

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Курсова робота

з дисципліни «Програмування»

на тему: «Електронна черга»

Виконав

Студент 1 курсу, групи ІА-33

Заранік Максим Юрійович

(підпис)

Керівник:

асистент кафедри ІСТ

Мягкий Михайло Юрійович

(підпис)

Засвідчую, що у цій курсовій роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент

(підпис)

Київ – 2024 року

ЗМІСТ

ВСТУП.....	1
1 ВИМОГИ ДО СИСТЕМИ.....	3
1.1 Функціональні вимоги до системи	3
1.2 Нефункціональні вимоги до системи	3
2 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ.....	4
2.1 Діаграма прецедентів	4
2.2 Опис сценаріїв використання системи.....	5
3 АРХІТЕКТУРА СИСТЕМИ	14
4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ	16
4.1 Загальна структура проекту	16
4.2 Компоненти рівня доступу даних.....	17
4.3 Компоненти рівня бізнес-логіки.....	21
4.4 Компоненти рівня інтерфейсу користувача	23
5 ІНСТРУКЦІЯ КОРИСТУВАЧА	25
ВИСНОВКИ.....	32
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	35
ДОДАТКИ.....	36
ДОДАТОК А.....	36
ДОДАТОК Б	56
ДОДАТОК В	68

ВСТУП

Актуальність електронної черги:

в сучасному світі, де люди постійно повинні вирішувати деякі завдання електронна черга стає необхідною, оскільки вони можуть забезпечити ефективне управління потоком людей та завдань, вони є необхідними для використання в деяких установах та закладах, саме завдяки електронній черзі ми можемо зарезервувати час прийому до лікаря, або відвідування якогось заходу

Деякі основні фактори, які роблять систему електронних черг актуальною та потрібною:

1)Доступність та зручність: система має простий та інтуїтивно зрозумілий інтерфейс, який значно полегшує взаємодію з системою та робить її більш зрозумілою і готовою до використання в будь-який момент часу.

2)Ефективність обслуговування: система електронних черг дозволяє значним чином оптимізувати розподіл часу та ресурсів системи, оскільки всі операції в системі виконуються досить швидко

3)Розподіл потоку завдань: система електронних черг допомагає ефективно розподіляти потік завдань, та організовувати ресурси, що в свою чергу допоможе уникнути не контрольованої ситуації з розподілом цих ресурсів.

4)Безпека використання: система значно зменшує фізичний контакт між людьми, що може бути досить корисним, особливо коли не бажано масове скупчення людей.

Масштабованість: система може одночасно обслуговувати велику кількість людей, що в свою чергу забезпечує можливість користування чергою без значного перенавантаження системи

Мета створення системи:

Основною метою створення цієї системи є надання користувачам можливості організовувати свій час та ресурси, шляхом додавання їх в електронну чергу. Система створена для забезпечення відкритого, зручного та надійного середовища, де користувачі зможуть створювати електронну чергу(користувач має право сам вказати назву, та автоматично стає її власником), переглядати свої електронні черги, або всі створені на даний момент часу черги, також користувач має можливість додатися в будь-яку чергу, яка існує на цей момент, або редагувати чергу(додавати елемент в задану чергу, видаляти елемент з початку черги, видаляти елемент за значенням, блокувати чергу, тобто зробити так, щоб після блокування було не можливо змінювати вміст черги, або видалити чергу, за умови що вона стала не потрібна)

Для цього застосунок має володіти певними властивостями:

- Зрозумілість використання системи: система повинна мати простий та інтуїтивно зрозумілий користувацький інтерфейс, для якого не потрібні зайві інструкції
- Зберігання та читання даних: система має мати зручне зберігання даних, що переходить від користувача, та можливість ці дані читати для того, щоб виводити певну інформацію користувачу
- Автоматична обробка операцій: система має повністю автоматично та без будь-якого втручання людини автоматично обробляти всі операції, які їй надходять та віддавати відповідні відповіді на конкретно поставлений запит.

1 ВИМОГИ ДО СИСТЕМИ

1.1 Функціональні вимоги до системи

Система має відповідати наступним функціональним вимогам:

- Зареєстрований користувач повинен мати можливість увійти в обліковий запис;
- Зареєстрований користувач повинен мати можливість створювати черги (стає їх хазяїном);
- Зареєстрований користувач може переглядати інформацію про черги (в тому числі свою позицію в цих чергах);
- Хазяїн черги має всі можливості, які є у звичайного зареєстрованого користувача та можливість редагувати черги (видаляти елемент з початку черги та видалення з черги заданого користувача);
- Хазяїн черги має можливість блокування черги(блокує можливість подальшої зміни черги);
- Хазяїн черги має можливість додавання елементу в кінець черги(при умові що черга не заблокована);
-

1.2 Нефункціональні вимоги до системи

Система має відповідати наступним нефункціональним вимогам:

- Система повинна мати відкриту архітектуру;
- Система повинна мати веб-інтерфейс;
- Інтерфейс для гостя має бути зручним та інтуїтивно-зрозумілим;
- Інтерфейс користувача має бути зручним та інтуїтивно-зрозумілим;
- Система повинна бути крос-платформенною.

2 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ

2.1 Діаграма прецедентів

Діаграма прецедентів системи представлена на рис. 2.1.

Акторами є користувачі системи: зареєстрований користувач та хазяїн.

Хазяїну черги доступна уся функціональність, що і зареєстрованому користувачу, а також можливість редагувати чергу (видаляти чергу, блокувати, додавати та видаляти елемент з початку або заданого користувача). Детально усі сценарії використання описані у наступному підрозділі.

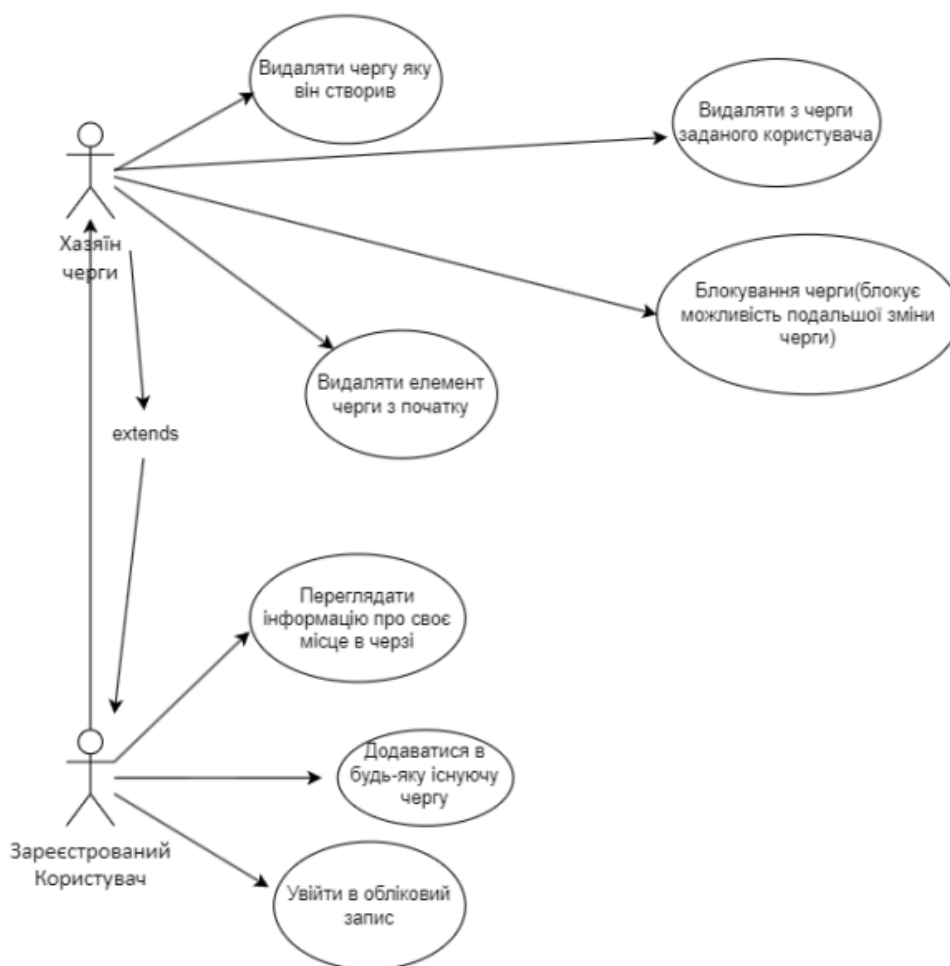


Рисунок 2.1 – Діаграма прецедентів

2.2 Опис сценаріїв використання системи

Детальні описи сценаріїв використання наведено у таблицях 2.1 – 2.8.

В таблиці 2.1 представлений сценарій «Реєстрація користувача».

Таблиця 2.1 – Сценарій використання «Реєстрація користувача»

Назва	Реєстрація користувача
ID	1
Опис	Користувач вводить своє ім'я(username), пароль(password) та підтверджений пароль(confirm password) після цього форма відправляє дані на сервер для подальшої обробки
Актори	Користувач
Вигоди компанії	Сервіси, на яких адміністратор буде вручну реєструвати користувачів будуть втрачати попит
Частота користування	Постійно
Тригери	Перевірка рівності пароля та його підтвердження, перевірка доступності імені користувача
Передумови	Кнопка реєстрації доступна на сторінці RegisterPage на яку можна перейти з LoginPage
Постумови	Користувача буде перенаправлено на сторінку LoginPage, де він зможе скористатися логіном та паролем для входу в систему, які він зазначив при реєстрації
Основний розвиток	Після заповнення всіх полів, користувач натискає кнопку “Register”
Альтернативні розвитки	—

Виняткові ситуації	Якщо паролі не співпадуть - то користувача перенаправить на сторінку з помилкою, де буде можливість перепройти реєстрації.
--------------------	--

В таблиці 2.2 представлений сценарій використання «Авторизація користувача».

Таблиця 2.2 – Сценарій використання «Авторизація користувача»

Назва	Авторизація користувача
ID	2
Опис	Користувач вводить своє ім'я(username), пароль(password) цього форма відправляє дані на сервер для подальшої обробки, та перевірки коректності даних
Актори	Користувач
Вигоди компанії	Забезпечує безпеку облікових даних користувача, та персоналізацію доступу до даних
Частота користування	Часто
Тригери	Користувач хоче отримати доступ до свого облікового запису
Передумови	У користувача є коректні дані для входу у систему(логін та пароль)
Постумови	Користувача буде перенаправлено на головну сторінку, де він отримує доступ до всіх його даних
Основний розвиток	Користувач відкриває сторінку з авторизацією, де він вводить всі необхідні дані для входу, після цього дані відправляються на сервер, де система перевіряє коректність введених даних, та у разі коректності даних, надає доступ до системи

Альтернативні розвитки	Якщо користувач вводить не коректні дані, система повідомляє йому про це, та просить ввести дані ще раз
Виняткові ситуації	—

В таблиці 2.3 представлений сценарій використання «Вихід з облікового запису».

Таблиця 2.3 – Сценарій використання «Вихід з облікового запису»

Назва	Вихід з облікового запису
ID	3
Опис	Користувач виконує, вихід з облікового запису
Актори	Користувач
Вигоди компанії	Забезпечує безпеку облікових даних користувача, та закриває доступ до функціональності облікового запису
Частота користування	Залежить від користувача
Тригери	Користувач хоче завершити сесію, та вийти зі свого облікового запису
Передумови	У користувача є коректні дані для входу у систему(логін та пароль), та він бажає з нього вийти
Постумови	Користувач завершує сеанс роботи з обліковим записом, та виходить з нього
Основний розвиток	Користувач знаходячись у системі, бажає завершити сеанс роботи з обліковим записом, та вийти з свого облікового запису, користувач

	натискає на кнопку Logout, після цього завершується сесія, та користувач перенаправляє на сторінку з авторизацією
Альтернативні розвитки	—
Виняткові ситуації	—

В таблиці 2.4 представлений сценарій використання «Створення черги».

Таблиця 2.4 – Сценарій використання «Створення черги»

Назва	Створення черги
ID	4
Опис	Користувач, який авторизований в системі, створює чергу
Актори	Користувач
Вигоди компанії	Забезпечує створення нових електронних черг, для подальшого додавання інформації
Частота користування	Залежить від користувача, може бути досить частою, або навпаки
Тригери	Користувач хоче створити свою електронну чергу(стати її власником)
Передумови	У користувача є коректні дані для входу у систему(логін та пароль), та він бажає створити свою чергу
Постумови	Користувач отримує нову електронну чергу, та стає її власником

Основний розвиток	Користувач знаходячись у системі, бажає створити свою електронну чергу для подальшого додавання туди елементів, або інших користувачів
Альтернативні розвитки	—
Виняткові ситуації	Якщо черга вже існує, то система повідомляє про те, що не можливо створити її ще раз, після цього система пропонує надати іншу назву черги та створити її

В таблиці 2.5 представлений сценарій використання «Редагування інформації про сеанси».

Таблиця 2.5 – Сценарій використання «Перегляд черг користувача»

Назва	Перегляд черг користувача
ID	5
Опис	Перегляд черг користувача
Актори	Власник
Вигоди компанії	Покращення контролю за чергами, оскільки їх можна переглядати
Частота користування	Залежить від користувача, може бути досить частою, або навпаки
Тригери	Виникнення необхідності переглянути дані черги
Передумови	Користувач є власником черги, та хоче її переглянути

Постумови	Користувача буде перенаправлено на сторінку, де він зможе переглянути інформацію про вміст черги, тобто все що в ній записано в даний момент часу
Основний розвиток	Після натискання кнопки View, користувача перенаправляє на відповідну сторінку з даними про чергу
Альтернативні розвитки	—
Виняткові ситуації	Якщо черга пуста то користувачу буде надано повідомлення, про те, що черга пуста та її не можливо переглянути

В таблиці 2.6 представлений сценарій використання «Перегляд усіх черг».

Таблиця 2.6 – Сценарій використання «Перегляд усіх черг»

Назва	Перегляд усіх черг
ID	6
Опис	Перегляд усіх черг
Актори	Користувач
Вигоди компанії	Покращення контролю за чергами, оскільки їх можна переглядати
Частота користування	Залежить від користувача, може бути досить частою, або навпаки
Тригери	Виникнення необхідності переглянути дані черги
Передумови	Користувач авторизований у системі та хоче переглянути черги які вже існують в системі

Постумови	Користувача буде перенаправлено на сторінку, де він зможе переглянути інформацію про вміст черги, тобто все що в ній записано в даний момент часу
Основний розвиток	Після натискання кнопки View, користувача перенаправляє на відповідну сторінку з даними про чергу
Альтернативні розвитки	—
Виняткові ситуації	Якщо черга пуста то користувачу буде надано повідомлення, про те, що черга пуста та її не можливо переглянути

В таблиці 2.7 представлений сценарій використання «Додавання користувача в чергу самостійно».

Таблиця 2.7 – Сценарій використання «Додавання користувача в чергу самостійно»

Назва	Додавання користувача в чергу самостійно
ID	7
Опис	Користувач, який авторизований в системі, хоче додатися в кінець черги
Актори	Користувач
Вигоди компанії	Можливість зайняти місце в в кінці черги
Частота користування	Залежить від користувача, може бути досить частою, або навпаки
Тригери	Виникнення потреби знаходитися в черзі

Передумови	Користувач авторизований у системі та хоче додатися в кінець черги
Постумови	Авторизований користувач, додається в доступну на даний момент часу чергу
Основний розвиток	Після вибору черги та натискання кнопки додатися, користувача буде перенаправлено на сторінку, де система повідомить про те що користувачу вдалося або не вдалося додатися в чергу
Альтернативні розвитку	—
Виняткові ситуації	Якщо черга заблокована, то в неї не можливо додаватися, і система повідомить про це відповідним повідомленням

В таблиці 2.8 представлений сценарій використання «Редагування черги».

Таблиця 2.8 – Сценарій використання «Редагування черги»

Назва	Редагування черги
ID	8
Опис	Користувач, який авторизований в системі, та є власником черги хоче її змінити
Актори	Власник
Вигоди компанії	Дає можливість користуватися чергою та змінювати її вміст за потреби
Частота користування	Залежить від власника, може бути досить частою, або навпаки

Тригери	Власник хоче змінити дані в черзі, або заблокувати її (та/або видалити чергу)
Передумови	Користувач авторизований у системі та є власником черги, і хоче змінити дані черги
Постумови	Власник черги хоче змінити свою чергу(додати елемент в кінець черги, видалити елемент з кінця черги, видалення елементу за назвою, блокування та/або видалення черги)
Основний розвиток	Після вибору черги та натискання кнопки додати елемент, він додається в кінець черги, після натискання видалення з черги елементу за назвою, елемент з цією назвою видаляється з черги, після натискання кнопки блокування черги, черга не може змінювати свій вміст(можливо лише після її розблокування), після натискання кнопки видалення черги, черга видаляється разом з усіма даними що в ній були
Альтернативні розвитки	—
Виняткові ситуації	Якщо черга пуста та власним намагається видалити з неї елемент то система повідомить про те, що неможливо видалити елемент, оскільки черга пуста

3 АРХІТЕКТУРА СИСТЕМИ

Загальна архітектура системи наведена на рис. 3.1

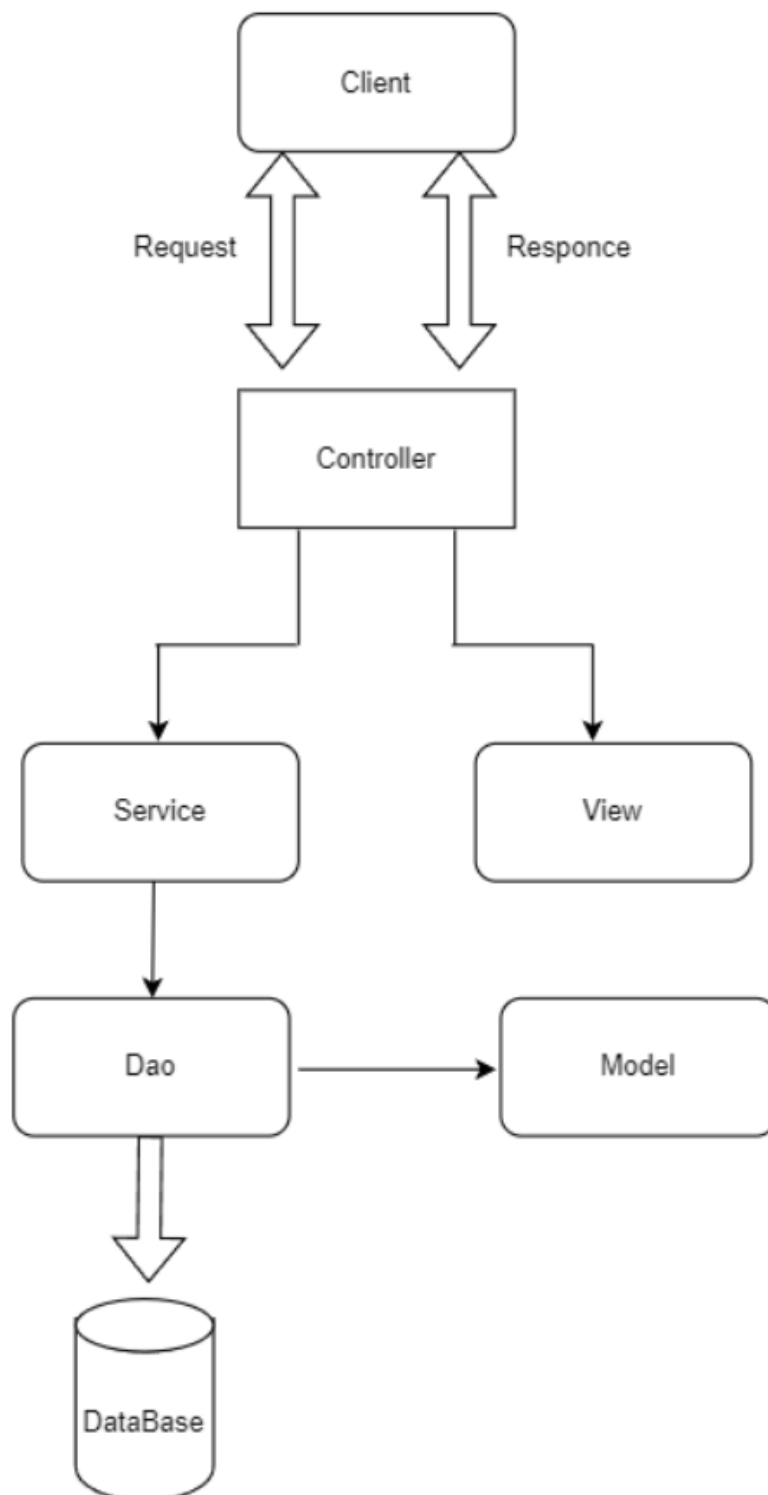


Рисунок 3.1 – Загальна архітектура системи

Система складається з наступних елементів:

1. графічний інтерфейс;
2. серверна частина;
3. база даних.

Графічний інтерфейс необхідний для взаємодії з користувачем. HTTP запит надходить до серверної частини, яка обробляє запит та повертає відповідь, саме на серверній частині відбувається основна логіка, дані які прийшли з графічної частини валідуються та конвертуються за допомогою `nameValidatorService`,

За потреби серверна частина формує запит до бази даних, через відповідний сервіс для роботи з базою даних, у свою чергу база даних може зберігати та повертати дані

До серверної частини належать наступні елементи:

1. контролер;
2. модель та вигляд;
3. сервіс;
4. репозиторій.

На сервлети (контролер) надходять дані з графічного інтерфейсу, вони валідуються за допомогою `namevalidatorservice` та за потреби вносяться до бази даних з використанням відповідного сервісу, у сервлеті відбувається вся необхідна бізнес логіка запиту та за потреби перенаправляє на сервлет, який відповідальний за відображення дани, і він в свою чергу перенаправляє на сторінку з відображенням даних.

4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

4.1 Загальна структура проекту

Загальна структура проекту представлена на рис.4.1

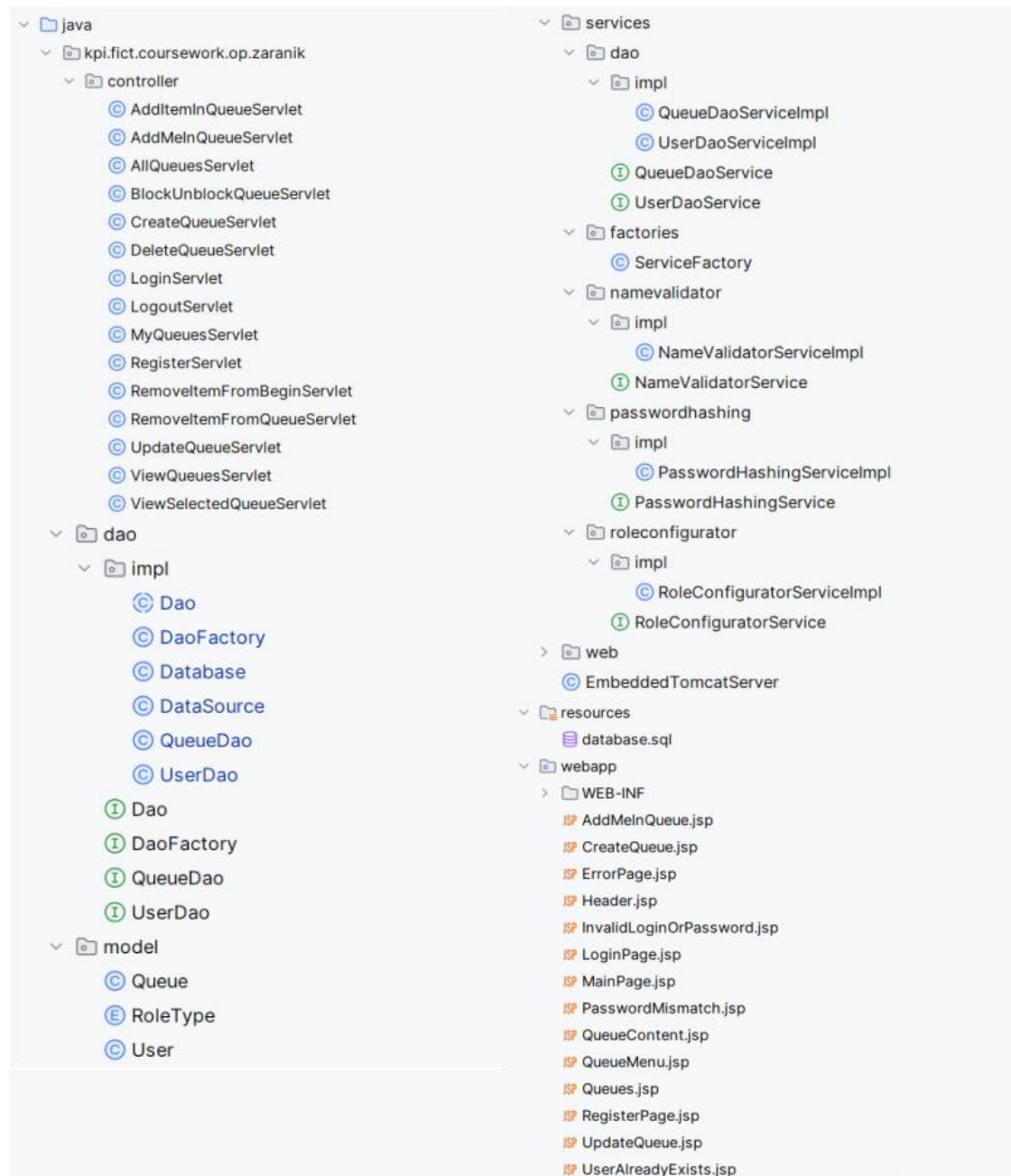


Рисунок 4.1 – Загальна структура проекту

Проект складається з веб-ресурсів та вихідного коду, який в свою чергу розділяються на компоненти бізнес-логіки, компоненти рівня доступу до даних, та веб-компоненти, далі наведено детальна схема.

4.2 Компоненти рівня доступу даних

Основні сутності та інтерфейси рівня доступу до даних наведені на рис. 4.2

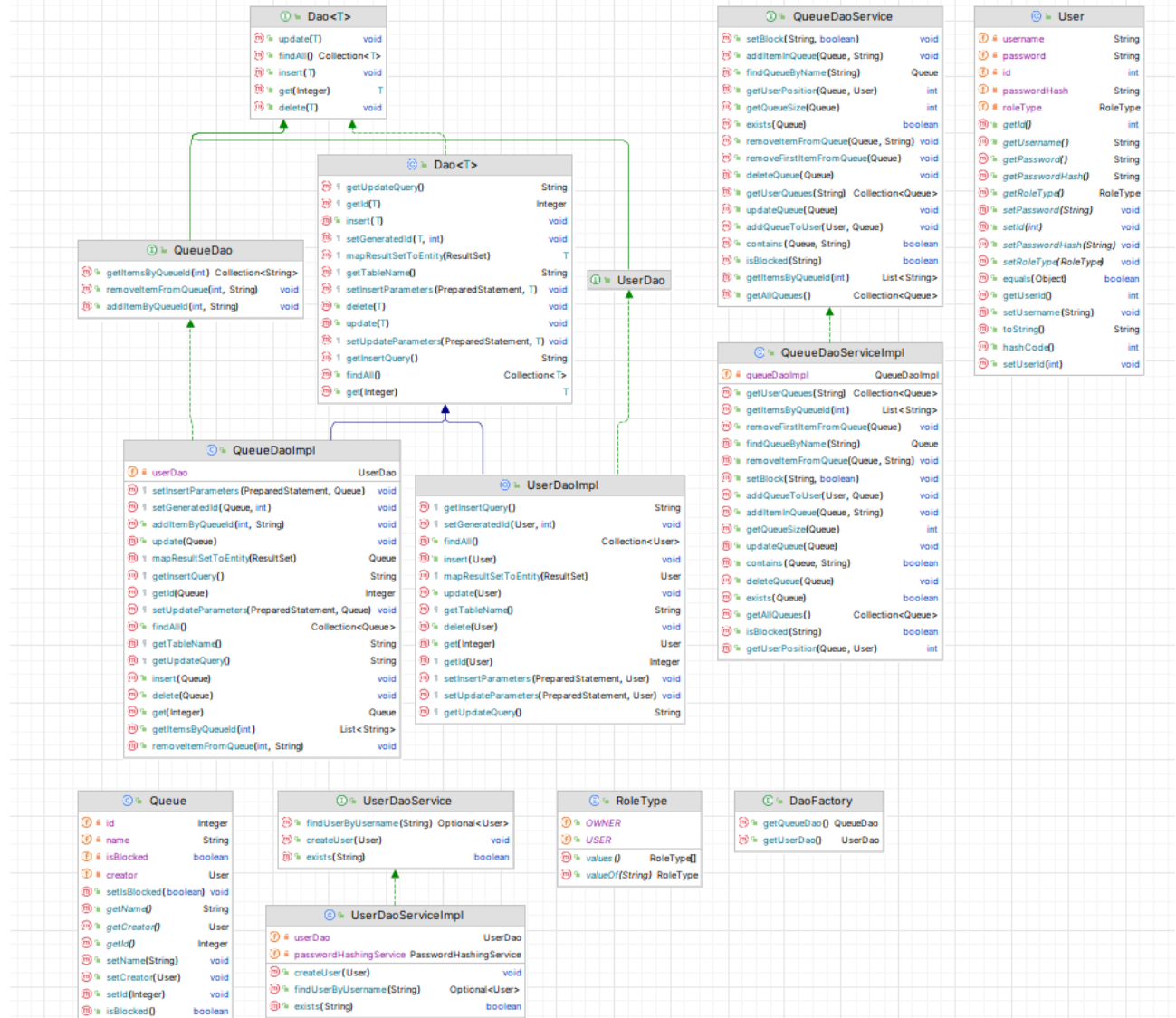


Рисунок 4.2 – Основні сутності та інтерфейси рівня доступу до даних

а) Клас User представляє модель користувача, який може створювати черги, переглядати та редагувати їх, тощо.

Встановлює відношення один до багатьох, з класом Queue, оскільки користувач має можливість створити декілька електронних черг.

Містить функціонал для отримання імені користувача(username), пароля(password), та типу ролі користувача в системі (roleType).

б) Клас Queue Представляє модель електронної черги, яка є безпосередньо чергою, в яку користувач буде згодом додавати та видаляти елементи

Встановлює відношення багато до одного, з класом “User”, оскільки в одного користувача може бути багато черг.

- с) Перерахування RoleType представляє модель за допомогою якої визначаються ролі користувачів для подальшого розподілення прав доступу до черг, в даному випадку можуть бути два типи ролі: «USER», «OWNER»
- d) Інтерфейс Dao представляє інтерфейс для роботи з базою даних, який представляє собою абстрактну базу даних, яка володіє операціями insert, delete, get, findAll, update (CRUD).
- е) Інтерфейс UserDao представляє інтерфейс для роботи з базою даних, у якій зберігається інформація про користувачів, цей інтерфейс наслідує базовий інтерфейс Dao, та отримує всі його методи, для роботи з базою даних та подальшого зберігання користувачів.
- f) Інтерфейс QueueDao представляє інтерфейс для роботи з базою даних, у якій зберігається інформація про черги, цей інтерфейс наслідує базовий інтерфейс Dao, та отримує всі його методи та реалізує додаткові для роботи з елементами черги.
- g) Клас Dao представляє абстрактну базу даних, яку потім будуть наслідувати та перевизначати безпосередньо UserDao та QueueDao.

Даний інтерфейс UserRepository оголошує наступні методи:

- метод getUpdateQuery використовується для отримання запиту для оновлення даних.
- метод getId використовується для отримання унікального id для сутності.
- метод insert використовується для вставлення сутності в базу даних.
- метод setGeneratedId використовується для встановлення згенерованого id сутності.
- метод mapResultSetToEntity використовується для того щоб зібрати об'єкт з бази даних через конструктор та повернути новий об'єкт сутності.
- метод getTableName використовується для повернення ім'я таблиці, з якою взаємодіє база даних.

- метод `setInsertParameters` використовується для встановлення параметрів для запиту на вставлення даних.
- метод `delete` використовується для видалення сутності з бази даних.
- метод `update` використовується для оновлення даних сутності з бази даних.
- метод `setUpdateParameters` використовується для встановлення параметрів, які будуть використані для запиту на оновлення даних.
- метод `getInsertQuery` використовується для отримання запиту на вставлення даних.
- метод `findAll` використовується для того щоб знайти всю інформацію яка знаходиться в базі даних.
- метод `get` використовується для того щоб повернути всю інформацію про сутність знаходиться в базі даних, за її унікальним `id`.

h) Клас `UserDaoImpl` конкретна реалізація абстрактної бази даних `UserDao`, яка реалізує всі методи для роботи за базою даних, та подальшого зберігання користувачів, наслідую абстрактну базу даних та перевизначає методи для отримання запитів на видалення, вставлення та оновлення даних.

i) Клас `UserDaoImpl` конкретна реалізація абстрактної бази даних `UserDao`, яка реалізує всі методи для роботи за базою даних, та подальшого зберігання користувачів, наслідую абстрактну базу даних та перевизначає методи для отримання запитів на видалення, вставлення та оновлення даних.

Даний клас `UserDaoImpl` наслідує всі методи `Dao` та оголошує наступні методи:

- метод `addItemByQueueId` використовується для додавання елемента в чергу за її `id`.
- метод `getItemsByQueueId` використовується для отримання всіх елементів черги за її `id`.
- метод `removeItemFromQueue` використовується для видалення заданого елемента з черги.

j) Інтерфейс `UserDaoService` є інтерфейсом сервіса для доступу до функціональності бази даних та управління ними.

Даний інтерфейс `UserDaoService` оголошує наступні методи:

- метод `findUserByUsername` використовується для отримання користувача за його логіном.
- метод `createUser` використовується для створення та занесення до бази даних користувача за його логіном та паролем.
- метод `exists` використовується для перевірки наявності користувача в базі даних.

k) Інтерфейс `QueueDaoService` є інтерфейсом сервіса для доступу до функціональності бази даних та управління ними.

Даний інтерфейс `UserDaoService` оголошує наступні методи:

- метод `setBlock` використовується для блокування черги.
- метод `addItemInQueue` використовується для додавання елементу в чергу.
- метод `findQueueByName` використовується пошуку черги за її назвою.
- метод `getUserPosition` використовується для знаходження позиції користувача в черзі.
- метод `getQueueSize` використовується для визначення кількості елементів черги.
- метод `exists` використовується для перевірки наявності черги.
- метод `removeItemFromQueue` використовується для видалення елементу з черги.
- метод `removeFirstItemFromQueue` використовується для видалення першого елементу з черги.
- метод `deleteQueue` використовується для видалення черги.
- метод `getUserQueues` використовується для отримання всіх черг користувача.
- метод `updateQueue` використовується для оновлення інформації про чергу.

- метод `addQueueToUser` використовується для додавання черги користувачу.
- метод `contains` використовується для перевірки вмісту заданого елементу в черзі.
- метод `IsBlocked` використовується для перевірки стану блокування черги.
- метод `getItemsByQueueId` використовується для отримання елементів черги.
- метод `getAllQueues` використовується для отримання всіх черг які існують.

1) Інтерфейс `QueueDaoServiceImpl` є класом для реалізації сервіса `QueueDaoService`, який реалізовує всі методи, які в ньому описані.

4.3 Компоненти рівня бізнес-логіки

Основні сутності та інтерфейси рівня бізнес-логіки на рис. 4.3

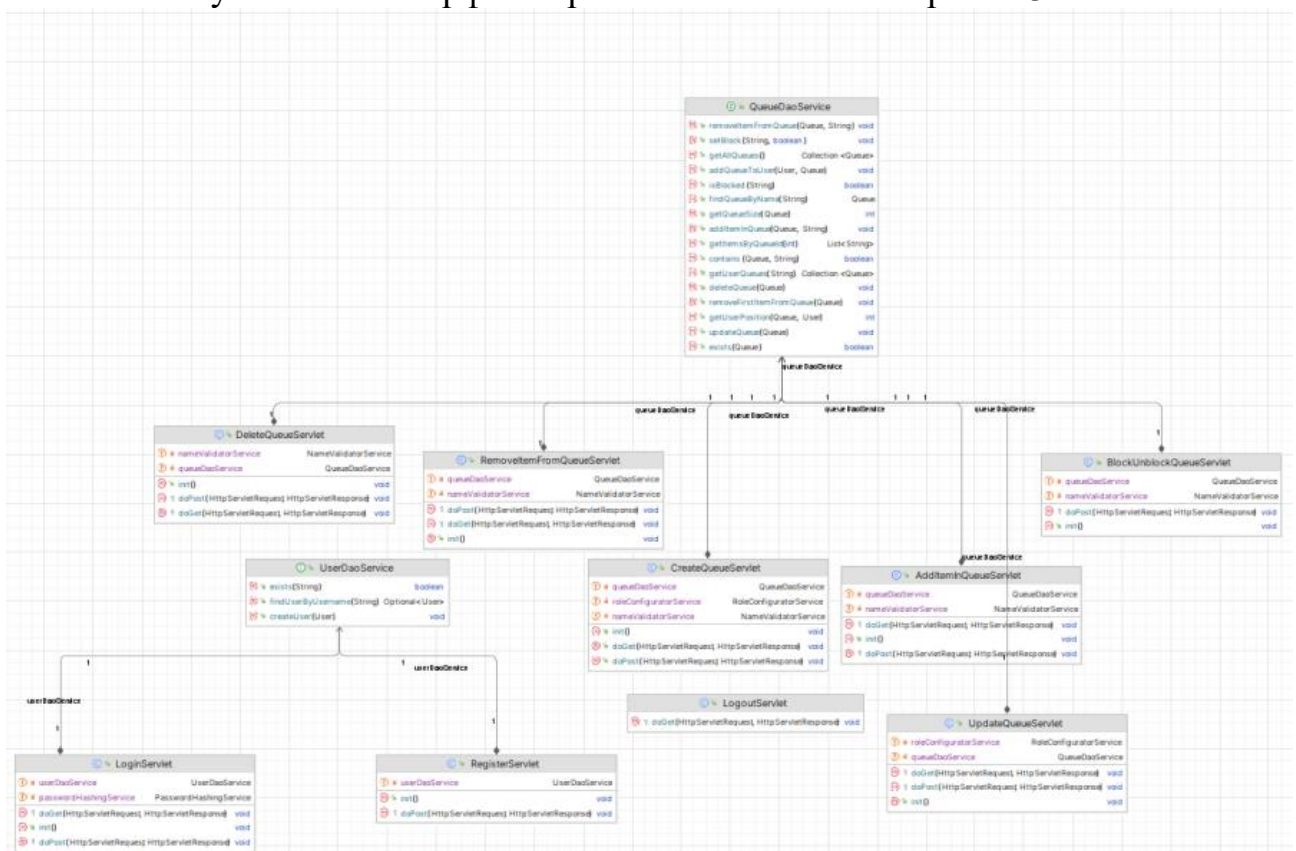


Рисунок 4.3 – Основні сутності рівня бізнес-логіки

а) Клас `LoginServlet` відповідає за контролювання входу в систему користувача та перевірку коректності вхідних даних(логіну та паролю), через

параметри request, отримаємо “username” та “password”, за допомогою методу “findUserByUserName” в сервісі userDaoService отримаємо користувача, перевіривши що користувач існує і нам не потрібно видавати помилку ми за допомогою методу “checkPasswords” перевіряємо чи співпадають пароль з бази даних(його хеш) та також хеш пароля який вводить користувач, якщо це так то система пропускає користувача, перенаправленням на головну сторінку “MainPage.jsp”.

b) Клас LogoutServlet відповідає за контролювання виходу з систему шляхом закриття сесії методом session.invalidate.

с) Клас RegisterServlet Відповідає за реєстрацію користувача в систем, шляхом отримання даних з request(такі самі як і LoginServlet.java), та перевірки чи такий користувач вже існує та чи співпадають пароль та його підтвердження, у разі успішного виконання всіх умов, сервлет створює нового користувача.

d) Клас CreateQueueServlet Відповідає за створення нової черги, в методі “doGet” перевіряє чи користувач авторизований за допомогою взяття користувача з сесії, та у методі “doPost” основна логіка роботи сервлету: з request береться назва черги яку необхідно створити, та за допомогою сервісу “nameValidatorService” перевіряється коректність імені, якщо воно не коректне користувач отримає відповідне повідомлення, потім перевіриться чи черга за заданим іменем вже існує через відповідний метод “queueDaoService” та у разі її відсутності вона створиться, за допомогою “roleConfiguratorService” настраюється роль користувачу (власник черги), і за допомогою методу “addQueueToUser” додається до відповідного користувача.

е) Клас DeleteQueueServlet Відповідає за видалення черги, в методі “doGet” перевіряє чи користувач авторизований за допомогою взяття користувача з сесії, та у методі “doPost” основна логіка роботи сервлету: з request береться назва черги яку необхідно видалити, та за допомогою сервісу “namevalidatorservice” перевіряється коректність імені, якщо воно не

коректне користувач отримає відповідне повідомлення, потім перевіриться чи черга за заданим іменем існує через відповідний метод “queueDaoService” та у разі її існування вона видаляється за допомогою “deleteQueue”.

f) Клас UpdateQueueServlet відповідає за видалення черги, в методі “doGet” перевіряє чи користувач авторизований за допомогою взяття користувача з сесії, та у методі “doPost” основна логіка роботи сервлету: з request береться назва черги яку необхідно видалити, та за допомогою сервісу “nameValidatorService” перевіряється коректність імені, якщо воно не коректне користувач отримає відповідне повідомлення, потім перевіриться чи черга за заданим іменем існує через відповідний метод “queueDaoService” та у разі її існування вона видаляється за допомогою “deleteQueue”.

g) Клас RoleConfiguratorService відповідає за налаштування ролі між користувачем та чергою.

Даний інтерфейс UserDaoService оголошує наступні методи:

- метод getConfiguration перевіряє який статус має задана черга для заданого користувача(чи є він звичайним користувачем, або власником цієї черги), для цього він робить запит у базу даних та знаходить чергу за іменем та повертає автора цієї черги через метод getCreator, та перевіряє чи заданий користувач власником цієї черги, і якщо так то повертає “RoleType.OWNER”, а в усіх інших випадках “RoleType.USER”.
- метод configureRole використовується для налаштування ролі між користувачем та заданою чергою.

4.4 Компоненти рівня інтерфейсу користувача

Основні сутності рівня інтерфейсу користувача наведені на рис. 4.4

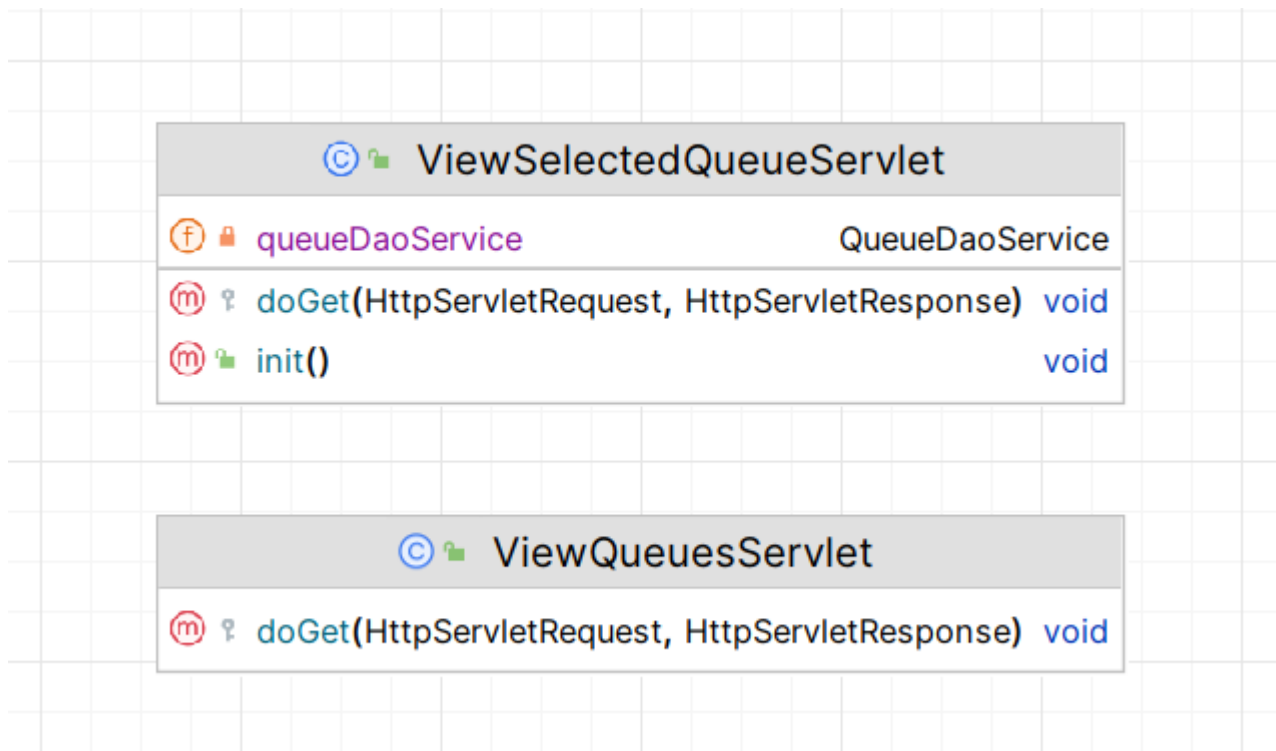


Рисунок 4.4 – Основні сутності рівня інтерфейсу користувача

- Контролер ViewQueuesServlet відповідає за перенаправлення на сторінку Queues.jsp, та подальшого відображення.
- Контролер ViewSelectedQueueServlet відповідає за відображення заданої черги шляхом запиту до відповідного сервісу задля отримання заданої черги та позиції користувача в цій черзі.

LoginPage.jsp представляє собою сторінку на якій користувач може ввести логін та пароль, який він заздалегідь отримав під час реєстрації, та у разі коректності введених даних користувача перенаправляє на головну сторінку.

RegisterPage.jsp представляє собою сторінку для реєстрації в системі. На ній знаходиться форма з ім'ям користувача (логіном), паролем та його підтвердження, та у разі не існування такого самого користувача і співпадіння пароля і його підтвердження користувача перенаправляє на сторінку з логіном, де він може ввести дані для входу.

MainPage.jsp представляє собою головну сторінку на якій користувач може керувати чергами.

CreateQueue.jsp представляє собою сторінку на якій користувач може створити електронну чергу шляхом введення назви, та у разі не існування черги вона створиться та користувача перенаправить на головну сторінку.

UpdateQueue.jsp представляє собою сторінку на якій користувач може оновити дані про чергу, тобто додати в неї елемент, видалити його, або заблокувати чергу, також є можливість видалення черги та її вмісту.

AddMeInQueue.jsp представляє собою сторінку на якій користувач може самостійно додатися в чергу шляхом вибору будь-якої існуючої на даний момент черги та натисканням кнопки add.

QueueMenu.jsp представляє собою сторінку на якій користувач може побачити основне меню з чергами для подальшої можливості їх перегляду.

QueueContent.jsp представляє собою сторінку на якій виводиться вміст черги, тобто всі елементи що в ній є там позицію поточного користувача в цій черзі.

5 ІНСТРУКЦІЯ КОРИСТУВАЧА

На рис. 5.1 наведена головна сторінка, в якій можна управляти чергами

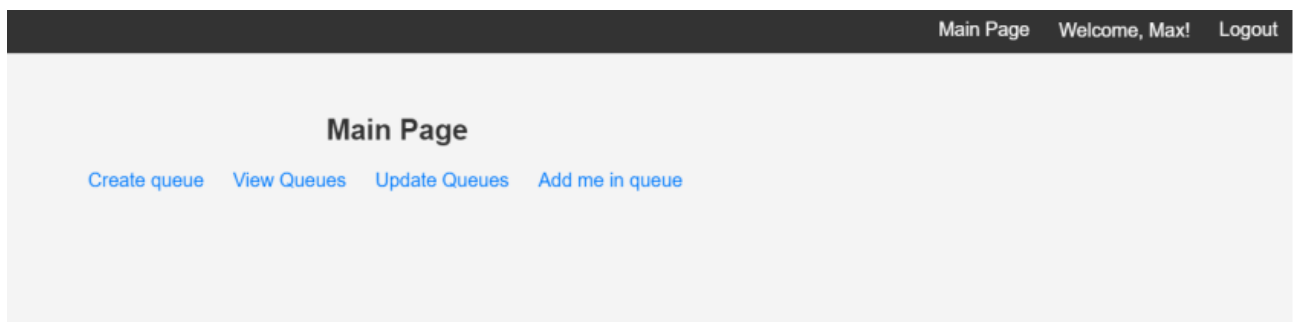
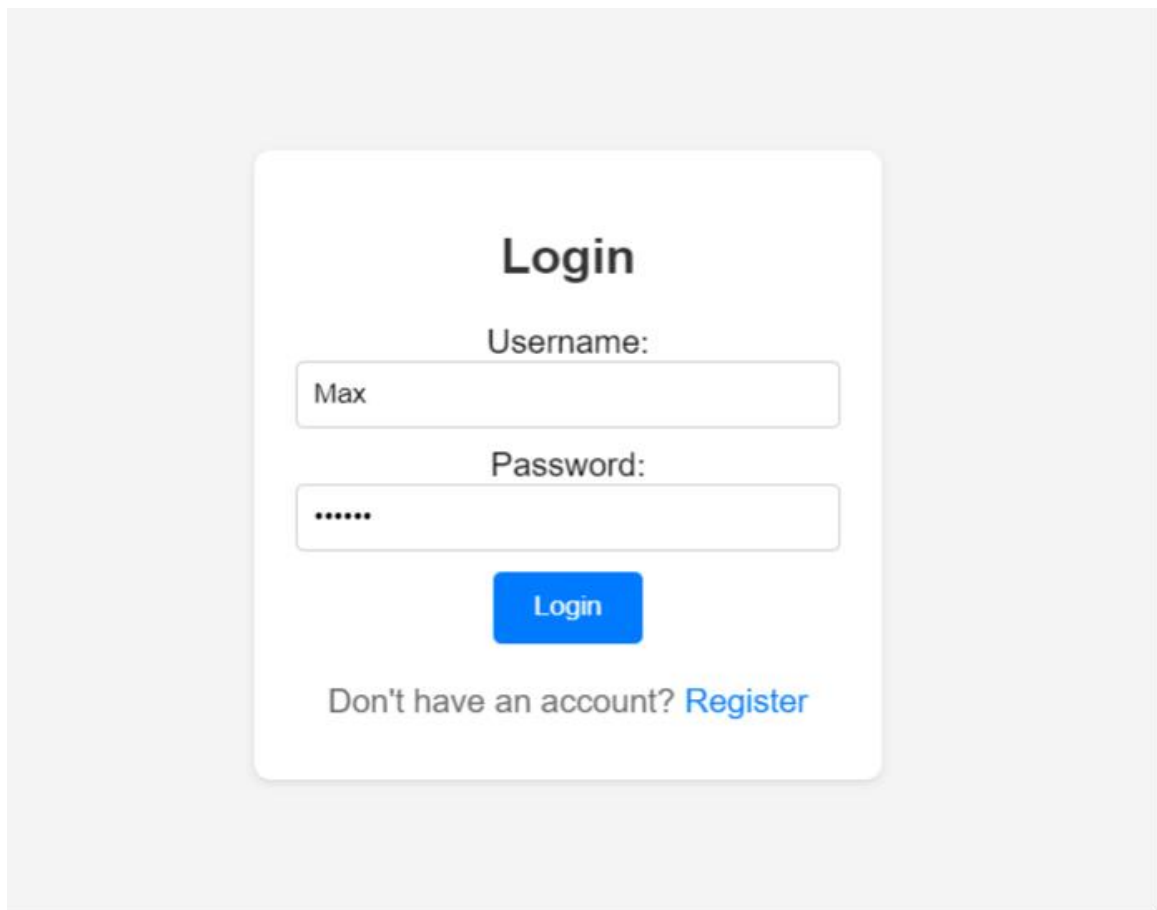
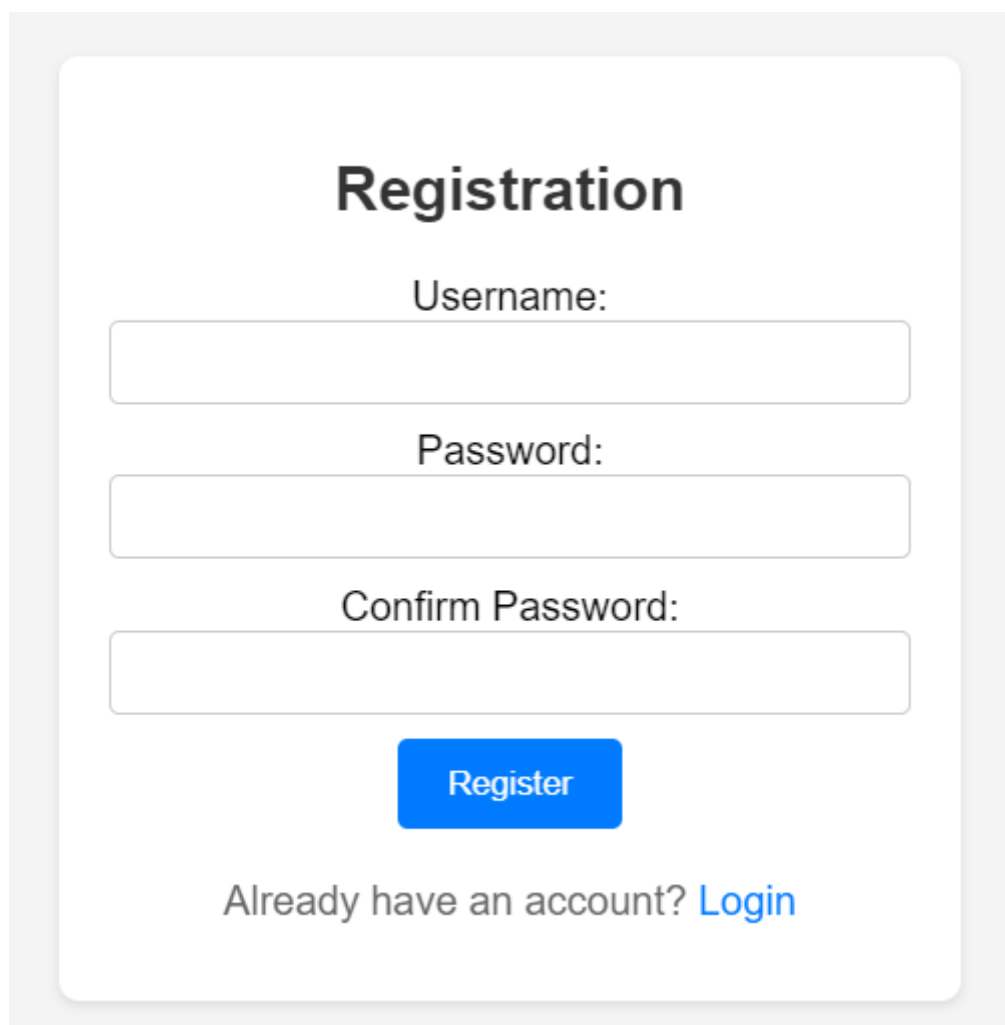


Рис 5.1 Головна сторінка системи



The image shows a login form centered on a light gray background. The form is a white rounded rectangle. At the top, the word "Login" is written in a bold, black, sans-serif font. Below it, the label "Username:" is followed by a text input field containing the text "Max". Underneath, the label "Password:" is followed by a password input field filled with seven dots. A blue rectangular button with the word "Login" in white text is positioned below the password field. At the bottom of the form, the text "Don't have an account?" is followed by a blue, underlined link labeled "Register".

Рис 5.2 Сторінка з входом до системи



Registration

Username:

Password:

Confirm Password:

Register

Already have an account? [Login](#)

Рис 5.3 Сторінка реєстрації

На рис. 5.3 зображена сторінка для реєстрації в системі. Для реєстрації потрібно ввести ім'я, пароль та його підтвердження, та натиснути на кнопку Register

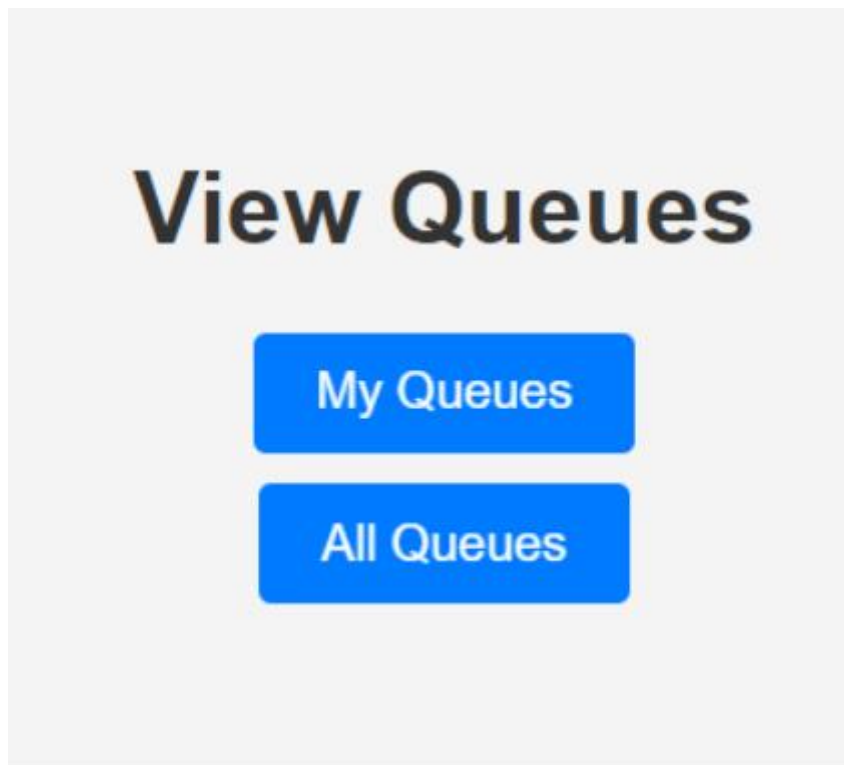


Рис 5.4 Сторінка з вибором перегляду черг

На рис. 5.4 зображена панель з вибором варіанту перегляду черг.

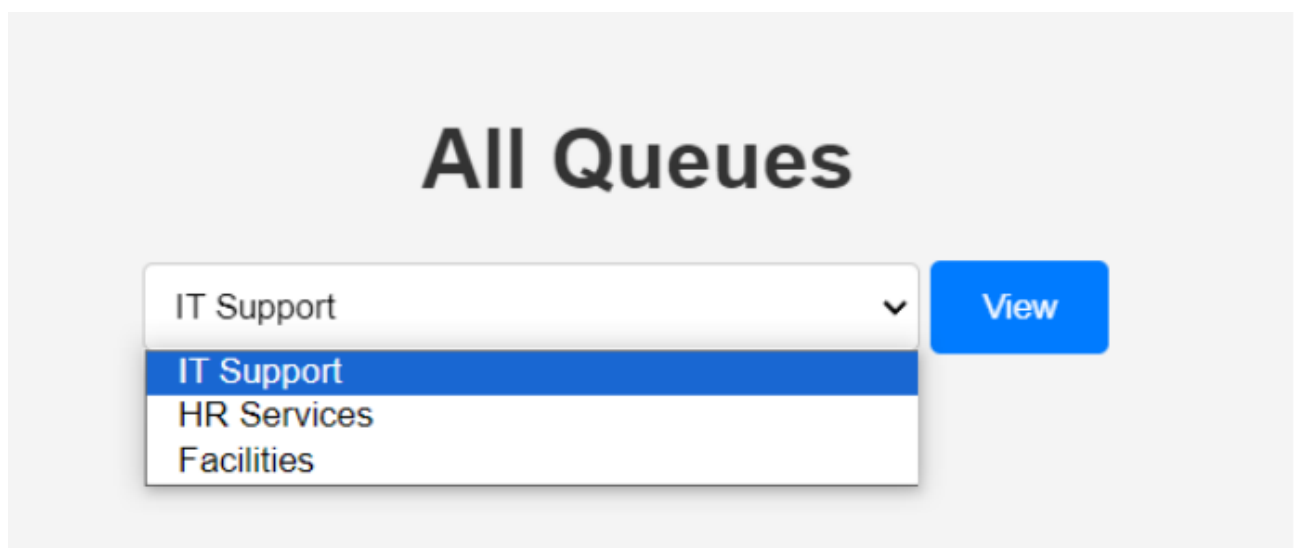
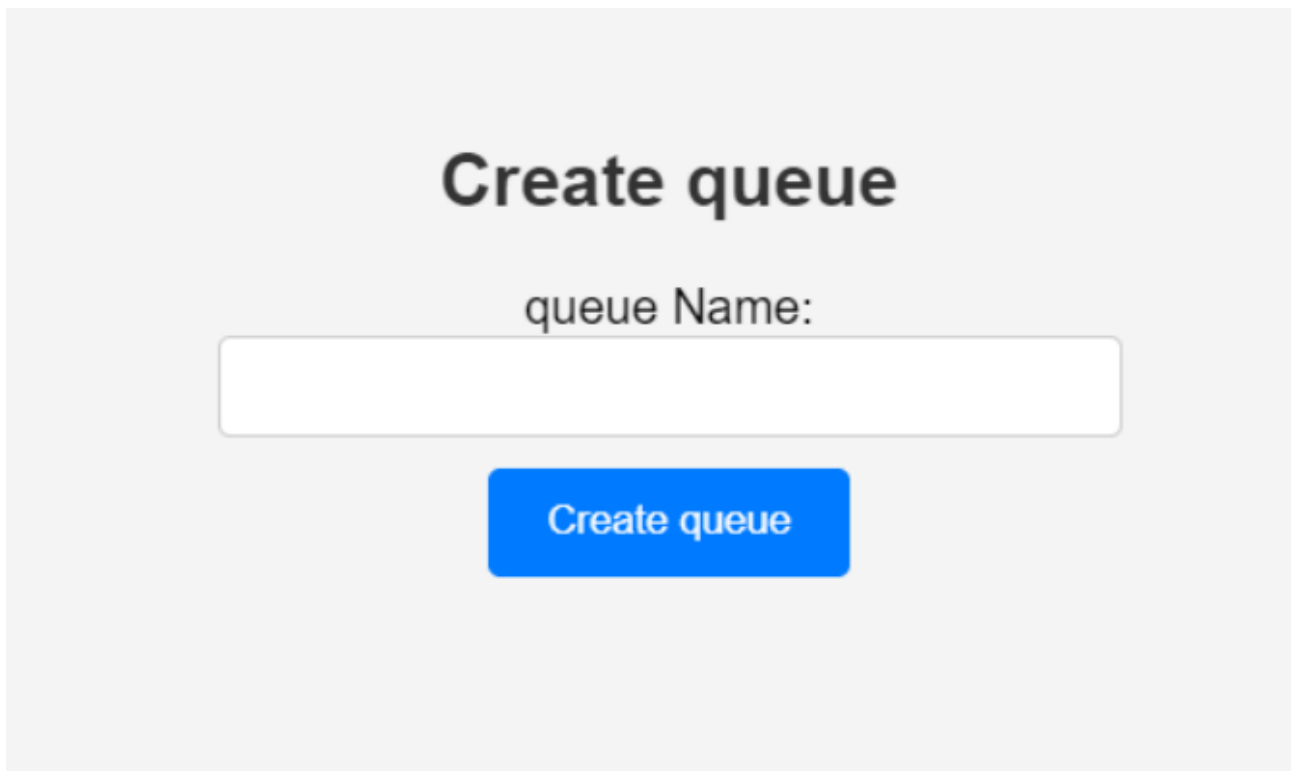


Рис 5.5 Сторінка для створення нової черги

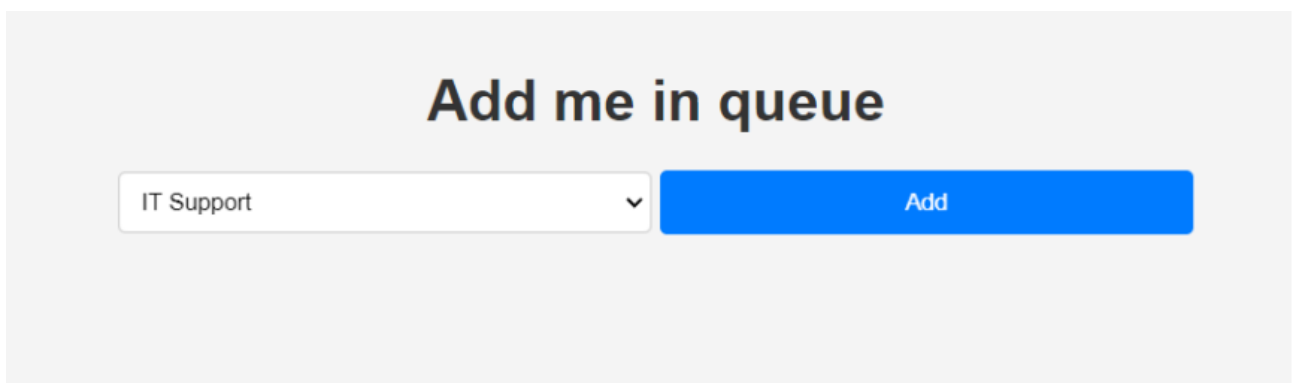
На рис. 5.5 зображена форма після натискання на будь-яку кнопку, користувач побачить меню де він може вибрати задану чергу для переглядання, та переглянути її шляхом натискання на кнопку "View"



The screenshot shows a web form titled "Create queue" in a large, bold, dark font. Below the title, the text "queue Name:" is displayed in a smaller, regular font. Underneath this label is a wide, empty rectangular text input field with a thin gray border. Centered below the input field is a solid blue rectangular button with the text "Create queue" in white, sans-serif font.

Рис 5.6 Сторінка перегляду інформації про сеанс

На рис. 5.6 форма для створення нової черги, яка містить випадаючий список та кнопку Create queue, після натискання якої створюється черга. Якщо адміністратор заповнив всі поля, він може натиснути на кнопку..



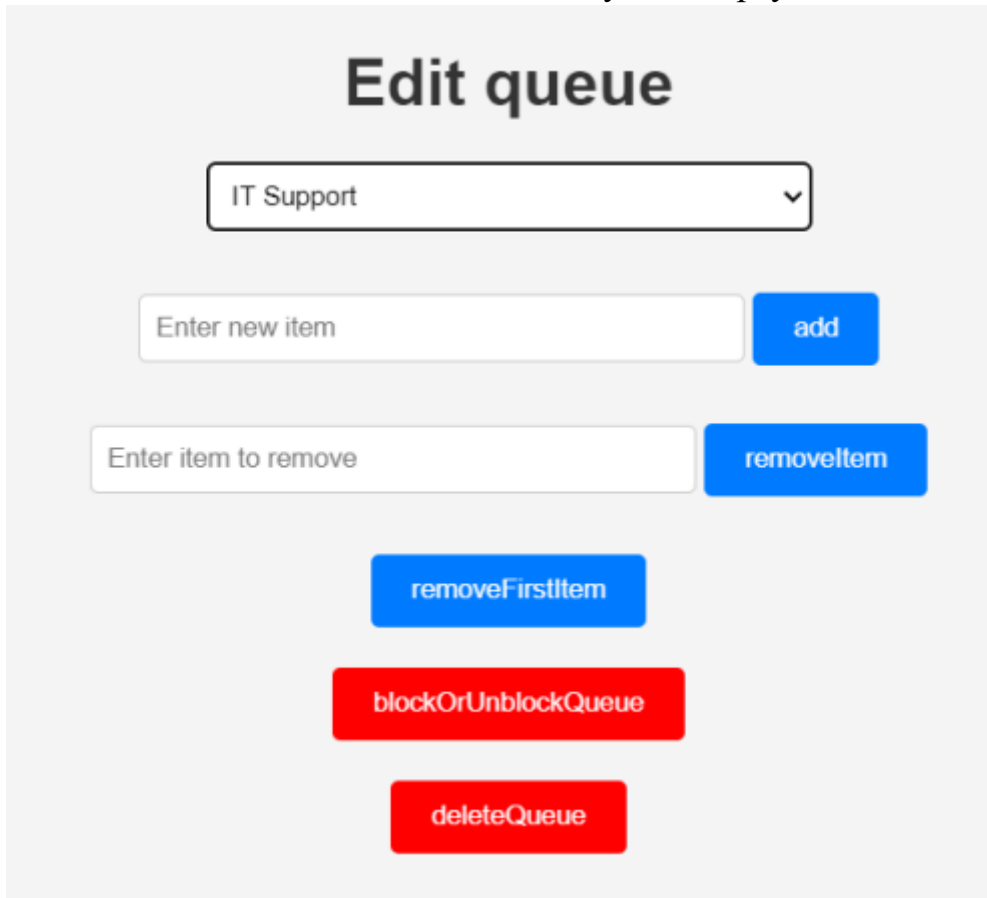
The screenshot shows a web form titled "Add me in queue" in a large, bold, dark font. Below the title, there is a dropdown menu with the text "IT Support" and a small downward-pointing arrow on the right. To the right of the dropdown menu is a solid blue rectangular button with the text "Add" in white, sans-serif font.

Рис 5.7 Сторінка для додавання в чергу самостійно

На рис 5.7 наведена сторінка з можливістю додавання користувача в чергу самостійно, в результаті натискання на кнопку Add, та вибору відповідної черги.

Рис 5.8 Сторінка редагування черги

На рис. 5.8 наведена сторінка для редагування черги, в якій є можливість додавати елементи, видаляти їх, блокувати чергу та видаляти її.



The screenshot shows a web interface titled "Edit queue". At the top, there is a dropdown menu with the text "IT Support" and a downward arrow. Below this, there are two input fields. The first is labeled "Enter new item" and is followed by a blue button labeled "add". The second is labeled "Enter item to remove" and is followed by a blue button labeled "removeItem". Below these, there are three more buttons: a blue button labeled "removeFirstItem", a red button labeled "blockOrUnblockQueue", and a red button labeled "deleteQueue".

Рис 5.9 Сторінка перегляду інформації про квиток

На рис. 5.9 зображена сторінка перегляду інформації про квиток. Тут користувачі можуть переглянути актуальну інформацію про квиток.

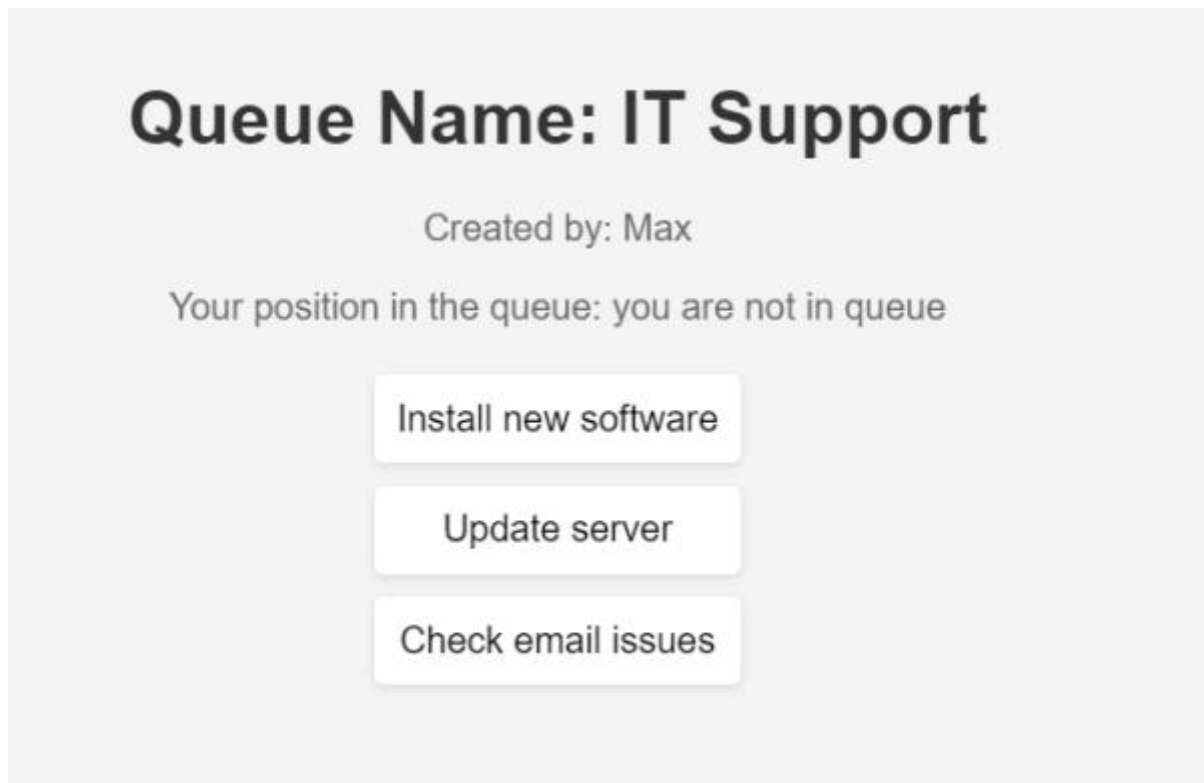


Рис 5.10 Сторінка з інформацією про чергу

На рис. 5.10 зображена сторінка у якій написано весь зміст черги, включаючи позицію користувача в цій черзі.

ВИСНОВКИ

В результаті виконання цієї курсової роботи я розробив додаток який реалізує функціонал електронної черги, в якому користувачі мають право зареєструватися та входити в систему під свої логіном та паролем який вони вказали під час реєстрації. Спочатку треба було сформулювати та чітко визначити функціональні та не функціональні вимоги до системи, потім було обрано перелік технологій якими я користувався в цій роботі.

В якості основної мови програмування мною була вибрана Java, оскільки вона має досить потужні можливості в розробці веб-застосунків, в якості середовища розробки в використовував IntelliJ IDEA Ultimate Edition, оскільки це одна з найзручніших та популярних IDE, в якості бази даних мною була вибрана мова PostgreSQL, за її відносну простоту та ефективність, для роботи з підключеннями до бази даних так звані “JDBC” я використовував сторонню бібліотеку під назвою “HikaryCP”, яка і керувала конекшн пулом, оскільки вона досить не складна в використанні та налаштуванні, в якості сервера мною був використаний “embedded tomcat server”, всі ці бібліотеки підключаються за допомогою збірника проекту “Maven”, та безпосередньо додавання залежностей(dependencies).

Оскільки дана система має мати графічний інтерфейс, мною було вирішено використовувати мови HTML та CSS для стилів, графічний інтерфейс в системі представлений через використання jsp сторінок.

Наступним кроком в цій роботі було описати сценарії використання системи. Користувач переходить в веб додаток та відразу має можливість увійти в свій аккаунт, але якщо його немає, користувач може перейти за покликанням внизу форми, та перейти на сторінку з реєстрацією користувача, де йому буде потрібно власноруч придумати та ввести логін та пароль до системи, також тут є перевірка на те, що заданий користувач вже існує та те що пароль та його підтвердження можуть не співпадати, після успішної реєстрації користувача перенаправляють на головну сторінку системи

в якій він може нативнувши на певні кнопки взаємодіяти з системою, також у користувача у верхньому правому куті є можливість повернутися на головну сторінку та вийти з аккаунту.

У користувача є 4 основні кнопки на головній сторінці:

Create queue, View queues, Add me in queue, та Update queue.

Далі наведена функціональність системи:.

1.Після того як користувач натисне на кнопку “Create queue” та створить нову чергу він автоматично стає її власником, там отримає права на редагування, також цю черга тепер відображається в меню View queues у розділі My queues, де користувач може переглянути інформацію про чергу(її вміст та свою позицію), аналогічно черга додається в розділі “All queues”, де користувач може побачити весь функціонал що є і в “My Queues”.

2.Після натискання кнопки “View queues” перед користувачем становиться вибір: передивитися свої черги, або передивитися усі існуючі на цей момент часу черги(“My Queues” та “All queues” відповідно), після того як користувач натискає на кнопку йому дається на вибір одна черга з випадаючого списку в якому він може вибрати потрібну йому чергу, натискаючи кнопку “View” користувача перенаправляє на сторінку з виведеною чергою, де він може переглянути як зміст так і свою позицію(якщо він є в черзі)

3.Після натискання на кнопку “Update queue” користувач перенаправляється на відповідну сторінку де йому дається змога редагувати черги, він може вибрати чергу з випадаючого списку, тим самими обрати її як чергу для редагування, потім система перевіряє чи може користувач редагувати чергу(чи є він її власником) після цього у разі успішної перевірки він може додавати елемент в чергу видаляти його, блокувати чергу, тобто чергу не можна буде змінювати до того моменту як її знову не розблокують,

також користувач має команду видалення з початку яка видаляє найперший елемент у списку.

4.Після натисканням кнопки “Add me in queue” користувач має змогу з випадającego списку вибрати будь-яку чергу яка існує на даний момент часу чергу та додатися в неї, після цього ім’я користувача додається в кінець черги.

Таким чином проект був розділений на декілька основних частин: контролери(сервлети), сервіси, модель, та репозиторії(dao), таким чином можна сказати що система написана на паттерні MVC(model, view, controller), що сприяє відкритості архітектури та можливості розширення функціоналу, що робить систему конкурентно спроможною та досить легкою в налаштуванні та використанні

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
2. Офіційна документація Lombok: [Електронний ресурс] – Режим доступу до ресурсу: <https://projectlombok.org/>
3. Офіційна документація HikariCP: [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/brettwooldridge/HikariCP>

ДОДАТКИ

ДОДАТОК А

Код основних сутностей та інтерфейсів рівня доступу до даних

User.java

```
package kpi.fict.coursework.op.zaranik.model;
import java.util.Objects;
import lombok.AllArgsConstructor;
import lombok.Data;
```

@Data

@AllArgsConstructor

```
public class User {
    private String username;
    private String password;
    private int id;
    private String passwordHash;
    private RoleType roleType;
```

```
    public User(Integer userId, String username, String password, RoleType
roleType) {
        this.id = userId;
        this.username = username;
        this.password = password;
        this.passwordHash = String.valueOf(password.hashCode());
        this.roleType = roleType;
    }
```

```
    public User(String username, String password, RoleType roleType) {
        this.username = username;
        this.password = password;
        this.roleType = roleType;
    }
```

```

public User(String username, String password) {
    this.username = username;
    this.password = password;
    this.roleType = RoleType.USER;
}

```

```

public void setUsername(String username) {
    this.username = username;
}

```

```

public int getUserId(){ return this.id;};

```

```

public void setId(int id) {this.id = id;}

```

@Override

```

public String toString() {
    return "User{" +
        "username=" + username + "\" +
        ", password=" + password + "\" +
        "}";
}

```

@Override

```

public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    User user = (User) o;
    return Objects.equals(username, user.username) &&

```

```
Objects.equals(password,
    user.password);
}
```

```
@Override
public int hashCode() {
    return Objects.hash(username, password);
}
```

```
}
```

Queue.java

```
package kpi.fict.coursework.op.zaranik.model;
```

```
import lombok.AllArgsConstructor;
import lombok.Getter;
```

```
@Getter
```

```
@AllArgsConstructor
```

```
public class Queue {
    private String name;
    private User creator;
    private boolean isBlocked;
    private Integer id;

    public Queue(String name, User creator) {
        this.name = name;
        this.creator = creator;
        this.id = this.name.hashCode();
    }
```

```
public boolean isBlocked() {
    return isBlocked;
}
```



```

public void setIsBlocked(boolean blocked) {
    isBlocked = blocked;
}

public void setId(Integer id){
    this.id = id;
}

public void setName(String name) {
    this.name = name;
}

public void setCreator(User creator) {
    this.creator = creator;
}
}

```

RoleType.java

```

package kpi.fict.coursework.op.zaranik.model;

public enum RoleType{
    USER, OWNER;
}

```

Dao.java

```

package kpi.fict.coursework.op.zaranik.dao.impl;

import java.sql.*;
import java.util.*;
import lombok.SneakyThrows;

```

```

public abstract class Dao<T> implements
kpi.fict.coursework.op.zaranik.dao.Dao<T> {

    protected abstract T mapResultSetToEntity(ResultSet rs) throws
SQLException;

    protected abstract String getTableName();

    protected abstract String getInsertQuery();

    protected abstract void setInsertParameters(PreparedStatement ps, T
entity) throws SQLException;

    protected abstract String getUpdateQuery();

    protected abstract void setUpdateParameters(PreparedStatement ps, T
entity) throws SQLException;

    protected abstract Integer getId(T entity);

    protected abstract void setGeneratedId(T entity, int id);

```

```
@Override
```

```
@SneakyThrows
```

```

public T get(Integer id) {
    String sqlQuery = "SELECT * FROM " + getTableName() + " WHERE id = ?";
    try (Connection connection = DataSource.getConnection();
        PreparedStatement ps = connection.prepareStatement(sqlQuery)) {
        ps.setInt(1, id);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                return mapResultSetToEntity(rs);
            }
        }
    }
}

```

```

    }
}
return null;
}

```

@Override

@SneakyThrows

```

public Collection<T> findAll() {
    List<T> entities = new ArrayList<>();
    String sqlQuery = "SELECT * FROM " + getTableName();
    try (Connection connection = DataSource.getConnection();
        PreparedStatement ps = connection.prepareStatement(sqlQuery);
        ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            entities.add(mapResultSetToEntity(rs));
        }
    }
    return entities;
}

```

@Override

@SneakyThrows

```

public void insert(T entity) {
    String sqlQuery = getInsertQuery();
    try (Connection connection = DataSource.getConnection();
        PreparedStatement ps = connection.prepareStatement(sqlQuery,
Statement.RETURN_GENERATED_KEYS)) {
        setInsertParameters(ps, entity);
        ps.executeUpdate();
        try (ResultSet generatedKeys = ps.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                setGeneratedId(entity, generatedKeys.getInt(1));
            }
        }
    }
}

```

```

    }
}

```

@Override

@SneakyThrows

public void delete(T entity) {

String sqlQuery = "DELETE FROM " + getTableName() + " WHERE id = ?";

try (Connection connection = DataSource.getConnection();

PreparedStatement ps = connection.prepareStatement(sqlQuery)) {

ps.setInt(1, getId(entity));

ps.executeUpdate();

}

}

@Override

@SneakyThrows

public void update(T entity) {

String sqlQuery = getUpdateQuery();

try (Connection connection = DataSource.getConnection();

PreparedStatement ps = connection.prepareStatement(sqlQuery)) {

setUpdateParameters(ps, entity);

ps.executeUpdate();

}

}

}

QueueDaoImpl.java

package kpi.fict.coursework.op.zaranik.dao.impl;

import java.sql.*;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

```
import kpi.fict.coursework.op.zaranik.dao.QueueDao;
import kpi.fict.coursework.op.zaranik.dao.UserDao;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.model.User;
```

```
public class QueueDaoImpl extends Dao<Queue> implements QueueDao {
```

```
    private UserDao userDao;
```

```
    public QueueDaoImpl(UserDao userDao) {
        super();
        this.userDao = userDao;
    }
```

```
@Override
```

```
protected Queue mapResultSetToEntity(ResultSet rs) throws SQLException {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int creatorId = rs.getInt("creatorId");
    boolean isBlocked = rs.getBoolean("isBlocked");
    User creator = userDao.get(creatorId);
    Queue queue = new Queue(name, creator);
    queue.setId(id);
    queue.setIsBlocked(isBlocked);
    return queue;
}
```

```
@Override
```

```
public List<String> getItemsByQueueId(int queueId) {
    List<String> items = new ArrayList<>();
    String query = "SELECT item FROM queueItems WHERE queueId = ?";
    try (Connection connection = DataSource.getConnection();
        PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt(1, queueId);
```

```

    try (ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            items.add(rs.getString("item"));
        }
    }
} catch (SQLException e) {
    throw new RuntimeException(e);
}
return items;
}

```

@Override

```

public void addItemByQueueId(int queueId, String item) {
    String query = "INSERT INTO queueItems (queueId, item) VALUES (?, ?)";
    try (Connection connection = DataSource.getConnection();
        PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt(1, queueId);
        ps.setString(2, item);
        ps.executeUpdate();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

@Override

```

public void removeItemFromQueue(int queueId, String item) {
    String query = "DELETE FROM queueItems WHERE id IN (" +
        "SELECT id FROM queueItems WHERE queueId = ? AND item = ? LIMIT 1)";
    try (Connection connection = DataSource.getConnection();
        PreparedStatement ps = connection.prepareStatement(query)) {
        ps.setInt(1, queueId);
        ps.setString(2, item);
        ps.executeUpdate();
    }
}

```

```

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

```

@Override
protected String getTableName() {
    return "queues";
}

```

```

@Override
protected String getInsertQuery() {
    return "INSERT INTO queues (name, creatorId, isBlocked) VALUES  
(?, ?, ?)";
}

```

```

@Override
protected void setInsertParameters(PreparedStatement ps, Queue entity)
throws SQLException {
    ps.setString(1, entity.getName());
    ps.setInt(2, entity.getCreator().getId());
    ps.setBoolean(3, entity.isBlocked());
}

```

```

@Override
protected String getUpdateQuery() {
    return "UPDATE queues SET name = ?, creatorId = ?, isBlocked = ?  
WHERE id = ?";
}

```

```

@Override
protected void setUpdateParameters(PreparedStatement ps, Queue entity)
throws SQLException {
    ps.setString(1, entity.getName());
}

```

```

    ps.setInt(2, entity.getCreator().getId());
    ps.setBoolean(3, entity.isBlocked());
    ps.setInt(4, entity.getId());
}

```

```

@Override
protected Integer getId(Queue entity) {
    return entity.getId();
}

```

```

@Override
protected void setGeneratedId(Queue entity, int id) {
    entity.setId(id);
}

```

```

@Override
public Collection<Queue> findAll() {
    return super.findAll();
}

```

```

@Override
public void insert(Queue queue) {
    super.insert(queue);
}

```

```

@Override
public void delete(Queue queue) {
    super.delete(queue);
}

```

```

@Override
public void update(Queue queue) {
    super.update(queue);
}

```



```

@Override
public Queue get(Integer id) {
    return super.get(id);
}
}

```

UserDaoImpl.java

```
package kpi.fict.coursework.op.zaranik.dao.impl;
```

```

import java.sql.*;
import java.util.Collection;
import kpi.fict.coursework.op.zaranik.dao.UserDao;
import kpi.fict.coursework.op.zaranik.model.RoleType;
import kpi.fict.coursework.op.zaranik.model.User;

```

```
public class UserDaoImpl extends Dao<User> implements UserDao {
```

```

    public UserDaoImpl() {
        super();
    }

```

```
@Override
```

```

protected User mapResultSetToEntity(ResultSet rs) throws SQLException {
    int id = rs.getInt("id");
    String username = rs.getString("username");
    String password = rs.getString("password");
    RoleType roleType = RoleType.valueOf(rs.getString("roleType"));
    return new User(id, username, password, roleType);
}

```

```
@Override
```

```
protected String getTableName() {
```

```

    return "users";
}

@Override
protected String getInsertQuery() {
    return "INSERT INTO users (username, password, roleType) VALUES
(?, ?, ?)";
}

@Override
protected void setInsertParameters(PreparedStatement ps, User entity)
throws SQLException {
    ps.setString(1, entity.getUsername());
    ps.setString(2, entity.getPassword());
    ps.setString(3, entity.getRoleType().name());
}

@Override
protected String getUpdateQuery() {
    return "UPDATE users SET username = ?, password = ?, roleType = ?
WHERE id = ?";
}

@Override
protected void setUpdateParameters(PreparedStatement ps, User entity)
throws SQLException {
    ps.setString(1, entity.getUsername());
    ps.setString(2, entity.getPassword());
    ps.setString(3, entity.getRoleType().name());
    ps.setInt(4, entity.getId());
}

@Override
protected Integer getId(User entity) {

```

```
    return entity.getUserId();  
}
```

```
@Override  
protected void setGeneratedId(User entity, int id) {  
    entity.setUserId(id);  
}
```

```
@Override  
public Collection<User> findAll() {  
    return super.findAll();  
}
```

```
@Override  
public void insert(User user) {  
    super.insert(user);  
}
```

```
@Override  
public void delete(User user) {  
    super.delete(user);  
}
```

```
@Override  
public void update(User user) {  
    super.update(user);  
}
```

```
@Override  
public User get(Integer id) {  
    return super.get(id);  
}  
}
```

Dao.java

```
package kpi.fict.coursework.op.zaranik.dao;
```

```
import java.util.Collection;
```

```
public interface Dao<T> {
    T get(Integer id);
    Collection<T> findAll();
    void insert(T entity);
    void delete(T entity);
    void update(T entity);
}
```

QueueDao.java

```
package kpi.fict.coursework.op.zaranik.dao;
```

```
import java.util.Collection;
```

```
import kpi.fict.coursework.op.zaranik.model.Queue;
```

```
public interface QueueDao extends Dao<Queue> {
    void addItemByQueueId(int queueId, String item);
    Collection<String> getItemsByQueueId(int queueId);
    void removeItemFromQueue(int queueId, String item);
}
```

UserDao.java

```
package kpi.fict.coursework.op.zaranik.dao;
```

```
import kpi.fict.coursework.op.zaranik.model.User;
```

```
public interface UserDao extends Dao<User> {}
```

QueueDaoServiceImpl.java

```

package kpi.fict.coursework.op.zaranik.services.dao.impl;

import java.util.Collection;
import java.util.List;
import kpi.fict.coursework.op.zaranik.dao.impl.QueueDaoImpl;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.QueueDaoService;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;

@RequiredArgsConstructor
public class QueueDaoServiceImpl implements QueueDaoService {

    private final QueueDaoImpl queueDaoImpl;

    @Override
    public Queue findQueueByName(String name) {
        return queueDaoImpl.findAll().stream()
            .filter(queue -> queue.getName().equals(name))
            .findFirst()
            .orElse(null);
    }

    @Override
    public void addQueueToUser(User user, Queue queue) {
        if (user != null && queue != null) {
            queue.setCreator(user);
            queueDaoImpl.insert(queue);
        }
    }

    @Override
    public Collection<Queue> getUserQueues(String username) {

```

```

return queueDaoImpl.findAll().stream()
    .filter(queue -> queue.getCreator().getUsername().equals(username))
    .toList();
}

```

@Override

```

public Collection<Queue> getAllQueues() {
    return queueDaoImpl.findAll();
}

```

@Override

@SneakyThrows

```

public int getUserPosition(Queue queue, User user) {
    Queue selectedQueue = queueDaoImpl.findAll().stream()
        .filter(q -> q.getName().equals(queue.getName()))
        .findFirst()
        .orElse(null);
    if (selectedQueue == null) {
        return -1;
    }
    List<String> items =
queueDaoImpl.getItemsByQueueId(selectedQueue.getId());
    if (!items.contains(user.getUsername())) {
        return -1;
    }
    return items.indexOf(user.getUsername()) + 1;
}

```

@Override

```

public void updateQueue(Queue queue) {
    queueDaoImpl.update(queue);
}

```

@Override

```

public void deleteQueue(Queue queue) {
    queueDaoImpl.delete(queue);
}

```

@Override

```

public boolean exists(Queue queue) {
    return findQueueByName(queue.getName()) != null;
}

```

@Override

@SneakyThrows

```

public void removeFirstItemFromQueue(Queue queue) {
    List<String> items = queueDaoImpl.getItemsByQueueId(queue.getId());
    if (!items.isEmpty()) {
        String firstItem = items.get(0);
        queueDaoImpl.removeItemFromQueue(queue.getId(), firstItem);
    }
}

```

@Override

```

public List<String> getItemsByQueueId(int queueId) {
    return queueDaoImpl.getItemsByQueueId(queueId);
}

```

@Override

```

public void addItemInQueue(Queue queue, String item) {
    int queueId = queue.getId();
    queueDaoImpl.addItemByQueueId(queueId, item);
}

```

@Override

```

public void removeItemFromQueue(Queue queue, String item) {
    int queueId = queue.getId();
    queueDaoImpl.removeItemFromQueue(queueId, item);
}

```

```

}

@Override
@SneakyThrows
public int getQueueSize(Queue queue) {
    return queueDaoImpl.getItemsByQueueId(queue.getId()).size();
}

@Override
@SneakyThrows
public boolean contains(Queue queue, String item) {
    List<String> items = queueDaoImpl.getItemsByQueueId(queue.getId());
    return items.contains(item);
}

@Override
public boolean isBlocked(String queueName) {
    Queue queue = findQueueByName(queueName);
    return queue != null && queue.isBlocked();
}

@Override
public void setBlock(String queueName, boolean isBlocked) {
    Queue queue = findQueueByName(queueName);
    if (queue != null) {
        queue.setIsBlocked(isBlocked);
        queueDaoImpl.update(queue);
    }
}
}

```

UserDaoServiceImpl.java

```

package kpi.fict.coursework.op.zaranik.services.dao.impl;

```



```

import java.util.Optional;
import kpi.fict.coursework.op.zaranik.dao.UserDao;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.UserDaoService;
import
kpi.fict.coursework.op.zaranik.services.passwordhashing.PasswordHashingService
;

```

```

public class UserDaoServiceImpl implements UserDaoService {
    private UserDao userDao;
    private PasswordHashingService passwordHashingService;

    public UserDaoServiceImpl(UserDao userDao, PasswordHashingService
passwordHashingService){
        this.userDao = userDao;
        this.passwordHashingService = passwordHashingService;
    }

```

@Override

```

public Optional<User> findUserByUsername(String username) {
    return userDao
        .findAll()
        .stream()
        .filter(user -> user.getUsername().equals(username))
        .findFirst();
}

```

@Override

```

public void createUser(User user) {
    Optional<User> userOpt = findUserByUsername(user.getUsername());
    if(userOpt.isPresent()){
        throw new RuntimeException("user already exists");
    }
    String password =

```

```
passwordHashingService.hashPassword(user.getPassword());
    user.setPassword(password);
    userDao.insert(user);
}
```

@Override

```
public boolean exists(String username) {
    Optional<User> userOpt = userDao
        .findAll()
        .stream()
        .filter(user -> user.getUsername().equals(username))
        .findFirst();

    return userOpt.isPresent();
}
}
```

ДОДАТОК Б

Код основних сутностей та інтерфейсів рівня бізнес-логіки

CreateQueueServlet.java

```
package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.model.RoleType;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.QueueDaoService;
```

```

import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import
kpi.fict.coursework.op.zaranik.services.namevalidator.NameValidatorService;
import
kpi.fict.coursework.op.zaranik.services.roleconfigurator.RoleConfiguratorService;
import lombok.SneakyThrows;

@WebServlet("/createQueue")
public class CreateQueueServlet extends HttpServlet {

    private QueueDaoService queueDaoService;
    private RoleConfiguratorService roleConfiguratorService;
    private NameValidatorService nameValidatorService;

    @Override
    @SneakyThrows
    public void init() {
        super.init();
        queueDaoService = ServiceFactory.getQueueDaoService();
        roleConfiguratorService = ServiceFactory.getRoleConfiguratorService();
        nameValidatorService = ServiceFactory.getNameValidatorService();
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        HttpSession session = request.getSession();
        User user = (User) session.getAttribute("user");
        if (user == null) {
            response.sendRedirect("LoginPage.jsp");
            return;
        }
        request.getRequestDispatcher("/CreateQueue.jsp").forward(request,
        response);
    }

```

```

}

@sneakyThrows
@Override
public void doPost(HttpServletRequest request, HttpServletResponse
response) {
    String selectedQueueName = request.getParameter("queueName");
    HttpSession session = request.getSession();
    User user = (User) session.getAttribute("user");

    if (user == null) {
        response.sendRedirect("LoginPage.jsp");
        return;
    }

    if (!nameValidatorService.isValidName(selectedQueueName)) {
        request.setAttribute("errorMessage", "Empty Form Submitted");
        request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
        return;
    }

    Queue selectedQueue =
queueDaoService.findQueueByName(selectedQueueName);
    if (selectedQueue != null && queueDaoService.exists(selectedQueue)) {
        request.setAttribute("errorMessage", "Queue Is Already Created");
        request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
        return;
    }

    Queue queue = new Queue(selectedQueueName, user);
    roleConfiguratorService.configureRole(user, queue, RoleType.OWNER);
    queueDaoService.addQueueToUser(user, queue);
    queueDaoService.updateQueue(queue);
    response.sendRedirect("/MainPage.jsp");
}

```

```
}
```

```
}
```

DeleteQueueServlet.java

```
package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.QueueDaoService;
import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import
kpi.fict.coursework.op.zaranik.services.namevalidator.NameValidatorService;
import lombok.SneakyThrows;

@WebServlet("/deleteQueue")
public class DeleteQueueServlet extends HttpServlet {

    private QueueDaoService queueDaoService;
    private NameValidatorService nameValidatorService;

    @Override
    @SneakyThrows
    public void init(){
        super.init();
        this.queueDaoService = ServiceFactory.getQueueDaoService();
        this.nameValidatorService = ServiceFactory.getNameValidatorService();
    }
}
```

```

@SneakyThrows
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) {
    HttpSession session = request.getSession(false);
    if (session != null) {
        User user = (User) session.getAttribute("user");
        if (user != null) {
            request.setAttribute("queues", queueDaoService.getAllQueues());
        }
    }
    request.getRequestDispatcher("/CreateQueue.jsp").forward(request,
response);
}

```

```

@SneakyThrows
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) {
    String selectedQueueName = request.getParameter("selectedQueue");
    HttpSession session = request.getSession();
    User user = (User) session.getAttribute("user");

    if(!nameValidatorService.isValidName(selectedQueueName)){
        request.setAttribute("errorMessage", "Empty Form Submitted");
        request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
        return;
    }

    if (user != null) {
        Queue selectedQueue =
queueDaoService.findQueueByName(selectedQueueName);
        if (selectedQueue != null) {

```

```

        queueDaoService.deleteQueue(selectedQueue);
        request.getRequestDispatcher("MainPage.jsp").forward(request, response);
    }
} else {
    response.sendRedirect("/LoginPage.jsp");
}
}
}

```

LoginServlet.java

```

package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Optional;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.UserDaoService;
import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import
kpi.fict.coursework.op.zaranik.services.passwordhashing.PasswordHashingService
;
import lombok.SneakyThrows;

public class LoginServlet extends HttpServlet {

    private UserDaoService userDaoService;
    private PasswordHashingService passwordHashingService;

    @Override
    @SneakyThrows
    public void init() {

```

```

    super.init();
    this.userService = ServiceFactory.getUserDaoService();
    this.passwordHashingService = ServiceFactory.getPasswordHashingService();
}

@Override
@SneakyThrows
protected void doGet(HttpServletRequest request, HttpServletResponse
response) {
    request.getRequestDispatcher("LoginPage.jsp").forward(request, response);
}

@SneakyThrows
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    Optional<User> userOpt = userService.findUserByUsername(username);
    if (userOpt.isEmpty()) {
        request.getRequestDispatcher("InvalidLoginOrPassword.jsp").forward(request,
response);
        return;
    }

    User user = userOpt.get();
    if (passwordHashingService.checkPassword(password, user.getPassword())) {
        request.getSession().setAttribute("username", username);
        request.getSession().setAttribute("user", user);
        response.sendRedirect("MainPage.jsp");
    } else {

```



```

request.getRequestDispatcher("InvalidLoginOrPassword.jsp").forward(request,
response);
    }
}
}

```

LogoutServlet.java

```

package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import lombok.SneakyThrows;

public class LogoutServlet extends HttpServlet {

    @Override
    @SneakyThrows
    protected void doGet(HttpServletRequest request, HttpServletResponse
response){
        HttpSession session = request.getSession();
        session.invalidate();
        response.sendRedirect("/LoginPage.jsp");
    }
}

```

RegisterServlet.java

```

package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;

```

```

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import kpi.fict.coursework.op.zaranik.model.RoleType;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.UserDaoService;
import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import lombok.SneakyThrows;

```

```

@WebServlet("/register")

```

```

public class RegisterServlet extends HttpServlet {
    UserDaoService userDaoService;

```

```

    @Override

```

```

    @SneakyThrows

```

```

    public void init(){

```

```

        super.init();

```

```

        this.userDaoService = ServiceFactory.getUserDaoService();

```

```

    }

```

```

    @Override

```

```

    @SneakyThrows

```

```

    public void doPost(HttpServletRequest request, HttpServletResponse
response){

```

```

        String username = request.getParameter("username");

```

```

        String password = request.getParameter("password");

```

```

        String confirmPassword = request.getParameter("confirm_password");

```

```

        if(userDaoService.exists(username)){

```

```

            request.setAttribute("errorMessage", "user already exists");

```

```

            request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);

```

```

        }

```

```

        if(!password.equals(confirmPassword)){

```

```

        request.setAttribute("errorMessage", "Password mismatching");
        request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
    }

```

```

        User user = new User(username, password);
        userDaoService.createUser(user);

```

```

        response.sendRedirect("LoginPage.jsp");
    }

```

```

}

```

UpdateQueueServlet.java

```

package kpi.fict.coursework.op.zaranik.controller;

```

```

import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.util.ArrayList;
import java.util.List;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.model.RoleType;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.QueueDaoService;
import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import
kpi.fict.coursework.op.zaranik.services.roleconfigurator.RoleConfiguratorService;
import lombok.SneakyThrows;

```

```

@WebServlet("/updateQueue")
public class UpdateQueueServlet extends HttpServlet {

```

```
private QueueDaoService queueDaoService;
private RoleConfiguratorService roleConfiguratorService;
```

```
@Override
@SneakyThrows
public void init(){
    super.init();
    this.queueDaoService = ServiceFactory.getQueueDaoService();
    this.roleConfiguratorService = ServiceFactory.getRoleConfiguratorService();
}
```

```
@Override
@SneakyThrows
protected void doGet(HttpServletRequest request, HttpServletResponse
response) {
    HttpSession session = request.getSession();
    User user = (User) session.getAttribute("user");
    if (user == null) {
        response.sendRedirect("LoginPage.jsp");
        return;
    }
    List<Queue> allQueues = new ArrayList<>(queueDaoService.getAllQueues());
    request.setAttribute("queues", allQueues);
    request.getRequestDispatcher("UpdateQueue.jsp").forward(request,
response);
}
```

```
@Override
@SneakyThrows
protected void doPost(HttpServletRequest request, HttpServletResponse
response) {
    String selectedAction = request.getParameter("action");
    String selectedQueueName = request.getParameter("selectedQueue");
    Queue selectedQueue =
```

```

queueDaoService.findQueueByName(selectedQueueName);
HttpSession session = request.getSession();
User user = (User) session.getAttribute("user");

if (selectedQueue == null) {
    System.out.println("Selected queue does not exist: " +
selectedQueueName);
    request.setAttribute("errorMessage", "Selected queue does not exist");
    request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
    return;
}

if (roleConfiguratorService.getConfiguration(user, selectedQueue) !=
RoleType.OWNER) {
    request.setAttribute("errorMessage", "Permission Denied");
    request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
    return;
}

if (queueDaoService.isBlocked(selectedQueueName)
&& !selectedAction.equals("blockOrUnblockQueue")) {
    request.setAttribute("errorMessage", "The queue is blocked. No
modifications allowed.");
    request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
    return;
}

switch (selectedAction) {
    case "add":
        request.getRequestDispatcher("/addItemInQueue").forward(request,
response);
        break;
    case "removeItem":

```

```

request.getRequestDispatcher("/removeItemFromQueue").forward(request,
response);
    break;
    case "removeFirstItem":

request.getRequestDispatcher("/removeItemFromBegin").forward(request,
response);
    break;
    case "blockOrUnblockQueue":
        request.getRequestDispatcher("/blockUnblockQueue").forward(request,
response);
        break;
    case "deleteQueue":
        request.getRequestDispatcher("/deleteQueue").forward(request, response);
        break;
    default:
        break;
}
}

}

```

ДОДАТОК В

Код основних сутностей та інтерфейсів рівня інтерфейсу користувача

AllQueuesServlet.java

```

package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

```

```

import java.io.IOException;
import java.util.Collection;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.services.dao.QueueDaoService;
import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import lombok.SneakyThrows;

@WebServlet("/allQueues")
public class AllQueuesServlet extends HttpServlet {

    private QueueDaoService queueDaoService;

    @Override
    @SneakyThrows
    public void init(){
        super.init();
        queueDaoService = ServiceFactory.getQueueDaoService();
    }

    @SneakyThrows
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        Collection<Queue> allQueues = queueDaoService.getAllQueues();
        request.setAttribute("queues", allQueues);
        request.setAttribute("tableName", "All Queues");

        if(queueDaoService.getAllQueues().isEmpty()){
            request.setAttribute("errorMessage", "No queues available");
            request.getRequestDispatcher("/ErrorPage.jsp").forward(request, response);
            return;
        }

        request.getRequestDispatcher("QueueMenu.jsp").forward(request, response);
    }
}

```

```

    }
}

```

ViewQueuesServlet.java

```

package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;
import kpi.fict.coursework.op.zaranik.model.User;

@WebServlet("/viewQueues")
public class ViewQueuesServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        HttpSession session = request.getSession(false);
        User user = (User) session.getAttribute("user");

        if (user == null) {
            response.sendRedirect("LoginPage.jsp");
            return;
        }

        request.getRequestDispatcher("/Queues.jsp").forward(request, response);
    }
}

```


ViewSelectedQueueServlet.java

```
package kpi.fict.coursework.op.zaranik.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;
import kpi.fict.coursework.op.zaranik.model.Queue;
import kpi.fict.coursework.op.zaranik.model.User;
import kpi.fict.coursework.op.zaranik.services.dao.QueueDaoService;
import kpi.fict.coursework.op.zaranik.services.factories.ServiceFactory;
import lombok.SneakyThrows;

@WebServlet("/viewSelectedQueue")
public class ViewSelectedQueueServlet extends HttpServlet {

    private QueueDaoService queueDaoService;

    @Override
    @SneakyThrows
    public void init(){
        super.init();
        queueDaoService = ServiceFactory.getQueueDaoService();
    }

    @SneakyThrows
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        HttpSession session = request.getSession(false);
        if (session == null) {
```

```

        response.sendRedirect("LoginPage.jsp");
        return;
    }

```

```

    User user = (User) session.getAttribute("user");
    if (user == null) {
        response.sendRedirect("LoginPage.jsp");
        return;
    }

```

```

    String selectedQueueName = request.getParameter("selectedQueue");
    Queue selectedQueue =
queueDaoService.findQueueByName(selectedQueueName);

```

```

    if (selectedQueue == null) {
        session.setAttribute("errorMessage", "Queue not found");
        response.sendRedirect("ErrorPage.jsp");
        return;
    }

```

```

    String userPosition =
String.valueOf(queueDaoService.getUserPosition(selectedQueue, user));
    if (userPosition.equals("-1")) userPosition = "you are not in queue";

```

```

        request.setAttribute("selectedQueue", selectedQueue);
        request.setAttribute("userPosition", userPosition);
        request.setAttribute("items",
queueDaoService.getItemsByQueueId(selectedQueue.getId()));
        request.getRequestDispatcher("QueueContent.jsp").forward(request,
response);
    }
}

```

