
  ],
  templateUrl: './template-item.component.html',
  styleUrl: './template-item.component.scss',
})
export class TemplateItemComponent
  extends AutoClosableComponent
  implements OnInit, OnDestroy
{
  @Input() template = new Template('placeholder', 'placeholder', 'placeholder');
  userIsAdmin: boolean = false;

  updateTemplateForm = new FormGroup({
    senderTemplateName: new FormControl<string>(''),
    senderTemplateDescription: new FormControl<string>(''),
  });

  constructor(
    private templateActionShare: TemplateActionsShareService,
    @Inject('templateYesNoShare')
    private yesNoShareService: YesNoShareService<Template>,
    private dialog: MatDialog,
    private authService: AuthenticationService,
    eventBus: NgEventBus
  ) {
    super();
    super.manage(
      yesNoShareService.onDecision((decision) => {
        if (decision.decision && decision.issuer.id === this.template.id) {
          templateActionShare.nextDelete(this.template.id);
        }
```

```
    })
  );
  this.setIsUserAdmin();
  super.manage(
    eventBus.on(Events.LOGGED_IN).subscribe(() => this.setIsUserAdmin())
  );
}

ngOnInit(): void {
  this.updateTemplateForm = new FormGroup({
    senderTemplateName: new FormControl<string>('', [
    Validators.maxLength(100),
    this.nameNotSameValidator(),
    ]),
    senderTemplateDescription: new FormControl<string>('', [
      Validators.maxLength(500),
    ]),
  });
}

private setIsUserAdmin() {
  this.userIsAdmin = this.authService.getParticipant()?.role.name === 'ADMIN';
}

updateTemplate() {
  const newName = this.updateTemplateForm.get('senderTemplateName')?.value;
  const newDescription = this.updateTemplateForm.get(
  'senderTemplateDescription'
  )?.value;
  if ((newName || newDescription) && this.updateTemplateForm.valid)
    this.templateActionShare.nextUpdate({
      id: this.template.id,
      name: newName ?? undefined,
      description: newDescription ?? undefined,
    });
}

deleteTemplate() {
  this.dialog.open(YesNoDialogComponent<Template>, {
  injector: Injector.create({
      providers: [
        {
          provide: YesNoShareService<Template>,
          useValue: this.yesNoShareService,
        },
      ],
    }),
    data: {
      title: $localize`Do you want to delete`+ ` "`+ this.template.name + '" ' + $localize`template`,
      message: $localize`This action can not be reversed.`,
      issuer: this.template,
    },
  });
}
```

```
  getErrorMessage(formControlName: string) {
    return getErrorMessage(formControlName, this.updateTemplateForm);
  }

  nameNotSameValidator(): ValidatorFn {
    return (control: AbstractControl<string>) => {
      if (control.value.trim() === this.template.name) {
        return {
          invalidTemplateName: $localize`Name is already applied.`,
        };
      } else {
        return null;
      }
    };
  }
}


import { Component, Inject, Input, LOCALE_ID } from '@angular/core';
import { MatButtonModule } from '@angular/material/button';
import { MatButtonToggleModule } from '@angular/material/button-toggle';
import { MatIconModule } from '@angular/material/icon';
import { RouterModule } from '@angular/router';
import { Role } from '../../shared/participant';
import { MatDialog } from '@angular/material/dialog';
import { ChangePasswordComponent } from '../change-password/change-password.component';

@Component({
  selector: 'side-nav',
  standalone: true,
  imports: [
  RouterModule,
  MatButtonModule,
  MatIconModule,
    MatButtonToggleModule,
  ],
  templateUrl: './side-nav.component.html',
  styleUrl: './side-nav.component.scss',
})
export class SideNavComponent {
  @Input() userRole?: Role;

  locales = [
    { localeId: 'uk-UA', selected: false, name: 'Українська' },
    { localeId: 'en-US', selected: false, name: 'English' },
  ];

  constructor(@Inject(LOCALE_ID) public locale: string, private dialog: MatDialog) {
    this.locales.forEach((l) => (l.selected = l.localeId === locale));
  }

  changePassword() {
    this.dialog.open(ChangePasswordComponent);
  }
```

```
}


import { Injectable } from '@angular/core';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router } from '@angular/router';
import { AxiosError } from 'axios';
import { MetaData, NgEventBus } from 'ng-event-bus';
import { Events } from '../../shared/duty-manager-events';
import { Participant } from '../../shared/participant';
import { AxiosService } from './axios.service';

@Injectable({
  providedIn: 'root',
})
export class AuthenticationService {
  constructor(
    private eventBus: NgEventBus,
    private axios: AxiosService,
    private snackBar: MatSnackBar,
    private router: Router
  ) {
    eventBus
      .on(Events.LOGIN)
      .subscribe((credentials: any) => this.login(credentials.data));
    eventBus.on(Events.LOGOUT).subscribe(() => this.logout());
    eventBus
      .on(Events.REGISTER)
      .subscribe((register: MetaData<any>) => this.register(register.data));
  }

  checkIfAlreadyLoggedIn(): void {
    if (this.axios.getParticipant() !== null) {
      this.eventBus.cast(Events.LOGGED_IN, this.axios.getParticipant());
    }
  }

  getParticipant() {
    return this.axios.getParticipant();
  }

  async login(credentials: {
    login: string;
    password: string;
  }): Promise<Participant> {
    this.axios.setParticipant(null);
    this.eventBus.cast(Events.LOGGED_OUT);
    try {
      const response = await this.axios.request('post', 'auth/jwt', {
        email: credentials.login,
        password: credentials.password,
      });
      const participant = response.data as Participant;
      this.axios.setParticipant(participant);
      this.eventBus.cast(Events.LOGGED_IN, participant);
```

```
      return participant;
    } catch (error) {
      this.axios.setParticipant(null);
      this.eventBus.cast(Events.LOGGED_OUT);
      return Promise.reject(error);
    }
  }

  logout() {
    this.axios.setParticipant(null);
    this.eventBus.cast(Events.LOGGED_OUT);
    this.snackBar.open($localize`You were logged out.`, undefined, { duration: 2000 });
    this.router.navigate(['']);
  }

  register(registerData: { email: string; password: string }) {
    this.axios
      .request('post', 'participants', registerData)
      .then(() =>
        this.eventBus.cast(Events.LOGIN, {
        login: registerData.email,
          password: registerData.password,
        })
      )
      .catch((error: AxiosError<any, any>) =>
        this.snackBar.open(error.response?.data?.message, undefined, {
        duration: 5000,
        })
      );
  }
}


import { Injectable } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router } from '@angular/router';
import axios, { AxiosError, AxiosHeaders, AxiosResponse } from 'axios';
import { NgxSpinnerService } from 'ngx-spinner';
import { environment } from '../../environments/environment';
import { Participant } from '../../shared/participant';
import { AuthenticationComponent } from '../authentication/authentication.component';

@Injectable({
  providedIn: 'root',
})
export class AxiosService {
  private loginFormOpened = false;
  private participant?: Participant;

  constructor(
    private dialog: MatDialog,
    router: Router,
    snackBar: MatSnackBar,
    private loadIndicator: NgxSpinnerService
```

```
) {
  axios.defaults.baseURL = environment.apiUrl + '/api/v1';
  axios.defaults.headers.post['Content-Type'] = 'application/json';
  axios.interceptors.response.use(null, (error: AxiosError) => {
  const notAuthenticated = error.response?.status === 401;
  const loginIsCurrentPage = router.url === '/login';
    if (notAuthenticated && !loginIsCurrentPage && !this.loginFormOpened) {
      const dialogRef = this.dialog.open(AuthenticationComponent);
      this.loginFormOpened = true;
      dialogRef.afterClosed().subscribe(() => {
        this.loginFormOpened = false;
      });
    }
    if (notAuthenticated && loginIsCurrentPage) {
      snackBar.open($localize`You are not logged in.`, $localize`close`, {
      duration: 2000,
      });
    }
    if (error.config!.url === 'auth/jwt') {
      snackBar.open(
        $localize`Login or password are incorrect!`,
        $localize`close`,
        {
          duration: 2000,
        }
      );
    }
    return Promise.reject(error);
  });
}

getParticipant(): Participant | null {
  if (this.participant) {
    return this.participant;
  } else {
    const stringParticipant = window.localStorage.getItem('participant');
    if (stringParticipant === null) {
      return null;
    }
    const participant = JSON.parse(stringParticipant);
    this.participant = participant as Participant;
    return participant;
  }
}

setParticipant(participant: Participant | null): void {
  if (participant !== null) {
    window.localStorage.setItem('participant', JSON.stringify(participant));
    this.participant = participant;
  } else {
    window.localStorage.removeItem('participant');
    this.participant = undefined;
  }
}
```

```
// TODO commit to git, check all functionality on staging server, deploy new version

  request<T>(
    method: string,
    url: string,
    data?: any,
    parameters?: any
  ): Promise<AxiosResponse<any, T>> {
    let timeout: any | undefined;
    let timeoutWasSet = false;
    let mainPageTimeout: any | undefined;
    let mainPageTimeoutWasSet = false;
    if (method === 'get') {
      timeout = setTimeout(() => this.loadIndicator.show(), 500);
      timeoutWasSet = true;
    }
    if((method === 'post' && url.includes('auth'))) {
      mainPageTimeout = setTimeout(() => this.loadIndicator.show('main-page-load-spinner'), 500);
      mainPageTimeoutWasSet = true;
    }
    let headers: AxiosHeaders = new AxiosHeaders();
    headers.set('Time-Zone', Intl.DateTimeFormat().resolvedOptions().timeZone);

    const participant = this.getParticipant();

    if (participant !== null) {
      headers.set('Authorization', 'Bearer ' + participant.jwt);
    }
    const response = axios({
      method: method,
      url: url,
      data: data,
      headers: headers,
      params: parameters,
    });
    response.finally(() => {
      if (timeout !== undefined && timeoutWasSet) {
        clearTimeout(timeout);
      }
      this.loadIndicator.hide();
      if (mainPageTimeout !== undefined && mainPageTimeoutWasSet) {
        clearTimeout(mainPageTimeout);
      }
      this.loadIndicator.hide('main-page-load-spinner');
    });
    return response;
  }
}


import { AxiosError } from 'axios';
import { AxiosService } from './axios.service';
import { MatSnackBar } from '@angular/material/snack-bar';

export class ErrorAlertingService {
```

```
  constructor(protected axios: AxiosService, protected snackBar: MatSnackBar) {}

  async alertingRequest(
    method: string,
    url: string,
    data?: any,
    parameters?: any
  ): Promise<any> {
    try {
      const response = await this.axios.request(method, url, data, parameters);
      return response.data;
    } catch (error: any) {
      this.alertUser(error);
      return Promise.reject(error);
    }
  }

  private alertUser(error: AxiosError<any, any>) {
    this.snackBar.open(
    error.response?.data.message ?? error.message,
      $localize`close`,
      {
        duration: 10000,
      }
    );
  }
}


import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';

export class DeferredPromise<T> {
  private promise: Promise<T>;
  resolve: (value: T | PromiseLike<T>) => void = () => {};
  reject: (error?: any) => void = () => {};

  constructor() {
    this.promise = new Promise<T>((resolve, reject) => {
      this.resolve = resolve;
      this.reject = reject;
    });
  }

  async then(fulfilled: (val: T) => void) {
    return this.promise.then(fulfilled);
  }

  async catch(rejected: (error: any) => void) {
    return this.promise.catch(rejected);
  }

  async finally(settled: () => void) {
    return this.promise.finally(settled);
  }
```

```
}

type RecordExecutionFactRequest = {
  fact: {
    templateId:  string;
    description: string;
    instant: boolean;
  };
  usedTemplateIdPromise?: DeferredPromise<string | null>;
  factRecordedPromise?: DeferredPromise<string | null>;
};

@Injectable({
  providedIn: 'root',
})
export class ExecutionFactActionsShareService {
  private onTestimonySubject = new Subject<string>();
  private onFactFinishSubject = new Subject<string>();
  private onCreateSubject = new Subject<RecordExecutionFactRequest>();
  private onFactDeleteSubject = new Subject<string>();

  onTestimony(callback: (id: string) => void) {
    return this.onTestimonySubject.subscribe(callback);
  }

  onFactFinish(callback: (id: string) => void) {
    return this.onFactFinishSubject.subscribe(callback);
  }

  onCreate(callback: (value: RecordExecutionFactRequest) => void) {
    return this.onCreateSubject.subscribe(callback);
  }

  onFactDelete(callback: (id: string) => void) {
    return this.onFactDeleteSubject.subscribe(callback);
  }

  nextTestimony(id: string) {
    this.onTestimonySubject.next(id);
  }

  nextFactFinish(id: string) {
    this.onFactFinishSubject.next(id);
  }

  nextCreate(request: RecordExecutionFactRequest) {
    this.onCreateSubject.next(request);
  }

  nextFactDelete(id: string) {
    this.onFactDeleteSubject.next(id);
  }
}
```

```
import { DatePipe } from '@angular/common';
import { Injectable } from '@angular/core';
import { MatSnackBar } from '@angular/material/snack-bar';
import { ExecutionFact } from '../../shared/execution-fact';
import { RecordExecutionFact } from '../../shared/execution-facts-types';
import { AxiosService } from './axios.service';
import { ErrorAlertingService } from './error-alerting-service';

@Injectable({
  providedIn: 'root',
})
export class ExecutionFactService extends ErrorAlertingService {
  constructor(
    axios: AxiosService,
    snackBar: MatSnackBar,
    private datePipe: DatePipe
  ) {
    super(axios, snackBar);
  }

  async fetchExecutionFact(id: string): Promise<ExecutionFact> {
    return super.alertingRequest('get', `execution-facts/${id}`);
  }

  async fetchExecutionFacts(
    from: Date = new Date('1/1/2024'),
    to?: Date,
    executorId: string = this.axios.getParticipant()?.id ?? '',
    pageSize?: number
  ): Promise<ExecutionFact[]> {
    return super.alertingRequest('get', 'execution-facts', null, {
      from: this.datePipe.transform(from, 'yyyy-MM-ddTHH:mm:ss'),
      executorId: executorId,
      to: this.datePipe.transform(to, 'yyyy-MM-ddTHH:mm:ss'),
      pageSize: pageSize,
    });
  }

  async fetchActiveExecutionFacts(
    from: Date = new Date('1/1/2024'),
    to?: Date,
    executorId: string = this.axios.getParticipant()?.id ?? '',
    pageSize?: number
  ): Promise<ExecutionFact[]> {
    return super.alertingRequest('get', 'execution-facts/active', null, {
      from: this.datePipe.transform(from, 'yyyy-MM-ddTHH:mm:ss'),
      executorId: executorId,
      to: this.datePipe.transform(to, 'yyyy-MM-ddTHH:mm:ss'),
      pageSize: pageSize,
    });
  }

  async finishExecutionFact(id: string): Promise<void> {
    return super.alertingRequest('post', 'execution-facts/finished', id);
  }
```

```
  async registerExecutionFact(
    recordExecutionFact: RecordExecutionFact
  ): Promise<string> {
    const response = await super.alertingRequest('post', 'execution-facts', {
      ...recordExecutionFact,
      executorId: this.axios.getParticipant()?.id,
    });
    if (recordExecutionFact.instant === true) {
      await this.finishExecutionFact(response);
    }
    return response;
  }

  async testifyExecutionFact(factId: string): Promise<string> {
    return super.alertingRequest('post', `execution-facts/${factId}/testimonies`);
  }

  async deleteExecutionFact(factId: string): Promise<void> {
    return super.alertingRequest('delete', `execution-facts/${factId}`);
  }
}


import { Injectable, OnDestroy } from '@angular/core';
import { NgEventBus } from 'ng-event-bus';
import { Subject, Subscription } from 'rxjs';
import { validate as idUuidValid } from 'uuid';
import { Events } from '../../shared/duty-manager-events';
import { Participant } from '../../shared/participant';
import { AuthenticationService } from './authentication.service';

type TemplateIdsToNumberOfItsUsages = { id: string; numberOfUsages: number };

@Injectable({
  providedIn: 'root',
})
export class MostUsedTemplatesService implements OnDestroy {
  private subscriptions: Subscription[] = [];
  private templateIdsToNumberOfItsUsages: TemplateIdsToNumberOfItsUsages[] = [];
  private newTemplateIdAdded = new Subject<string>();

  constructor(
    private authService: AuthenticationService,
    eventBus: NgEventBus
  ) {
    this.loadCurrentUserPreferences();
    this.subscriptions.push(
      eventBus
        .on(Events.LOGGED_IN)
        .subscribe(() => this.loadCurrentUserPreferences())
    );
  }

  ngOnDestroy(): void {
```

```
  this.subscriptions.forEach((s) => s.unsubscribe());
}

loadCurrentUserPreferences() {
  const currentUser = this.authService.getParticipant();
  if (currentUser) {
    this.templateIdsToNumberOfItsUsages = JSON.parse(
      window.localStorage.getItem(this.getStorageKeyForUser(currentUser)) ??
      '[]'
    ) as TemplateIdsToNumberOfItsUsages[];
  }
}

templateUsed(templateId: string) {
  const existingTemplateIndex = this.templateIdsToNumberOfItsUsages.findIndex(
    (t) => t.id === templateId
  );
  if (existingTemplateIndex !== -1) {
    const existingTemplate = this.templateIdsToNumberOfItsUsages.at(
      existingTemplateIndex
    )!!;
    existingTemplate.numberOfUsages += 1;
    this.templateIdsToNumberOfItsUsages.splice(
    existingTemplateIndex,
      1,
      existingTemplate
    );
  } else {
    this.templateIdsToNumberOfItsUsages.push({
    id: templateId,
      numberOfUsages: 1,
    });
    this.newTemplateIdAdded.next(templateId);
  }
}

removeGlitched(templateId: string) {
  this.templateIdsToNumberOfItsUsages.splice(
  this.templateIdsToNumberOfItsUsages.findIndex((t) => t.id === templateId),
  1
  );
}

write10MostUsedTemplateIdsToLocalStorage() {
  const currentUser = this.authService.getParticipant();
  if (currentUser) {
    window.localStorage.setItem(
      this.getStorageKeyForUser(currentUser),
      JSON.stringify(
      this.templateIdsToNumberOfItsUsages
          .filter((e) => idUuidValid(e.id))
          .sort((a, b) => b.numberOfUsages - a.numberOfUsages)
          .slice(0, 10)
      )
    );
```

```
    }
  }

  getMostUsedTemplateIds() {
    return this.templateIdsToNumberOfItsUsages.map((t) => t.id);
  }

  onNewTemplateId(callback: (id: string) => void) {
    return this.newTemplateIdAdded.subscribe(callback);
  }

  private getStorageKeyForUser(participant: Participant) {
    return participant.id + '_most_used_template_ids';
  }
}
```

```
import { Injectable } from '@angular/core';
import { MatSnackBar } from '@angular/material/snack-bar';
import { AxiosError } from 'axios';
import { Template } from '../../shared/template';
import { AxiosService } from './axios.service';
import { CreateTemplate, UpdateTemplate } from '../../shared/templates-types';
import { ErrorAlertingService } from './error-alerting-service';

@Injectable({
  providedIn: 'root',
})
export class TemplateService extends ErrorAlertingService {

  constructor(axios: AxiosService, snackBar: MatSnackBar) {
    super(axios, snackBar);
  }

  async fetchTemplates(
    page: number = 0,
    pageSize: number = 50
  ): Promise<Template[]> {
    return super.alertingRequest('get', 'templates', undefined, {
      page: page,
      pageSize: pageSize,
    });
  }

  async fetchTemplate(id: String): Promise<Template> {
    return super.alertingRequest('get', `templates/${id}`);
  }

  async getNumberOfTemplates(): Promise<number> {
    return super.alertingRequest('get', 'templates/quantity');
  }

  async createTemplate(newTemplate: CreateTemplate): Promise<string> {
    return super.alertingRequest('post', 'templates', newTemplate);
  }
```

```
  async updateTemplate(updatedTemplate: UpdateTemplate) {
    if(updatedTemplate.name === '') {
      updatedTemplate.name = undefined;
    }
    if(updatedTemplate.description === '') {
      updatedTemplate.description = undefined;
    }
    return super.alertingRequest('put', `templates/${updatedTemplate.id}`, updatedTemplate);
  }

  async deleteTemplate(id: String): Promise<void> {
    return super.alertingRequest('delete', `templates/${id}`);
  }

  async fuzzySearchByName(name: string): Promise<Template[]> {
    return super.alertingRequest('get', 'templates', undefined, {
    fuzzyName: name
    })
  }
}


import { CommonModule } from '@angular/common';
import { Component, Optional } from '@angular/core';
import {
  FormControl,
  FormGroup,
  ReactiveFormsModule,
  Validators,
} from '@angular/forms';
import { MatButtonModule } from '@angular/material/button';
import { MatDialogRef } from '@angular/material/dialog';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
import { MatInputModule } from '@angular/material/input';
import { NgEventBus } from 'ng-event-bus';
import { validatePassword } from './password-validator';
import { getErrorMessage } from '../../shared/validation-errors-getter';
import { Events } from '../../shared/duty-manager-events';

@Component({
  selector: 'registration-form',
  standalone: true,
  imports: [
    MatInputModule,
    MatFormFieldModule,
    ReactiveFormsModule,
    CommonModule,
    MatIconModule,
    MatButtonModule,
  ],
  templateUrl: './registration-form.component.html',
  styleUrl: './registration-form.component.scss',
})
```

```
export class RegistrationFormComponent {
  hide = true;
  constructor(
    private eventBus: NgEventBus,
    @Optional() private dialogRef?: MatDialogRef<any>
  ) {}

  registerForm = new FormGroup({
    senderFullName: new FormControl('', [Validators.required]),
    senderEmail: new FormControl('', [Validators.required, Validators.email]),
    senderPassword: new FormControl('', [
      Validators.required,
      validatePassword,
      Validators.minLength(8)
    ]),
  });

  register() {
    if (this.registerForm.valid) {
      this.dialogRef?.close();
      this.eventBus.cast(Events.REGISTER, {
        fullName: this.registerForm.get('senderFullName')?.value,
        email: this.registerForm.get('senderEmail')?.value,
        password: this.registerForm.get('senderPassword')?.value,
      });
    }
  }

  getErrorMessage(formControlName: string) {
    return getErrorMessage(formControlName, this.registerForm);
  }
}


import { AsyncPipe, CommonModule } from '@angular/common';
import {
  Component,
  ElementRef,
  HostListener,
  OnDestroy,
  OnInit,
} from '@angular/core';
import { FormControl, ReactiveFormsModule, Validators } from '@angular/forms';
import { MatAutocompleteModule } from '@angular/material/autocomplete';
import { MatButtonModule } from '@angular/material/button';
import { MatCheckboxModule } from '@angular/material/checkbox';
import { MatChipsModule } from '@angular/material/chips';
import { MatDividerModule } from '@angular/material/divider';
import { MatExpansionModule } from '@angular/material/expansion';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
import { MatInputModule } from '@angular/material/input';
import { MatTabsModule } from '@angular/material/tabs';
import { RouterLink } from '@angular/router';
import { NgEventBus } from 'ng-event-bus';
```

```
import { SlickCarouselModule } from 'ngx-slick-carousel';
import { AutoClosableComponent } from '../../shared/AutoClosableComponent';
import { Events } from '../../shared/duty-manager-events';
import { Template } from '../../shared/template';
import {
  DeferredPromise,
  ExecutionFactActionsShareService,
} from '../services/execution-fact-actions-share.service';
import { MostUsedTemplatesService } from '../services/most-used-templates.service';
import { TemplateService } from '../services/template.service';

type SlickChangeEvent = {
  event: any;
  slick: any;
  currentSlide: number;
  first: boolean;
  last: boolean;
};

@Component({
  selector: 'record-execution-fact',
  standalone: true,
  imports: [
    MatButtonModule,
    MatChipsModule,
    MatInputModule,
    MatFormFieldModule,
    ReactiveFormsModule,
    MatIconModule,
    CommonModule,
    MatCheckboxModule,
    SlickCarouselModule,
    MatDividerModule,
    MatExpansionModule,
    MatTabsModule,
    MatAutocompleteModule,
    AsyncPipe,
    RouterLink,
  ],
  templateUrl: './record-execution-fact.component.html',
  styleUrl: './record-execution-fact.component.scss',
})
export class RecordExecutionFactComponent
  extends AutoClosableComponent
  implements OnDestroy, OnInit
{
  senderDescription = new FormControl('', [Validators.required]);
  senderInstant = new FormControl('');
  senderFuzzyName = new FormControl('');
  slideConfig?: any;
  mostUsedTemplates: Template[] = [];
  fuzzySearchResult: Template[] = [];
  recordButtonDisabled = false;

  private selectedTemplate?: Template;
```

```
private currentTemplatePage = { pageIndex: 0, pageSize: 50 };

constructor(
  private templatesService: TemplateService,
  private factsActionsService: ExecutionFactActionsShareService,
  eventBus: NgEventBus,
  private elementRef: ElementRef,
  private templateCache: MostUsedTemplatesService
) {
  super();
  super.manage(eventBus.on(Events.LOGGED_IN).subscribe(() => this.setData()));
  super.manage(
    templateCache.onNewTemplateId((id) =>
      templatesService
        .fetchTemplate(id)
        .then((t) => this.mostUsedTemplates.push(t))
    )
  );
}

ngOnInit(): void {
  this.setData();
}

private setData() {
  this.setMostUsedTemplates();
}

override ngOnDestroy(): void {
  super.ngOnDestroy();
  this.writeMostUsedTemplatesToCache();
}

@HostListener('window:beforeunload')
private writeMostUsedTemplatesToCache() {
  this.templateCache.write10MostUsedTemplateIdsToLocalStorage();
}

initCarouselArrows() {
  if (this.slideConfig === undefined) {
    const nextArrow = (
      this.elementRef.nativeElement as HTMLElement
    ).querySelector('#next-arrow');
    const prevArrow = (
      this.elementRef.nativeElement as HTMLElement
    ).querySelector('#prev-arrow');
    this.slideConfig = {
    slidesToScroll: 1,
      infinite: false,
      variableWidth: true,
      nextArrow: nextArrow,
      prevArrow: prevArrow,
      swipe: false,
    };
  }
```

```
}

private async loadTemplates() {
  return this.templatesService.fetchTemplates(
    this.currentTemplatePage.pageIndex,
    this.currentTemplatePage.pageSize
  );
}

private setMostUsedTemplates() {
  this.mostUsedTemplates = [];
  this.templateCache.getMostUsedTemplateIds().forEach((id) =>
    this.templatesService
      .fetchTemplate(id)
      .then((t) => this.mostUsedTemplates.push(t))
      .catch(() => this.templateCache.removeGlitched(id))
  );
}

getErrorMessage() {
  return $localize`Either select parent or describe what you did.`;
}

recordExecutionFact() {
  let templateId = '';
  if (this.selectedTemplate) {
    templateId = this.selectedTemplate.id;
  }
  if (this.senderDescription.value && this.senderDescription.valid) {
    const usedTemplateIdPromise = new DeferredPromise<string | null>();
    usedTemplateIdPromise.then((id) => {
      if (id !== null) {
        this.templatesService
          .fetchTemplate(id)
          .then((t) => this.templateCache.templateUsed(t.id));
      }
    });
    const factRecordedPromise = new DeferredPromise<string | null>();
    factRecordedPromise.finally(() => (this.recordButtonDisabled = false));
    this.recordButtonDisabled = true;
    this.factsActionsService.nextCreate({
      fact: {
        templateId: templateId,
        description: this.senderDescription.value,
        instant: Boolean(this.senderInstant.value),
      },
      usedTemplateIdPromise: usedTemplateIdPromise,
      factRecordedPromise: factRecordedPromise,
    });
  }
}

selectionChanged(templateName: string) {
  const pastTemplate = this.selectedTemplate;
  const nameFilter = (template: Template): boolean =>
```

```
      template.name === templateName;
    this.selectedTemplate =
    this.fuzzySearchResult.find(nameFilter) ??
    this.mostUsedTemplates.find(nameFilter);
    const senderDescription = (this.senderDescription.value ?? '').trim();
    if (
      pastTemplate !== undefined &&
      this.selectedTemplate !== undefined &&
      templateName === pastTemplate.name
    ) {
      if (senderDescription === this.selectedTemplate.description) {
        this.senderDescription.setValue('');
      }
      this.selectedTemplate = undefined;
    } else if (
      senderDescription === '' ||
      this.mostUsedTemplates.find(
        (template) => template.description === senderDescription
      ) !== undefined ||
      this.fuzzySearchResult.find(
        (template) => template.description === senderDescription
      ) !== undefined
    ) {
      this.senderDescription.setValue(this.selectedTemplate?.description  ??  '');
    }
  }

  async setTemplatesByFuzzyName() {
    const templateName = this.senderFuzzyName.value;
    if (
      this.senderFuzzyName.dirty &&
      templateName &&
      templateName.trim() !== ''
    ) {
      this.fuzzySearchResult = await this.templatesService.fuzzySearchByName(
        templateName
      );
    }
  }
}


import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
import { Participant } from '../../shared/participant';
import { MatCardModule } from '@angular/material/card';
import { MatButtonModule } from '@angular/material/button';
import { ExecutionFactService } from '../services/execution-fact.service';
import { ExecutionFact } from '../../shared/execution-fact';
import { MatChipsModule } from '@angular/material/chips';
import { MatIconModule } from '@angular/material/icon';
import { AuthenticationService } from '../services/authentication.service';
import { subDays } from 'date-fns';
import { NgScrollbarModule } from 'ngx-scrollbar';
import { DatePipe } from '@angular/common';
```

```
//This component has limitation, it is that if there was more than 200 facts in n day range,
// the list of facts will not correctly show facts that need to be testified, where n is number of most recent days to be shown.
// This needs to be fixed on server side.

@Component({
  selector: 'participant-item',
  standalone: true,
  imports: [MatCardModule, MatButtonModule, MatChipsModule, MatIconModule, NgScrollbarModule, DatePipe],
  templateUrl: './participant-item.component.html',
  styleUrl:  './participant-item.component.scss',
})
export class ParticipantItemComponent implements OnInit {
  private static readonly MAX_EXECUTION_FACTS = 3;

  @Input() participant: Participant = new Participant();
  @Output() navigateToFactsPage = new EventEmitter<string>();

  executionFacts: ExecutionFact[] = [];
  userAllowedToFinishFact = false;

  constructor(
    private factService: ExecutionFactService,
    private authService: AuthenticationService
  ) {}
  ngOnInit(): void {
    this.initializeFacts();
  }

  private initializeFacts() {
    this.userAllowedToFinishFact =
      this.participant.id === this.authService.getParticipant()?.id;
    if (this.userAllowedToFinishFact) {
      this.loadActiveFacts();
    } else {
      this.loadNotTestified();
    }
  }

  private async loadActiveFacts() {
    this.executionFacts = await this.factService.fetchActiveExecutionFacts(
      new Date('1/1/2024'),
      undefined,
      this.participant.id,
      ParticipantItemComponent.MAX_EXECUTION_FACTS
    );
  }

  private async loadNotTestified() {
    const loaded = await this.factService.fetchExecutionFacts(
      subDays(new Date(), 3),
      undefined,
      this.participant.id
    );
    this.executionFacts = loaded.filter(
```

```
      (fact) =>
        fact.testimonies.find(
        (testimony) =>
            testimony.witnessId === this.authService.getParticipant()?.id
        ) === undefined
    ).slice(0, ParticipantItemComponent.MAX_EXECUTION_FACTS);
  }

  emitParticipantId() {
    this.navigateToFactsPage.emit(this.participant.id);
  }

  finishExecutionFact(fact: ExecutionFact) {
    this.factService
      .finishExecutionFact(fact.id)
      .then(() => this.initializeFacts());
  }

  testifyExecutionFact(fact: ExecutionFact) {
    this.factService
      .testifyExecutionFact(fact.id)
      .then(() => this.initializeFacts());
  }
}


import { CommonModule } from '@angular/common';
import { Component, EventEmitter, Output } from '@angular/core';
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MetaData, NgEventBus } from 'ng-event-bus';
import { Participant } from '../../shared/participant';
import { Events } from '../../shared/duty-manager-events';
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-header',
  standalone: true,
  imports: [MatToolbarModule, MatButtonModule, MatIconModule, CommonModule, RouterModule],
  templateUrl: './header.component.html',
  styleUrl: './header.component.scss',
})
export class HeaderComponent {
  private _participant?: Participant;
  @Output() sideNawStateChanged = new EventEmitter<void>()
  constructor(private eventBus: NgEventBus) {
    eventBus
      .on(Events.LOGGED_IN)
      .subscribe((participantData: MetaData<any>) =>
        this.setParticipant(participantData.data as Participant)
      );
    eventBus
      .on(Events.LOGGED_OUT)
      .subscribe(() => (this._participant = undefined));
```

```
  }

  private setParticipant(participant: Participant) {
    if (!this._participant) {
      this._participant = participant;
    } else if (!this._participant.equals(participant)) {
      this._participant = participant;
    }
  }

  logout() {
    this.eventBus.cast(Events.LOGOUT);
  }

  get participant() {
    return this._participant;
  }

  toggleMenu() {
    this.sideNawStateChanged.emit();
  }
}


import { CommonModule, DatePipe } from '@angular/common';
import {
  Component,
  EventEmitter,
  Injectable,
  Input,
  OnDestroy,
  OnInit,
  Output,
} from '@angular/core';
import { FormControl, ReactiveFormsModule, Validators } from '@angular/forms';
import { MatButtonModule } from '@angular/material/button';
import {
  MatButtonToggleChange,
  MatButtonToggleModule,
} from '@angular/material/button-toggle';
import { MatDatepickerModule } from '@angular/material/datepicker';
import { MatDividerModule } from '@angular/material/divider';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
import { MatInputModule } from '@angular/material/input';
import { MatMenuModule } from '@angular/material/menu';
import {
  MatPaginatorIntl,
  MatPaginatorModule,
  PageEvent,
} from '@angular/material/paginator';
import {
  add,
  differenceInDays,
  isSameDay,
```

```
  subDays,
  subMilliseconds,
} from 'date-fns';
import { Subject, Subscription } from 'rxjs';
import { AutoClosableComponent } from '../../shared/AutoClosableComponent';
import {
  ExecutionFactFilter,
  ExecutionFactLoadParameters,
} from '../../shared/execution-facts-types';
import { ExecutionFactsLoadParametersShareService } from '../services/execution-facts-load-settings-share.service';

@Injectable()
export class ExecutionFactPaginator implements MatPaginatorIntl, OnDestroy {
  private subscriptions: Subscription[] = [];
  activeLoadSettings?: ExecutionFactLoadParameters;
  changes: Subject<void> = new Subject<void>();
  itemsPerPageLabel = $localize`Days per page`;
  nextPageLabel = $localize`Next period`;
  previousPageLabel = $localize`Previous period`;
  firstPageLabel = '';
  lastPageLabel = '';
  constructor(
    private datePipe: DatePipe,
    factsLoadSettingsService: ExecutionFactsLoadParametersShareService
  ) {
    this.subscriptions.push(
      factsLoadSettingsService.onLoadParameters((settings) => {
      this.activeLoadSettings = settings;
        this.changes.next();
      })
    );
  }
  ngOnDestroy(): void {
    this.subscriptions.forEach((sub) => sub.unsubscribe());
  }
  getRangeLabel(page: number, pageSize: number, length: number) {
    return `${this.datePipe.transform(
      this.activeLoadSettings?.from,
      'MMMM d'
    )} - ${this.datePipe.transform(this.activeLoadSettings?.to, 'MMMM d')}`;
  }
}

@Component({
  selector: 'execution-facts-filter',
  standalone: true,
  imports: [
    MatFormFieldModule,
    MatInputModule,
    MatDatepickerModule,
    MatPaginatorModule,
    MatButtonToggleModule,
    MatButtonModule,
    MatMenuModule,
    MatIconModule,
```

```
    ReactiveFormsModule,
    CommonModule,
    MatDividerModule,
  ],
  providers: [{ provide: MatPaginatorIntl, useClass: ExecutionFactPaginator }],
  templateUrl: './execution-facts-filter.component.html',
  styleUrl:  './execution-facts-filter.component.scss',
})
export class ExecutionFactsFilterComponent
  extends AutoClosableComponent
  implements OnDestroy, OnInit
{
  private static readonly DEFAULT_PAGE_SIZE: number = 7;
  private static readonly DEFAULT_FROM_DATE =
  ExecutionFactsFilterComponent.initDefaultFromDate();
  private static readonly DEFAULT_TO_DATE = new Date();
  private static readonly DEFAULT_LOAD_PARAMETERS = {
    from: ExecutionFactsFilterComponent.DEFAULT_FROM_DATE,
    to: ExecutionFactsFilterComponent.DEFAULT_TO_DATE,
  };

  private static initDefaultFromDate(): Date {
    return subDays(new Date(), ExecutionFactsFilterComponent.DEFAULT_PAGE_SIZE);
  }

  senderTemplateName = new FormControl<string>('');
  senderFromDate = new FormControl<Date>(
  ExecutionFactsFilterComponent.DEFAULT_FROM_DATE,
  [Validators.required]
  );
  senderToDate = new FormControl<Date>(
    ExecutionFactsFilterComponent.DEFAULT_TO_DATE
  );
  stateFilter = { active: true, finished: true };
  pageOptions = [1, 3, 7, 31];
  private _loadSettingsChanged = false;
  activeFilters: ExecutionFactFilter[] = [];
  activeStateFilter: ExecutionFactFilter = () => true;

  @Input() pageSize = ExecutionFactsFilterComponent.DEFAULT_PAGE_SIZE;
  @Input() defaultLoadParameters: ExecutionFactLoadParameters =
  ExecutionFactsFilterComponent.DEFAULT_LOAD_PARAMETERS;
  @Output() private filterChanged = new EventEmitter<ExecutionFactFilter[]>();
  @Output() private stateFilterChanged =
    new EventEmitter<ExecutionFactFilter>();

  _activeLoadSettings: ExecutionFactLoadParameters =
    ExecutionFactsFilterComponent.DEFAULT_LOAD_PARAMETERS;

  constructor(
    private factsLoadSettingsService: ExecutionFactsLoadParametersShareService
  ) {
    super();
    super.manage(
      this.factsLoadSettingsService.onLoadParameters((settings) => {
```

```
      this.senderFromDate.setValue(settings.from);
      this.senderToDate.setValue(settings.to);
    })
  );
}

ngOnInit(): void {
  this._activeLoadSettings = this.defaultLoadParameters;
}
updateFilters() {
  this.activeFilters = new Array<ExecutionFactFilter>();
  const templateName = this.senderTemplateName.getRawValue();
  if (templateName) {
    this.activeFilters.push((fact) => {
      return fact.templateName == templateName;
    });
  }
}

emitFilters() {
  this.filterChanged.emit(this.activeFilters);
}

clearFilters() {
  this.activeFilters = [];
}
updateLoadSetting() {
  this.activeLoadSettings = {
  from:
      this.senderFromDate.getRawValue() ?? this.defaultLoadParameters.from,
    to: this.senderToDate.getRawValue() ?? this.defaultLoadParameters.to,
  };
}

emitLoadSettings() {
  this.factsLoadSettingsService.nextLoadParameters(this.activeLoadSettings);
}
clearLoadSettings() {
  this.activeLoadSettings = this.defaultLoadParameters;
}

changeSelectedLoadState(event: MatButtonToggleChange) {
  let activeOption = event.value as string[];

  if (activeOption.includes('active') && activeOption.includes('finished')) {
    this.activeStateFilter = () => true;
    this.stateFilter.active = true;
    this.stateFilter.finished = true;
  } else if (activeOption.includes('finished')) {
    this.activeStateFilter = (fact) => fact.finishTime !== null;
    this.stateFilter.active = false;
    this.stateFilter.finished = true;
  } else if (activeOption.includes('active')) {
    this.activeStateFilter = (fact) => fact.finishTime === null;
    this.stateFilter.active = true;
```

```
    this.stateFilter.finished = false;
  } else {
    this.activeStateFilter = () => false;
    this.stateFilter.active = false;
    this.stateFilter.finished = false;
  }

  this.stateFilterChanged.emit(this.activeStateFilter);
}

changePage(page: PageEvent) {
  this.pageSize = page.pageSize;
  const pageOffset = this.getDaysOffset(page);
  let fromDate = this.activeLoadSettings.from;
  if (
    !(
      page.previousPageIndex &&
      Math.abs(page.pageIndex - page.previousPageIndex) !== 1
    )
  ) {
    fromDate = add(fromDate, { days: pageOffset });
  }
  const toDate = subMilliseconds(
    add(fromDate, { days: this.pageSize + 1 }),
    500
  );
  this.activeLoadSettings = {
    from: fromDate,
    to: toDate,
  };
  this.factsLoadSettingsService.nextLoadParameters(this.activeLoadSettings);
}

getDaysOffset(page: PageEvent): number {
  if (!page.previousPageIndex || page.pageIndex > page.previousPageIndex) {
    return page.pageSize;
  } else {
    return 0 - page.pageSize;
  }
}

get maxLength() {
  return Number.MAX_SAFE_INTEGER;
}

get loadSettingsChanged() {
  return this._loadSettingsChanged;
}

get activeLoadSettings() {
  return this._activeLoadSettings;
}

set activeLoadSettings(settings: ExecutionFactLoadParameters) {
  this._activeLoadSettings = settings;
```

```
    if (
      isSameDay(settings.from, this.defaultLoadParameters.from) &&
      isSameDay(settings.to, this.defaultLoadParameters.to)
    ) {
      this._loadSettingsChanged = false;
    } else {
      this._loadSettingsChanged = true;
    }
  }

  calculateCurrentIndex(): number {
    const daysOffset = differenceInDays(
    new Date(),
    this.activeLoadSettings.from
    );
    const offsetRelatedToMaxValueAllowed =
      Number.MAX_SAFE_INTEGER - 1 * this.pageSize - daysOffset; // (1 * this.pageSize) is here to allow user make
at least one request with 'from' date being in the future
    const paginatedOffset = offsetRelatedToMaxValueAllowed / this.pageSize;
    return paginatedOffset;
  }
}
import { AsyncPipe } from '@angular/common';
import { Component, OnDestroy, OnInit } from '@angular/core';
import { MatDividerModule } from '@angular/material/divider';
import { ActivatedRoute } from '@angular/router';
import { endOfDay, startOfDay, subDays } from 'date-fns';
import { NgEventBus } from 'ng-event-bus';
import { ProcessingComponent } from '../../shared/ProcessingComponent';
import { Events } from '../../shared/duty-manager-events';
import { ExecutionFact } from '../../shared/execution-fact';
import {
  ExecutionFactFilter,
  ExecutionFactLoadParameters,
} from '../../shared/execution-facts-types';
import { Participant } from '../../shared/participant';
import { ExecutionFactsFilterComponent } from '../execution-facts-filter/execution-facts-filter.component';
import { ExecutionFactsListComponent } from '../execution-facts-list/execution-facts-list.component';
import { RecordExecutionFactComponent } from '../record-execution-fact/record-execution-fact.component';
import { AuthenticationService } from '../services/authentication.service';
import { ExecutionFactActionsShareService } from '../services/execution-fact-actions-share.service';
import { ExecutionFactService } from '../services/execution-fact.service';
import { ExecutionFactsLoadParametersShareService } from '../services/execution-facts-load-settings-share.service';
import { MostUsedTemplatesService } from '../services/most-used-templates.service';
import { YesNoShareService } from '../services/yes-no-share.service';

@Component({
  selector: 'app-execution-facts',
  standalone: true,
  imports: [
    ExecutionFactsListComponent,
    RecordExecutionFactComponent,
    ExecutionFactsFilterComponent,
    MatDividerModule,
    AsyncPipe,
```

```
  ],
  providers: [
    ExecutionFactsLoadParametersShareService,
    ExecutionFactActionsShareService,
    MostUsedTemplatesService,
    {
      provide: 'deleteFactShare',
      useValue: new YesNoShareService<ExecutionFact>(),
    },
  ],
  templateUrl: './execution-facts.component.html',
  styleUrl: './execution-facts.component.scss',
})
export class ExecutionFactsComponent
  extends ProcessingComponent<ExecutionFact>
  implements OnInit, OnDestroy
{
  daysToBeShown = 1;

  defaultLoadParameters = this.getDefaultLoadParameters();

  private getDefaultLoadParameters(): { from: Date; to: Date } {
    const currentDate = new Date();
    return {
      from: startOfDay(subDays(currentDate, this.daysToBeShown)),
      to: endOfDay(currentDate),
    };
  }
  private viewingParticipantId?: string;
  private _loggedInParticipant?: Participant;
  private _executionFactFilters: ExecutionFactFilter[] = [];
  private _executionFactsLoadParameters: ExecutionFactLoadParameters =
    this.defaultLoadParameters;
  private _executionFactsStateFilter: ExecutionFactFilter = () => true;
  private _userAllowedToChangeFacts = true;
  private filteredExecutionFacts: ExecutionFact[] = [];

  constructor(
    private factService: ExecutionFactService,
    private factsLoadingParametersService: ExecutionFactsLoadParametersShareService,
    private authService: AuthenticationService,
    route: ActivatedRoute,
    factsActionsService: ExecutionFactActionsShareService,
    eventBus: NgEventBus
  ) {
    super(
      eventBus,
      () => this.loadExecutionFacts(),
      () => this.setFilteredExecutionFacts()
    );
    super.manage(
      eventBus
        .on(Events.LOGGED_IN)
        .subscribe(
          (participant: any) => (this._loggedInParticipant = participant)
```

```
    )
);
super.manage(
  factsActionsService.onFactDelete((id) =>
  this.factService
      .deleteExecutionFact(id)
      .then(() =>
        super.replaceDataPiece(this.factHasSameId(id), () =>
        Promise.resolve(null)
        )
      )
  )
);
super.manage(
  route.queryParamMap.subscribe((params) => {
    this.viewingParticipantId = params.get('participant-id') ?? undefined;
    if (this.viewingParticipantId) {
      this._userAllowedToChangeFacts =
        authService.getParticipant()?.id === this.viewingParticipantId;
    }
    this.loadExecutionFacts().then((facts) => super.replaceData(facts));
  })
);
super.manage(
  this.factsLoadingParametersService.onLoadParameters((parameters) => {
  if (this._executionFactsLoadParameters !== parameters) {
      this.parameters = parameters;
    }
  })
);
super.manage(
  factsActionsService.onCreate((recordFact) =>
  this.factService
      .registerExecutionFact(recordFact.fact)
      .then((id) => {
        super.addDataPiece(() =>
          this.factService.fetchExecutionFact(id).then((f) => {
            recordFact.usedTemplateIdPromise?.resolve(f.templateId);
            return f;
          })
        );
        return id;
      })
      .then(
        recordFact.factRecordedPromise?.resolve,
        recordFact.factRecordedPromise?.reject
      )
  )
);
super.manage(
  factsActionsService.onTestimony((id) => {
  this.factService
      .testifyExecutionFact(id)
      .then(() =>
        super.replaceDataPiece(this.factHasSameId(id), () =>
```

```
          this.factService.fetchExecutionFact(id)
        )
      );
    })
  );
  super.manage(
    factsActionsService.onFactFinish((id) =>
    this.factService
      .finishExecutionFact(id)
      .then(() =>
        super.replaceDataPiece(this.factHasSameId(id), () =>
        this.factService.fetchExecutionFact(id)
        )
      )
    )
  );
}
private factHasSameId(id: string) {
  return (fact: ExecutionFact) => fact.id === id;
}

private async loadExecutionFacts() {
  return this.factService.fetchExecutionFacts(
    this._executionFactsLoadParameters.from,
    this._executionFactsLoadParameters.to,
    this.viewingParticipantId
  );
}
ngOnInit(): void {
  this.factsLoadingParametersService.nextLoadParameters(
  this._executionFactsLoadParameters
  );
  this._loggedInParticipant = this.authService.getParticipant() ?? undefined;
}

get executionFacts(): ExecutionFact[] {
  return this.filteredExecutionFacts;
}

set filters(filters: ExecutionFactFilter[]) {
  this._executionFactFilters = filters;
  this.setFilteredExecutionFacts();
}

set stateFilter(filter: ExecutionFactFilter) {
  this._executionFactsStateFilter = filter;
  this.setFilteredExecutionFacts();
}

set parameters(parameters: ExecutionFactLoadParameters) {
  this._executionFactsLoadParameters = parameters;
  this.loadExecutionFacts()
    .then((facts) => {
      super.replaceData(facts);
      return facts;
```

```
      })
      .then((facts) => (this.filteredExecutionFacts = facts));
  }

  get userAllowedToChangeFacts() {
    return this._userAllowedToChangeFacts;
  }

  get loggedInParticipant() {
    return this._loggedInParticipant;
  }

  private setFilteredExecutionFacts() {
    this.filteredExecutionFacts = super.data.filter(
    (fact) =>
        this._executionFactFilters.every((filter) => filter(fact)) &&
        this._executionFactsStateFilter(fact)
    );
  }
}
import { CommonModule } from '@angular/common';
import { Component, Inject, Injector, Input, OnInit } from '@angular/core';
import { MatButtonModule } from '@angular/material/button';
import { MatCardModule } from '@angular/material/card';
import { MatChipsModule } from '@angular/material/chips';
import { MatDialog } from '@angular/material/dialog';
import { RouterModule } from '@angular/router';
import { differenceInSeconds } from 'date-fns';
import { AutoClosableComponent } from '../../shared/AutoClosableComponent';
import { ExecutionFact } from '../../shared/execution-fact';
import { Participant } from '../../shared/participant';
import { ExecutionFactActionsShareService } from '../services/execution-fact-actions-share.service';
import { YesNoShareService } from '../services/yes-no-share.service';
import { YesNoDialogComponent } from '../yes-no-dialog/yes-no-dialog.component';

@Component({
  selector: 'execution-fact-item',
  standalone: true,
  imports: [
    MatCardModule,
    MatButtonModule,
    CommonModule,
    MatChipsModule,
    RouterModule,
  ],
  templateUrl: './execution-fact-item.component.html',
  styleUrl: './execution-fact-item.component.scss',
})
export class ExecutionFactItemComponent
  extends AutoClosableComponent
  implements OnInit
{
  @Input() executionFact = new ExecutionFact();
  @Input() loggedInParticipant?: Participant;
  timeSpent = '0:0:0';
```

```
constructor(
  private factsActionsService: ExecutionFactActionsShareService,
  @Inject('deleteFactShare')
  private deleteFactShare: YesNoShareService<ExecutionFact>,
  private dialog: MatDialog
) {
  super();
  super.manage(
    deleteFactShare.onDecision((decision) => {
      if (decision.issuer.id === this.executionFact.id && decision.decision) {
        this.factsActionsService.nextFactDelete(decision.issuer.id);
      }
    })
  );
}
ngOnInit(): void {
  this.timeSpent = this.getTimeSpent();
}
finish() {
  this.factsActionsService.nextFactFinish(this.executionFact.id);
}
testify() {
  this.factsActionsService.nextTestimony(this.executionFact.id);
}
deleteFact() {
  this.dialog.open(YesNoDialogComponent<ExecutionFact>, {
  injector: Injector.create({
      providers: [
        {
          provide: YesNoShareService<ExecutionFact>,
          useValue: this.deleteFactShare,
        },
      ],
    }),
    data: {
      title:
        $localize`Do you want to delete`+
        ' "' +
        this.executionFact.description +
        '" ' +
        $localize`execution fact`,
      message: $localize`This action can not be reversed.`,
      issuer: this.executionFact,
    },
  });
}

get loggedInUserViewingHisFacts() {
  return this.executionFact.executorId === this.loggedInParticipant?.id;
}

get userIsAdmin() {
  return this.loggedInParticipant?.role.name === 'ADMIN';
}
```

```
  get userHaveNotTestified() {
    return (
      this.executionFact.testimonies.find(
        (t) => t.witnessId === this.loggedInParticipant?.id
      ) == undefined
    );
  }

  private getTimeSpent(): string {
    let secondsDifference = differenceInSeconds(
      this.executionFact.finishTime ?? '',
      this.executionFact.startTime
    );
    const hours =
      secondsDifference >= 3600 ? Math.floor(secondsDifference / 3600) : 0;
    secondsDifference = secondsDifference - hours * 3600;
    const minutes =
      secondsDifference >= 60 ? Math.floor(secondsDifference / 60) : 0;
    secondsDifference = secondsDifference - minutes * 60;
    return
`${this.getFormattedTime(hours)}:${this.getFormattedTime(minutes)}:${this.getFormattedTime(secondsDifference)
}`;
  }

  private getFormattedTime(time: number) {
    const stringifiedTime = time.toString();
    if (stringifiedTime.length > 1) {
      return time.toString();
    }
    return `0${stringifiedTime}`;
  }
}
import { Component } from '@angular/core';
import {
  FormControl,
  FormGroup,
  ReactiveFormsModule,
  Validators,
} from '@angular/forms';
import { MatButtonModule } from '@angular/material/button';

import { MatExpansionModule } from '@angular/material/expansion';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
import { MatInputModule } from '@angular/material/input';
import { getErrorMessage } from '../../shared/validation-errors-getter';
import { TemplateActionsShareService } from '../services/template-actions-share.service';

@Component({
  selector: 'create-template',
  standalone: true,
  imports: [
  MatExpansionModule,
  MatButtonModule,
  ReactiveFormsModule,
```

```
    MatInputModule,
    MatFormFieldModule,
    MatIconModule,
  ],
  templateUrl: './create-template.component.html',
  styleUrl: './create-template.component.scss',
})
export class CreateTemplateComponent {
  createTemplateForm = new FormGroup({
  senderTemplateName: new FormControl<string>('', [
      Validators.maxLength(100),
      Validators.required,
    ]),
    senderTemplateDescription: new FormControl<string>('', [
      Validators.maxLength(500),
      Validators.required,
    ]),
  });

  constructor(private templateActionShare: TemplateActionsShareService) {}

  create() {
    const name = this.createTemplateForm.get('senderTemplateName')?.value;
    const description = this.createTemplateForm.get(
    'senderTemplateDescription'
    )?.value;
    if (name && description && this.createTemplateForm.valid) {
      this.templateActionShare.nextCreate({
        name: name,
        description: description,
      });
    }
  }

  getErrorMessage(formControlName: string) {
    return getErrorMessage(formControlName, this.createTemplateForm);
  }
}

import { Component, Optional } from '@angular/core';
import { ParticipantService } from '../services/participant.service';
import { MatInputModule } from '@angular/material/input';
import { MatFormFieldModule } from '@angular/material/form-field';
import {
  FormControl,
  FormGroup,
  ReactiveFormsModule,
  Validators,
} from '@angular/forms';
import { CommonModule } from '@angular/common';
import { MatIconModule } from '@angular/material/icon';
import { MatButtonModule } from '@angular/material/button';
import { validatePassword } from '../registration-form/password-validator';
import { getErrorMessage } from '../../shared/validation-errors-getter';
import { AuthenticationService } from '../services/authentication.service';
```

```
import { MatSnackBar } from '@angular/material/snack-bar';
import {
  MatDialogClose,
  MatDialogContent,
  MatDialogRef,
} from '@angular/material/dialog';

@Component({
  selector: 'app-change-password',
  standalone: true,
  imports: [
    MatInputModule,
    MatFormFieldModule,
    ReactiveFormsModule,
    CommonModule,
    MatIconModule,
    MatButtonModule,
    MatDialogContent,
    MatDialogClose,
  ],
  templateUrl: './change-password.component.html',
  styleUrl: './change-password.component.scss',
})
export class ChangePasswordComponent {
  changePasswordForm = new FormGroup({
  senderOldPassword: new FormControl('', [
      Validators.required,
      validatePassword,
      Validators.minLength(8),
    ]),
    senderNewPassword: new FormControl('', [
      Validators.required,
      validatePassword,
      Validators.minLength(8),
    ]),
  });

  hideOldPassword = false;
  hideNewPassword = false;

  constructor(
    private participantService: ParticipantService,
    private authService: AuthenticationService,
    private snackBar: MatSnackBar,
    @Optional() private dialogRef?: MatDialogRef<any>
  ) {}
  change() {
    const id = this.authService.getParticipant()?.id;
    const oldPasswordForm = this.changePasswordForm.get('senderOldPassword') !!;
    const newPasswordForm = this.changePasswordForm.get('senderNewPassword') !!;
    if (id == undefined) {
      this.snackBar.open(
        $localize`You have to be logged in to perform this action.`,
        $localize`close`,
        {
```

```
          duration: 2000,
        }
      );
      return;
    }
    if (oldPasswordForm.valid && newPasswordForm.valid) {
      this.participantService
        .changePassword(id, {
          oldPassword: oldPasswordForm.value !!,
          newPassword: newPasswordForm.value !!,
        })
        .then(() => {
          this.dialogRef?.close();
          this.snackBar.open(
            $localize`Your password was changed.`,
            $localize`close`,
            {
              duration: 2000,
            }
          );
        });
    }
  }

  getErrorMessage(formControlName: string) {
    return getErrorMessage(formControlName, this.changePasswordForm);
  }
}
import { Component } from '@angular/core';
import { MatTabsModule } from '@angular/material/tabs';
import { MatDialogContent } from '@angular/material/dialog';
import { LoginFormComponent } from '../login-form/login-form.component';
import { RegistrationFormComponent } from '../registration-form/registration-form.component';
import { AuthenticationService } from '../services/authentication.service';
import { NgEventBus } from 'ng-event-bus';
import { AutoClosableComponent } from '../../shared/AutoClosableComponent';
import { Events } from '../../shared/duty-manager-events';

@Component({
  selector: 'app-authentication',
  standalone: true,
  imports: [MatTabsModule, MatDialogContent, LoginFormComponent, RegistrationFormComponent],
  templateUrl: './authentication.component.html',
  styleUrl: './authentication.component.scss',
})
export class AuthenticationComponent extends AutoClosableComponent {
  userIsAdmin = false;
  constructor(authService: AuthenticationService, eventBus: NgEventBus) {
    super();
    this.userIsAdmin = authService.getParticipant()?.role.name === 'ADMIN';
    super.manage(eventBus.on(Events.LOGGED_IN).subscribe(() => {
    this.userIsAdmin = authService.getParticipant()?.role.name === 'ADMIN';
    }))
  }
}
```

Додаток Б

Код основних сервісів та компонентів на бекенді

```
package com.duty.manager.service.impl;

import com.duty.manager.entity.Role;
import jakarta.annotation.Nullable;
import jakarta.validation.constraints.NotNull;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;

import java.util.function.Consumer;
import java.util.function.Function;

public class AuthenticationAwareService {

  public <T> T roleAwareCall(Function<Boolean, T> roleAwareFunction, @Nullable Role role) {
    boolean found = SecurityContextHolder.getContext().getAuthentication()
        .getAuthorities().stream()
        .anyMatch(authority -> {
          if (role == null) {
            return true;
          } else {
            return authority.getAuthority().equals(role.getAuthority());
          }
        }
        );
    return roleAwareFunction.apply(found);
  }

  public Authentication authAwareCall(Consumer<Authentication> authConsumer) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    authConsumer.accept(authentication);
    return authentication;
  }

  public boolean containsRole(@NotNull Authentication authentication, @Nullable Role role) {
    if (role == null) {
      return true;
    } else {
      return authentication.getAuthorities().stream()
          .anyMatch(
              authority -> authority.getAuthority().equals(role.getAuthority())
          );
    }
  }

}

package com.duty.manager.service.impl;

import com.duty.manager.service.UserDetailsProvider;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class CredentialsService implements UserDetailsService {

  private final List<UserDetailsProvider<? extends UserDetails>> userDetailsProviders;

  @Override
  public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
    Optional<? extends UserDetails> userDetails = userDetailsProviders.stream()
        .map(p -> p.getUserByIdentifier(email))
        .filter(Optional::isPresent)
        .findFirst()
        .orElseThrow(() -> new UsernameNotFoundException("Email or password are not correct"));
    return userDetails.get();
  }
}
```

```
package com.duty.manager.service.impl;

import com.duty.manager.dto.GetExecutionFactDTO;
import com.duty.manager.dto.RecordExecutionFactDTO;
import com.duty.manager.entity.ExecutionFact;
import com.duty.manager.entity.Role;
import com.duty.manager.repository.ExecutionFactRepository;
import com.duty.manager.repository.ParticipantRepository;
import com.duty.manager.repository.TemplateRepository;
import com.duty.manager.service.ExecutionFactService;
import com.duty.manager.service.ServiceException;
import com.duty.manager.service.TimeService;
import jakarta.annotation.Nullable;
import jakarta.annotation.PostConstruct;
import jakarta.transaction.Transactional;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.NotNull;
import org.modelmapper.Converter;
import org.modelmapper.ModelMapper;
import org.modelmapper.PropertyMap;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.format.datetime.standard.DateTimeContext;
import org.springframework.format.datetime.standard.DateTimeContextHolder;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
```

```
import java.time.ZoneId;
import java.time.ZoneOffset;
import java.util.LinkedList;
import java.util.List;
import java.util.UUID;
import java.util.function.Function;
import java.util.function.Supplier;

@Service
@Transactional
public class ExecutionFactServiceImpl extends RoleBasedMappingService implements ExecutionFactService {

  public static final int MAXIMAL_PAGE_SIZE = 200;

  private final ExecutionFactRepository executionFactRepository;

  private final TimeService timeService;

  private final ModelMapper modelMapper;

  private final TemplateRepository templateRepository;

  private final ParticipantRepository participantRepository;

  public ExecutionFactServiceImpl(ModelMapper modelMapper,
                                  ExecutionFactRepository executionFactRepository,
                                  TimeService timeService,
                                  TemplateRepository templateRepository,
                                  ParticipantRepository participantRepository) {
    super(modelMapper);
    this.executionFactRepository = executionFactRepository;
    this.timeService = timeService;
    this.modelMapper = modelMapper;
    this.templateRepository = templateRepository;
    this.participantRepository = participantRepository;
  }

  @PostConstruct
  private void configureModelMapper() {
    modelMapper.addMappings(new PropertyMap<RecordExecutionFactDTO, ExecutionFact>() {
      @Override
      protected void configure() {
        skip().setId(null);
        using(idToEntityConvertor(id -> getNullableReferenceById(id, templateRepository::getReferenceById)))
            .map(source.getTemplateId()).setTemplate(null);
        using(idToEntityConvertor(participantRepository::getReferenceById))
            .map(source.getExecutorId()).setExecutor(null);
        with(req -> timeService.utcNow()).map().setStartTime(null);
      }
    });
    this.modelMapper.createTypeMap(ExecutionFact.class, GetExecutionFactDTO.class).addMappings(mapping ->
{
      mapping.map(src -> src.getExecutor().getId(), GetExecutionFactDTO::setExecutorId);
      mapping.map(src -> src.getTemplate().getId(), GetExecutionFactDTO::setTemplateId);
    });
```

```
    }

    private <E> Converter<UUID, E> idToEntityConvertor(Function<UUID, E> supplier) {
        return ctx -> supplier.apply(ctx.getSource());
    }

    private <E> E getNullableReferenceById(UUID id, Function<UUID, E> supplier) {
        if (id == null) {
            return null;
        } else {
            return supplier.apply(id);
        }
    }

    @Override
    public List<GetExecutionFactDTO> getFinishedForDateRange(@NotNull LocalDateTime from, @Nullable
LocalDateTime to,
                                                             @Nullable @Max(200) Integer pageSize) {
        return getExecutionFactDTOS(from, to, pageSize, executionFactRepository::getAllFinishedInRange);
    }

    private List<GetExecutionFactDTO> getExecutionFactDTOS(LocalDateTime from, LocalDateTime to, Integer
pageSize,
                                                           TripletFunction<LocalDateTime,
                                                                   LocalDateTime,
                                                                   Pageable,
                                                                   List<ExecutionFact>> factSupplier) {
        DateTimeContext ctx = DateTimeContextHolder.getDateTimeContext();
        if (to == null) {
            to = timeService.utcNow();
        } else {
            to = getUtcZonedTime(to, ctx);
        }
        from = getUtcZonedTime(from, ctx);
        if (pageSize == null) {
            pageSize = MAXIMAL_PAGE_SIZE;
        }
        if (from.isAfter(to)) {
            throw new ServiceException("From date can not be after to date");
        } else if (from.isAfter(timeService.utcNow())) {
            return new LinkedList<>();
        }
        return factSupplier.apply(from, to, PageRequest.ofSize(pageSize)).stream()
                .map(this::mapEntityToGetDTO)
                .toList();
    }

    private LocalDateTime getUtcZonedTime(LocalDateTime dateTime, DateTimeContext ctx) {
        ZoneId id;
        if(ctx == null) {
            return dateTime;
        } else {
            id = ctx.getTimeZone();
            if(id == null) {
                return dateTime;
```

```
            }
        }
        return dateTime.atZone(id).withZoneSameInstant(ZoneOffset.UTC).toLocalDateTime();
    }

    private GetExecutionFactDTO mapEntityToGetDTO(ExecutionFact fact) {
        GetExecutionFactDTO getDTO = super.mapToDto(fact, GetExecutionFactDTO.class, Role.GUEST);
        getDTO.getTestimonies().forEach(t -> {
            t.setTemplateName(getDTO.getTemplateName());
            t.setSecured(getDTO.isSecured());
        });
        return getDTO;
    }

    @Override
    public UUID recordExecutionFact(RecordExecutionFactDTO factDTO) {
        ExecutionFact fact = modelMapper.map(factDTO, ExecutionFact.class);
        if (factDTO.getDescription() == null ||
                factDTO.getDescription().trim().isEmpty() && factDTO.getTemplateId() != null) {
            fact.setDescription(fact.getTemplate().getDescription());
        }
        return executionFactRepository.save(fact).getId();
    }

    @Override
    public List<GetExecutionFactDTO> getFinishedForDateRangeForParticipant(@NotNull LocalDateTime from,
                                                                           @Nullable LocalDateTime to,
                                                                           @NotNull UUID participantId,
                                                                           @Nullable @Max(200) Integer pageSize) {
        return getExecutionFactDTOS(from, to, pageSize, (validatedFrom, notNullTo, pageable) ->
executionFactRepository
                .getAllFinishedInRangeForParticipant(validatedFrom, notNullTo, participantId, pageable)
        );
    }

    @Override
    public void finishExecution(UUID id, Authentication authentication) {
        ExecutionFact executionFact = getRawExecutionFact(id);
        if (!authentication.getName().equals(executionFact.getExecutor().getEmail()) &&
                !authentication.getAuthorities().contains(Role.ADMIN)) {
            throw new ServiceException("You are not allowed to finish this execution fact", HttpStatus.FORBIDDEN);
        }
        if (executionFact.getFinishTime() != null) {
            throw new ServiceException("Execution fact with id %s is already finished".formatted(id));
        }
        executionFact.setFinishTime(timeService.utcNow());
        executionFactRepository.flush();
    }

    private ExecutionFact getRawExecutionFact(UUID id) {
        return executionFactRepository.findById(id).orElseThrow(notFound(id.toString()));
    }

    private Supplier<ServiceException> notFound(String identifier) {
```

```
        return () -> new ServiceException("Execution fact with id %s not found".formatted(identifier),
HttpStatus.NOT_FOUND);
    }

    @Override
    public List<GetExecutionFactDTO> getActiveForDateRange(@NotNull LocalDateTime from, @Nullable
LocalDateTime to,
                                                           @Nullable @Max(200) Integer pageSize) {
        return getExecutionFactDTOS(from, to, pageSize, executionFactRepository::getAllActiveInRange);
    }

    @Override
    public List<GetExecutionFactDTO> getActiveForDateRangeForParticipant(@NotNull LocalDateTime from,
                                                                         @Nullable LocalDateTime to,
                                                                         @NotNull UUID participantId,
                                                                         @Nullable @Max(200) Integer pageSize) {
        return getExecutionFactDTOS(from, to, pageSize, (validatedFrom, notNullTo, pageable) ->
executionFactRepository
                .getAllActiveInRangeForParticipant(validatedFrom, notNullTo, participantId, pageable)
        );
    }

    @Override
    public GetExecutionFactDTO getById(UUID id) {
        return mapEntityToGetDTO(getRawExecutionFact(id));
    }

    @Override
    public List<GetExecutionFactDTO> getInRange(@NotNull LocalDateTime from, @Nullable LocalDateTime to,
                                                @Nullable @Max(200) Integer pageSize) {
        return getExecutionFactDTOS(from, to, pageSize, executionFactRepository::getAllInRange);
    }

    @Override
    public List<GetExecutionFactDTO> getInRangeForParticipant(@NotNull LocalDateTime from, @Nullable
LocalDateTime to,
                                                              @NotNull UUID participantId, @Nullable @Max(200) Integer pageSize) {
        return getExecutionFactDTOS(from, to, pageSize, (validatedFrom, notNullTo, pageable) ->
executionFactRepository
                .getAllInRangeForParticipant(validatedFrom, notNullTo, participantId, pageable)
        );
    }

    @Override
    public void deleteExecutionFact(UUID factId) {
        ExecutionFact fact = getRawExecutionFact(factId);
        if (!fact.getTestimonies().isEmpty()) {
            throw new ServiceException("You can not delete this execution fact as it was testified.",
HttpStatus.BAD_REQUEST);
        }
        authAwareCall(authentication -> {
            if (fact.getExecutor().getEmail().equals(authentication.getName()) ||
                    containsRole(authentication, Role.ADMIN)) {
                executionFactRepository.delete(fact);
            }
```

```
        });
    }

    @FunctionalInterface
    public interface TripletFunction<A1, A2, A3, R> {
        R apply(A1 argument1, A2 argument2, A3 argument3);
    }

}

package com.duty.manager.service.impl;

import com.duty.manager.dto.GetParticipantDTO;
import com.duty.manager.service.ParticipantService;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Service
public class JwtService {

    private final String jwtKey;

    private final Long expirationTime;

    private final ParticipantService participantService;

    public JwtService(@Qualifier("jwtKey") String jwtKey, @Qualifier("expirationTime") Long expirationTime,
                      ParticipantService participantService) {
        this.jwtKey = jwtKey;
        this.expirationTime = expirationTime;
        this.participantService = participantService;
    }

    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimResolver) {
        Claims claims = extractAllClaims(token);
        return claimResolver.apply(claims);
    }

    public GetParticipantDTO generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {
```

```
        GetParticipantDTO participant = participantService.getParticipant(userDetails.getUsername());
        String jwt = Jwts.builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + expirationTime))
                .signWith(getSigningKey(), SignatureAlgorithm.HS256)
                .compact();
        participant.setJwt(jwt);
        return participant;
    }

    public GetParticipantDTO generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }

    public boolean isTokenValid(String token, UserDetails userDetails) {
        String username = extractUserName(token);
        return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
    }

    private boolean isTokenExpired(String token) {
        return extractClaim(token, Claims::getExpiration).before(new Date());
    }

    private Claims extractAllClaims(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(getSigningKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSigningKey() {
        byte[] keyBytes = Decoders.BASE64.decode(jwtKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}

package com.duty.manager.service.impl;

import com.duty.manager.dto.ChangePasswordDTO;
import com.duty.manager.dto.GetParticipantDTO;
import com.duty.manager.dto.RegisterParticipantDTO;
import com.duty.manager.entity.Participant;
import com.duty.manager.entity.Role;
import com.duty.manager.repository.ParticipantRepository;
import com.duty.manager.repository.RoleRepository;
import com.duty.manager.service.ParticipantService;
import com.duty.manager.service.ServiceException;
import com.duty.manager.service.UserDetailsProvider;
import jakarta.transaction.Transactional;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotNull;
```

```
import org.modelmapper.ModelMapper;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.HttpStatus;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.UUID;
import java.util.function.Supplier;

@Service
@Transactional
public class ParticipantServiceImpl extends RoleBasedMappingService implements ParticipantService,
UserDetailsProvider<Participant> {

  private final ParticipantRepository participantRepository;
  private final ModelMapper modelMapper;
  private final PasswordEncoder passwordEncoder;
  private final RoleRepository roleRepository;

  public ParticipantServiceImpl(ModelMapper modelMapper,
                        ParticipantRepository participantRepository,
                        PasswordEncoder passwordEncoder,
                        RoleRepository roleRepository) {
    super(modelMapper);
    this.participantRepository = participantRepository;
    this.modelMapper = modelMapper;
    this.passwordEncoder = passwordEncoder;
    this.roleRepository = roleRepository;
  }

  @Override
  public UUID registerParticipant(RegisterParticipantDTO registerParticipantDTO) {
    throwExceptionIfEmailIsTaken(registerParticipantDTO);
    Participant newParticipant = modelMapper.map(registerParticipantDTO, Participant.class);
    newParticipant.setPassword(passwordEncoder.encode(newParticipant.getPassword()));
    newParticipant.setRole(roleRepository.findByName(Role.PARTICIPANT.getName())
          .orElseThrow(()->new ServiceException("Roles are not configured in the database")));
    return participantRepository.saveAndFlush(newParticipant).getId();
  }

  private void throwExceptionIfEmailIsTaken(RegisterParticipantDTO registerParticipantDTO) {
    if (participantRepository.findByEmail(registerParticipantDTO.getEmail()).isPresent()) {
      throw new ServiceException("Email is already taken");
    }
  }

  @Override
  public GetParticipantDTO getParticipant(String identifier) {
    return mapToGetDTO(getRawParticipant(identifier));
  }

  private Participant getRawParticipant(String identifier) {
    try {
```

```
            UUID id = UUID.fromString(identifier);
            return participantRepository.findById(id).orElseThrow(notFound(identifier));
        } catch (IllegalArgumentException e) {
            return participantRepository.findByEmail(identifier).orElseThrow(notFound(identifier));
        }
    }

    private GetParticipantDTO mapToGetDTO(Participant participant) {
        return super.mapToDto(participant, GetParticipantDTO.class, Role.GUEST);
    }

    private Supplier<ServiceException> notFound(String identifier) {
        return () -> new ServiceException("Participant with identifier %s not found".formatted(identifier));
    }

    @Override
    public List<GetParticipantDTO> getParticipants(@Min(0) @NotNull Integer page, @Max(200) @NotNull Integer
pageSize) {
        return participantRepository.findAll(PageRequest.of(page,
pageSize)).stream().map(this::mapToGetDTO).toList();
    }

    @Override
    public void changePassword(String identifier, @NotNull ChangePasswordDTO changePasswordDTO) {
        Participant participant = getRawParticipant(identifier);
        if(passwordEncoder.matches(changePasswordDTO.getOldPassword(), participant.getPassword())) {
            participant.setPassword(passwordEncoder.encode(changePasswordDTO.getNewPassword()));
        } else {
            throw new ServiceException("Old password incorrect.", HttpStatus.BAD_REQUEST);
        }
    }

    @Override
    public Optional<Participant> getUserByIdentifier(String identifier) {
        return participantRepository.findByEmail(identifier);
    }
}

package com.duty.manager.service.impl;

import com.duty.manager.dto.GetSecuredDTO;
import com.duty.manager.entity.Role;
import jakarta.annotation.Nullable;
import lombok.RequiredArgsConstructor;
import org.modelmapper.ModelMapper;

@RequiredArgsConstructor
public class RoleBasedMappingService extends AuthenticationAwareService {

    private final ModelMapper modelMapper;

    public <S, D extends GetSecuredDTO> D mapToDto(S source, Class<D> destination, @Nullable Role
forbiddenRole) {
        return roleAwareCall(searchResult -> {
            D mappedDTO = modelMapper.map(source, destination);
```

```
            mappedDTO.setSecured(searchResult);
            return mappedDTO;
        }, forbiddenRole);
    }

    public <S, D extends GetSecuredDTO> D mapToDto(S source, Class<D> destination) {
        return mapToDto(source, destination, null);
    }

}

package com.duty.manager.service.impl;

import com.duty.manager.dto.CreateTemplateDTO;
import com.duty.manager.dto.GetTemplateDTO;
import com.duty.manager.dto.UpdateTemplateDTO;
import com.duty.manager.entity.ExecutionFact;
import com.duty.manager.entity.Role;
import com.duty.manager.entity.Template;
import com.duty.manager.repository.TemplateRepository;
import com.duty.manager.service.ServiceException;
import com.duty.manager.service.TemplateService;
import jakarta.transaction.Transactional;
import jakarta.validation.Valid;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotNull;
import org.hibernate.validator.constraints.Length;
import org.modelmapper.ModelMapper;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

@Service
@Transactional
public class TemplateServiceImpl extends RoleBasedMappingService implements TemplateService {

    private final TemplateRepository templateRepository;

    private final ModelMapper modelMapper;

    public TemplateServiceImpl(ModelMapper modelMapper, TemplateRepository templateRepository) {
        super(modelMapper);
        this.templateRepository = templateRepository;
        this.modelMapper = modelMapper;
    }

    @Override
    public UUID createTemplate(CreateTemplateDTO templateDTO) {
        throwExceptionIfExist(templateDTO);
        Template template = modelMapper.map(templateDTO, Template.class);
        return templateRepository.save(template).getId();
```

```
    }

    private void throwExceptionIfExist(CreateTemplateDTO templateDTO) {
        String name = templateDTO.getName();
        try {
            getRowTemplate(name);
            throw new IllegalArgumentException(
                    "Template with name %s already exists".formatted(name)
            );
        } catch (ServiceException ignored) {
        }
    }

    @Override
    public List<GetTemplateDTO> getTemplates(@NotNull @Min(0) Integer page, @Max(50) @NotNull Integer
pageSize) {
        return templateRepository.findAll(PageRequest.of(page,  pageSize)).get()
                .map(this::templateEntityToGetDTO).toList();
    }

    private GetTemplateDTO templateEntityToGetDTO(Template template) {
        return super.mapToDto(template, GetTemplateDTO.class, Role.GUEST);
    }

    @Override
    public GetTemplateDTO getTemplate(@NotNull String identifier) {
        return templateEntityToGetDTO(getRowTemplate(identifier));
    }

    private Template getRowTemplate(String identifier) {
        Optional<Template> template;
        try {
            template = templateRepository.findById(UUID.fromString(identifier));
        } catch (IllegalArgumentException e) {
            template = templateRepository.findByName(identifier);
        }
        return template.orElseThrow(
                () -> new ServiceException("Template with identifier %s not found".formatted(identifier))
        );
    }

    @Override
    public void updateTemplate(@NotNull String identifier, @Valid UpdateTemplateDTO templateUpdates) {
        Template template = getRowTemplate(identifier);
        if (templateUpdates.getName() != null &&
templateRepository.findByName(templateUpdates.getName()).isEmpty()) {
            template.setName(templateUpdates.getName());
        }
        if (templateUpdates.getDescription() != null) {
            template.setDescription(templateUpdates.getDescription());
        }
        templateRepository.flush();
    }

    @Override
```

```
    public void deleteTemplate(@NotNull String identifier) {
        try {
            Template template = getRowTemplate(identifier);
            List<ExecutionFact> relatedFacts = template.getFacts();
            if (!relatedFacts.isEmpty()) {
                relatedFacts.forEach(fact -> fact.setTemplate(null));
            }
            templateRepository.delete(template);
            templateRepository.flush();
        } catch (Exception e) {
            throw new ServiceException(e.getMessage());
        }
    }

    @Override
    public Long getNumberOfEntities() {
        return templateRepository.count();
    }

    @Override
    public List<GetTemplateDTO> getTemplatesByFuzzyName(@NotNull @Length(max = 100) String fuzzyName) {
        return templateRepository.fuzzySearchByName(fuzzyName).stream()
                .map(this::templateEntityToGetDTO)
                .toList();
    }

}
```

```
package com.duty.manager.service.impl;

import com.duty.manager.dto.GetTestimonyDTO;
import com.duty.manager.entity.Role;
import com.duty.manager.entity.Testimony;
import com.duty.manager.repository.ExecutionFactRepository;
import com.duty.manager.repository.ParticipantRepository;
import com.duty.manager.repository.TestimonyRepository;
import com.duty.manager.service.ServiceException;
import com.duty.manager.service.TestimonyService;
import com.duty.manager.service.TimeService;
import jakarta.transaction.Transactional;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotNull;
import org.modelmapper.ModelMapper;
import org.springframework.data.domain.PageRequest;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.UUID;
import java.util.function.Supplier;

@Service
@Transactional
public class TestimonyServiceImpl extends RoleBasedMappingService implements TestimonyService {
```

```
private final TimeService timeService;

private final TestimonyRepository testimonyRepository;

private final ParticipantRepository participantRepository;

private final ExecutionFactRepository executionFactRepository;


public TestimonyServiceImpl(ModelMapper modelMapper,
                            TimeService timeService,
                            TestimonyRepository testimonyRepository,
                            ParticipantRepository participantRepository,
                            ExecutionFactRepository executionFactRepository) {
   super(modelMapper);
   this.timeService = timeService;
   this.testimonyRepository = testimonyRepository;
   this.participantRepository = participantRepository;
   this.executionFactRepository = executionFactRepository;
}

@Override
public UUID testifyExecutionFact(@NotNull UUID executionFactId, @NotNull Authentication authentication) {
   UUID witnessId = participantRepository.findByEmail(authentication.getName())
         .orElseThrow(notFound(authentication.getName())).getId();
   checkIfExist(executionFactId);
   throwIfPersonAlreadyTestifiedFact(executionFactId, witnessId);
   return testimonyRepository.save(
         Testimony.builder()
               .withExecutionFact(executionFactRepository.getReferenceById(executionFactId))
               .withWitness(participantRepository.getReferenceById(witnessId))
               .withTimestamp(timeService.utcNow())
               .build()
   ).getId();
}

private Supplier<ServiceException> notFound(String identifier) {
   return () -> new ServiceException("Participant with email %s not found".formatted(identifier));
}

private void checkIfExist(UUID executionFactId) {
   executionFactRepository.getReferenceById(executionFactId);
}

private void throwIfPersonAlreadyTestifiedFact(UUID executionFactId, UUID witnessId) {
   testimonyRepository.findByExecutionFactIdAndWitnessId(executionFactId, witnessId)
         .ifPresent(t -> {
            throw new ServiceException("Execution fact with id %s was already witnesses by %s"
                  .formatted(executionFactId,  witnessId));
         });
}

@Override
```

```
  public List<GetTestimonyDTO> getTestimoniesForExecutionFact(@NotNull UUID id, @Min(0) @NotNull
Integer page,
                                                @Max(200) @NotNull Integer pageSize) {
      return testimonyRepository.findAllByExecutionFactId(id,  PageRequest.of(page,  pageSize)).stream()
            .map(this::mapToGetDTO)
            .toList();
  }

  private GetTestimonyDTO mapToGetDTO(Testimony t) {
      GetTestimonyDTO getDTO = super.mapToDto(t, GetTestimonyDTO.class, Role.GUEST);
      getDTO.setTemplateName(t.getExecutionFact().getTemplate().getName());
      return getDTO;
  }

}
```

```
package com.duty.manager.service.impl;

import com.duty.manager.service.TimeService;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.time.ZoneOffset;

@Service
public class TimeServiceImpl implements TimeService {

  @Override
  public LocalDateTime utcNow() {
      return LocalDateTime.now(ZoneOffset.UTC);
  }

}
```

```
package com.duty.manager.repository;

import com.duty.manager.entity.Testimony;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

@Repository
public interface TestimonyRepository extends JpaRepository<Testimony, UUID> {

  List<Testimony> findAllByExecutionFactId(UUID id, Pageable pageable);

  Optional<Testimony>  findByExecutionFactIdAndWitnessId(UUID executionFactId, UUID witnessId);
}
```

```
package com.duty.manager.repository;
```

```
import com.duty.manager.entity.Template;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

@Repository
public interface TemplateRepository extends JpaRepository<Template, UUID> {

  Optional<Template> findByName(String name);

  @Query(
        nativeQuery = true,
        value = """
              WITH similarity_query AS (
                 SELECT *, word_similarity(name, :name) AS name_similarity
                 FROM templates
              )
              SELECT id, name, description, version
              FROM similarity_query
              WHERE name_similarity >= 0.1
              ORDER BY name_similarity desc
              LIMIT 50;
              """
  )
  List<Template> fuzzySearchByName(@Param("name") String name);

}

package com.duty.manager.repository;

import com.duty.manager.entity.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;
import java.util.UUID;

@Repository
public interface RoleRepository extends JpaRepository<Role, UUID> {
  Optional<Role> findByName(String name);
}

package com.duty.manager.repository;

import com.duty.manager.entity.Participant;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;
import java.util.UUID;
```

```
@Repository
public interface ParticipantRepository extends JpaRepository<Participant, UUID> {
  Optional<Participant> findByEmail(String email);

}

package com.duty.manager.repository;

import com.duty.manager.entity.ExecutionFact;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.time.LocalDateTime;
import java.util.List;
import java.util.UUID;

@Repository
public interface ExecutionFactRepository extends JpaRepository<ExecutionFact, UUID> {

  @Query("select f from ExecutionFact f "
      + "where f.startTime >= :from and f.startTime <= :to or f.finishTime >= :from and f.finishTime <= :to")
  List<ExecutionFact> getAllInRange(@Param("from") LocalDateTime from, @Param("to") LocalDateTime to,
                    Pageable pageable);

  @Query("select f from ExecutionFact f "
      + "where (f.startTime >= :from and f.startTime <= :to or"
      + " f.finishTime >= :from and f.finishTime <= :to) and f.executor.id = :participantId")
  List<ExecutionFact> getAllInRangeForParticipant(@Param("from") LocalDateTime from,
                          @Param("to") LocalDateTime to,
                          @Param("participantId") UUID participantId,
                          Pageable pageable);

  @Query("select f from ExecutionFact f where f.finishTime >= :from and f.finishTime <= :to")
  List<ExecutionFact> getAllFinishedInRange(@Param("from") LocalDateTime from, @Param("to")
LocalDateTime to,
                        Pageable pageable);

  @Query("select f from ExecutionFact f where f.finishTime >= :from and f.finishTime <= :to and f.executor.id =
:participantId")
  List<ExecutionFact> getAllFinishedInRangeForParticipant(@Param("from") LocalDateTime from,
                            @Param("to") LocalDateTime to,
                            @Param("participantId") UUID participantId,
                            Pageable pageable);

  @Query("select f from ExecutionFact f where f.startTime >= :from and f.startTime <= :to and f.finishTime is null")
  List<ExecutionFact> getAllActiveInRange(@Param("from") LocalDateTime from, @Param("to") LocalDateTime
to,
                      Pageable pageable);

  @Query("select f from ExecutionFact f where f.startTime >= :from and f.startTime <= :to and f.executor.id =
:participantId and f.finishTime is null")
```

```
    List<ExecutionFact> getAllActiveInRangeForParticipant(@Param("from") LocalDateTime from,
                                                          @Param("to") LocalDateTime to,
                                                          @Param("participantId") UUID participantId,
                                                          Pageable pageable);

}

package com.duty.manager.controller;

import com.duty.manager.dto.CreateTemplateDTO;
import com.duty.manager.dto.GetTemplateDTO;
import com.duty.manager.dto.UpdateTemplateDTO;
import com.duty.manager.service.TemplateService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.ArraySchema;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.UUID;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/templates")
public class TemplateController {

    private final TemplateService templateService;

    @GetMapping(params = {"page", "pageSize"})
    @Operation(description = """
            Has two different sets of parameters, page && pageSize || fuzzyName.

            When page && pageSize are specified returns templates at specified page with
            specified page size. Page count starts from 0.

            When fuzzyName is specified return all templates which names are fuzzy matched.
            """)
    @ApiResponse(
            responseCode = "200",
            description = "Returns templates that are at specified page with specified page size",
```

```
        content = @Content(
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                array = @ArraySchema(
                        schema = @Schema(implementation = GetTemplateDTO.class)
                )
        )
)
public List<GetTemplateDTO> getTemplates(@RequestParam(defaultValue = "0")
                                         Integer page,
                                         @RequestParam(defaultValue = "50")
                                         Integer pageSize) {
    return templateService.getTemplates(page, pageSize);
}

@GetMapping(params = {"fuzzyName"})
public List<GetTemplateDTO> getByFuzzyName(@RequestParam String fuzzyName) {
    return templateService.getTemplatesByFuzzyName(fuzzyName);
}

@GetMapping("/{templateIdentifier}")
@Operation(description = "Returns template by specified id or name. Returns 404 if no template was found")
@ApiResponse(
        responseCode = "200",
        description = "Returns template with specified template id or template name",
        content = @Content(
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                schema = @Schema(implementation = GetTemplateDTO.class)
        )
)
@ApiResponse(
        responseCode = "404",
        description = "Template with specified template id or template name was not found",
        content = @Content(
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                schema = @Schema(implementation = ExceptionResponse.class)
        )
)
public GetTemplateDTO getTemplate(@PathVariable String templateIdentifier) {
    return templateService.getTemplate(templateIdentifier);
}

@PostMapping
@Operation(
        description = "Adds given template to database. Returns 400 if template with given name already exists"
)
@ResponseStatus(HttpStatus.CREATED)
@ApiResponse(
        responseCode = "201",
        description = "Returns id of template that was added",
        content = @Content(
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                schema = @Schema(implementation = UUID.class)
        )
)
@ApiResponse(
```

```
            responseCode = "400",
            description = "Means that name already exist",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON_VALUE,
                    schema = @Schema(implementation = ExceptionResponse.class)
            )
    )
    public ResponseEntity<UUID> addTemplate(@RequestBody CreateTemplateDTO addTemplateDto) {
        return
ResponseEntity.status(HttpStatus.CREATED.value()).body(templateService.createTemplate(addTemplateDto));
    }

    @PutMapping("/{templateIdentifier}")
    @Operation(
            description = "Updates template with given identifier with data from UpdateTemplateDTO." +
                    " If UpdateTemplateDTO has null fields then that specific field will not be effected"
    )
    @ApiResponse(
            responseCode = "200",
            description = "Means that template was updated and all given parameters were changed"
    )
    public void updateTemplate(@PathVariable String templateIdentifier, @RequestBody UpdateTemplateDTO
updateTemplateDTO) {
        templateService.updateTemplate(templateIdentifier,   updateTemplateDTO);
    }

    @DeleteMapping("/{templateIdentifier}")
    @Operation(
            description = "Deletes template with given identifier"
    )
    @ApiResponse(
            responseCode = "200",
            description = "Means that template was deleted"
    )
    @ApiResponse(
            responseCode = "400",
            description = "Means that template with given identifier does not exist",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON_VALUE,
                    schema = @Schema(implementation = ExceptionResponse.class)
            )
    )
    public void deleteTemplate(@PathVariable String templateIdentifier) {
        templateService.deleteTemplate(templateIdentifier);
    }

    @GetMapping("/quantity")
    @Operation(
            description = "Returns total number of templates."
    )
    public Long getNumberOfTemplates() {
        return templateService.getNumberOfEntities();
    }

}
```

```
package com.duty.manager.controller;

import com.duty.manager.service.ServiceException;
import io.jsonwebtoken.ExpiredJwtException;
import lombok.Generated;
import org.springframework.core.convert.ConversionFailedException;
import org.springframework.http.HttpStatus;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.time.LocalDateTime;


@RestControllerAdvice
@Generated
public class RestControllerExceptionHandler {

  @ExceptionHandler(jakarta.validation.ConstraintViolationException.class)
  @ResponseStatus(HttpStatus.BAD_REQUEST)
  public ExceptionResponse handleConstraintViolationException(jakarta.validation.ConstraintViolationException e)
{
      return ExceptionResponse.builder()
            .withMessage(e.getMessage())
          .withHttpStatus(HttpStatus.BAD_REQUEST)
          .withDate(LocalDateTime.now())
          .build();
  }

  @ExceptionHandler(org.hibernate.exception.ConstraintViolationException.class)
  @ResponseStatus(HttpStatus.BAD_REQUEST)
  public ExceptionResponse handleHibernateConstraintViolationException(
       org.hibernate.exception.ConstraintViolationException e) {
      return ExceptionResponse.builder()
            .withMessage(e.getMessage())
          .withHttpStatus(HttpStatus.BAD_REQUEST)
          .withDate(LocalDateTime.now())
          .build();
  }

  @ExceptionHandler(ServiceException.class)
  @ResponseStatus(HttpStatus.BAD_REQUEST)
  public ExceptionResponse handleServiceException(
       ServiceException e) {
      return ExceptionResponse.builder()
            .withMessage(e.getMessage())
          .withHttpStatus(e.getStatusCode())
          .withDate(LocalDateTime.now())
          .build();
  }

  @ExceptionHandler(ConversionFailedException.class)
```

```
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ExceptionResponse handleConversionFailedException(
            ConversionFailedException e) {
        return ExceptionResponse.builder()
                .withMessage(e.getMessage())
                .withHttpStatus(HttpStatus.BAD_REQUEST)
                .withDate(LocalDateTime.now())
                .build();
    }

    @ExceptionHandler(ExpiredJwtException.class)
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    public ExceptionResponse handleExpiredJwtException(
            ExpiredJwtException e) {
        return ExceptionResponse.builder()
                .withMessage("Your session expired.")
                .withHttpStatus(HttpStatus.UNAUTHORIZED)
                .withDate(LocalDateTime.now())
                .build();
    }

    @ExceptionHandler(HttpMessageNotReadableException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ExceptionResponse handleHttpMessageNotReadableException(
            HttpMessageNotReadableException e) {
        return ExceptionResponse.builder()
                .withMessage(e.getMessage())
                .withHttpStatus(HttpStatus.BAD_REQUEST)
                .withDate(LocalDateTime.now())
                .build();
    }

    @ExceptionHandler(UsernameNotFoundException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ExceptionResponse handleUsernameNotFoundException(
            UsernameNotFoundException e) {
        return ExceptionResponse.builder()
                .withMessage(e.getMessage())
                .withHttpStatus(HttpStatus.BAD_REQUEST)
                .withDate(LocalDateTime.now())
                .build();
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ExceptionResponse handleIllegalArgumentExceptionException(
            IllegalArgumentException e) {
        return ExceptionResponse.builder()
                .withMessage(e.getMessage())
                .withHttpStatus(HttpStatus.BAD_REQUEST)
                .withDate(LocalDateTime.now())
                .build();
    }

}
```

```
package com.duty.manager.controller;

import com.duty.manager.dto.ChangePasswordDTO;
import com.duty.manager.dto.GetParticipantDTO;
import com.duty.manager.dto.RegisterParticipantDTO;
import com.duty.manager.service.ParticipantService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.enums.ParameterIn;
import io.swagger.v3.oas.annotations.media.ArraySchema;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.UUID;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/participants")
public class ParticipantController {

  private final ParticipantService participantService;

  @PostMapping
  @Operation(
        description = "Register participant."
  )
  @ApiResponse(
        responseCode = "201",
        description = "Participant was successfully registered. It returns id of created participant",
        content = @Content(
              mediaType = MediaType.APPLICATION_JSON_VALUE,
              schema = @Schema(implementation = UUID.class)
        )
  )
  public ResponseEntity<UUID> register(@RequestBody RegisterParticipantDTO participantDTO) {
     return ResponseEntity.status(HttpStatus.CREATED.value())
           .body(participantService.registerParticipant(participantDTO));
  }

  @GetMapping("/{identifier}")
  @Operation(description = "Returns participant by specified id or email. Returns 404 if no participant was found")
```

```
  @ApiResponse(
        responseCode = "200",
        description = "Returns participant with specified id or email",
        content = @Content(
              mediaType = MediaType.APPLICATION_JSON_VALUE,
              schema = @Schema(implementation = GetParticipantDTO.class)
        )
  )
  @ApiResponse(
        responseCode = "404",
        description = "Participant with specified id or email was not found",
        content = @Content(
              mediaType = MediaType.APPLICATION_JSON_VALUE,
              schema = @Schema(implementation = ExceptionResponse.class)
        )
  )
  public GetParticipantDTO getParticipant(@PathVariable String identifier) {
     return participantService.getParticipant(identifier);
  }

  @GetMapping
  @Operation(
        description = "Returns list of registered participants",
        parameters = @Parameter(name = "Authorization", in = ParameterIn.HEADER, required = true)
  )
  @ApiResponse(
        responseCode = "200",
        description = "You are allowed to get this list, defaults: page = 0, pageSize = 200," +
              " restrictions: page min = 0, pageSize max = 200",
        content = @Content(
              mediaType = MediaType.APPLICATION_JSON_VALUE,
              array = @ArraySchema(
                    schema = @Schema(implementation = GetParticipantDTO.class)
              )
        )
  )
  public List<GetParticipantDTO> getParticipants(@RequestParam(required = false, defaultValue = "0")
                                        Integer page,
                                        @RequestParam(required = false, defaultValue = "200")
                                        Integer pageSize) {
     return participantService.getParticipants(page, pageSize);
  }

  @PostMapping("/{identifier}/password")
  @Operation(description = "Changes password for participant with given identifier, identifier can be either email or
id.")
  public void changePassword(@PathVariable String identifier, @RequestBody ChangePasswordDTO
passwordDTO) {
     participantService.changePassword(identifier,  passwordDTO);
  }

}

package com.duty.manager.controller;
```

```
import com.duty.manager.dto.GetExecutionFactDTO;
import com.duty.manager.dto.GetTestimonyDTO;
import com.duty.manager.dto.RecordExecutionFactDTO;
import com.duty.manager.service.ExecutionFactService;
import com.duty.manager.service.TestimonyService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.ArraySchema;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDateTime;
import java.util.List;
import java.util.UUID;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/execution-facts")
public class ExecutionFactController {

  private final ExecutionFactService executionFactService;

  private final TestimonyService testifyExecutionFact;

  @GetMapping(path = "/finished", params = {"from"})
  @Operation(description = "Returns execution facts based on specified date range, \"finish date\" is used for search,"
+
      "Default value for \"to\" is current day. The limit for maximal list size is 200")
  @ApiResponse(
      responseCode = "200",
      description = "Retrieved",
      content = @Content(
          mediaType = MediaType.APPLICATION_JSON_VALUE,
          array = @ArraySchema(
              schema = @Schema(implementation = GetExecutionFactDTO.class)
          )
      )
  )
  public List<GetExecutionFactDTO> getFinishedForDateRange(@RequestParam LocalDateTime from,
                                      @RequestParam(required = false) LocalDateTime to,
```

```
                                        @RequestParam(required = false) Integer pageSize) {
        return executionFactService.getFinishedForDateRange(from, to, pageSize);
    }

    @PostMapping
    @Operation(
            description = "Records given execution fact."
    )
    @ResponseStatus(HttpStatus.CREATED)
    @ApiResponse(
            responseCode = "201",
            description = "Returns id of execution fact that was recorded",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON_VALUE,
                    schema = @Schema(implementation = UUID.class)
            )
    )
    public ResponseEntity<UUID> recordExecutionFact(@RequestBody RecordExecutionFactDTO factDTO) {
        return
ResponseEntity.status(HttpStatus.CREATED.value()).body(executionFactService.recordExecutionFact(factDTO));
    }

    @PostMapping("/finished")
    @Operation(
            description = "Sets finish time for execution fact with given id to current time." +
                    " You can finish only your execution task, ADMIN can finish any."
            ,parameters = @Parameter()
    )
    @ApiResponse(
            responseCode = "200",
            description = "Finish time was set successfully"
    )
    public void finishExecution(@RequestBody UUID executionFactId, Authentication authentication) {
        executionFactService.finishExecution(executionFactId, authentication);
    }

    @PostMapping("/{executionFactId}/testimonies")
    @Operation(
            description = "Authenticated person testifies execution fact."
    )
    @ApiResponse(
            responseCode = "201",
            description = "Execution fact was successfully testified. It returns id of created testimony",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON_VALUE,
                    schema = @Schema(implementation = UUID.class)
            )
    )
    public ResponseEntity<UUID> testifyExecutionFact(@PathVariable UUID executionFactId, Authentication
authentication) {
        return ResponseEntity.status(HttpStatus.CREATED.value())
                .body(testifyExecutionFact.testifyExecutionFact(executionFactId,    authentication));
    }

    @GetMapping("/{executionFactId}/testimonies")
```

```
@Operation(description = "Returns testimonies for specified execution fact.  Default value for \"page\" is 0,\n" +
      "                \"pageSize\" is 200. Min value for page is 0, Max value for page size is 200.")
@ApiResponse(
      responseCode = "200",
      description = "Returns testimonies for specified execution fact.",
      content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            array = @ArraySchema(
                  schema = @Schema(implementation = GetTestimonyDTO.class)
            )
      )
)
public List<GetTestimonyDTO> getTestimoniesForExecutionFact(@PathVariable UUID executionFactId,
                                          @RequestParam(required = false, defaultValue = "0")
                                          Integer page,
                                          @RequestParam(required = false, defaultValue = "200")
                                          Integer pageSize) {
   return testifyExecutionFact.getTestimoniesForExecutionFact(executionFactId,  page,  pageSize);
}

@GetMapping(path = "/finished", params = {"from", "executorId"})
@Operation(description = "Returns execution facts based on specified date range and for specific executor," +
      " \"finish date\" is used for search, Default value for \"to\" is current day. The limit for maximal" +
      " list size is 200")
@ApiResponse(
      responseCode = "200",
      description = "Retrieved",
      content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            array = @ArraySchema(
                  schema = @Schema(implementation = GetExecutionFactDTO.class)
            )
      )
)
public List<GetExecutionFactDTO> getFinishedForDateRangeForExecutor(@RequestParam LocalDateTime from,
                                            @RequestParam(required = false) LocalDateTime to,
                                            @RequestParam UUID executorId,
                                            @RequestParam(required = false) Integer pageSize) {
   return executionFactService.getFinishedForDateRangeForParticipant(from,  to,  executorId,  pageSize);
}

@GetMapping(path = "/active", params = {"from"})
@Operation(description = "Returns execution facts based on specified date range," +
      " \"start date\" is used for search, ignores all records that have \"finish time\", Default value for \"to\"" +
      " is current day. The limit for maximal list size is 200")
@ApiResponse(
      responseCode = "200",
      description = "Retrieved",
      content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            array = @ArraySchema(
                  schema = @Schema(implementation = GetExecutionFactDTO.class)
            )
      )
)
```

```
public List<GetExecutionFactDTO> getActiveForDateRange(@RequestParam LocalDateTime from,
                                        @RequestParam(required = false) LocalDateTime to,
                                        @RequestParam(required = false) Integer pageSize) {
   return executionFactService.getActiveForDateRange(from, to, pageSize);
}

@GetMapping(path = "/active", params = {"from", "executorId"})
@Operation(description = "Returns execution facts based on specified date range and for specific executor," +
      " \"start date\" is used for search, ignores all records that have \"finish time\", Default value for \"to\"" +
      " is current day. The limit for maximal list size is 200")
@ApiResponse(
      responseCode = "200",
      description = "Retrieved",
      content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            array = @ArraySchema(
                  schema = @Schema(implementation = GetExecutionFactDTO.class)
            )
      )
)
public List<GetExecutionFactDTO> getActiveForDateRangeForExecutor(@RequestParam LocalDateTime from,
                                            @RequestParam(required = false) LocalDateTime to,
                                            @RequestParam UUID executorId,
                                            @RequestParam(required = false) Integer pageSize) {
   return executionFactService.getActiveForDateRangeForParticipant(from, to, executorId, pageSize);
}

@GetMapping(params = {"from"})
@Operation(description = "Returns execution facts based on specified date range," +
      " \"start date\" or \"finish time\" is used for search, Default value for \"to\"" +
      " is current day. The limit for maximal list size is 200")
@ApiResponse(
      responseCode = "200",
      description = "Retrieved",
      content = @Content(
            mediaType = MediaType.APPLICATION_JSON_VALUE,
            array = @ArraySchema(
                  schema = @Schema(implementation = GetExecutionFactDTO.class)
            )
      )
)
public List<GetExecutionFactDTO> getForDateRange(@RequestParam LocalDateTime from,
                                    @RequestParam(required = false) LocalDateTime to,
                                    @RequestParam(required = false) Integer pageSize) {
   return executionFactService.getInRange(from, to, pageSize);
}

@GetMapping(params = {"from", "executorId"})
@Operation(description = "Returns execution facts based on specified date range and for specific executor," +
      " \"start date\" or \"finish time\" is used for search, Default value for \"to\"" +
      " is current day. The limit for maximal list size is 200")
@ApiResponse(
      responseCode = "200",
      description = "Retrieved",
      content = @Content(
```

```
                mediaType = MediaType.APPLICATION_JSON_VALUE,
                array = @ArraySchema(
                        schema = @Schema(implementation = GetExecutionFactDTO.class)
                )
        )
    )
    public List<GetExecutionFactDTO> getForDateRangeForExecutor(@RequestParam LocalDateTime from,
                                                @RequestParam(required = false) LocalDateTime to,
                                                @RequestParam UUID executorId,
                                                @RequestParam(required = false) Integer pageSize) {
        return executionFactService.getInRangeForParticipant(from, to, executorId, pageSize);
    }

    @GetMapping("/{id}")
    @Operation(description = "Returns execution fact by given id.")
    @ApiResponse(
            responseCode = "200",
            description = "Retrieved",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON_VALUE,
                    schema = @Schema(implementation = GetExecutionFactDTO.class)
            )
    )
    public GetExecutionFactDTO getByID(@PathVariable UUID id) {
        return executionFactService.getById(id);
    }

    @DeleteMapping("/{id}")
    @Operation(
            description = "Deletes execution fact with given id"
    )
    @ApiResponse(
            responseCode = "200",
            description = "Means that execution fact was deleted"
    )
    @ApiResponse(
            responseCode = "400",
            description = "Means that execution fact with given identifier does not exist",
            content = @Content(
                    mediaType = MediaType.APPLICATION_JSON_VALUE,
                    schema = @Schema(implementation = ExceptionResponse.class)
            )
    )
    public void deleteExecutionFact(@PathVariable UUID id) {
        executionFactService.deleteExecutionFact(id);
    }

}

package com.duty.manager.controller;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;
import lombok.Builder;
```

```
import lombok.Data;
import org.springframework.http.HttpStatus;

import java.time.LocalDateTime;

@Data
@Builder(setterPrefix = "with")
public class ExceptionResponse {

  private final String message;

  private final HttpStatus httpStatus;

  @JsonSerialize(using = LocalDateTimeSerializer.class)
  @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
  private final LocalDateTime date;

}
package com.duty.manager.controller;

import com.duty.manager.dto.AuthenticationRequestDTO;
import com.duty.manager.dto.GetParticipantDTO;
import com.duty.manager.service.impl.JwtService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/auth")
public class AuthenticationController {

  private final AuthenticationManager authenticationManager;
  private final UserDetailsService userDetailsService;
  private final JwtService jwtService;

  @PostMapping("/jwt")
  public ResponseEntity<GetParticipantDTO> login(@RequestBody AuthenticationRequestDTO
authenticationRequest) {
    authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(
            authenticationRequest.getEmail(),
            authenticationRequest.getPassword())
    );
    return ResponseEntity.ok(
        jwtService.generateToken(
            userDetailsService.loadUserByUsername(authenticationRequest.getEmail())
        )
    );
```

```
    }
}
```