**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ**



**ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

Навчально-науковий інститут інфокомунікацій та програмної інженерії

Кафедра інформаційних технологій

**Курсовий проект**

з дисципліни «Організація баз даних та знань»

на тему: **«СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПОЛІКЛІНІКИ»**

Виконав: студент 3 курсу, групи ІПЗ-3.04 спеціальності

121 Інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бухта М.М.

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів\_\_\_\_\_\_\_Оцінка ECTS\_\_\_\_\_\_\_

**Одеса  2024**

ЗМІСТ

|  |  |
| --- | --- |
| ВСТУП................................................................................................................ | ....3 |
| 1. ПОСТАНОВКА ЗАВДАННЯ....................................................................... | ....4 |
| 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.................................. | ....7 |
| 3. АНАЛІЗ ФУНКЦІЙНИХ ВИМОГ ТА USE-CASE ДІАГРАМИ............... | ..10 |
| 4. ДІАГРАМИ ПОСЛІДОВНОСТІ ДОДАТКА.............................................. | ..13 |
| 5. ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ........................... | ..15 |
| 6. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ................................................ | ..20 |
| 7. МОДЕЛЬ ПРОГРАМНОГО ДОДАТКУ...................................................... | ..22 |
| 8. СТВОРЕННЯ БАЗИ ДАНИХ....................................................................... | ..26 |
| 9. ПРОГРАМНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОГО ДОДАТКУ..................... | ..38 |
| 10. ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ........................... | ..44 |
| 11. ІНСТРУКЦІЯ КОРИСТУВАЧА З ІЛЮСТРАЦІЯМИ............................. | ..48 |
| ПЕРЕЛІК ПОСИЛАНЬ...................................................................................... | ..51 |
| Додаток А. СХЕМА БАЗИ ДАНИХ................................................................. | ..52 |
| Додаток Б. ЗАПИТИ ДЛЯ СТВОРЕННЯ БАЗИ ДАНИХ.............................. | ..53 |
| Додаток В. КОД КЛІЄНТСЬКОГО ЗАСТОСУНКУ...................................... | ..90 |
| Додаток Г. КОД СЕРВЕРНОЇ ЧАСТИНИ ЗАСТОСУНКУ........................... | 125 |

**ВСТУП**

В сучасних поліклініках електронна картотека пацієнтів є однією з найважливіших складових інформаційної системи. Ця система забезпечує збереження та організацію медичних даних пацієнтів, а також дозволяє зручно керувати інформацією про прийоми, лікування та інші медичні процедури.

Метою даного проекту є розроблена інформаційна система - електронна картотека поліклініки. Дані пацієнтів будуть зберігатися в базі даних та зчитуватися під час роботи програми.

Задачами проекту буде розробка інформаційної моделі предметної області та створення бази даних, яка забезпечить потрібною інформацією електронну картотеку

Метою курсового проекту є проектування та створення інформаційної системи для автоматизації повсякденних процесів у електронній картотеці пацієнтів поліклініки. Для досягнення цієї мети потрібно розв'язати наступні задачі:

* Проаналізувати предметну область, відокремити користувачів системи;
* виконати проектування бази даних;
* Обґрунтувати вибір засобів та технологій розробки;
* Розробити базу даних на основі системи керування базами даних PostgreSQL;
* Розробити Front-End частину застосунку;
* Розробити Back-End частину застосунку;
* Протестувати програмний застосунок, переконавшись у його коректному функціонуванні

**1. ПОСТАНОВКА ЗАВДАННЯ**

В інформаційній системі передбачено три типи користувачів:

* **Адміністратор** — відповідая за підтримку працездатності, змінює ролі, слідкує за базою, куреє співробітниками та їх розкладом;
* **Доктор** — приймає записи пацієнтів, виписує рецепти, результати прийому. Оновлює мед картку пацієнта;
* **Пацієнт** — перегляд розкладу лікарів, запис до лікаря, робить запроси у онлайн консультації.

У табл. 1.1. наведено перелік задач для кожного з користувачів із зазначенням вхідної та вихідної інформації. Схема бази даних знаходиться в додатку А.

Таблиця 1.1 — задачі користувачів.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Задача | Вхідна інформація | | | Вихідна інформація |
| 1. **Пацієнт** | | | | |
| 1.1. Перегляд лікарів | ПІБ лікаря  Спеціальність | | | ПІБ лікаря  Спеціальність  Графік роботи  Номер телефону |
| 1.2. Реєстрація | Пошта  Пароль | | |  |
| 1.3. Перегляд результатів прийомів | Ідентифікатор пацієнта | | | ПІБ лікаря  Скарги  Діагнози  Рецепт (назва ліків)  ПІБ пацієнта  Дата прийому  Статус прийому |
| 1.4. Перегляд виписаних рецептів | Ідентифікатор пацієнта | | | Назва препарату  Виробник  Рекомендації прийому |
| 1.5 Запис на прийом | Ідентифікатор співробітника  Скарги  Ідентифікатор пацієнта  Дата прийома Час прийома | | | Прийом |
| Задача | | Вхідна інформація | Вихідна інформація | |
| 1. **Лікар** | | | | |
| 2.1. Перегляд даних про пацієнта | ПІБ пацієнта | | | ПІБ пацієнта  Дата народження  Адреса  Номер телефону |
| 2.2 Перегляд результатів прийому | Ідентифікатор співробітника | | | Ідентифікатор прийому  ПІБ лікаря  Скарга  Діагноз  Рецепт  ПІБ пацієнта  Дата  Статус прийома |
| 2.3 Перегляд рецептів | Ідентифікатор рецепту | | | Номер препарату  Назва препарату  Рекомендації прийому |
| 2.4 Додавання нового рецепту | Номер препарату  Назва препарату  Рекомендації по прийому  Ідентифікатор прийому | | | Рецепт |
| 2.5 Перегляд свого робочого графіку | Ідентифікатор співробітника | | | Час початку  Час закінчення  День тижня |
| 2.6 Редагування прийому | Ідентифікатор прийому  Діагноз | | | Прийом |
| 2.7 Зміна статусу прийому | Статус = «Открыт /Закрыт/Отменен»  Діагноз  Ідентифікатор рецепту | | | Прийом |
| 1. **Адміністратор** | | | | |
| 3.1. Пошук зареєстрованих користувачів | Роль  Пошта | | | Пошта  Роль  Дата створення |
| 3.2 Редагування користувачів | Ідентифікатор користувача  Роль | | | Користувач |
| 3.3. Пошук зареєстрованого пацієнта | ПІБ пацієнта | | | ПІБ пацієнта  Дата народження  Адреса  Номер телефону |
| 3.4 Створення медичної картки | ПІБ пацієнта  Дата народження  Адреса  Номер телефонy  Ідентифікатор користувача | | | Медична картка пацієнта |
| Задача | Вхідна інформація | | | Вихідна інформація |
| 3.5 Редагування медичної картки | Ідентифікатор пацієнта  ПІБ пацієнта  Дата народження  Адреса | | | Медична картка пацієнта |
| 3.6. Редагування препаратів | Ідентифікатор препарату  Номер перпарату  Назва препарату  Виробник | | | Препарат |
| 3.7 Додавання нового співробітника | ПІБ співробітника  Спеціальність  Номер телефону  Ідентифікатор користувача | | | Медпрацівник |
| 3.8 Видалення препаратів з бази | Ідентифікатор препарату | | |  |
| 3.9 Створювати графік роботи лікарів | Лікар  День тижня прийому  Початку прийому  Кінець прийому | | | Графік |

**2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ**

Для вирішення поставлених задач була обрана **трьохланкова клієнт-серверна архітектура**, яка складається з сервера бази даних (рівень управління ресурсами), сервера додатків (рівень прикладного компоненту) та клієнтів (рівень представлення даних). Взаємодія між цими компонентами відбувається таким чином: клієнт надсилає запит на перегляд сторінки, який обробляється сервером додатків. При необхідності, сервер додатків звертається до сервера бази даних та отримує відповідь у вигляді запитуваної сторінки, яку відправляє клієнту. Схема цієї взаємодії наведена на рисунку 2.1.

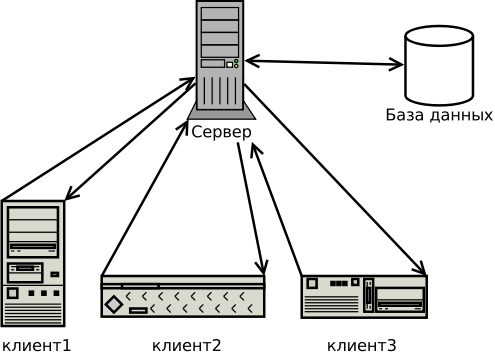


Рисунок 2.1 — схема триланкової архітектури.

Для проектування логічної структури додатку був обраний шаблон **MVC (Model-View-Controller)**. У цьому архітектурному патерну йде переросподіл обов’язків між трьома компонентами, що дозволяє робити розробки більш гнучно та підтриємаємо.

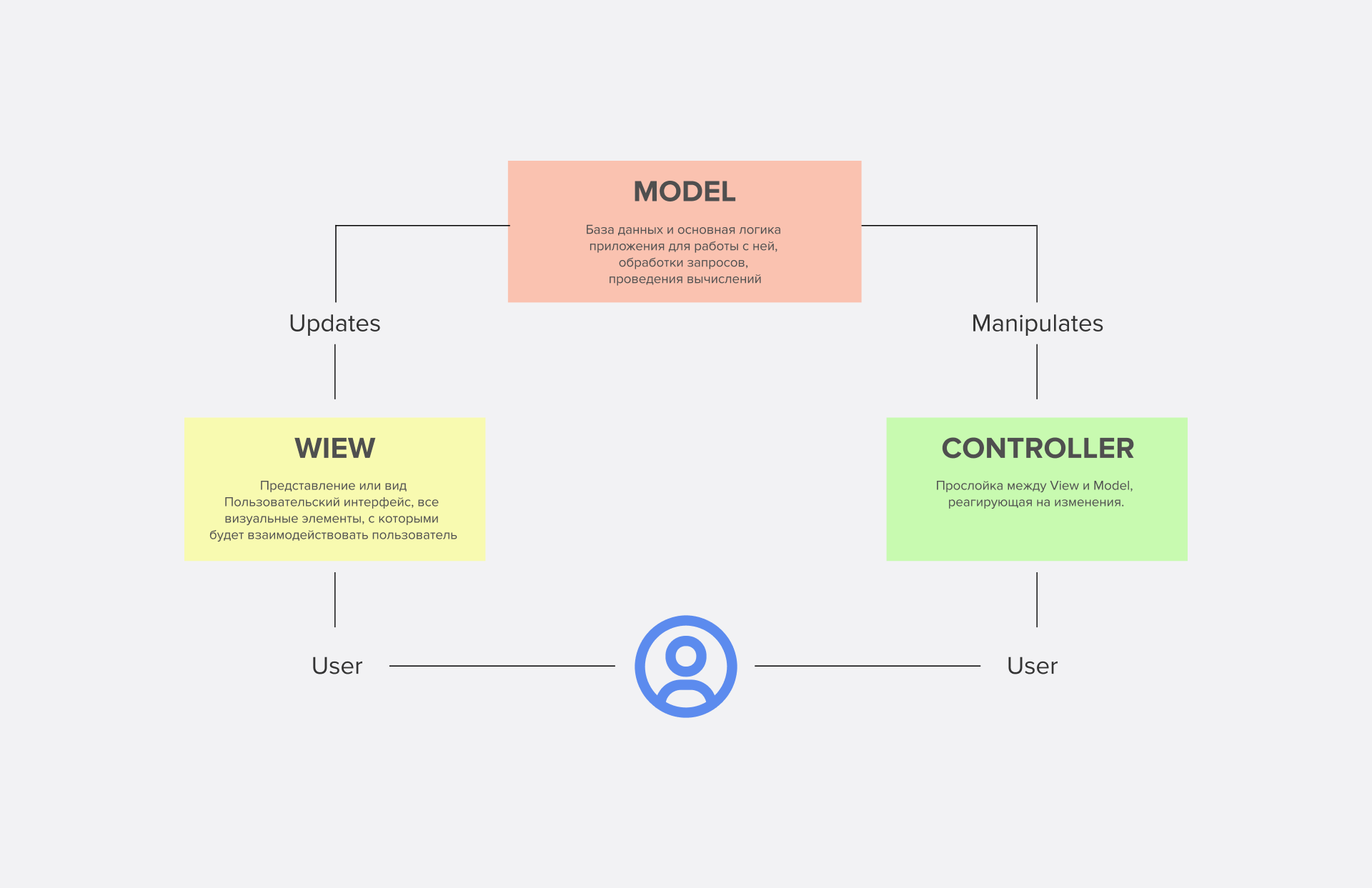
**Model (модель)** — відповідає за управління даними. Зберегає усю мета інформацію.

**View (представлення)** — відповадіє за відображення даних користувачу. Якщо це графічна частина, то відображає графічно, якщо ми говоримо про програмну, то це, частіше усього, інтерфейс взаємодії, який можно використовувати для комунікації між іншими компонентами програми.

**Controller (контролер)** — приймає вхідні дани та обробляє їх. Тут описана основна логіка додатка та бізнес-логіка.

Схема архітектурного шаблону наведена на рисунку 2.2.

Рисунок 2.2 — схема архітектурного патерна MVC.



**Переваги MVC:**

* Розділення обов’язків — розділення на три компоненти робить код більш організованим та підтримуємим.
* Модульність — усі компаненти незалежні та можуть бути заменіми.
* Тестованість — усі компаненти можуть бути легко протестовані без залежності один від одного
* Гнучність — архітектура легко адаптується до змін вимог

Завдяки цим перевагам, шаблон MVC дозволяє швидко побудувати прототип додатку та легко вносити зміни без значних змін у вихідний код.

**3. АНАЛІЗ ФУНКЦІЙНИХ ВИМОГ ТА USE-CASE ДІАГРАМИ**

Аналіз функціональних вимог та розробка use-case діаграм є важливим етапом у процесі розробки програмного забезпечення. Він дозволяє визначити основні функціональні можливості системи, взаємодію користувачів з нею, а також забезпечити розуміння основних процесів, що відбуваються в межах системи.

Функціональні вимоги визначають поведінку системи, що повинна бути реалізована для задоволення потреб користувачів. Вони описують, що система повинна робити, які послуги надавати та як вона повинна реагувати на різні події. Для цього проводиться збір вимог від усіх зацікавлених сторін та їх аналіз для виявлення необхідних функцій. Основні кроки аналізу функціональних вимог:

* **Збір вимог**: Включає інтерв'ю, опитування, спостереження за користувачами та аналіз документів.
* **Аналіз вимог**: Систематизація та деталізація зібраних вимог, визначення пріоритетів.
* **Документування вимог**: Створення документа, що містить опис всіх функціональних вимог.

**Use-case діаграми** (діаграми прецедентів) використовуються для візуального моделювання функціональних вимог системи. Вони відображають взаємодію користувачів (акторів) із системою через різні сценарії використання. Основні елементи use-case діаграм:

* **Актори**: Зовнішні суб’єкти, що взаємодіють із системою (наприклад, користувачі, інші системи).
* **Прецеденти**: Функціональні можливості або сценарії використання системи, що виконуються акторами.
* **Зв'язки**: Відношення між акторами та прецедентами, що показують, які саме функції виконуються кожним актором.

Процес створення use-case діаграм

* **Ідентифікація акторів**: Визначення всіх можливих користувачів системи та зовнішніх систем, які взаємодіють із нею.
* **Визначення прецедентів**: Опис усіх функцій, які повинні бути реалізовані в системі для задоволення потреб акторів.
* **Встановлення зв'язків**: З’єднання акторів з відповідними прецедентами для відображення їх взаємодії.

Для більшого розуміння розглянемо приклад створення use-case діаграми для інформаційної системи “Поліклініка "ЗаЗдоров'я"”. Приклади зображені на рисунках 3.1-3.3.

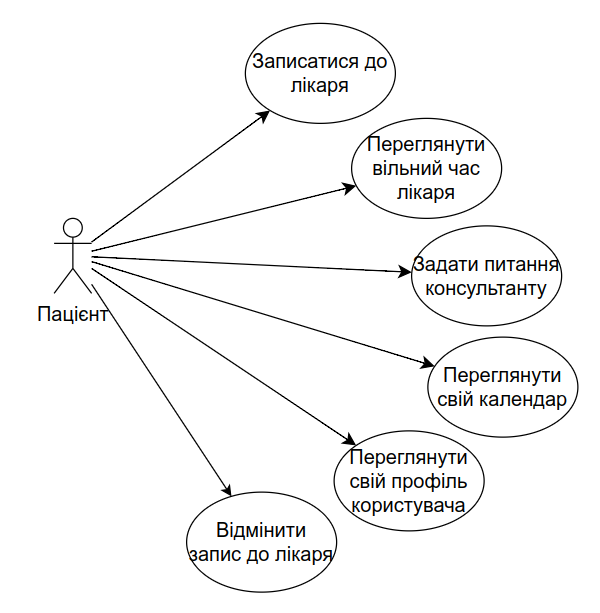


Рисунок 3.1 – діаграма прецедентів для пацієнта

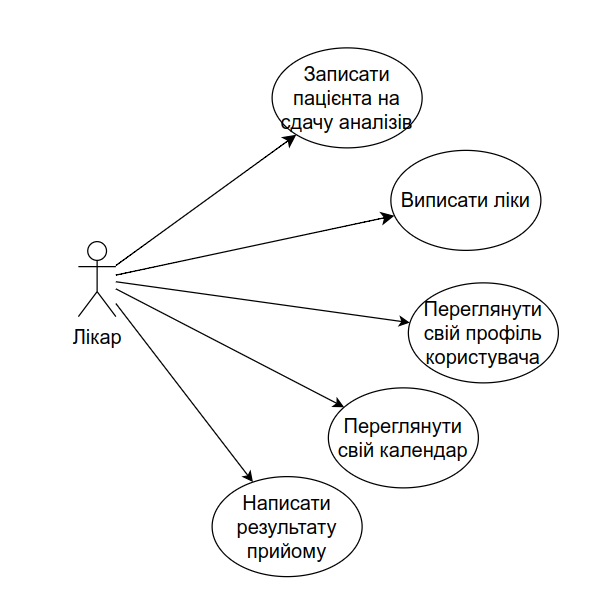


Рисунок 3.2 – діаграма прецедентів для лікаря

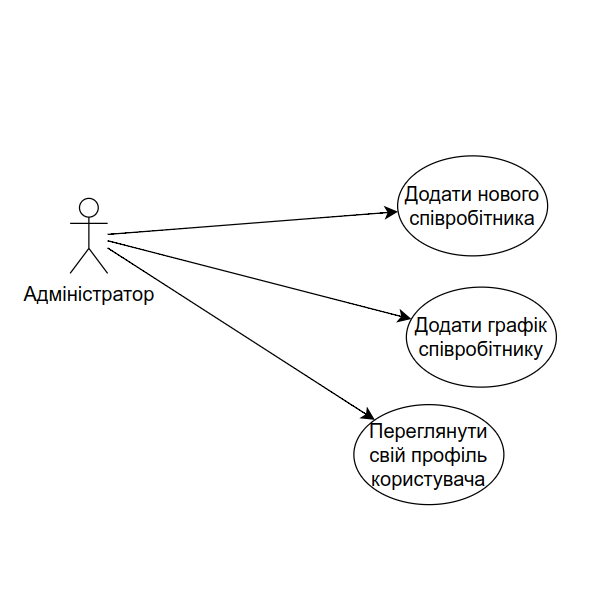


Рисунок 3.3 – діаграма прецедентів для адміністратора

**4. ДІАГРАМИ ПОСЛІДНОВСТІ ДОДАТКА**

Діаграма послідовності (sequence diagram) — це один з типів діаграм UML (Unified Modeling Language), який використовується для моделювання динамічних аспектів системи. Вона показує взаємодію між різними об'єктами системи у вигляді послідовності повідомлень, які вони обмінюються між собою в певному часовому порядку.

Діаграми послідовності допомагають зрозуміти, як об'єкти взаємодіють між собою протягом часу для виконання певної функції або бізнес-процесу. Вони є ефективним інструментом для документування архітектури програмного забезпечення, що полегшує розуміння та підтримку системи. Також вони використовуються під час етапів аналізу та проектування для моделювання різних сценаріїв використання системи та виявлення можливих проблем або вдосконалень. Діаграми послідовності мають наступні основні компоненти:

* **Актори (Actors)**: представляють користувачів або зовнішні системи, які взаємодіють із системою. Вони зображуються у вигляді паличкових чоловічків або прямокутників.
* **Об'єкти (Objects)**: екземпляри класів, які взаємодіють між собою. Зображуються у вигляді прямокутників із підкресленими іменами.
* **Лінії життя (Lifelines)**: вертикальні пунктирні лінії, що виходять від об'єктів або акторів, показують їх існування протягом певного часу.
* **Активні періоди (Activation bars)**: прямокутники на лініях життя, що показують періоди, коли об'єкти виконують певні дії.
* **Повідомлення (Messages)**: стрілки між лініями життя, що показують обмін повідомленнями між об'єктами. Види повідомлень можуть бути синхронними (суцільна стрілка) або асинхронними (стрілка з півколом).
* **Примітки (Notes)**: Використовуються для додаткових пояснень і зображуються як прямокутники з загнутим кутом, з'єднані пунктирними лініями з відповідними елементами.

На рис 4.1 приклад діаграми послідовності для веб-сервера, що обробляє запити на відкриття файлу.

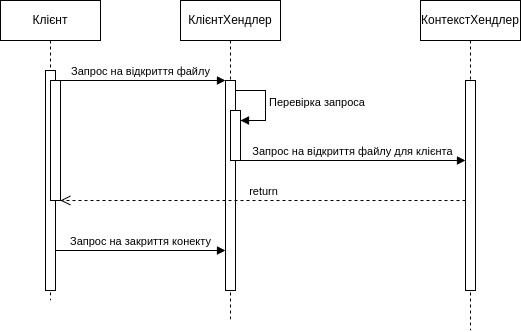


Рисунок 4.1 – діаграма послідовності для відкриття файлу

Це лише один з багатьох можливих сценаріїв, який ілюструє, як діаграми послідовності допомагають зрозуміти та моделювати взаємодію між компонентами системи.

**5. ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ**

Сутінсть та їх властивості з описом обмежень, що потрібні для розв’язання поставлених задач, наведені в таблиці 5.1.

Таблиця 5.1 — опис сутностей та їх властивостей

|  |  |  |
| --- | --- | --- |
| **Властивість** | **Опис** | **Обмеження** |
| **Об’єкт Users/Користувач** | | |
| id | Ідентифакатор | PRIMARY KEY (PK) |
| name | Ім’я | До 50 симв. NOT NULL |
| surname | Прізвище | До 50 симв. NOT NULL |
| middle\_name | По-батькові | До 50 симв. |
| email | Пошта | custom\_email, NOT NULL, UNIQUE |
| **Об’єкт Passwords/Пароль** | | |
| user\_id | Ідентифікатор користувача | PRIMARY KEY (PK), FOREIGN KEY (FK) |
| pass | Пароль | TEXT, NOT NULL |
| **Об’єкт PhoneNumber/НомерТелефону** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| number | Номер телефону | VARCHAR(10), NOT NULL, UNIQUE, CHECK (CHAR\_LENGTH(number) = 10) |
| country\_code | Код країни | VARCHAR(3), NOT NULL |
| user\_id | Ідентифікатор користувача | FOREIGN KEY (FK) |
| **Об’єкт Roles/Ролі** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| name | Назва ролі | VARCHAR(50), NOT NULL, UNIQUE |
| salary | Зарплата | DECIMAL(8, 2), CHECK (salary >= 0) |
| **Властивість** | **Опис** | **Обмеження** |
| **Об’єкт UsersRoles/РолиКористувачів** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| user\_id | Ідентифікатор користувача | FOREIGN KEY (FK), UNIQUE (user\_id, role\_id) |
| role\_id | Ідентифікатор ролі | FOREIGN KEY (FK) |
| **Об’єкт Messages/Повідомлення** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| user\_1\_id | Ідентифікатор користувача 1 | FOREIGN KEY (FK) |
| user\_2\_id | Ідентифікатор користувача 2 | FOREIGN KEY (FK) |
| destination\_time | Час відправки | TIMESTAMPTZ, DEFAULT CURRENT\_TIMESTAMP, NOT NULL |
| data | Повідомлення | До 2048 симв., NOT NULL |
| **Об’єкт Allergens/Алергени** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| name | Назва алергену | До 50 симв. NOT NULL |
| **Об’єкт UsersAllergens/АлергениКористувача** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| user\_id | Ідентифікатор користувача | FOREIGN KEY (FK) |
| allergen\_id | Ідентифікатор алергену | FOREIGN KEY (FK) |
| **Об’єкт BookedDoctors/БронюванняДоктора** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| patient\_id | Ідентифікатор пацієнта | FOREIGN KEY (FK), CONSTRAINT valid\_patient\_role CHECK (is\_valid\_patient\_role(patient\_id)) |
| doctor\_id | Ідентифікатор лікаря | FOREIGN KEY (FK), CONSTRAINT valid\_doctor\_role CHECK (is\_valid\_doctor\_role(doctor\_id)) |
| **Властивість** | **Опис** | **Обмеження** |
| book\_time | Час бронювання | TIMESTAMPTZ, DEFAULT CURRENT\_TIMESTAMP, NOT NULL |
| appointment\_time | Час прийому | TIMESTAMPTZ, NOT NULL |
| **Об’єкт DoctorAppointments/ПрийомУЛікаря** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| booked\_doctor\_id | Ідентифікатор бронювання лікаря | FOREIGN KEY (FK) |
| complaint | Скарга | До 2048 симв. |
| **Об’єкт Analyses/Аналізи** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| name | Назва аналізу | До 128 симв., NOT NULL UNIQUE |
| **Об’єкт AnalyseAppointments/НазначеніАналізи** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| analyse\_id | Ідентифікатор аналізу | FOREIGN KEY (FK) |
| doctor\_appintment\_id | Ідентифікатор прийому лікаря | FOREIGN KEY (FK) |
| is\_completed | Виконано | BOOLEAN |
| **Об'єкт PatientSickLeave/Больнічні** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| doctor\_appointment\_id | Ідентифікатор прийому лікаря | FOREIGN KEY (FK) |
| start\_date | Дата початку | DATE, DEFAULT CURRENT\_DATE, NOT NULL |
| end\_date | Дата закінчення | DATE, DEFAULT CURRENT\_DATE, NOT NULL |
| **Об'єкт Drugs/Ліки** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| **Властивість** | **Опис** | **Обмеження** |
| name | Назва препарату | До 128 симв., NOT NULL UNIQUE |
| description | Опис | До 2048 симв., NOT NULL |
| **Об’єкт PrescriptionDrugs/ВиписаніЛіки** | | |
| id | Ідентифікатор | PRIMARY KEY (PK) |
| drug\_id | Ідентифікатор препарату | FOREIGN KEY (FK) |
| doctor\_appointment\_id | Ідентифікатор прийому лікаря | FOREIGN KEY (FK) |

Між сутностямин наявні 2 типи зв’язків:

* 1:N — “один-до-бататьох”. Наявні в таблицях:
  + phone\_numbers та users
  + messages та users (user\_1\_id, user\_2\_id)
  + users\_allergen та users
  + users\_allergen та allergens
  + booked\_doctors та users (patient\_id, doctor\_id)
  + analyse\_appointments та analyses
  + prescription\_drugs та drugs
* 1:1 — “один-до-одного”. Наявні у таблицях:
  + users\_roles та users
  + doctor\_appointments та booked\_doctors
  + analyse\_appointments та doctor\_appointments
  + passwords та users
  + patient\_sick\_leave та doctor\_appointments
  + prescription\_drugs та doctor\_appointments

Для формалізації даного зв’язку первинний ключ однозв’язної сутності додається до схеми N-зв’язної сутності у якості завнішнього ключа. Схема бази даних наведена в додатку А.

**6. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Для роботи з базою даних обрано СКБД **PostgreSQL**, для Back-End частини використано мову програмування **C++** і бібліотеки **POSIX**, **jsoncpp**, **gtests**, **plog**, **cmake**, **pqxx** та стандартну бібліотеку шаблонів (**STL**). Для Front-End використовувалися нативні JavaScript, HTML та CSS. Весь процес розробки виконувався у текстовому редакторі **Visual Studio Code** з використанням системи контролю версій **Git**.

Сьогодні існує багато систем керування базами даних (Oracle, MongoDB, PostgreSQL та інші). Для **реалізації бази даних** цієї системи обрано діалект PostgreSQL з наступних причин:

* Об'єктно-реляційна модель даних, що використовується в PostgreSQL, є зрозумілою для кінцевого користувача. Це дозволяє зручно працювати з даними і зробити систему більш доступною для користувачів.
* Має гнучкий механізм управління правами користувачів бази даних за допомогою ролей. Це дозволяє точно налаштувати доступ до даних для різних користувачів і забезпечити безпеку інформації.
* Мова програмування plpgSQL, яка є розширенням стандартного SQL, підтримується в PostgreSQL. Вона дозволяє створювати ефективні збережені процедури і спрощує програмування логіки обробки даних.
* Постійно підтримується розробниками, що гарантує наявність оновлень, виправлення помилок та підтримку у майбутньому.

Back-End частина була розроблена з використанням мови програмування C++ і бібліотек POSIX, jsoncpp, gtests, plog, cmake, pqxx та стандартної бібліотеки шаблонів (STL). Це дозволяє створювати високопродуктивні та надійні серверні додатки.

Основні можливості та бібліотеки **Back-End:**

* POSIX: Працює на UNIX системах і забезпечує прямий доступ до команд системи. Це дозволяє значно економити ресурси і підвищує швидкість обробки та передачі даних.
* jsoncpp: Використовується для роботи з форматом даних JSON, що дозволяє легко парсити та генерувати JSON-дані.
* gtests: Бібліотека для створення та виконання модульних тестів, що забезпечує високу якість коду.
* plog: Легковага бібліотека для логування, яка дозволяє відстежувати події та помилки під час виконання програми. Являється базою для собственого логера.
* cmake: Система автоматичного складання, що дозволяє легко керувати процесом компіляції та налаштуванням проекту.
* pqxx: C++ бібліотека для роботи з PostgreSQL, що забезпечує зручний інтерфейс для виконання SQL-запитів та роботи з базою даних.

**Front-End** частина була розроблена з використанням нативних JavaScript, HTML та CSS, що дозволяє створювати інтерактивні і привабливі веб-інтерфейси. HTML використовується для структурування веб-сторінок, CSS для стилізації елементів, а JavaScript для додавання динамічної поведінки та взаємодії з сервером.

Для розробки використовувався текстовий редактор Visual Studio Code, що забезпечує зручне середовище для написання коду з підтримкою великої кількості плагінів та розширень. Система контролю версій Git використовувалася для управління змінами коду, що дозволяє відстежувати історію змін, співпрацювати з іншими розробниками і забезпечувати резервне копіювання коду.

**7. МОДЕЛЬ ПРОГРАМНОГО ДОДАТКУ**

У додатку використовуються три типи користувачів: адміністратор, лікар та пацієнт. Ці користувачі мають спільний доступ до одного додатку. Модель рівня представлення даних побудована у вигляді ієрархії сторінок, яка відображає послідовність переходів між сторінками в додатку. Цю ієрархію можна побачити на рисунках 7.1-3 відповідно.

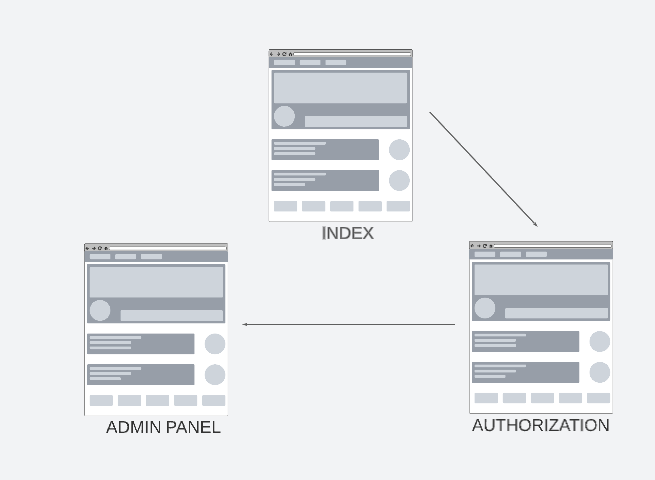


Рисунок 7.1 – ієрархія сторінок адміністратора

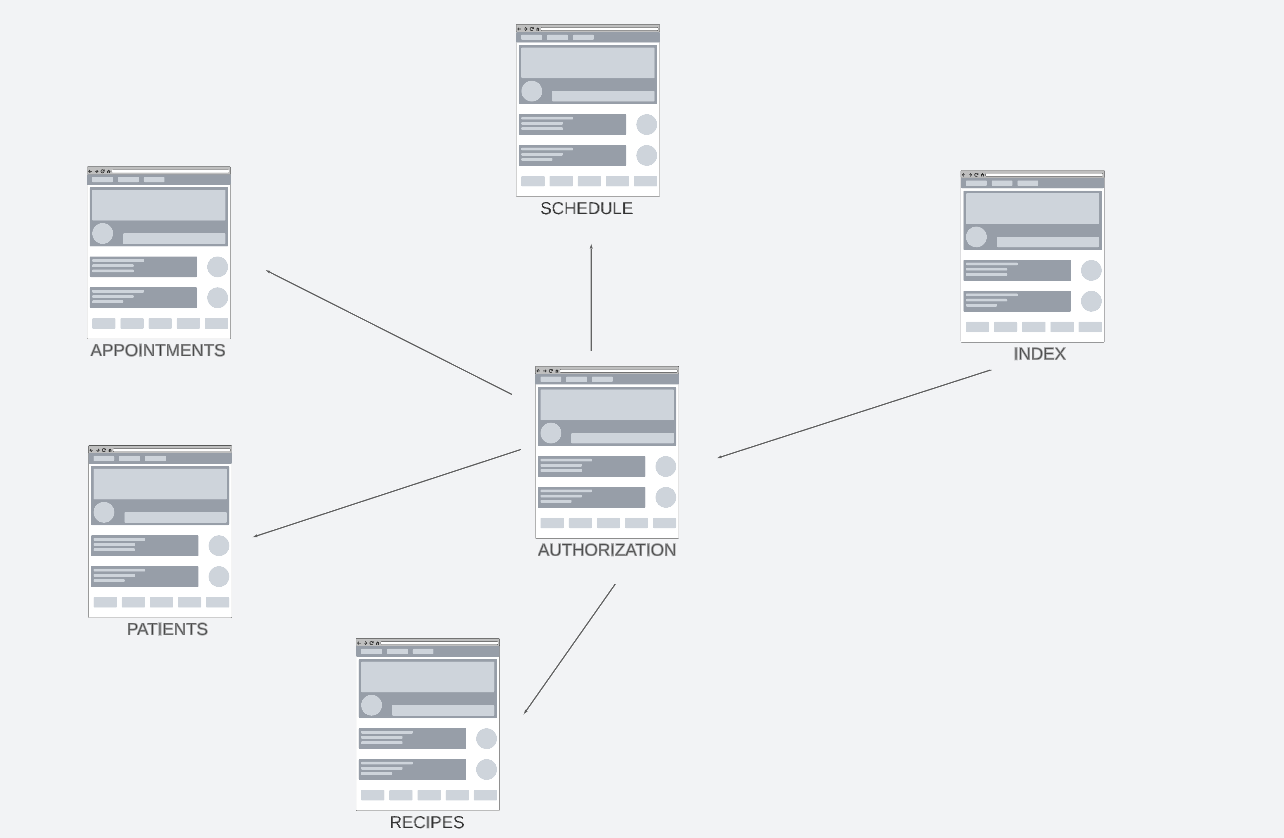


Рисунок 7.2 – ієрархія сторінок лікаря

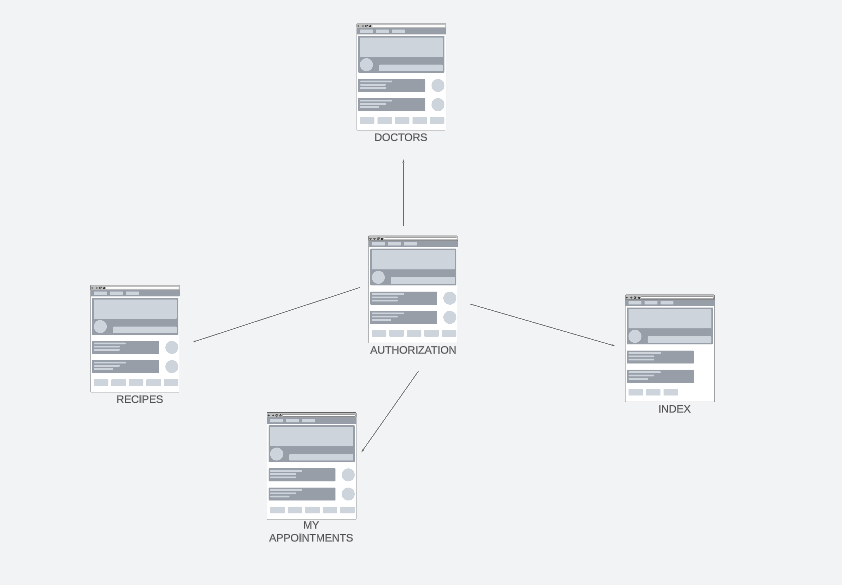


Рисунок 7.3 – ієрархія сторінок пацієнта

Шаблон МVC реалізується за допомогою відповідних класів (контролерів), які відповідають за регулювання відображення видів, авторизацію користувачів у системі, а також обробку даних до передачі запитів до сервера бази даних або після отримання відповідей від нього.

Моделі рівня прикладного компоненту та управління ресурсами можуть бути представлені у вигляді діаграм класів, які демонструють загальну структуру ієрархії класів системи, їх атрибути, методи та взаємозв'язки. На рисунку 7.4 наведена діаграма рівня прикладного компоненту для даного застосунку.

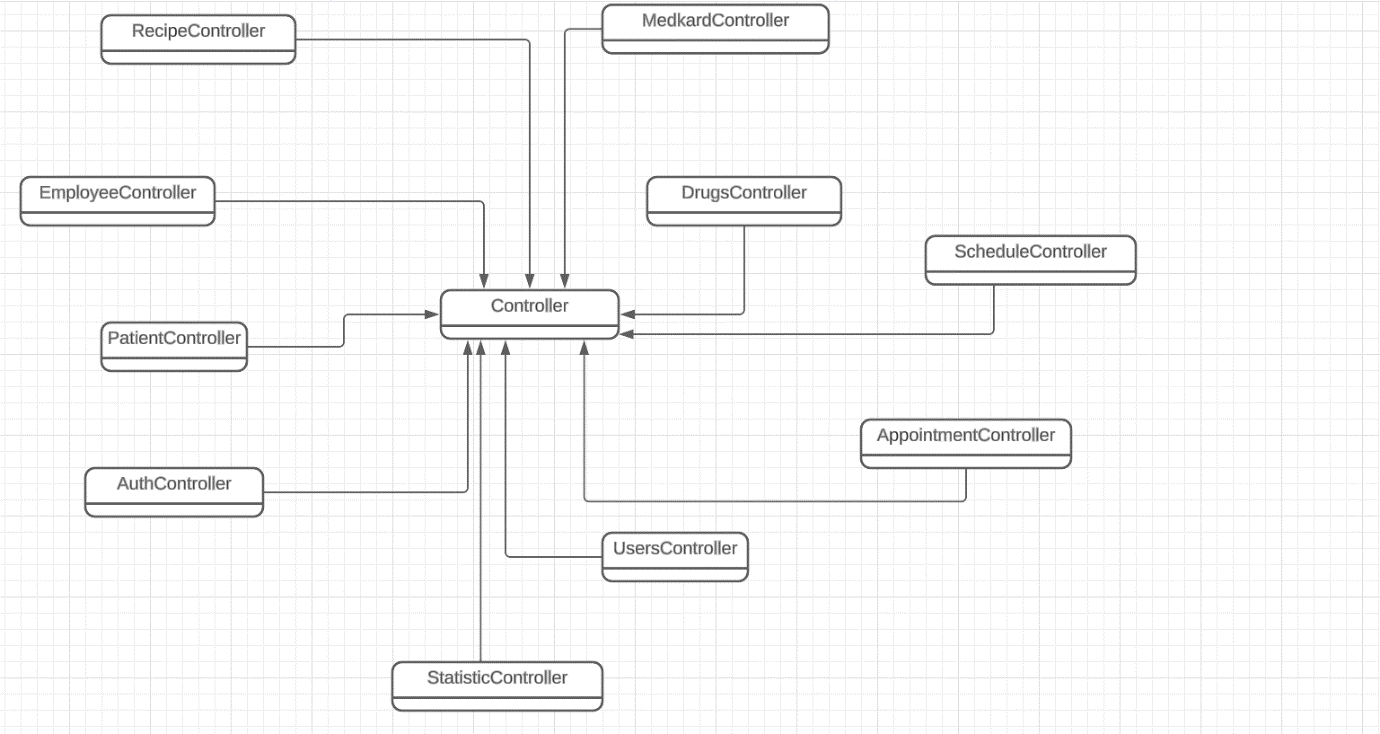


Рисунок 7.4 – Ієрархія класів – прикладного компоненту

На рисунку 7.5 наведена модель рівня управління ресурсами для застосунку клієнта. Класи на цьому рівні відповідають за забезпечення доступу до бази даних та обмін даними з сервером додатків. Вони реалізують методи, які дозволяють виконувати запити до бази даних, отримувати результати запитів та передавати їх до сервера додатків для подальшої обробки. Ці класи виконують важливу роль у взаємодії між клієнтською частиною та сервером додатків, забезпечуючи ефективну обробку та зберігання даних.

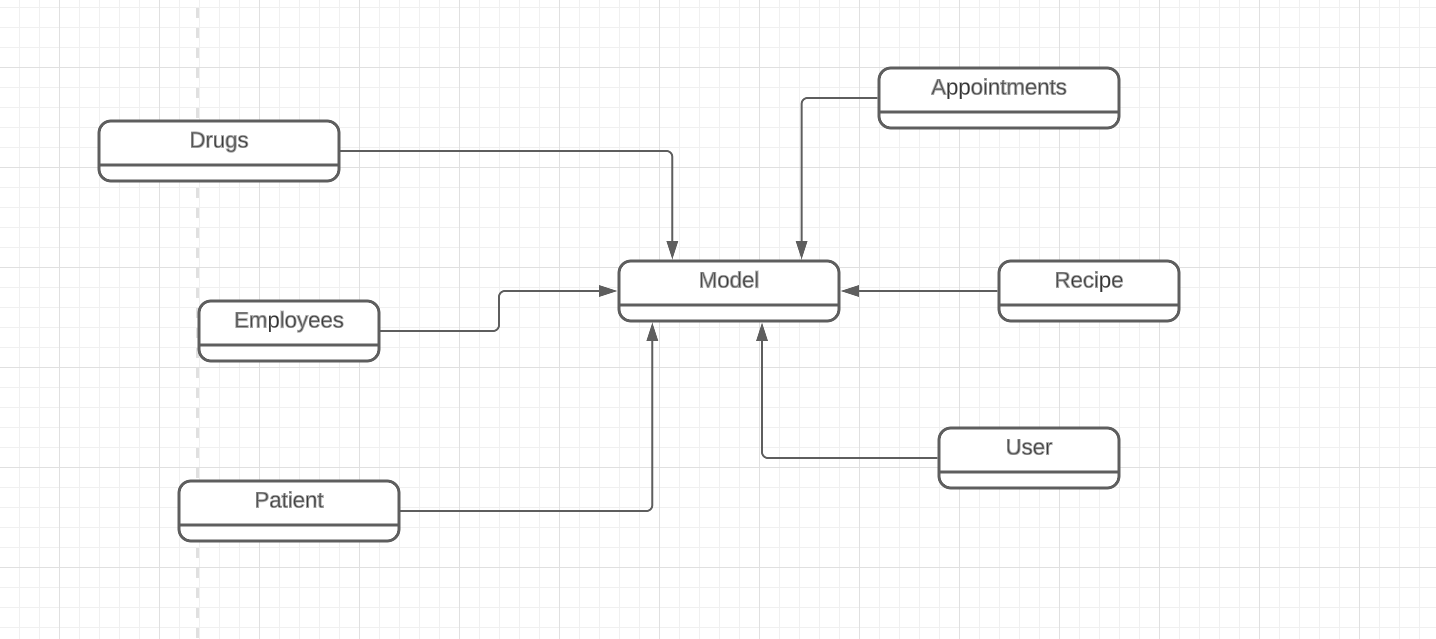


Рисунок 7.5 – Ієрархія класів – рівень управління ресурсами

**8. СТВОРЕННЯ БАЗИ ДАНИХ**

В цьому розділі описані об’єкти бази даних. Повний код створення бази даних наведено у додатку Б.

Наша база даних має на даний час наступні таблиці:

1. Users – Користувачі;
2. Passwords – Паролі;
3. PhoneNumbers – ТелефоніНомери;
4. Roles – Ролі;
5. UserRoles – РоліКористувачів;
6. Messages – Повідомлення;
7. Allergens – Алергени;
8. UserAllergen – АлергениКористувачів;
9. BookedDoctors – ЗаписаніЛікарі;
10. DoctorAppointments – ПрийомиУЛікарів;
11. Analyses – Аналізи;
12. AnalyseAppointments – ПризначенняАналізів;
13. PatientSickLeave – ЛікарняніЛистиПацієнтів;
14. Drugs – Ліки;
15. PrescriptionDrugs – ПризначеніЛіки;

Створення таблиці на прикладе “booked\_doctors”:

CREATE TABLE booked\_doctors (

id BIGSERIAL,

patient\_id BIGINT NOT NULL,

doctor\_id BIGINT NOT NULL,

book\_time TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL,

appointment\_time TIMESTAMPTZ NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (patient\_id) REFERENCES users(id),

FOREIGN KEY (doctor\_id) REFERENCES users(id),

UNIQUE(patient\_id, doctor\_id, appointment\_time),

CONSTRAINT valid\_patient\_role CHECK (is\_valid\_patient\_role(patient\_id)),

CONSTRAINT valid\_doctor\_role CHECK (is\_valid\_doctor\_role(doctor\_id))

);

Створення таблиці відбувається за допомогою команди **CREATE TABLE**, далі вказується ім'я таблиці. Кожному полю таблиці зіставляються тип даних обраної СУБД (PostgreSQL) і обмеження цілісності. Поле id є **первинним ключем (PRIMARY KEY)** і має тип **BIGSERIAL** (цей тип не є справжнім типом даних, а являє собою зручний спосіб створення унікального ідентифікатора шляхом збільшення попереднього значення в даному стовпці на вказане значення, за замовчуванням на 1). **NOT NULL** при визначенні полів таблиці означає, що вони не можуть містити порожні значення. **CHECK ((is\_valid\_patient\_role(patient\_id))** при визначенні поля patient\_id вказує, що значення у цьому полі має бути id корситувача з ролью пацієнта. **UNIQUE** при визначенні полів patient\_id, doctor\_id, appointment\_time вказує на те, що комбінація даних полів не має повторюватися у таблиці.

**Представлення (views)** – це віртуальні таблиці, які не зберігають дані самі по собі, але зберігають запити для отримання даних з однієї або декількох таблиць. Вони надають спосіб перегляду і маніпулювання даними безпосередньо, дозволяючи абстрагувати складність базових запитів. В нашій базі даних приведені наступні представлення:

* **max\_diseases\_per\_doctor** – відображає список хвороб для максимальної кількості пацієнтів кожного лікаря.
* **doctor\_visit\_trend** – відображає тренд кількості візитів до лікаря (за датами візитів).

Приклад створення “max\_diseases\_per\_doctor”:

CREATE VIEW max\_diseases\_per\_doctor AS

WITH doctor\_patient\_disease AS (

SELECT

bd.doctor\_id,

da.complaint AS disease,

COUNT(DISTINCT bd.patient\_id) AS patient\_count

FROM

booked\_doctors bd

JOIN

doctor\_appointments da ON bd.id = da.booked\_doctor\_id

GROUP BY

bd.doctor\_id, da.complaint

),

max\_patient\_disease AS (

SELECT

doctor\_id,

disease,

patient\_count,

ROW\_NUMBER() OVER (PARTITION BY doctor\_id ORDER BY patient\_count DESC, disease) AS rn

FROM

doctor\_patient\_disease

)

SELECT

d.id AS doctor\_id,

d.name AS doctor\_name,

d.surname AS doctor\_surname,

d.middle\_name AS doctor\_middle\_name,

m.disease AS most\_popular\_disease,

m.patient\_count

FROM

max\_patient\_disease m

JOIN

users d ON m.doctor\_id = d.id

WHERE

m.rn = 1;

Створення або заміна представлення з назвою max\_diseases\_per\_doctor відбувається за допомогою команди **CREATE VIEW**. У виразі WITH створюються два CTE (Common Table Expressions). Перший CTE називається doctor\_patient\_disease і вибирає дані про лікарів, захворювання та кількість пацієнтів з цим захворюванням. Для цього використовуються поля doctor\_id, complaint як disease, та кількість унікальних patient\_id як patient\_count. Виконується з'єднання таблиць booked\_doctors і doctor\_appointments, і групування даних за doctor\_id та disease. Другий CTE називається max\_patient\_disease і визначає найбільш поширені захворювання для кожного лікаря. Використовується функція **ROW\_NUMBER() OVER**, яка присвоює номери рядкам у кожній групі лікарів, відсортованих за кількістю пацієнтів у спадному порядку та за назвою захворювання. Розділення даних відбувається за doctor\_id.

Основний **SELECT** вибирає дані з max\_patient\_disease та з'єднує їх з таблицею users, щоб отримати інформацію про лікарів (ім'я, прізвище, по батькові). Вибираються тільки ті записи, де **rn = 1**, тобто найбільш поширене захворювання для кожного лікаря. Виконується з'єднання з таблицею users для отримання додаткової інформації про лікарів. Записи сортуються за кількістю пацієнтів із захворюванням у спадному порядку, а також за назвою захворювання.

Умови з'єднань вказані в розділі **ON**, де вказується спільний стовпець між таблицями для з'єднання. Завдяки з'єднанням можна отримувати значення з інших таблиць, які відповідають умовам з'єднань.

**Тригери** – це спеціальні об'єкти, які автоматично виконуються при виникненні певних подій у таблиці чи поданні. Тригери використовуються для підтримки цілісності даних, автоматизації завдань і забезпечення виконання бізнес-правил без необхідності вносити зміни в прикладний код. Основні характеристики тригерів:

* **Події:** Тригери можуть бути викликані різними подіями:
  + **INSERT:** Додавання нового рядка до таблиці.
  + **UPDATE:** Оновлення існуючого рядка в таблиці.
  + **DELETE:** Видалення рядка з таблиці.
* **Час спрацьовування:**
  + **BEFORE:** Тригери, що спрацьовують до виконання операції.
  + **AFTER:** Тригери, що спрацьовують після виконання операції.
  + **INSTEAD OF:** Тригери, що замінюють собою операцію (зазвичай використовуються для подань).
* **Рівні:**
  + **ROW:** Тригери, що спрацьовують для кожного окремого рядка, який піддається зміні.
  + **STATEMENT:** Тригери, що спрацьовують один раз для всієї операції.

В базі є наступні тригери:

* **trigger\_check\_allergens** – забороняє виписувати ліки, в описі яких є алерген пацієнта

Приклад створення тригера “trigger\_check\_allergens”:

CREATE OR REPLACE FUNCTION check\_allergens() RETURNS TRIGGER AS $$

DECLARE

patient\_id BIGINT;

patient\_allergens TEXT[];

allergen TEXT;

drug\_description TEXT;

BEGIN

SELECT bd.patient\_id INTO patient\_id

FROM booked\_doctors bd

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

WHERE da.id = NEW.doctor\_appointment\_id;

SELECT description INTO drug\_description

FROM drugs

WHERE id = NEW.drug\_id;

SELECT ARRAY\_AGG(a.name)

INTO patient\_allergens

FROM users\_allergen ua

JOIN allergens a ON ua.allergen\_id = a.id

WHERE ua.user\_id = patient\_id;

IF patient\_allergens IS NOT NULL THEN

FOR i IN ARRAY\_LOWER(patient\_allergens, 1) .. ARRAY\_UPPER(patient\_allergens, 1) LOOP

allergen := patient\_allergens[i];

-- Отладочное сообщение для проверки значений

RAISE NOTICE 'Checking allergen: % in drug description: %', allergen, drug\_description;

IF position(lower(allergen) IN lower(drug\_description)) > 0 THEN

RAISE EXCEPTION 'Cannot prescribe drug with allergen %', allergen;

END IF;

END LOOP;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_check\_allergens

BEFORE INSERT ON prescription\_drugs

FOR EACH ROW

EXECUTE FUNCTION check\_allergens();

Створення або заміна функції з назвою check\_allergens відбувається за допомогою команди **CREATE OR REPLACE FUNCTION**. Функція повертає тип **TRIGGER**, що означає, що вона буде викликатися автоматично при вставці даних у таблицю prescription\_drugs. У блоці **DECLARE** визначаються змінні, що будуть використовуватися в функції: patient\_id типу **BIGINT** для збереження ідентифікатора пацієнта, patient\_allergens типу TEXT[] для збереження масиву алергенів пацієнта, allergen типу TEXT для ітерації через алергени, та drug\_description типу TEXT для збереження опису ліків. У блоці **BEGIN** починається виконання функції. Спочатку виконується запит SELECT bd.patient\_id INTO patient\_id, який отримує ідентифікатор пацієнта з таблиць booked\_doctors і doctor\_appointments на основі NEW.doctor\_appointment\_id. Далі виконується запит SELECT description INTO drug\_description, який отримує опис ліків з таблиці drugs на основі NEW.drug\_id. Після цього виконується запит **SELECT ARRAY\_AGG(a.name) INTO patient\_allergens**, який отримує масив алергенів пацієнта з таблиць users\_allergen і allergens на основі patient\_id. У блоці розгалуження **IF** перевіряється, чи є масив алергенів пацієнта непорожнім (patient\_allergens IS NOT NULL). Якщо масив непорожній, то за допомогою циклу **FOR** відбувається ітерація через алергени. Для кожного алергену allergen перевіряється, чи міститься він у описі ліків за допомогою функції **position(lower(allergen) IN lower(drug\_description)) > 0**. Якщо алерген знайдено, викликається виняток за допомогою команди **RAISE EXCEPTION** з повідомленням про виявлений алерген. Якщо алергени не знайдені в описі ліків, функція повертає новий запис **NEW**.

Створення триггера з назвою trigger\_check\_allergens відбувається за допомогою команди **CREATE TRIGGER**. Тригер спрацьовує перед вставкою у таблицю prescription\_drugs. Для кожного рядка виконується функція check\_allergens(). Тобто, кожного разу, коли відбувається вставка даних у таблицю prescription\_drugs, виконується функція check\_allergens().

**Збережені функції (stored procedures)** є потужним інструментом для виконання повторюваних задач в базі даних. Вони дозволяють об'єднати серію SQL-інструкцій в одну логічну одиницю і виконувати її при необхідності. Збережені функції можуть мати параметри входу та виходу, а також можуть повертати значення або таблиці.

* **get\_patients\_by\_doctor\_and\_date** – повертає список пацієнтів, яких обстежував вказаний лікар у вказану дату (id, ім'я, прізвище, по батькові, email).
* **get\_top\_doctors\_last\_month** – повертає список лікарів, ранжованих за кількістю пацієнтів, яких вони обстежили за останній місяць (id, ім'я, прізвище, по батькові, email, кількість пацієнтів).
* **get\_patients\_with\_heart\_disease\_drugs** – повертає список пацієнтів, яким було призначено ліки від серцевих захворювань (id, ім'я, прізвище, по батькові, email).
* **get\_patient\_count\_per\_doctor\_last\_month** – рахує кількість пацієнтів, яких кожен лікар обстежив за останній місяць (id, ім'я, прізвище, по батькові, email, кількість пацієнтів).
* **get\_medications\_and\_tests\_for\_patient** – повертає загальний список ліків і тестів, які були призначені вказаному пацієнту за останні шість місяців без повторень (назва, тип).
* **check\_user\_login** – перевіряє логін користувача (email) та пароль і повертає `TRUE`, якщо вони співпадають, або `FALSE`, якщо ні.
* **get\_last\_appointments\_by\_surname** – повертає останні прийоми лікарів для пацієнта з вказаним прізвищем (ім'я лікаря, час прийому, скарга).
* **get\_clinic\_statistics** – повертає статистику по клініці за вказаний проміжок часу (загальна кількість пацієнтів, кількість інфекційних захворювань, гіпертонії, діабету, захворювань легень, загальна кількість виданих лікарняних, загальна кількість днів лікарняних).
* **check\_allergens** – тригерна функція для перевірки, чи містять призначені ліки алергени, на які у пацієнта є алергія, і піднімає виключення у разі їх наявності.

Приклад створення функції “get\_patient\_count\_per\_doctor\_last\_month”:

CREATE OR REPLACE FUNCTION get\_patient\_count\_per\_doctor\_last\_month()

RETURNS TABLE (

id BIGINT,

name VARCHAR(50),

surname VARCHAR(50),

middle\_name VARCHAR(50),

email custom\_email,

clients\_count BIGINT

) AS $$

DECLARE

last\_month\_start DATE := date\_trunc('month', CURRENT\_DATE) - interval '1 month';

last\_month\_end DATE := date\_trunc('month', CURRENT\_DATE);

BEGIN

RETURN QUERY

SELECT u.id, u.name, u.surname, u.middle\_name, u.email, COUNT(bd.patient\_id) AS patients\_count

FROM users u

JOIN booked\_doctors bd ON u.id = bd.doctor\_id

WHERE bd.appointment\_time >= last\_month\_start

AND bd.appointment\_time < last\_month\_end

GROUP BY u.id;

END;

$$ LANGUAGE plpgsql;

/\* Description:

\* Merge: Give a total list of medications and tests that have been prescribed for the specified

\* patient in the last six months, without repeats, with status.

\*/

CREATE OR REPLACE FUNCTION get\_medications\_and\_tests\_for\_patient(patient\_id BIGINT)

RETURNS TABLE

(

name varchar(128),

type varchar(20)

)

AS

$$

BEGIN

RETURN QUERY

-- Список анализов и лекарств которые были выписаны определенному пациенту за последние 6 месяцев

(

SELECT CAST(analyses.name AS varchar(128)) AS name, CAST('Анализы' AS varchar(20)) AS type

FROM users

INNER JOIN booked\_doctors ON booked\_doctors.patient\_id = users.id

INNER JOIN doctor\_appointments ON doctor\_appointments.booked\_doctor\_id = booked\_doctors.id

INNER JOIN analyse\_appointments ON analyse\_appointments.doctor\_appointment\_id = doctor\_appointments.id

INNER JOIN analyses ON analyses.id = analyse\_appointments.analyse\_id

WHERE users.id = get\_medications\_and\_tests\_for\_patient.patient\_id

AND booked\_doctors.appointment\_time >= CURRENT\_DATE - INTERVAL '6 months'

UNION

SELECT CAST(drugs.name AS varchar(128)) AS name, CAST('Лекарства' AS varchar(20)) AS type

FROM users

INNER JOIN booked\_doctors ON booked\_doctors.patient\_id = users.id

INNER JOIN doctor\_appointments ON doctor\_appointments.booked\_doctor\_id = booked\_doctors.id

INNER JOIN prescription\_drugs ON prescription\_drugs.doctor\_appointment\_id = doctor\_appointments.id

INNER JOIN drugs ON drugs.id = prescription\_drugs.drug\_id

WHERE users.id = get\_medications\_and\_tests\_for\_patient.patient\_id

AND booked\_doctors.appointment\_time >= CURRENT\_DATE - INTERVAL '6 months'

);

END;

$$ LANGUAGE plpgsql;

**9. ПРОГРАМНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОГО ДОДАТКУ**

Клієнтська частина нашого веб-застосунку розроблена з використанням стандартних веб-технологій: **HTML**, **CSS** та **JavaScript**. Вона забезпечує інтуїтивно зрозумілий інтерфейс для користувачів, дозволяючи їм реєструватися, входити в систему та керувати своїми профілями. Основна увага приділена модульності та повторному використанню компонентів для спрощення підтримки та розширення застосунку.

Структура проекту організована таким чином, щоб код був легко читабельним та підтримуваним. Основні директорії включають “bars” для загальних елементів інтерфейсу, “profile\_page” для сторінок, пов'язаних із профілем користувача, “scripts” для JavaScript-файлів, та “src” і “fonts” для ресурсів, таких як зображення та шрифти.

* **bars:** Містить HTML і CSS файли для верхньої панелі навігації, використовуваної на всіх сторінках.
* **profile\_page:** Містить сторінки профілю користувача, реєстрації та входу, а також відповідні стилі.
* **scripts:** Містить JavaScript-файли, що забезпечують функціональність застосунку, такі як обробка автентифікації та включення HTML-компонентів.
* **src:** Містить зображення, використовувані в проекті.
* **fonts:** Містить шрифти, використовувані в проекті.

Верхня панель навігації (`top\_bar.html` і `top\_bar.css`) забезпечує єдиний зовнішній вигляд на всіх сторінках застосунку. Вона включає контактну інформацію та посилання на основні розділи сайту. Це дозволяє користувачам легко переміщатися між різними сторінками:

html

<div class="header-top">

<div class="contact-info">

<span>CALL US: +41 678-678-456 | +41 678-678-903</span>

<span>594 Sunset Ave. Manahawkin, NJ 08050</span>

<span>Email: jenniewilkerson@info.com</span>

</div>

</div>

<div class="header-main">

<img src="../src/logo.svg" alt="Логотип" class="logo">

<nav class="nav">

<ul>

<li><a href="../index.html">HOME</a></li>

<li><a href="../profile\_page/profile.html">PROFILE</a></li>

<li><a href="#">BLOG</a></li>

<li><a href="#">TEAM</a></li>

<li><a href="#">FAQ</a></li>

<li><a href="#">PRESS</a></li>

<li><a href="#">TESTIMONIALS</a></li>

<li><a href="#">CONTACT</a></li>

</ul>

</nav>

<a href="#" class="btn-appointment">Book an Appointment</a>

</div>

Кожна з цих сторінок має свою структуру та стилі, щоб забезпечити зручність використання та відповідність дизайну. Сторінки реєстрації та входу включають форми для введення даних користувачами, а сторінка профілю відображає особисту та медичну інформацію користувача. Нижче зображена частина коду profile.html.

```html

<div class="container">

<div class="profile-header">

<img src="../src/default\_avatar.jpg" alt="User Avatar" class="avatar">

<h1>Profile</h1>

<h2>Welcome, [User's Name]</h2>

</div>

<div class="profile-content">

<div class="personal-info">

<h3>Personal Information</h3>

<p><strong>Name:</strong> [User's Name]</p>

<p><strong>Email:</strong> [User's Email]</p>

<p><strong>Phone:</strong> [User's Phone Number]</p>

</div>

<div class="medical-info">

<h3>Medical Card</h3>

<p><strong>Allergens:</strong> [User's Allergens] <a href="#" class="more-link">More...</a></p>

<h4>Booked Doctors</h4>

<ul>

<li>Doctor: [Doctor's Name], Appointment: latest... <a href="#" class="more-link">More...</a></li>

</ul>

<h4>Doctor Appointments</h4>

<ul>

<li>Complaint: latest..., Date: latest... <a href="#" class="more-link">More...</a></li>

</ul>

<h4>Prescription Drugs</h4>

<ul>

<li>Drug: latest..., Description: latest... <a href="#" class="more-link">More...</a></li>

</ul>

</div>

</div>

</div>

<button id="logoutButton" class="btn-logout">Logout</button>

Для обробки входу та реєстрації використовується файл `auth.js`. Цей файл містить логіку для відправки даних форм на сервер та обробки відповідей. При успішному вході дані користувача зберігаються в `localStorage`, що дозволяє визначити стан входу при подальших запитах:

document.addEventListener('DOMContentLoaded', function() {

const signinForm = document.getElementById('signinForm');

if (signinForm) {

signinForm.addEventListener('submit', async (event) => {

event.preventDefault();

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

try {

const response = await fetch('/api/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ email, password }),

credentials: 'include'

});

const data = await response.json();

if (response.ok && data.is\_login\_success) {

localStorage.setItem('isLoggedIn', 'true');

window.location.href = 'profile.html';

} else {

alert(data.message || 'Login failed');

}

} catch (error) {

console.error('Error:', error);

}

});

}

});

Для перенаправлення користувачів залежно від їх стану входу використовується файл `redirect.js`. Він перевіряє, чи збережено статус входу в `localStorage`, і перенаправляє користувача на відповідну сторінку:

document.addEventListener('DOMContentLoaded', function() {

const isLoggedIn = localStorage.getItem('isLoggedIn');

if (isLoggedIn === 'true') {

window.location.href = 'profile.html';

} else {

window.location.href = 'signin.html';

}

});

```

Клієнтська частина веб-застосунку реалізована з урахуванням принципів модульності та повторного використання коду. Це дозволяє легко підтримувати та розширювати функціональність застосунку. Використання сучасних технологій та підходів забезпечує високу продуктивність і зручність для користувачів.

**10. ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ**

При розробці веб-сервера було обрано сучасні та потужні технології, що забезпечують високу продуктивність, гнучкість і надійність. Основною мовою програмування було обрано C++17, що забезпечує розширений набір функцій та можливостей у порівнянні з попередніми стандартами. C++17 включає покращення в області роботи з пам'яттю, багатопоточності та іншими критично важливими аспектами для серверних додатків.

Для реалізації основної логіки та роботи з даними активно використовуються бібліотеки стандартного шаблонного бібліотечного інтерфейсу (**STL**), які надають широкий спектр готових рішень для роботи з контейнерами, алгоритмами та іншими базовими структурами.

Для забезпечення сумісності з різними операційними системами та ефективної роботи з системними ресурсами було використано **POSIX API**. Це дозволяє реалізувати високопродуктивні та стабільні рішення для управління потоками, синхронізацією, вхідними/вихідними операціями та іншими низькорівневими аспектами системної роботи.

POSIX (Portable Operating System Interface) - це набір стандартів, розроблених IEEE для забезпечення сумісності між різними операційними системами. POSIX визначає стандарти для роботи з системними викликами, такими як управління процесами, потоками, файловими системами та мережею. У контексті програмування на C++, POSIX надає низку API для управління потоками, синхронізацією, асинхронними вхідними/вихідними операціями та іншими аспектами, що є критично важливими для мережевих додатків.

Для роботи з базами даних було обрано бібліотеку **pqxx**, яка надає зручний інтерфейс для взаємодії з PostgreSQL. Це дозволяє ефективно виконувати SQL-запити, працювати з транзакціями та іншими функціями бази даних.

Система автоматизованого збирання **CMake** була використана для управління процесом компіляції та налаштування проекту. Це забезпечує гнучкість і зручність при роботі з великими проектами, дозволяючи легко налаштовувати залежності та параметри збірки.

Для забезпечення якості коду та тестування було використано бібліотеку **gtest** (Google Test), що дозволяє створювати модульні та інтеграційні тести для перевірки коректності роботи компонентів сервера.

Компіляція проекту виконується за допомогою компілятора **gcc**, який є потужним і широко використовуваним інструментом для компіляції C++ коду. Він забезпечує високу продуктивність та сумісність з багатьма платформами.

Для логування та відстеження подій у додатку використовується бібліотека **plog**, яка надає простий та ефективний спосіб зберігання логів, що допомагає в налагодженні та моніторингу роботи сервера. Бібліотека plog була використана для написання обертки для особистого логера, де формат визову лог методів був перероблен у більш зручний формат.

Для роботи з JSON-даними було обрано бібліотеку **jsoncpp**, що дозволяє зручно парсити, генерувати та маніпулювати JSON-об'єктами, забезпечуючи взаємодію сервера з клієнтами через сучасні API.

Всі ці технології разом створюють надійну та ефективну основу для розробки високопродуктивного веб-сервера, що відповідає сучасним вимогам до продуктивності, гнучкості та надійності.

Для розуміння того, як функціонує веб-сервер на C++, важливо розглянути ключові компоненти, що забезпечують його роботу. Кожен компонент має свою специфічну роль та відповідальність, що в сукупності дозволяє серверу обробляти запити, надавати ресурси та підтримувати стабільну роботу. Основними компонентами є **ServerStarter**, **ClientHandler** та **ContextHandler**.

**ServerStarter** відповідає за ініціалізацію та запуск веб-сервера. Його головна мета – налаштувати всі необхідні параметри для роботи сервера, такі як порт для прослуховування вхідних з'єднань, максимальна кількість одночасних підключень, а також інші конфігураційні параметри, що забезпечують стабільну роботу сервера. Після налаштування параметрів, ServerStarter створює та налаштовує об'єкти для обробки мережевих з'єднань і запускає головний цикл обробки запитів. Цей компонент також відповідає за запуск потоку або пулу потоків, що дозволяє серверу обробляти запити асинхронно та ефективно розподіляти навантаження.

**ClientHandler** має ключову роль в обробці індивідуальних з'єднань з клієнтами. Коли клієнт встановлює з'єднання з сервером, ClientHandler приймає вхідні HTTP запити, розбирає HTTP пакети та після цього починає взаємодіяти з іншими компонентами програми для обробки запиту. Він аналізує метод запиту, URL, заголовки та тіло запиту, передаючи розібрані дані до ContextHandler для подальшої обробки.

**ContextHandler** відповідає за обробку логіки бізнес-процесів, відкриття нових сторінок та надсилання необхідних ресурсів клієнту. Після отримання запиту від **ClientHandler**, ContextHandler визначає необхідні дії на основі URL та методу запиту. Він виконує всі необхідні бізнес-операції, такі як перевірка прав доступу, взаємодія з базою даних для отримання або збереження даних, та формування відповіді. ContextHandler також відповідає за підготовку всіх необхідних ресурсів (HTML файли, зображення, скрипти) для відправки клієнту, що дозволяє забезпечити коректну роботу веб-сторінки.

ServerStarter, ClientHandler та ContextHandler тісно взаємодіють один з одним, забезпечуючи ефективну обробку HTTP запитів та відповідей. ServerStarter запускає сервер та налаштовує потоки для обробки запитів. Коли клієнт надсилає запит, ClientHandler приймає та розбирає його, після чого взаємодіє з ContextHandler для виконання необхідних бізнес-операцій. ContextHandler, у свою чергу, обробляє запит, виконує логіку бізнес-процесів, відкриває нові сторінки та готує всі необхідні ресурси для клієнта, після цього відправляє її назад клієнту. Ця чітко визначена структура та розподіл ролей дозволяють досягти високої продуктивності, масштабованості та надійності веб-сервера.

**11. ІНСТРУКЦІЯ КОРИСТУВАЧА З ІЛЮСТРАЦІЯМИ**

В цьому розділі розглядається спосіб вирішення завдань користувачів за допомогою розробленого інтерфейсу.

При переході на сайт заявляється головна сторінка. Рисунок10.1

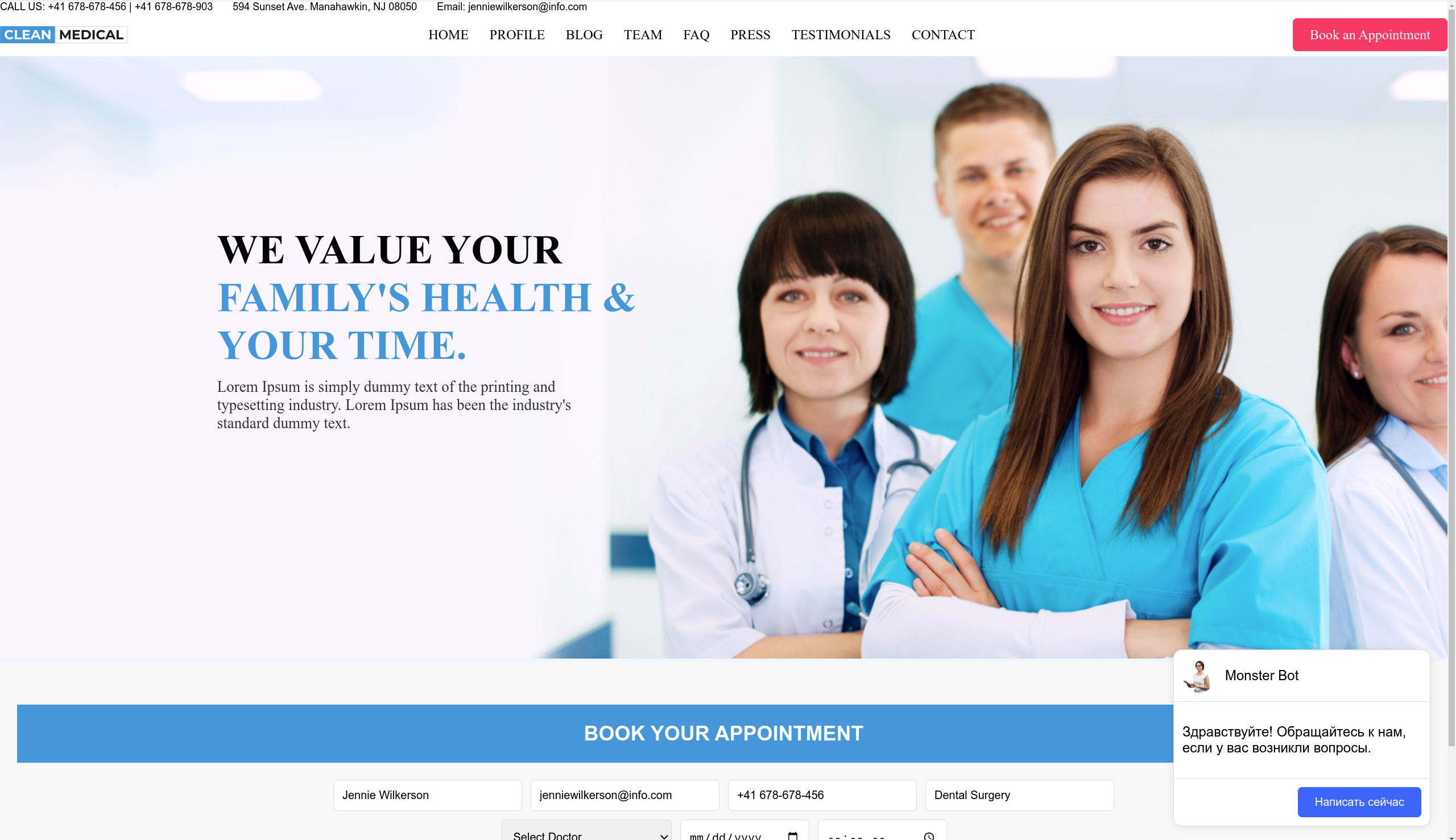


Рисунок 10.1 – головна сторінка веб-сайту

Щоб увійти у систему потрібно ввести дані користувача такі як пошта та пароль (рис 10.2). Якщо ж цих данних ще немає треба зареєструватися (рис 10.3). Після реєстрації вас перекидує на сторінку профіля користувача. Там вказані основні дані, такі як ПІБ, номер телефону, електрона адреса. Також має виводитися останні дані із колонки алергій, запису до лікарів, прийоми та виписані препарати (рис 10.4). Якщо нажати на кнопку more..., то можна дізнатися усю інформацію

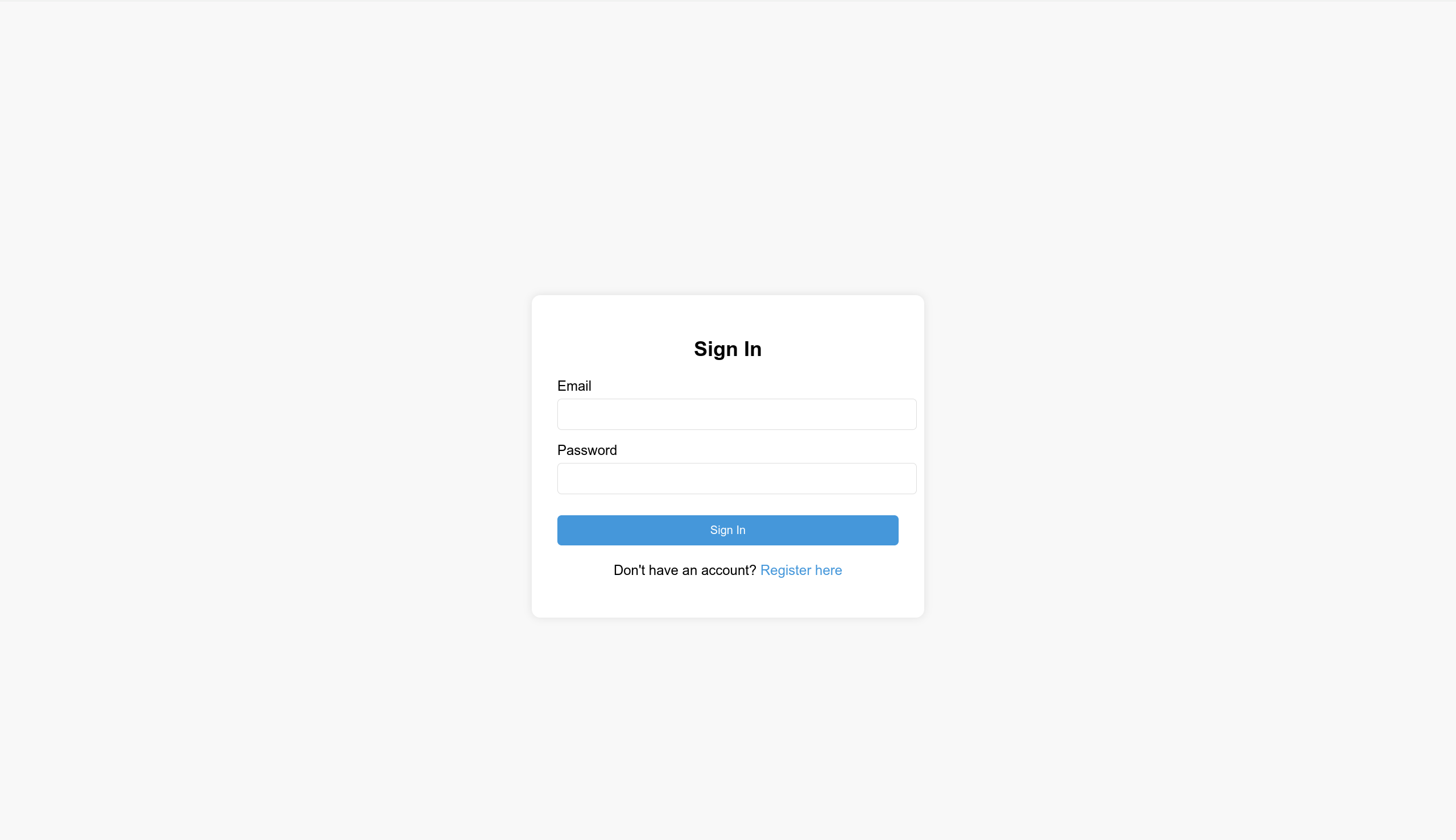


Рисунок 10.2 – вікно входу в профіль

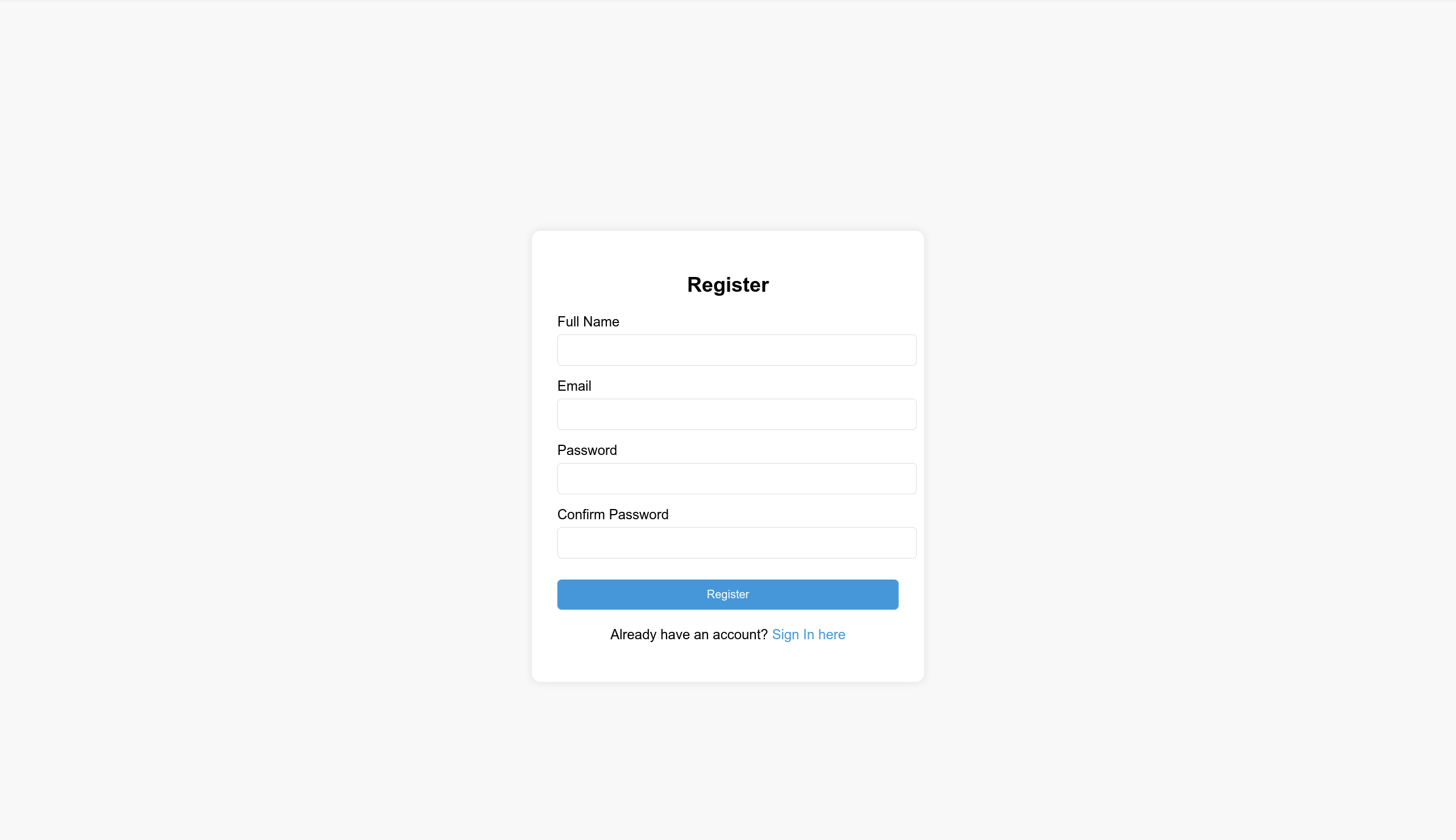


Рисунок 10.3 – вікно реєстрації

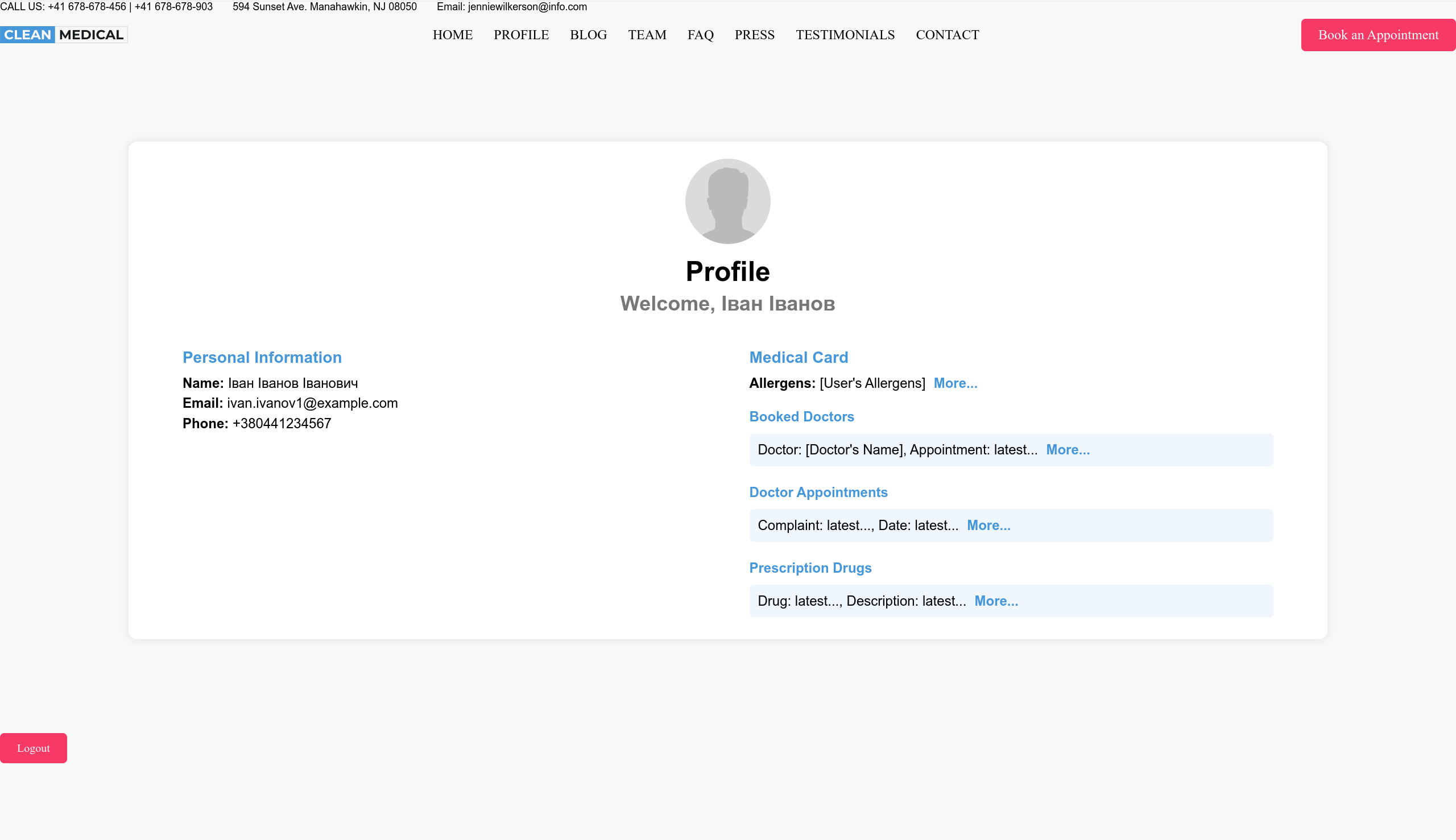


Рисунок 10.4 – профіль користувача

**ПЕРЕЛІК ПОСИЛАНЬ**

* Документація по C++ [Електронний ресурс]-Режим доступу: https://en.cppreference.com/w/
* Документація по JavaScript [Електронний ресурс]-Режим доступу: <https://www.w3schools.com/js/>
* Документація POSIX API [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E19048-01/chorus5/806-6897/auto1/index.html
* PostgreSQL : Документация: 9.6: CREATE TRIGGER [Електронний ресурс] – Режим доступу: <https://postgrespro.ru/docs/postgresql/9.6/sql-createtrigger>
* PostgresSQL: Документація: CREATE PRODUCE [Елктроний ресурс]-Режим доступу: <https://postgrespro.ru/docs/postgresql/11/sql-createprocedure>
* Малахов Є.В., Проектування баз даних та їх реалізація засобами стандартного SQL та PostgreSQL: Навч. посіб. для студ. вищих навч. закладів / Є.В. Малахов, О.А. Блажко, М.Г. Глава // Одеса: ВМВ, 2012. – 248 с.
* Документація: Шаблон MVC [Елктроний ресурс]-Режим доступу: <https://wiki.merionet.ru/servernye-resheniya/100/chto-takoe-arhitektura-mvc-model-view-controller/>

**Додаток А**

**СХЕМА БАЗИ ДАНИХ**



**Додаток Б**

**ЗАПИТИ ДЛЯ СТВОРЕННЯ БАЗИ ДАНИХ**

---------------------- INCLUDES ----------------------

CREATE EXTENSION IF NOT EXISTS citext;

---------------------- CUSTOM DATA TYPES ----------------------

CREATE DOMAIN custom\_email AS citext

CHECK ( value ~ '^(?!.\*[.\_-]$)[a-zA-Z0-9\_.-]+@[a-zA-Z0-9]+(?:[a-zA-Z0-9-]\*[a-zA-Z0-9])?(?:\.[a-zA-Z0-9]+(?:[a-zA-Z0-9-]\*[a-zA-Z0-9])?)\*$');

---------------------- CUSTOM FUNCTIONS ----------------------

CREATE OR REPLACE FUNCTION is\_valid\_patient\_role(patient\_id BIGINT) RETURNS BOOLEAN AS $$

BEGIN

-- ur.role\_id = 1, where 1 - is a patient role ID;

RETURN EXISTS (SELECT 1 FROM users\_roles AS ur WHERE ur.user\_id = patient\_id AND ur.role\_id = 1);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION is\_valid\_doctor\_role(doctor\_id BIGINT) RETURNS BOOLEAN AS $$

BEGIN

-- ur.role\_id = 2, where 2 - is a doctor role ID;

RETURN EXISTS (SELECT 1 FROM users\_roles AS ur WHERE ur.user\_id = doctor\_id AND ur.role\_id = 2);

END;

$$ LANGUAGE plpgsql;

---------------------- CREATE TABLES ----------------------

CREATE TABLE users (

id BIGSERIAL,

name VARCHAR(50) NOT NULL,

surname VARCHAR(50) NOT NULL,

middle\_name VARCHAR(50),

email custom\_email NOT NULL,

PRIMARY KEY (id),

UNIQUE (email)

);

CREATE TABLE passwords (

user\_id BIGINT PRIMARY KEY,

pass TEXT NOT NULL,

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

);

CREATE TABLE phone\_numbers (

id BIGSERIAL,

number VARCHAR(10) NOT NULL,

country\_code VARCHAR(3) NOT NULL,

user\_id INT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (user\_id) REFERENCES users(id),

UNIQUE (number),

CHECK (CHAR\_LENGTH(number) = 10)

);

CREATE TABLE roles (

id SMALLSERIAL,

name VARCHAR(50) NOT NULL,

salary DECIMAL(8, 2) DEFAULT NULL,

PRIMARY KEY (id),

UNIQUE (name),

CHECK (salary >= 0)

);

INSERT INTO roles (name, salary)

VALUES ('patient', DEFAULT),

('doctor', 23000);

CREATE TABLE users\_roles (

id BIGSERIAL,

user\_id BIGINT NOT NULL,

role\_id SMALLINT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (user\_id) REFERENCES users(id),

FOREIGN KEY (role\_id) REFERENCES roles(id),

UNIQUE (user\_id, role\_id)

);

CREATE TABLE messages (

id BIGSERIAL,

user\_1\_id BIGINT NOT NULL,

user\_2\_id BIGINT NOT NULL,

destination\_time TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL,

data VARCHAR(2048) NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (user\_1\_id) REFERENCES users(id),

FOREIGN KEY (user\_2\_id) REFERENCES users(id)

);

-- Make optimization for ofter filtering by user\_1 and user\_2 IDs;

CREATE INDEX idx\_user\_1\_id ON messages (user\_1\_id);

CREATE INDEX idx\_user\_2\_id ON messages (user\_2\_id);

CREATE TABLE allergens (

id SMALLSERIAL,

name VARCHAR(32) NOT NULL,

PRIMARY KEY (id)

);

CREATE TABLE users\_allergen (

id BIGSERIAL,

user\_id BIGINT NOT NULL,

allergen\_id SMALLINT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY(user\_id) REFERENCES users(id),

FOREIGN KEY (allergen\_id) REFERENCES allergens(id)

);

CREATE TABLE booked\_doctors (

id BIGSERIAL,

patient\_id BIGINT NOT NULL,

doctor\_id BIGINT NOT NULL,

book\_time TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL,

appointment\_time TIMESTAMPTZ NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (patient\_id) REFERENCES users(id),

FOREIGN KEY (doctor\_id) REFERENCES users(id),

UNIQUE(patient\_id, doctor\_id, appointment\_time),

CONSTRAINT valid\_patient\_role CHECK (is\_valid\_patient\_role(patient\_id)),

CONSTRAINT valid\_doctor\_role CHECK (is\_valid\_doctor\_role(doctor\_id))

);

CREATE TABLE doctor\_appointments (

id BIGSERIAL,

booked\_doctor\_id BIGINT NOT NULL,

complaint VARCHAR(2048),

PRIMARY KEY (id),

FOREIGN KEY (booked\_doctor\_id) REFERENCES booked\_doctors(id)

);

CREATE TABLE analyses (

id SMALLSERIAL,

name VARCHAR(128) NOT NULL,

PRIMARY KEY (id),

UNIQUE (name)

);

CREATE TABLE analyse\_appointments (

id BIGSERIAL,

analyse\_id SMALLINT NOT NULL,

doctor\_appointment\_id BIGINT NOT NULL,

is\_completed BOOLEAN,

PRIMARY KEY (id),

FOREIGN KEY (analyse\_id) REFERENCES analyses(id),

FOREIGN KEY (doctor\_appointment\_id) REFERENCES doctor\_appointments(id)

);

CREATE TABLE patient\_sick\_leave (

id BIGSERIAL,

doctor\_appointment\_id BIGINT NOT NULL,

start\_date DATE DEFAULT CURRENT\_DATE NOT NULL,

end\_date DATE DEFAULT CURRENT\_DATE NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (doctor\_appointment\_id) REFERENCES doctor\_appointments(id)

);

CREATE TABLE drugs (

id BIGSERIAL,

name varchar(64) NOT NULL,

description varchar(2048) NOT NULL,

PRIMARY KEY (id),

UNIQUE (name)

);

CREATE TABLE prescription\_drugs (

id BIGSERIAL,

drug\_id BIGINT NOT NULL,

doctor\_appointment\_id BIGINT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (drug\_id) REFERENCES drugs(id),

FOREIGN KEY (doctor\_appointment\_id) REFERENCES doctor\_appointments(id)

);

/\* Descripiton:

\* Create a view displaying the list of diseases for the maximum number of patients of each doctor.

\*/

CREATE VIEW max\_diseases\_per\_doctor AS

WITH doctor\_patient\_disease AS (

SELECT

bd.doctor\_id,

da.complaint AS disease,

COUNT(DISTINCT bd.patient\_id) AS patient\_count

FROM

booked\_doctors bd

JOIN

doctor\_appointments da ON bd.id = da.booked\_doctor\_id

GROUP BY

bd.doctor\_id, da.complaint

),

max\_patient\_disease AS (

SELECT

doctor\_id,

disease,

patient\_count,

ROW\_NUMBER() OVER (PARTITION BY doctor\_id ORDER BY patient\_count DESC, disease) AS rn

FROM

doctor\_patient\_disease

)

SELECT

d.id AS doctor\_id,

d.name AS doctor\_name,

d.surname AS doctor\_surname,

d.middle\_name AS doctor\_middle\_name,

m.disease AS most\_popular\_disease,

m.patient\_count

FROM

max\_patient\_disease m

JOIN

users d ON m.doctor\_id = d.id

WHERE

m.rn = 1;

-- DROP VIEW IF EXISTS max\_diseases\_per\_doctor

/\* Description:

\* Create a view to display the trend in the number of doctor visits (by visit dates).

\*/

CREATE VIEW doctor\_visit\_trend AS

WITH daily\_visits AS (

SELECT

date\_trunc('day', bd.appointment\_time) AS visit\_date,

COUNT(\*) AS daily\_visit\_count

FROM

booked\_doctors bd

GROUP BY

date\_trunc('day', bd.appointment\_time)

),

visit\_trend AS (

SELECT

visit\_date,

daily\_visit\_count,

LAG(daily\_visit\_count) OVER (ORDER BY visit\_date) AS previous\_daily\_visit\_count,

LEAD(daily\_visit\_count) OVER (ORDER BY visit\_date) AS next\_daily\_visit\_count

FROM

daily\_visits

)

SELECT

visit\_date,

daily\_visit\_count,

COALESCE(daily\_visit\_count - previous\_daily\_visit\_count, 0) AS difference\_with\_previous\_day,

COALESCE(next\_daily\_visit\_count - daily\_visit\_count, 0) AS difference\_with\_next\_day

FROM

visit\_trend

ORDER BY

visit\_date;

-- DROP VIEW IF EXISTS doctor\_visit\_trend

CREATE OR REPLACE VIEW user\_details\_view AS

SELECT

u.id,

u.name,

u.surname,

u.middle\_name,

u.email,

STRING\_AGG(pn.country\_code || pn.number, ',') AS phone\_numbers,

r.name AS role

FROM

users u

LEFT JOIN

phone\_numbers pn ON u.id = pn.user\_id

LEFT JOIN

users\_roles ur ON u.id = ur.user\_id

LEFT JOIN

roles r ON ur.role\_id = r.id

GROUP BY

u.id, u.name, u.surname, u.middle\_name, u.email, r.name

ORDER BY

u.surname, u.name;

-- DROP VIEW IF EXISTS user\_details\_view

SELECT \* FROM max\_diseases\_per\_doctor;

SELECT \* FROM doctor\_visit\_trend;

SELECT \* FROM user\_details\_view WHERE email = 'ivan.ivanov1@example.com';

/\* Description:

\* Sampled Projection: Get a list of patients seen by a specified doctor on a specified date.

\*/

SELECT u.id, u.name, u.surname, u.middle\_name, u.email

FROM users u

JOIN booked\_doctors bd ON u.id = bd.patient\_id

JOIN users\_roles ur ON bd.doctor\_id = ur.user\_id

JOIN roles r ON ur.role\_id = r.id

WHERE r.name = 'doctor'

AND bd.doctor\_id = 1 -- 1 - requeired doc ID

AND DATE(bd.appointment\_time) = '2024-05-25';

/\* Description:

\* Get a list of doctors ranked by the number of patients admitted in the last month.

\*/

WITH date\_range AS (

SELECT

date\_trunc('month', CURRENT\_DATE) - interval '1 month' AS last\_month\_start,

date\_trunc('month', CURRENT\_DATE) AS last\_month\_end

)

SELECT

u.id,

u.name,

u.surname,

u.middle\_name,

u.email,

COUNT(bd.patient\_id) AS patients\_count

FROM users u

JOIN booked\_doctors bd ON u.id = bd.doctor\_id

JOIN date\_range dr ON bd.appointment\_time >= dr.last\_month\_start

AND bd.appointment\_time < dr.last\_month\_end

GROUP BY u.id

ORDER BY patients\_count DESC;

/\* Description:

\* Get a list of patients who have been prescribed medications from the heart disease category.

\*/

SELECT DISTINCT u.id, u.name, u.surname, u.middle\_name, u.email

FROM users u

JOIN booked\_doctors bd ON u.id = bd.patient\_id

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

JOIN prescription\_drugs pd ON da.id = pd.doctor\_appointment\_id

JOIN drugs d ON pd.drug\_id = d.id

WHERE LOWER(d.description) LIKE '%серц%';

/\* Description:

\* Count the number of patients each physician has seen per appointment during the last month.

\*/

WITH date\_range AS (

SELECT

date\_trunc('month', CURRENT\_DATE) - interval '1 month' AS last\_month\_start,

date\_trunc('month', CURRENT\_DATE) AS last\_month\_end

)

SELECT

u.id,

u.name,

u.surname,

u.middle\_name,

u.email,

COUNT(bd.patient\_id) AS patients\_count

FROM users u

JOIN booked\_doctors bd ON u.id = bd.doctor\_id

JOIN date\_range dr ON bd.appointment\_time >= dr.last\_month\_start

AND bd.appointment\_time < dr.last\_month\_end

GROUP BY u.id;

/\* Description:

\* Merge: Give a total list of medications and tests that have been prescribed for the specified

\* patient in the last six months, without repeats, with status.

\*/

SELECT analyses.name AS name, 'Анализы' AS type

FROM users

INNER JOIN booked\_doctors ON booked\_doctors.patient\_id = users.id

INNER JOIN doctor\_appointments ON doctor\_appointments.booked\_doctor\_id = booked\_doctors.id

INNER JOIN analyse\_appointments ON analyse\_appointments.doctor\_appointment\_id = doctor\_appointments.id

INNER JOIN analyses ON analyses.id = analyse\_appointments.analyse\_id

WHERE users.id = 5

AND booked\_doctors.appointment\_time >= CURRENT\_DATE - INTERVAL '6 months'

UNION

SELECT drugs.name AS name, 'Лекарства' AS type

FROM users

INNER JOIN booked\_doctors ON booked\_doctors.patient\_id = users.id

INNER JOIN doctor\_appointments ON doctor\_appointments.booked\_doctor\_id = booked\_doctors.id

INNER JOIN prescription\_drugs ON prescription\_drugs.doctor\_appointment\_id = doctor\_appointments.id

INNER JOIN drugs ON drugs.id = prescription\_drugs.drug\_id

WHERE users.id = 13

AND booked\_doctors.appointment\_time >= CURRENT\_DATE - INTERVAL '6 months';

/\* Description:

\* Get the ranking of each doctor based on:

\* - the number of patients seen;

\* - the number of analyses prescribed;

\* - the number of drugs prescribed in the last quarter.

\*/

WITH patient\_count AS (

SELECT

bd.doctor\_id,

COUNT(DISTINCT bd.patient\_id) AS patient\_count

FROM

booked\_doctors bd

WHERE

bd.appointment\_time >= date\_trunc('quarter', CURRENT\_DATE) - interval '3 month'

GROUP BY

bd.doctor\_id

),

analyse\_count AS (

SELECT

bd.doctor\_id,

COUNT(aa.id) AS analyse\_count

FROM

analyse\_appointments aa

JOIN

doctor\_appointments da ON aa.doctor\_appointment\_id = da.id

JOIN

booked\_doctors bd ON da.booked\_doctor\_id = bd.id

WHERE

aa.is\_completed = TRUE

AND bd.appointment\_time >= date\_trunc('quarter', CURRENT\_DATE) - interval '3 month'

GROUP BY

bd.doctor\_id

),

drug\_count AS (

SELECT

bd.doctor\_id,

COUNT(pd.id) AS drug\_count

FROM

prescription\_drugs pd

JOIN

doctor\_appointments da ON pd.doctor\_appointment\_id = da.id

JOIN

booked\_doctors bd ON da.booked\_doctor\_id = bd.id

WHERE

bd.appointment\_time >= date\_trunc('quarter', CURRENT\_DATE) - interval '3 month'

GROUP BY

bd.doctor\_id

),

combined AS (

SELECT

u.id AS doctor\_id,

u.name AS doctor\_name,

u.surname AS doctor\_surname,

u.middle\_name AS doctor\_middle\_name,

COALESCE(pc.patient\_count, 0) AS patient\_count,

COALESCE(ac.analyse\_count, 0) AS analyse\_count,

COALESCE(dc.drug\_count, 0) AS drug\_count

FROM

users u

LEFT JOIN

patient\_count pc ON u.id = pc.doctor\_id

LEFT JOIN

analyse\_count ac ON u.id = ac.doctor\_id

LEFT JOIN

drug\_count dc ON u.id = dc.doctor\_id

WHERE

EXISTS (SELECT 1 FROM users\_roles ur WHERE ur.user\_id = u.id AND ur.role\_id = 2)

)

SELECT

doctor\_id,

doctor\_name,

doctor\_surname,

doctor\_middle\_name,

patient\_count,

RANK() OVER (ORDER BY patient\_count DESC) AS patient\_rank,

analyse\_count,

RANK() OVER (ORDER BY analyse\_count DESC) AS analyse\_rank,

drug\_count,

RANK() OVER (ORDER BY drug\_count DESC) AS drug\_rank

FROM

combined

ORDER BY

doctor\_id;

/\* Description:

\* Sampled Projection: Get a list of patients seen by a specified doctor on a specified date.

\*/

CREATE OR REPLACE FUNCTION get\_patients\_by\_doctor\_and\_date(doctor\_id INT, appointment\_date DATE)

RETURNS TABLE (

id BIGINT,

name VARCHAR(50),

surname VARCHAR(50),

middle\_name VARCHAR(50),

email custom\_email

) AS $$

BEGIN

RETURN QUERY

SELECT u.id, u.name, u.surname, u.middle\_name, u.email

FROM users u

JOIN booked\_doctors bd ON u.id = bd.patient\_id

JOIN users\_roles ur ON bd.doctor\_id = ur.user\_id

JOIN roles r ON ur.role\_id = r.id

WHERE r.name = 'doctor'

AND bd.doctor\_id = get\_patients\_by\_doctor\_and\_date.doctor\_id

AND DATE(bd.appointment\_time) = get\_patients\_by\_doctor\_and\_date.appointment\_date;

END;

$$ LANGUAGE plpgsql;

/\* Description:

\* Get a list of doctors ranked by the number of patients admitted in the last month.

\*/

CREATE OR REPLACE FUNCTION get\_top\_doctors\_last\_month()

RETURNS TABLE (

id BIGINT,

name VARCHAR(50),

surname VARCHAR(50),

middle\_name VARCHAR(50),

email custom\_email,

clients\_count BIGINT

) AS $$

BEGIN

RETURN QUERY

SELECT \*

FROM get\_patient\_count\_per\_doctor\_last\_month() AS bd

ORDER BY bd.clients\_count DESC;

END;

$$ LANGUAGE plpgsql;

/\* Description:

\* Get a list of patients who have been prescribed medications from the heart disease category.

\*/

CREATE OR REPLACE FUNCTION get\_patients\_with\_heart\_disease\_drugs()

RETURNS TABLE (

id BIGINT,

name VARCHAR(50),

surname VARCHAR(50),

middle\_name VARCHAR(50),

email custom\_email

) AS $$

BEGIN

RETURN QUERY

SELECT DISTINCT u.id, u.name, u.surname, u.middle\_name, u.email

FROM users u

JOIN booked\_doctors bd ON u.id = bd.patient\_id

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

JOIN prescription\_drugs pd ON da.id = pd.doctor\_appointment\_id

JOIN drugs d ON pd.drug\_id = d.id

WHERE LOWER(d.description) LIKE '%серд%';

END;

$$ LANGUAGE plpgsql;

/\* Description:

\* Count the number of patients each physician has seen per appointment during the last month.

\*/

CREATE OR REPLACE FUNCTION get\_patient\_count\_per\_doctor\_last\_month()

RETURNS TABLE (

id BIGINT,

name VARCHAR(50),

surname VARCHAR(50),

middle\_name VARCHAR(50),

email custom\_email,

clients\_count BIGINT

) AS $$

DECLARE

last\_month\_start DATE := date\_trunc('month', CURRENT\_DATE) - interval '1 month';

last\_month\_end DATE := date\_trunc('month', CURRENT\_DATE);

BEGIN

RETURN QUERY

SELECT u.id, u.name, u.surname, u.middle\_name, u.email, COUNT(bd.patient\_id) AS patients\_count

FROM users u

JOIN booked\_doctors bd ON u.id = bd.doctor\_id

WHERE bd.appointment\_time >= last\_month\_start

AND bd.appointment\_time < last\_month\_end

GROUP BY u.id;

END;

$$ LANGUAGE plpgsql;

/\* Description:

\* Merge: Give a total list of medications and tests that have been prescribed for the specified

\* patient in the last six months, without repeats, with status.

\*/

CREATE OR REPLACE FUNCTION get\_medications\_and\_tests\_for\_patient(patient\_id BIGINT)

RETURNS TABLE

(

name varchar(128),

type varchar(20)

)

AS

$$

BEGIN

RETURN QUERY

-- Список анализов и лекарств которые были выписаны определенному пациенту за последние 6 месяцев

(

SELECT CAST(analyses.name AS varchar(128)) AS name, CAST('Анализы' AS varchar(20)) AS type

FROM users

INNER JOIN booked\_doctors ON booked\_doctors.patient\_id = users.id

INNER JOIN doctor\_appointments ON doctor\_appointments.booked\_doctor\_id = booked\_doctors.id

INNER JOIN analyse\_appointments ON analyse\_appointments.doctor\_appointment\_id = doctor\_appointments.id

INNER JOIN analyses ON analyses.id = analyse\_appointments.analyse\_id

WHERE users.id = get\_medications\_and\_tests\_for\_patient.patient\_id

AND booked\_doctors.appointment\_time >= CURRENT\_DATE - INTERVAL '6 months'

UNION

SELECT CAST(drugs.name AS varchar(128)) AS name, CAST('Лекарства' AS varchar(20)) AS type

FROM users

INNER JOIN booked\_doctors ON booked\_doctors.patient\_id = users.id

INNER JOIN doctor\_appointments ON doctor\_appointments.booked\_doctor\_id = booked\_doctors.id

INNER JOIN prescription\_drugs ON prescription\_drugs.doctor\_appointment\_id = doctor\_appointments.id

INNER JOIN drugs ON drugs.id = prescription\_drugs.drug\_id

WHERE users.id = get\_medications\_and\_tests\_for\_patient.patient\_id

AND booked\_doctors.appointment\_time >= CURRENT\_DATE - INTERVAL '6 months'

);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION check\_user\_login(user\_email TEXT, user\_password TEXT) RETURNS BOOLEAN AS $$

DECLARE

stored\_password TEXT;

BEGIN

-- Извлечение пароля из таблицы паролей

SELECT p.pass INTO stored\_password

FROM users u

JOIN passwords p ON u.id = p.user\_id

WHERE u.email = user\_email;

-- Если пользователь не найден, вернуть false

IF NOT FOUND THEN

RETURN FALSE;

END IF;

-- Проверка пароля

IF stored\_password = user\_password THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_last\_appointments\_by\_surname(patient\_surname VARCHAR(50))

RETURNS TABLE (

doctor\_name TEXT,

appointment\_time TIMESTAMPTZ,

complaint VARCHAR(2048)

) AS $$

BEGIN

RETURN QUERY

SELECT

CONCAT(d.name, ' ', d.surname) AS doctor\_name,

bd.appointment\_time,

da.complaint

FROM

users p

JOIN booked\_doctors bd ON p.id = bd.patient\_id

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

JOIN users d ON bd.doctor\_id = d.id

WHERE

p.surname = patient\_surname

ORDER BY

bd.appointment\_time DESC

LIMIT 1;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_clinic\_statistics(start\_date DATE, end\_date DATE)

RETURNS TABLE (

total\_patients INT,

infectious\_diseases INT,

hypertension INT,

diabetes INT,

lung\_diseases INT,

total\_sick\_leaves INT,

total\_sick\_leave\_days INT

) AS $$

DECLARE

total\_patients\_count INT;

infectious\_diseases\_count INT;

hypertension\_count INT;

diabetes\_count INT;

lung\_diseases\_count INT;

total\_sick\_leaves\_count INT;

total\_sick\_leave\_days\_count INT;

BEGIN

-- Подсчет общего количества пациентов

SELECT COUNT(DISTINCT bd.patient\_id)

INTO total\_patients\_count

FROM booked\_doctors bd

WHERE bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

-- Подсчет количества пациентов с инфекционными заболеваниями

SELECT COUNT(DISTINCT bd.patient\_id)

INTO infectious\_diseases\_count

FROM booked\_doctors bd

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

WHERE da.complaint ILIKE '%infection%'

AND bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

-- Подсчет количества пациентов с гипертонией

SELECT COUNT(DISTINCT bd.patient\_id)

INTO hypertension\_count

FROM booked\_doctors bd

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

WHERE da.complaint ILIKE '%hypertension%'

AND bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

-- Подсчет количества пациентов с диабетом

SELECT COUNT(DISTINCT bd.patient\_id)

INTO diabetes\_count

FROM booked\_doctors bd

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

WHERE da.complaint ILIKE '%diabetes%'

AND bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

-- Подсчет количества пациентов с заболеваниями легких

SELECT COUNT(DISTINCT bd.patient\_id)

INTO lung\_diseases\_count

FROM booked\_doctors bd

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

WHERE da.complaint ILIKE '%lung%'

AND bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

-- Подсчет общего количества выданных больничных

SELECT COUNT(\*)

INTO total\_sick\_leaves\_count

FROM patient\_sick\_leave psl

JOIN doctor\_appointments da ON psl.doctor\_appointment\_id = da.id

JOIN booked\_doctors bd ON da.booked\_doctor\_id = bd.id

WHERE bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

-- Подсчет общего количества дней больничных

SELECT SUM(psl.end\_date - psl.start\_date)

INTO total\_sick\_leave\_days\_count

FROM patient\_sick\_leave psl

JOIN doctor\_appointments da ON psl.doctor\_appointment\_id = da.id

JOIN booked\_doctors bd ON da.booked\_doctor\_id = bd.id

WHERE bd.appointment\_time BETWEEN get\_clinic\_statistics.start\_date AND get\_clinic\_statistics.end\_date;

RETURN QUERY SELECT

COALESCE(total\_patients\_count, 0),

COALESCE(infectious\_diseases\_count, 0),

COALESCE(hypertension\_count, 0),

COALESCE(diabetes\_count, 0),

COALESCE(lung\_diseases\_count, 0),

COALESCE(total\_sick\_leaves\_count, 0),

COALESCE(total\_sick\_leave\_days\_count, 0);

END;

$$ LANGUAGE plpgsql;

-- Создание функции триггера для проверки аллергенов

CREATE OR REPLACE FUNCTION check\_allergens() RETURNS TRIGGER AS $$

DECLARE

patient\_id BIGINT;

patient\_allergens TEXT[];

allergen TEXT;

drug\_description TEXT;

BEGIN

-- Получение ID пациента

SELECT bd.patient\_id INTO patient\_id

FROM booked\_doctors bd

JOIN doctor\_appointments da ON bd.id = da.booked\_doctor\_id

WHERE da.id = NEW.doctor\_appointment\_id;

-- Получение описания лекарства

SELECT description INTO drug\_description

FROM drugs

WHERE id = NEW.drug\_id;

-- Получение аллергенов пациента в виде массива

SELECT ARRAY\_AGG(a.name)

INTO patient\_allergens

FROM users\_allergen ua

JOIN allergens a ON ua.allergen\_id = a.id

WHERE ua.user\_id = patient\_id;

-- Проверка на наличие аллергенов в описании лекарства

IF patient\_allergens IS NOT NULL THEN

FOR i IN ARRAY\_LOWER(patient\_allergens, 1) .. ARRAY\_UPPER(patient\_allergens, 1) LOOP

allergen := patient\_allergens[i];

-- Отладочное сообщение для проверки значений

RAISE NOTICE 'Checking allergen: % in drug description: %', allergen, drug\_description;

IF position(lower(allergen) IN lower(drug\_description)) > 0 THEN

RAISE EXCEPTION 'Cannot prescribe drug with allergen %', allergen;

END IF;

END LOOP;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM get\_patients\_by\_doctor\_and\_date(8, '2024-05-10');

SELECT \* FROM get\_top\_doctors\_last\_month();

SELECT \* FROM get\_patients\_with\_heart\_disease\_drugs();

SELECT \* FROM get\_patient\_count\_per\_doctor\_last\_month();

SELECT \* FROM get\_medications\_and\_tests\_for\_patient(5);

SELECT \* FROM check\_user\_login('ivan.ivanov1@example.com', '123456')

SELECT \* FROM get\_last\_appointments\_by\_surname('Павлов');

SELECT \* FROM get\_clinic\_statistics('2024-01-01', '2024-12-31');

/\* Descripiton:

\* Create a view displaying the list of diseases for the maximum number of patients of each doctor.

\*/

CREATE VIEW max\_diseases\_per\_doctor AS

WITH doctor\_patient\_disease AS (

SELECT

bd.doctor\_id,

da.complaint AS disease,

COUNT(DISTINCT bd.patient\_id) AS patient\_count

FROM

booked\_doctors bd

JOIN

doctor\_appointments da ON bd.id = da.booked\_doctor\_id

GROUP BY

bd.doctor\_id, da.complaint

),

max\_patient\_disease AS (

SELECT

doctor\_id,

disease,

patient\_count,

ROW\_NUMBER() OVER (PARTITION BY doctor\_id ORDER BY patient\_count DESC, disease) AS rn

FROM

doctor\_patient\_disease

)

SELECT

d.id AS doctor\_id,

d.name AS doctor\_name,

d.surname AS doctor\_surname,

d.middle\_name AS doctor\_middle\_name,

m.disease AS most\_popular\_disease,

m.patient\_count

FROM

max\_patient\_disease m

JOIN

users d ON m.doctor\_id = d.id

WHERE

m.rn = 1;

-- DROP VIEW IF EXISTS max\_diseases\_per\_doctor

/\* Description:

\* Create a view to display the trend in the number of doctor visits (by visit dates).

\*/

CREATE VIEW doctor\_visit\_trend AS

WITH daily\_visits AS (

SELECT

date\_trunc('day', bd.appointment\_time) AS visit\_date,

COUNT(\*) AS daily\_visit\_count

FROM

booked\_doctors bd

GROUP BY

date\_trunc('day', bd.appointment\_time)

),

visit\_trend AS (

SELECT

visit\_date,

daily\_visit\_count,

LAG(daily\_visit\_count) OVER (ORDER BY visit\_date) AS previous\_daily\_visit\_count,

LEAD(daily\_visit\_count) OVER (ORDER BY visit\_date) AS next\_daily\_visit\_count

FROM

daily\_visits

)

SELECT

visit\_date,

daily\_visit\_count,

COALESCE(daily\_visit\_count - previous\_daily\_visit\_count, 0) AS difference\_with\_previous\_day,

COALESCE(next\_daily\_visit\_count - daily\_visit\_count, 0) AS difference\_with\_next\_day

FROM

visit\_trend

ORDER BY

visit\_date;

-- DROP VIEW IF EXISTS doctor\_visit\_trend

CREATE OR REPLACE VIEW user\_details\_view AS

SELECT

u.id,

u.name,

u.surname,

u.middle\_name,

u.email,

STRING\_AGG(pn.country\_code || pn.number, ',') AS phone\_numbers,

r.name AS role

FROM

users u

LEFT JOIN

phone\_numbers pn ON u.id = pn.user\_id

LEFT JOIN

users\_roles ur ON u.id = ur.user\_id

LEFT JOIN

roles r ON ur.role\_id = r.id

GROUP BY

u.id, u.name, u.surname, u.middle\_name, u.email, r.name

ORDER BY

u.surname, u.name;

-- DROP VIEW IF EXISTS user\_details\_view

SELECT \* FROM max\_diseases\_per\_doctor;

SELECT \* FROM doctor\_visit\_trend;

SELECT \* FROM user\_details\_view WHERE email = 'ivan.ivanov1@example.com';

-- Создание триггера для таблицы prescription\_drugs

CREATE TRIGGER trigger\_check\_allergens

BEFORE INSERT ON prescription\_drugs

FOR EACH ROW

EXECUTE FUNCTION check\_allergens();

**Додаток В**

**КОД КЛІЄНТСЬКОГО ЗАСТОСУНКУ**

**Index.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Медицинский Сайт</title>

<link rel="stylesheet" href="styles.css">

<link rel="stylesheet" href="bars/top\_bar.css">

</head>

<body>

<header class="header" w3-include-html="bars/top\_bar.html"></header>

<main>

<section class="hero">

<div class="hero-content">

<h1>WE VALUE YOUR</h1>

<h2>FAMILY'S HEALTH & <br> YOUR TIME.</h2>

<p>Lorem Ipsum is simply dummy text of the printing and <br> typesetting industry. Lorem Ipsum has been the industry's <br> standard dummy text.</p>

</div>

<div class="hero-image">

<img src="src/successful-medical-team.png" alt="Medical Team">

</div>

</section>

<section class="appointment-form">

<div class="appointment-title">

<h2>BOOK YOUR APPOINTMENT</h2>

</div>

<form>

<div class="form-row">

<input type="text" placeholder="Full Name" value="Jennie Wilkerson">

<input type="email" placeholder="Email" value="jenniewilkerson@info.com">

<input type="text" placeholder="Phone Number" value="+41 678-678-456">

<input type="text" placeholder="Requested Service" value="Dental Surgery">

</div>

<div class="form-row">

<select>

<option>Select Doctor</option>

<option>Doctor A</option>

<option>Doctor B</option>

<option>Doctor C</option>

</select>

<input type="date">

<input type="time">

</div>

<button type="submit">Book Now</button>

</form>

</section>

</main>

<div class="chat-widget">

<div class="chat-header">

<img src="src/young-asian-woman.png" alt="Profile" class="profile-image">

<span>Monster Bot</span>

</div>

<div class="chat-body">

<p>Здравствуйте! Обращайтесь к нам, если у вас возникли вопросы.</p>

</div>

<div class="chat-footer">

<button class="chat-button">Написать сейчас</button>

</div>

</div>

<script src="scripts/include.js"></script>

</body>

</html>

**style.css**

/\* Подключение шрифтов \*/

@font-face {

font-family: 'fa-solid-900';

src: url('fonts/fa-solid-900.woff2') format('woff2'),

url('fonts/fa-solid-900.woff') format('woff');

font-weight: 900;

font-style: normal;

}

@font-face {

font-family: 'fa-regular-400';

src: url('fonts/fa-regular-400.woff2') format('woff2'),

url('fonts/fa-regular-400.woff') format('woff');

font-weight: 400;

font-style: normal;

}

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

.hero {

display: flex;

align-items: center;

justify-content: center;

position: relative;

overflow: hidden;

background-color: #eef6fd;

}

.hero-content {

position: absolute;

top: 45%; /\* Поднимет текст выше \*/

left: 15%;

transform: translateY(-50%);

max-width: 50%;

}

.hero-content h1,

.hero-content h2,

.hero-content p {

margin: 0;

}

.hero-content h1 {

font-size: 48px;

font-weight: bold;

color: black;

font-family: 'fa-solid-900';

}

.hero-content h2 {

font-size: 48px;

font-weight: bold;

color: #3498db;

font-family: 'fa-regular-400';

}

.hero-content p {

color: #333333;

font-size: 18px;

margin-top: 10px;

font-family: 'fa-regular-400';

}

.hero-image {

width: 100%;

height: 100%;

}

.appointment-form {

background-color: #f8f8f8;

padding: 50px 20px;

text-align: center;

}

.appointment-title {

background-color: #3498db;

color: #fff;

padding: 20px;

margin-bottom: 20px;

}

.appointment-title h2 {

margin: 0;

}

.appointment-form form {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 10px;

}

.form-row {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 10px;

width: 100%;

}

.appointment-form input,

.appointment-form select {

width: 200px;

padding: 10px;

border: 1px solid #ddd;

border-radius: 5px;

}

.appointment-form input[type="date"],

.appointment-form input[type="time"] {

width: 130px;

}

.appointment-form button {

background-color: #ff3366;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

margin-top: 10px;

}

.chat-widget {

position: fixed;

bottom: 20px;

right: 20px;

width: 300px;

background-color: #fff;

border: 1px solid #ddd;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

.chat-header {

display: flex;

align-items: center;

padding: 10px;

border-bottom: 1px solid #ddd;

}

.profile-image {

width: 40px;

height: 40px;

border-radius: 50%;

margin-right: 10px;

}

.chat-body {

padding: 10px;

}

.chat-footer {

padding: 10px;

border-top: 1px solid #ddd;

text-align: right;

}

.chat-button {

background-color: #3366ff;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

}

**bars/top\_bar.html**

<div class="header-top">

<div class="contact-info">

<span>CALL US: +41 678-678-456 | +41 678-678-903</span>

<span>594 Sunset Ave. Manahawkin, NJ 08050</span>

<span>Email: jenniewilkerson@info.com</span>

</div>

</div>

<div class="header-main">

<img src="../src/logo.svg" alt="Логотип" class="logo">

<nav class="nav">

<ul>

<li><a href="../">HOME</a></li>

<li><a href="../profile\_page/">PROFILE</a></li>

<li><a href="#blog">BLOG</a></li>

<li><a href="#team">TEAM</a></li>

<li><a href="#faq">FAQ</a></li>

<li><a href="#press">PRESS</a></li>

<li><a href="#testimonials">TESTIMONIALS</a></li>

<li><a href="#contact">CONTACT</a></li>

</ul>

</nav>

<a href="#" class="btn-appointment">Book an Appointment</a>

</div>

**bars/top\_bar.css**

.header {

background-color: #f8f8f8;

padding: 10px 20px;

}

.header-top {

display: flex;

justify-content: space-between;

align-items: center;

font-size: 12px;

}

.contact-info span {

margin-right: 20px;

}

.header-main {

display: flex;

justify-content: space-between;

align-items: center;

}

.logo {

width: 150px;

}

.nav ul {

list-style-type: none;

padding: 0;

}

.nav li {

display: inline;

margin-right: 20px;

}

.nav a {

text-decoration: none;

color: #000;

font-family: 'fa-solid-900';

}

.btn-appointment {

background-color: #ff3366;

color: #fff;

padding: 10px 20px;

text-decoration: none;

border-radius: 5px;

font-family: 'fa-solid-900';

}

**profile\_page/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Index</title>

</head>

<body>

<script src="../scripts/redirect.js"></script>

</body>

</html>

**profile\_page/profile.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Profile</title>

<link rel="stylesheet" href="profile.css">

<link rel="stylesheet" href="../bars/top\_bar.css">

</head>

<body>

<header class="header" w3-include-html="../bars/top\_bar.html"></header>

<div class="container">

<div class="profile-header">

<img src="../src/default\_avatar.jpg" alt="User Avatar" class="avatar">

<h1>Profile</h1>

<h2 id="welcome-message">Welcome, [User's Name]</h2>

</div>

<div class="profile-content">

<div class="personal-info">

<h3>Personal Information</h3>

<p><strong>Name:</strong> <span id="user-name">[User's Name]</span></p>

<p><strong>Email:</strong> <span id="user-email">[User's Email]</span></p>

<p><strong>Phone:</strong> <span id="user-phone">[User's Phone Number]</span></p>

</div>

<div class="medical-info">

<h3>Medical Card</h3>

<p><strong>Allergens:</strong> [User's Allergens] <a href="#" class="more-link">More...</a></p>

<h4>Booked Doctors</h4>

<ul>

<li>Doctor: [Doctor's Name], Appointment: latest... <a href="#" class="more-link">More...</a></li>

</ul>

<h4>Doctor Appointments</h4>

<ul>

<li>Complaint: latest..., Date: latest... <a href="#" class="more-link">More...</a></li>

</ul>

<h4>Prescription Drugs</h4>

<ul>

<li>Drug: latest..., Description: latest... <a href="#" class="more-link">More...</a></li>

</ul>

</div>

</div>

</div>

<button id="logoutButton" class="btn-logout">Logout</button>

<script src="../scripts/include.js"></script>

<script src="../scripts/auth.js"></script>

<script>

document.addEventListener('DOMContentLoaded', function() {

const name = localStorage.getItem('name');

const surname = localStorage.getItem('surname');

const middleName = localStorage.getItem('middle\_name');

const email = localStorage.getItem('email');

const phoneNumbers = JSON.parse(localStorage.getItem('phone\_numbers') || '[]');

document.getElementById('welcome-message').textContent = `Welcome, ${name} ${surname}`;

document.getElementById('user-name').textContent = `${name} ${surname} ${middleName}`;

document.getElementById('user-email').textContent = email;

document.getElementById('user-phone').textContent = phoneNumbers.join(', ');

});

</script>

</body>

</html>

**profile\_page/profile.css**

body {

font-family: Arial, sans-serif;

background-color: #f8f8f8;

margin: 0;

padding: 0;

}

header {

width: 100%;

position: fixed;

top: 0;

z-index: 1000;

}

.container {

width: 80%;

background-color: #fff;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

margin: 100px auto; /\* Добавим auto чтобы центрировать контейнер \*/

text-align: center; /\* Центрируем текст внутри контейнера \*/

}

.profile-header {

text-align: center;

margin-bottom: 20px;

}

.profile-header h1 {

margin: 0;

}

.profile-header h2 {

margin: 5px 0;

color: #777;

}

.avatar {

width: 100px;

height: 100px;

border-radius: 50%;

margin-bottom: 10px;

}

.profile-content {

display: flex;

justify-content: center;

align-items: flex-start; /\* Обеспечим выравнивание контента по началу \*/

gap: 50px; /\* Добавим промежуток между колонками \*/

}

.personal-info, .medical-info {

width: 45%;

text-align: left; /\* Выравниваем текст по левому краю внутри колонок \*/

}

.personal-info h3, .medical-info h3, .medical-info h4 {

margin-bottom: 10px;

color: #3498db;

}

.personal-info p, .medical-info p, .medical-info ul {

margin: 5px 0;

}

.medical-info ul {

list-style-type: none;

padding: 0;

}

.medical-info ul li {

background-color: #eef6fd;

padding: 10px;

margin-bottom: 5px;

border-radius: 5px;

}

.more-link {

color: #3498db;

text-decoration: none;

font-weight: bold;

margin-left: 5px;

}

.more-link:hover {

text-decoration: underline;

}

.btn-logout {

background-color: #ff3366;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

margin-top: 10px;

font-family: 'fa-solid-900';

}

**profile\_page/registration.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Register</title>

<link rel="stylesheet" href="registration.css">

<link rel="stylesheet" href="../bars/top\_bar.css">

</head>

<body>

<header class="header"></header>

<div class="content-wrapper">

<div class="container">

<div class="form-container">

<h2>Register</h2>

<form id="registerForm">

<div class="form-group">

<label for="fullname">Full Name</label>

<input type="text" id="fullname" required>

</div>

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" required>

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" required>

</div>

<div class="form-group">

<label for="confirmpassword">Confirm Password</label>

<input type="password" id="confirmpassword" required>

</div>

<button type="submit" class="btn">Register</button>

<p>Already have an account? <a href="signin.html">Sign In here</a></p>

</form>

</div>

</div>

</div>

<script src="../scripts/include.js"></script>

<script src="../scripts/auth.js"></script>

</body>

</html>

**profile\_page/registration.css**

body {

font-family: Arial, sans-serif;

background-color: #f8f8f8;

margin: 0;

padding: 0;

}

.content-wrapper {

display: flex;

justify-content: center;

align-items: center;

height: calc(100vh - 80px); /\* Учитывая высоту топбара \*/

padding-top: 60px; /\* Высота топбара с небольшим подъемом \*/

}

.container {

display: flex;

justify-content: center;

align-items: center;

width: 100%;

}

.form-container {

background-color: #fff;

padding: 30px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

max-width: 400px;

width: 100%;

text-align: center;

}

.form-container h2 {

margin-bottom: 20px;

}

.form-group {

margin-bottom: 15px;

text-align: left;

}

.form-group label {

display: block;

margin-bottom: 5px;

}

.form-group input {

width: 100%;

padding: 10px;

border: 1px solid #ddd;

border-radius: 5px;

}

.btn {

background-color: #3498db;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

width: 100%;

margin-top: 10px;

}

p {

margin-top: 20px;

}

a {

color: #3498db;

text-decoration: none;

}

**profile\_page/signin.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sign In</title>

<link rel="stylesheet" href="signin.css">

<link rel="stylesheet" href="../bars/top\_bar.css">

</head>

<body>

<header class="header"></header>

<div class="content-wrapper">

<div class="container">

<div class="form-container">

<h2>Sign In</h2>

<form id="signinForm">

<div class="form-group">

<label for="email">Email</label>

<input type="email" id="email" required>

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" id="password" required>

</div>

<button type="submit" class="btn">Sign In</button>

<p>Don't have an account? <a href="registration.html">Register here</a></p>

</form>

</div>

</div>

</div>

<script src="../scripts/include.js"></script>

<script src="../scripts/auth.js"></script>

</body>

</html>

**profile\_page/signin.css**

body {

font-family: Arial, sans-serif;

background-color: #f8f8f8;

margin: 0;

padding: 0;

}

.content-wrapper {

display: flex;

justify-content: center;

align-items: center;

height: calc(100vh - 80px);

padding-top: 60px;

}

.container {

display: flex;

justify-content: center;

align-items: center;

width: 100%;

}

.form-container {

background-color: #fff;

padding: 30px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

max-width: 400px;

width: 100%;

text-align: center;

}

.form-container h2 {

margin-bottom: 20px;

}

.form-group {

margin-bottom: 15px;

text-align: left;

}

.form-group label {

display: block;

margin-bottom: 5px;

}

.form-group input {

width: 100%;

padding: 10px;

border: 1px solid #ddd;

border-radius: 5px;

}

.btn {

background-color: #3498db;

color: #fff;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

width: 100%;

margin-top: 10px;

}

p {

margin-top: 20px;

}

a {

color: #3498db;

text-decoration: none;

}

**scripts/auth.js**

document.addEventListener('DOMContentLoaded', function() {

const registerForm = document.getElementById('registerForm');

const signinForm = document.getElementById('signinForm');

const logoutButton = document.getElementById('logoutButton');

if (signinForm) {

signinForm.addEventListener('submit', async (event) => {

event.preventDefault();

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

try {

const response = await fetch('/api/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ email, password }),

credentials: 'include'

});

const data = await response.json();

if (response.ok && data.is\_login\_success) {

localStorage.setItem('isLoggedIn', 'true');

localStorage.setItem('id', data.id);

localStorage.setItem('name', data.name);

localStorage.setItem('surname', data.surname);

localStorage.setItem('middle\_name', data.middle\_name);

localStorage.setItem('email', data.email);

localStorage.setItem('role', data.role);

localStorage.setItem('phone\_numbers', JSON.stringify(data.phone\_numbers));

window.location.href = 'profile.html';

} else {

alert(data.message || 'Login failed');

}

} catch (error) {

console.error('Error:', error);

}

});

}

if (registerForm) {

registerForm.addEventListener('submit', async (event) => {

event.preventDefault();

const fullname = document.getElementById('fullname').value;

const email = document.getElementById('email').value;

const password = document.getElementById('password').value;

const confirmpassword = document.getElementById('confirmpassword').value;

if (password !== confirmpassword) {

alert('Passwords do not match');

return;

}

try {

const response = await fetch('/api/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ fullname, email, password }),

credentials: 'include'

});

const data = await response.json();

if (response.ok && data.is\_login\_success) {

localStorage.setItem('isLoggedIn', 'true');

window.location.href = 'profile.html';

} else {

alert(data.message);

}

} catch (error) {

console.error('Error:', error);

}

});

}

if (logoutButton) {

logoutButton.addEventListener('click', () => {

localStorage.clear(); // Видаляємо всі дані

window.location.href = 'signin.html';

});

}

});

**scripts/include.js**

document.addEventListener("DOMContentLoaded", function() {

var includes = document.querySelectorAll('[w3-include-html]');

includes.forEach(function(include) {

var file = include.getAttribute('w3-include-html');

fetch(file)

.then(response => {

if (!response.ok) throw new Error('Network response was not ok');

return response.text();

})

.then(data => include.outerHTML = data)

.catch(error => console.error('There was a problem with the fetch operation:', error));

});

});

**scripts/auth.js**

document.addEventListener('DOMContentLoaded', function() {

const isLoggedIn = localStorage.getItem('isLoggedIn');

if (isLoggedIn === 'true') {

window.location.href = '/profile\_page/profile.html';

} else {

window.location.href = 'signin.html';

}

});

**Додаток Г**

**КОД СЕРВЕРНОЇ ЧАСТИНИ ЗАСТОСУНКУ**

#include "EnumStringConvertor.hpp"

namespace server::common {

EnumStringConvertor::EnumStringConvertor(void)

: m\_http\_request\_types {

{HttpHeaders::HttpRequestType::NONE, "NONE"},

{HttpHeaders::HttpRequestType::GET, "GET"},

{HttpHeaders::HttpRequestType::POST, "POST"},

{HttpHeaders::HttpRequestType::PUT, "PUT"},

{HttpHeaders::HttpRequestType::DELETE, "DELETE"}

}

{

}

EnumStringConvertor\* EnumStringConvertor::init() {

static EnumStringConvertor instance;

return &instance;

}

std::string EnumStringConvertor::to\_string(HttpHeaders::HttpRequestType val) {

return m\_http\_request\_types[val];

}

HttpHeaders::HttpRequestType EnumStringConvertor::str\_to\_request\_type(std::string val) {

for (const auto& elem\_it : m\_http\_request\_types) {

if (elem\_it.second == val) {

return elem\_it.first;

}

}

return HttpHeaders::HttpRequestType::NONE;

}

} // !server::common;#include "Slot.hpp"

#include <utility>

namespace server::common {

template<typename... Args>

Slot<Args...>::Slot(std::function<void(Args...)> func)

: m\_bind\_method{std::move(func)}

{

}

template<typename... Args>

void Slot<Args...>::invoke(Args... args) {

m\_bind\_method(std::forward<Args>(args)...);

}

} // !server::common;

#include "Signal.hpp"

#include <thread>

#include <utility>

namespace server::common {

template<typename... Args>

void Signal<Args...>::emit(Args... args) {

for (auto &slot : m\_slots\_container) {

std::thread(&Slot<Args...>::invoke, &slot, std::forward<Args>(args)...).detach();

}

}

template<typename... Args>

void Signal<Args...>::connect(const Slot<Args...>& slot) {

m\_slots\_container.push\_back(slot);

}

} // !server::common;#include "Socket.hpp"

#include "config.hpp"

#include "Logger.hpp"

#include <algorithm>

#include <cstring>

#include <sstream>

#include <memory>

#include <arpa/inet.h>

#include <cerrno>

#include <unistd.h>

#include <sys/sendfile.h>

namespace server::common {

Socket::Socket()

: m\_socket\_fd{0}

{

m\_sock\_address.sin\_addr.s\_addr = INADDR\_ANY;

m\_sock\_address.sin\_family = AF\_INET;

m\_sock\_address.sin\_port = ::htons(0);

std::fill(m\_sock\_address.sin\_zero, m\_sock\_address.sin\_zero + sizeof(m\_sock\_address.sin\_zero), config::STR\_EOF);

}

Socket::Socket(Socket&& other) {

operator=(std::move(other));

}

Socket::~Socket() {

// temporary commented due to closing fd after destroy

// bug description;

// when we copied socket in some area and the life cycle is ending

// this object is destroyed and the other stay in life.

// Due to the fd becomed invalid.

// deinit();

}

Socket& Socket::operator= (Socket&& other) {

this->m\_socket\_fd = other.m\_socket\_fd;

other.m\_socket\_fd = 0; // null the socket to escape the closing;

this->m\_sock\_address = other.m\_sock\_address;

return \*this;

}

bool Socket::operator== (const Socket& other) const {

return this->m\_socket\_fd == other.m\_socket\_fd;

}

bool Socket::operator!= (const Socket& other) const {

return this->m\_socket\_fd != other.m\_socket\_fd;

}

bool Socket::operator< (const Socket& other) const {

return this->m\_socket\_fd < other.m\_socket\_fd;

}

bool Socket::operator> (const Socket& other) const {

return this->m\_socket\_fd > other.m\_socket\_fd;

}

std::string Socket::to\_string() const {

std::stringstream ret;

ret << "ip = " << ip\_address\_as\_string() << "; port = " << m\_sock\_address.sin\_port

<< "; fd = " << m\_socket\_fd;

return ret.str();

}

std::int32\_t Socket::init() {

BDECLARE\_TAG\_SCOPE("Socket", \_\_FUNCTION\_\_);

// сделать этот момент более гибким под разные протоколы

m\_socket\_fd = ::socket(AF\_INET, SOCK\_STREAM, 0);

return m\_socket\_fd <= 0 ? -1 : 0;

}

std::int32\_t Socket::deinit() {

return close();

}

void Socket::set\_port(uint16\_t port) {

m\_sock\_address.sin\_port = ::htons(port);

}

int Socket::bind() {

return ::bind(m\_socket\_fd, reinterpret\_cast<sockaddr\*>(&m\_sock\_address), sizeof(m\_sock\_address));

}

int Socket::listen(std::int32\_t count) {

return ::listen(m\_socket\_fd, count);

}

Socket Socket::accept() {

socklen\_t address\_len = sizeof(m\_sock\_address);

Socket accepted\_socket;

accepted\_socket.m\_socket\_fd = ::accept(m\_socket\_fd, reinterpret\_cast<sockaddr\*>(&m\_sock\_address), &address\_len);

getsockname(accepted\_socket);

return accepted\_socket;

}

std::int32\_t Socket::send(const std::string& data) {

return ::send(m\_socket\_fd, data.c\_str(), data.size(), 0);

}

std::int32\_t Socket::sendfile(const File& file) {

off\_t offset{0};

while (offset < file.size()) {

if(::sendfile(m\_socket\_fd, file.file\_fd(), &offset, file.size() - offset) < 0) {

break;

}

}

return offset;

}

std::int32\_t Socket::read(std::string &ret\_buf, std::int32\_t max\_buf\_size) {

auto read\_buf = std::make\_unique<char[]>(max\_buf\_size);

std::memset(read\_buf.get(), 0, max\_buf\_size);

std::int32\_t bytes\_read = ::read(m\_socket\_fd, read\_buf.get(), max\_buf\_size);

if (bytes\_read > 0) {

ret\_buf.assign(read\_buf.get(), bytes\_read);

}

return bytes\_read;

}

std::int32\_t Socket::close() {

std::int32\_t ret = 0;

if (m\_socket\_fd > 0) {

ret = ::close(m\_socket\_fd);

}

if (!ret) {

m\_socket\_fd = 0;

}

return ret;

}

std::int32\_t Socket::getsockname(Socket &sock) const {

socklen\_t sock\_address\_len = sizeof(sock.m\_sock\_address);

return ::getsockname(sock.m\_socket\_fd, reinterpret\_cast<sockaddr\*>(&sock.m\_sock\_address), &sock\_address\_len);

}

std::string Socket::latest\_error() const {

return std::strerror(errno);

}

bool Socket::is\_valid() const {

return ::fcntl(m\_socket\_fd, F\_GETFL) != -1 || errno != EBADF;

}

std::string Socket::ip\_address\_as\_string() const {

return ::inet\_ntoa(m\_sock\_address.sin\_addr);

}

} // !server::common;#include "HttpHeaders.hpp"

#include "EnumStringConvertor.hpp"

#include <sstream>

// #include <cinttypes>

namespace server::common {

HttpHeaders::HttpHeaders(const std::string& request) {

std::istringstream iss(request);

std::string line;

// parse the first line to get request type (method), uri, http version

std::getline(iss, line);

std::istringstream first\_line\_stream(line);

std::string method\_str;

first\_line\_stream >> method\_str >> uri >> http\_version;

method = EnumStringConvertor::init()->str\_to\_request\_type(method\_str);

// parse least;

while (std::getline(iss, line) && !line.empty()) {

size\_t pos = line.find(':');

if (pos == std::string::npos) {

break;

}

std::string name = line.substr(0, pos);

std::string value = line.substr(pos + 2); // to skip ": "

headers.emplace(name, value);

}

// Парсим тело сообщения

std::stringstream body\_stream;

body\_stream << iss.rdbuf();

body = body\_stream.str();

}

std::string HttpHeaders::extension\_to\_content\_type(const std::string& extension) const {

std::stringstream ret\_ss;

if (extension == "html") {

ret\_ss << "Content-Type: text/html\r\n";

} else if (extension == "css") {

ret\_ss << "Content-Type: text/css\r\n";

} else if (extension == "jpg" || extension == "jpeg") {

ret\_ss << "Content-Type: image/jpeg\r\n";

} else if (extension == "png") {

ret\_ss << "Content-Type: image/png\r\n";

} else if (extension == "svg") {

ret\_ss << "Content-Type: image/svg+xml\r\n";

} else if (extension.find("wof") != std::string::npos) {

ret\_ss << "Content-Type: font/woff\r\n";

} else if (extension == "ico") {

ret\_ss << "Content-Type: image/vnd.microsoft.icon\r\n";

} else if (extension == "json") {

ret\_ss << "Content-Type: application/json\r\n";

}

else {

ret\_ss << "Content-Type: application/octet-stream\r\n";

}

return ret\_ss.str();

}

} // !server::common;#include "File.hpp"

#include <cstring>

#include <unistd.h>

namespace server::common {

std::int32\_t File::open(std::string file\_path, OpenTypeEnum modificator) {

std::int32\_t ret = -1;

m\_file\_fd = ::open(file\_path.c\_str(), static\_cast<std::int32\_t>(modificator));

if (m\_file\_fd >= 0) {

ret = update\_file\_stat();

}

return ret;

}

std::int32\_t File::close(void) {

return ::close(m\_file\_fd);

}

std::string File::latest\_error(void) const {

return std::strerror(errno);

}

std::int32\_t File::size(void) const noexcept {

return m\_file\_stat.st\_size;

}

std::int32\_t File::file\_fd(void) const noexcept {

return m\_file\_fd;

}

std::int32\_t File::update\_file\_stat(void) {

return ::fstat(m\_file\_fd, &m\_file\_stat);

}

} // !server::common;#ifndef POLYCLINIC\_SERVER\_COMMON\_SOCKET\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_SOCKET\_HPP

#include "File.hpp"

#include <string>

#include <cinttypes>

#include <netinet/in.h>

namespace server::common {

class Socket {

public:

Socket();

Socket(const Socket& other) = default;

Socket(Socket&& other);

~Socket();

constexpr Socket& operator= (const Socket& other) = default;

Socket& operator= (Socket&& other);

bool operator== (const Socket& other) const;

bool operator!= (const Socket& other) const;

bool operator< (const Socket& other) const;

bool operator> (const Socket& other) const;

std::string to\_string() const;

std::int32\_t init();

std::int32\_t deinit();

void set\_port(uint16\_t port);

std::int32\_t bind();

std::int32\_t listen(std::int32\_t count);

Socket accept();

std::int32\_t send(const std::string& data);

std::int32\_t sendfile(const File& file);

std::int32\_t read(std::string &ret\_buf, std::int32\_t max\_buf\_size);

std::int32\_t close();

std::string latest\_error() const;

bool is\_valid() const;

std::string ip\_address\_as\_string() const;

private:

std::int32\_t getsockname(Socket &sock) const;

private:

std::int32\_t m\_socket\_fd = 0;

sockaddr\_in m\_sock\_address;

};

} // !server::common;

#endif // !POLYCLINIC\_SERVER\_COMMON\_SOCKET\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_SLOT\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_SLOT\_HPP

#include <string>

#include <vector>

#include <functional>

namespace server::common {

template<typename... Args>

class Slot {

public:

Slot(std::function<void(Args...)> func);

void invoke(Args... args);

private:

std::function<void(Args...)> m\_bind\_method;

};

} // !server::common;

#endif // !POLYCLINIC\_SERVER\_COMMON\_SLOT\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_SIGNAL\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_SIGNAL\_HPP

#include "Slot"

#include <string>

#include <vector>

#include <functional>

namespace server::common {

template<typename... Args>

class Signal {

public:

void emit(Args... args);

void connect(const Slot<Args...>& slot);

private:

std::vector<Slot<Args...>> m\_slots\_container;

};

template<typename... Args>

void connect(Signal<Args...>\* signal, Slot<Args...>\* slot) {

signal->connect(\*slot);

}

} // !server::common;

#endif // !POLYCLINIC\_SERVER\_COMMON\_SIGNAL\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_HTTP\_HEADERS\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_HTTP\_HEADERS\_HPP

#include <string>

#include <unordered\_map>

namespace server::common {

class HttpHeaders {

public:

// Internal classes STARTS

enum class HttpRequestType {

NONE,

GET,

POST,

PUT,

DELETE

};

// Internal classes ENDS

public:

HttpRequestType method; // Тип HTTP запроса (GET, POST, PUT, и т.д.)

std::string uri; // URI ресурса

std::string http\_version; // Версия HTTP протокола

std::unordered\_multimap<std::string, std::string> headers; // Множество заголовков

std::string body; // Тело сообщения

HttpHeaders(void) = default;

HttpHeaders(const std::string& request);

std::string extension\_to\_content\_type(const std::string& extension) const;

};

} // !server::common;

#endif // !POLYCLINIC\_SERVER\_COMMON\_HTTP\_HEADERS\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_CONFIG\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_CONFIG\_HPP

#include <string>

#include <cinttypes>

namespace server::common::config {

const std::string LOCALHOST {"127.0.0.1"};

const std::string DB\_HOST {LOCALHOST};

const std::string DB\_USERNAME {"mbukhta"};

const std::string DB\_PASSWORD {"1qa@WS3ed"};

const std::string DB\_NAME {"polyclinic"};

const std::uint16\_t DB\_PORT {5432};

const std::uint16\_t SERVER\_PORT {8081};

const std::uint8\_t SERVER\_LISTEN\_SIZE = 10;

const char STR\_EOF = '\0';

const char EXTENSTION\_SEPARATOR = '.';

const std::uint16\_t BUFFER\_SIZE = 2048;

const std::string DEFAULT\_HTML\_FILE = "index.html";

} // !server::common::config;

#endif // !POLYCLINIC\_SERVER\_COMMON\_CONFIG\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_FILE\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_FILE\_HPP

#include <string>

#include <cinttypes>

#include <sys/stat.h>

#include <fcntl.h>

namespace server::common {

class File {

public:

enum class OpenTypeEnum {

READ\_ONLY = O\_RDONLY,

WRITE\_ONLY = O\_WRONLY,

READ\_WRITE = O\_RDWR

};

public:

std::int32\_t open(std::string file\_path, OpenTypeEnum modificator);

std::int32\_t close(void);

std::string latest\_error(void) const;

std::int32\_t size(void) const noexcept;

std::int32\_t file\_fd(void) const noexcept;

private:

std::int32\_t update\_file\_stat(void);

private:

std::int32\_t m\_file\_fd;

struct stat m\_file\_stat;

};

} // !server::common;

#endif // !POLYCLINIC\_SERVER\_COMMON\_FILE\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_ENUM\_STRING\_CONVERTOR\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_ENUM\_STRING\_CONVERTOR\_HPP

#include "HttpHeaders.hpp"

#include <string>

#include <unordered\_map>

namespace server::common {

class EnumStringConvertor {

private:

EnumStringConvertor(void);

public:

static EnumStringConvertor\* init();

std::string to\_string(HttpHeaders::HttpRequestType val);

HttpHeaders::HttpRequestType str\_to\_request\_type(std::string val);

private:

std::unordered\_map<HttpHeaders::HttpRequestType, std::string> m\_http\_request\_types;

};

} // !server::common;

#endif // !POLYCLINIC\_SERVER\_COMMON\_ENUM\_STRING\_CONVERTOR\_HPP;#ifndef POLYCLINIC\_SERVER\_COMMON\_SLOT

#define POLYCLINIC\_SERVER\_COMMON\_SLOT

#include "Slot.hpp"

#include "src/Slot.cpp"

#endif // !POLYCLINIC\_SERVER\_COMMON\_SLOT;#ifndef POLYCLINIC\_SERVER\_COMMON\_SIGNAL

#define POLYCLINIC\_SERVER\_COMMON\_SIGNAL

#include "Signal.hpp"

#include "src/Signal.cpp"

#endif // !POLYCLINIC\_SERVER\_COMMON\_SIGNAL;#include "DBQuery.hpp"

#include "config.hpp"

#include "Logger.hpp"

#include <pqxx/result>

#include <sstream>

#include <thread>

namespace config = server::common::config;

namespace server::db

{

DBQuery::DBQuery() {

BDECLARE\_TAG\_SCOPE("DBQuery", \_\_FUNCTION\_\_);

std::stringstream db\_connection\_command\_stream;

db\_connection\_command\_stream << "host=" << config::DB\_HOST << " port=" << config::DB\_PORT

<< " user=" << config::DB\_USERNAME << " password=" << config::DB\_PASSWORD

<< " dbname=" << config::DB\_NAME;

BLOG\_INFO("constructor called on thread #", std::this\_thread::get\_id());

BLOG\_INFO("Make connection to db. ", db\_connection\_command\_stream.str());

m\_db\_connection = std::make\_unique<pqxx::connection>(db\_connection\_command\_stream.str());

if (m\_db\_connection->is\_open()) {

BLOG\_DEBUG("Connection success");

} else {

BLOG\_FATAL("Connection failed");

abort();

}

};

DBQuery::~DBQuery(void) {

m\_db\_connection->disconnect();

}

void DBQuery::output\_all\_users(void) {

BDECLARE\_TAG\_SCOPE("DBQuery", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

auto db\_transaction = std::make\_unique<pqxx::work>(\*m\_db\_connection);

pqxx::result res = db\_transaction->exec("SELECT \* FROM users");

db\_transaction->commit();

for (const auto &row : res) {

std::stringstream table\_info\_output;

for (const auto &field : row) {

table\_info\_output << field.c\_str() << '\t';

}

BLOG\_DEBUG(table\_info\_output.str());

}

}

std::pair<bool, UserInfo> DBQuery::check\_signin\_is\_valid(std::string email, std::string password) {

BDECLARE\_TAG\_SCOPE("DBQuery", \_\_FUNCTION\_\_);

std::stringstream db\_request;

auto db\_transaction = std::make\_unique<pqxx::work>(\*m\_db\_connection);

db\_request << "SELECT \* FROM check\_user\_login('"

<< email << "', '" << password << "')";

BLOG\_INFO("Current request: ", db\_request.str());

pqxx::result res = db\_transaction->exec(db\_request.str());

db\_transaction->commit();

bool first = res.size() > 0 && res[0][0].as<bool>();

if (first) {

return {first, get\_user\_info\_by\_email(email)};

}

return {first, {}};

}

UserInfo DBQuery::get\_user\_info\_by\_email(const std::string& email) {

BDECLARE\_TAG\_SCOPE("DBQuery", \_\_FUNCTION\_\_);

std::stringstream db\_request;

UserInfo ret;

auto db\_transaction = std::make\_unique<pqxx::work>(\*m\_db\_connection);

db\_request << "SELECT \* FROM user\_details\_view WHERE email = '"

<< email << "'";

BLOG\_INFO("Current request: ", db\_request.str());

pqxx::result res = db\_transaction->exec(db\_request.str());

db\_transaction->commit();

for (const auto& row : res) {

ret.id = row["id"].as<std::uint64\_t>();

ret.name = row["name"].as<std::string>();

ret.surname = row["surname"].as<std::string>();

ret.middle\_name = row["middle\_name"].as<std::string>();

ret.email = row["email"].as<std::string>();

ret.role = row["role"].as<std::string>();

// ret.phone\_numbers = row["name"].as<std::string>();

std::stringstream phone\_numbers\_ss(row["phone\_numbers"].as<std::string>());

std::string phone\_number;

while(std::getline(phone\_numbers\_ss, phone\_number, ',')) {

ret.phone\_numbers.push\_back(phone\_number);

};

}

return ret;

}

} // !server::db;#ifndef POLYCLINIC\_SERVER\_DB\_DBQUERY\_HPP

#define POLYCLINIC\_SERVER\_DB\_DBQUERY\_HPP

#include <pqxx/connection>

#include <pqxx/transaction>

#include <string>

#include <vector>

#include <memory>

namespace server::db {

struct UserInfo {

std::uint64\_t id;

std::string name;

std::string surname;

std::string middle\_name;

std::string email;

std::string role;

std::vector<std::string> phone\_numbers;

};

class DBQuery {

public:

DBQuery(void);

DBQuery(const DBQuery &other) = delete;

~DBQuery(void);

void operator=(const DBQuery &other) = delete;

void output\_all\_users(void);

std::pair<bool, UserInfo> check\_signin\_is\_valid(std::string email, std::string password);

UserInfo get\_user\_info\_by\_email(const std::string& email);

private:

std::unique\_ptr<pqxx::connection> m\_db\_connection;

};

} // !server::db;

#endif // !POLYCLINIC\_SERVER\_DB\_DBQUERY\_HPP;#include <gtest/gtest.h>

TEST(MyTestSuite, TestName) {

EXPECT\_EQ(2 + 2, 4);

}#include "ClientHandlerController.hpp"

#include "ClientHandlerModel.hpp"

#include "config.hpp"

#include "EnumStringConvertor.hpp"

#include "HttpHeaders.hpp"

#include "Socket.hpp"

#include "Logger.hpp"

#include <json/json.h>

#include <thread>

#include <unordered\_map>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/sendfile.h>

namespace server::client\_handler::controllers {

ClientHandlerController::ClientHandlerController(std::weak\_ptr<serverstarter::models::IServerStarterModel> server\_model,

std::weak\_ptr<view::IClientHandlerInterface> client\_handler\_interface,

std::weak\_ptr<context\_handler::view::IContextHandlerInterface> context\_handler\_interface

) : m\_SERVER\_STARTER\_MODEL{server\_model}, m\_client\_handler\_interface{client\_handler\_interface.lock()}

, m\_context\_handler\_interface{context\_handler\_interface.lock()}

, on\_page\_updated\_slot{std::bind(&ClientHandlerController::on\_page\_updated, this, std::placeholders::\_1, std::placeholders::\_2 )}

{

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("constructor called on thread #", std::this\_thread::get\_id());

m\_db\_query = std::make\_shared<db::DBQuery>();

}

ClientHandlerController::~ClientHandlerController(void) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("destructor called on thread #", std::this\_thread::get\_id());

for (auto &client : m\_client\_handler\_model\_container) {

client->socket().close();

if (!client->socket().is\_valid()) {

BLOG\_DEBUG("client = ", client->socket().to\_string(), " is closed");

} else {

BLOG\_WARNING("Impossible to close the client = ", client->socket().to\_string(), ". ", client->socket().latest\_error());

}

}

}

void ClientHandlerController::start(void) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

init();

catch\_new\_connection();

}

void ClientHandlerController::init() {

common::connect(&m\_context\_handler\_interface->page\_updated, &on\_page\_updated\_slot);

}

void ClientHandlerController::on\_page\_updated(std::string address, common::Socket socket) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("Page address = ", address);

}

void ClientHandlerController::catch\_new\_connection(void) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

auto server\_socket = m\_SERVER\_STARTER\_MODEL->socket();

while (server\_socket.is\_valid()) {

BLOG\_DEBUG("waiting for connection");

auto client\_socket = server\_socket.accept();

if (!client\_socket.is\_valid()) {

BLOG\_ERROR("impossible to connect to client: ", client\_socket.latest\_error());

continue;

}

handle\_connect(client\_socket);

}

}

void ClientHandlerController::disconnect(std::weak\_ptr<models::IClientHandlerModel> weak\_client) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

auto client = weak\_client.lock();

BLOG\_INFO("client: ", client->socket().to\_string());

m\_client\_handler\_interface->on\_client\_disconnected.emit(client->socket());

m\_client\_handler\_model\_container.erase(client);

if (client->socket().close()) {

BLOG\_ERROR("Cannot to close the client!");

}

}

void ClientHandlerController::read\_data(std::weak\_ptr<models::IClientHandlerModel> weak\_client) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

auto client = weak\_client.lock();

BLOG\_INFO("client: ", client->socket().to\_string());

while (client->socket().is\_valid()) {

BLOG\_DEBUG("Reading client: ", client->socket().to\_string());

std::string read\_buf;

auto bytes\_read = client->socket().read(read\_buf, common::config::BUFFER\_SIZE);

handle\_read(client, std::move(read\_buf), bytes\_read);

}

}

void ClientHandlerController::send\_data(std::weak\_ptr<models::IClientHandlerModel> weak\_client) {

}

void ClientHandlerController::handle\_connect(const common::Socket& client\_socket) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

auto client = std::make\_shared<models::ClientHandlerModel>();

client->set\_socket(std::move(client\_socket));

BLOG\_INFO("client: ", client->socket().to\_string());

m\_client\_handler\_model\_container.insert(client);

std::thread(&ClientHandlerController::read\_data, this, client).detach();

}

void ClientHandlerController::handle\_read(std::weak\_ptr<models::IClientHandlerModel> weak\_client, std::string&& read\_data, std::int32\_t bytes\_read) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

auto client = weak\_client.lock();

BLOG\_INFO("client: ", client->socket().to\_string(), "bytes\_read = ", bytes\_read, "; data = ", read\_data);

if (bytes\_read > 0) {

common::HttpHeaders http\_headers(read\_data);

handle\_http\_request(client, http\_headers);

} else if (bytes\_read == 0) {

BLOG\_INFO("Disconnect client", client->socket().to\_string());

disconnect(client);

// TODO: create signal-slot for disconnect client;

} else {

BLOG\_ERROR("Error in reading data: ", client->socket().latest\_error());

}

}

void ClientHandlerController::handle\_http\_request(std::weak\_ptr<models::IClientHandlerModel> weak\_client, const common::HttpHeaders &header) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

auto client = weak\_client.lock();

BLOG\_INFO("client: ", client->socket().to\_string(), "; http request type = ", common::EnumStringConvertor::init()->to\_string(header.method), "; http body = ", header.body);

switch (header.method) {

case common::HttpHeaders::HttpRequestType::GET:

m\_context\_handler\_interface->request\_to\_open\_uri(header.uri, client->socket());

break;

case common::HttpHeaders::HttpRequestType::POST:

BLOG\_WARNING("Tmp logic for the POST request");

std::thread(&ClientHandlerController::handle\_json\_post, this, client->socket(), header.uri, header.body).detach();

break;

default:

BLOG\_WARNING("HTTP request method \"", common::EnumStringConvertor::init()->to\_string(header.method), "\" is not handledsss");

break;

}

}

void ClientHandlerController::handle\_json\_post(common::Socket socket, std::string uri, std::string json\_pkg) {

BDECLARE\_TAG\_SCOPE("ClientHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("Socket = ", socket.to\_string(), "; uri = ", uri, " json content = ", json\_pkg);

Json::Value json\_data;

Json::Value json\_response;

Json::CharReaderBuilder json\_reader;

std::istringstream json\_stream(json\_pkg);

std::string err;

common::HttpHeaders http\_header;

std::stringstream request;

if (!Json::parseFromStream(json\_reader, json\_stream, &json\_data, &err)) {

BLOG\_ERROR("Impossible to read json data: ", err);

return;

}

if (uri == "/api/login") {

BLOG\_DEBUG("email = ", json\_data["email"].asString(), "; password = ", json\_data["password"].asString());

auto login\_res = m\_db\_query->check\_signin\_is\_valid(json\_data["email"].asString(), json\_data["password"].asString());

json\_response["is\_login\_success"] = login\_res.first;

json\_response["token"] = login\_res.second.id;

json\_response["name"] = login\_res.second.name;

json\_response["surname"] = login\_res.second.surname;

json\_response["middle\_name"] = login\_res.second.middle\_name;

json\_response["email"] = login\_res.second.email;

json\_response["role"] = login\_res.second.role;

Json::Value phone\_numbers(Json::arrayValue);

for (const auto& number : login\_res.second.phone\_numbers) {

phone\_numbers.append(number);

}

json\_response["phone\_numbers"] = phone\_numbers;

request << "HTTP/1.1 200 OK\r\n";

} else {

BLOG\_WARNING("uri = ", uri, "; is not handled!");

}

request << http\_header.extension\_to\_content\_type("json");

auto json\_response\_str = std::move(json\_response.toStyledString());

request << "Content-Length: " << json\_response\_str.size() << "\r\n";

request << "Connection: close\r\n\r\n";

request << json\_response\_str;

socket.send(request.str());

BLOG\_DEBUG("Send http response : ", request.str());

}

} // !server::client\_handler::controllers;#ifndef POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_CONTROLLERS\_CLIENTHANDLERCONTROLLER\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_CONTROLLERS\_CLIENTHANDLERCONTROLLER\_HPP

#include "IClientHandlerModel.hpp"

#include "IServerStarterModel.hpp"

#include "IContextHandlerInterface.hpp"

#include "IClientHandlerInterface.hpp"

#include "DBQuery.hpp"

#include "HttpHeaders.hpp"

#include "Signal"

#include <set>

#include <memory>

namespace server::client\_handler::controllers {

class ClientHandlerController {

public:

ClientHandlerController(std::weak\_ptr<serverstarter::models::IServerStarterModel> server\_model,

std::weak\_ptr<view::IClientHandlerInterface> client\_handler\_interface,

std::weak\_ptr<context\_handler::view::IContextHandlerInterface> context\_handler\_interface

);

~ClientHandlerController(void);

void start(void);

/\*

\* Slots starts;

\*/

public:

common::Slot<std::string, common::Socket> on\_page\_updated\_slot;

void on\_page\_updated(std::string address, common::Socket socket);

/\*

\* Slots ends;

\*/

private:

void init();

void catch\_new\_connection(void);

void disconnect(std::weak\_ptr<models::IClientHandlerModel> weak\_client);

void read\_data(std::weak\_ptr<models::IClientHandlerModel> weak\_client);

void send\_data(std::weak\_ptr<models::IClientHandlerModel> weak\_client);

void handle\_connect(const common::Socket& client\_socket);

void handle\_read(std::weak\_ptr<models::IClientHandlerModel> weak\_client, std::string&& read\_data, std::int32\_t bytes\_read);

void handle\_http\_request(std::weak\_ptr<models::IClientHandlerModel> weak\_client, const common::HttpHeaders &header);

void handle\_json\_post(common::Socket socket, std::string uri, std::string json\_pkg);

private:

std::shared\_ptr<context\_handler::view::IContextHandlerInterface> m\_context\_handler\_interface;

std::shared\_ptr<view::IClientHandlerInterface> m\_client\_handler\_interface;

std::shared\_ptr<db::DBQuery> m\_db\_query;

std::shared\_ptr<const serverstarter::models::IServerStarterModel> m\_SERVER\_STARTER\_MODEL;

std::set<std::shared\_ptr<models::IClientHandlerModel>> m\_client\_handler\_model\_container;

};

} // !server::client\_handler::controllers;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_CONTROLLERS\_CLIENTHANDLERCONTROLLER\_HPP;#include "ClientHandlerModel.hpp"

namespace server::client\_handler::models {

common::Socket ClientHandlerModel::socket() const {

std::lock\_guard guard(m\_socket\_mutex);

return m\_socket;

}

void ClientHandlerModel::set\_socket(const common::Socket& val) {

std::lock\_guard guard(m\_socket\_mutex);

m\_socket = val;

}

void ClientHandlerModel::set\_socket(common::Socket&& val) {

std::lock\_guard guard(m\_socket\_mutex);

m\_socket = std::move(val);

}

} // !server::client\_handler::models;#ifndef POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_MODELS\_CLIENTHANDLERMODEL\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_MODELS\_CLIENTHANDLERMODEL\_HPP

#include "IClientHandlerModel.hpp"

#include <atomic>

#include <mutex>

namespace server::client\_handler::models {

class ClientHandlerModel : public IClientHandlerModel {

public:

common::Socket socket(void) const override;

void set\_socket(const common::Socket& val) override;

void set\_socket(common::Socket&& val) override;

private:

mutable std::mutex m\_socket\_mutex;

common::Socket m\_socket;

};

} // !server::client\_handler::models;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_MODELS\_CLIENTHANDLERMODEL\_HPP;#ifndef POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_MODELS\_ICLIENTHANDLERMODEL\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_MODELS\_ICLIENTHANDLERMODEL\_HPP

#include "Socket.hpp"

#include <cinttypes>

#include <netinet/in.h>

namespace server::client\_handler::models {

class IClientHandlerModel {

public:

virtual common::Socket socket(void) const = 0;

virtual void set\_socket(const common::Socket& val) = 0;

virtual void set\_socket(common::Socket&& val) = 0;

};

} // !server::client\_handler::models;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_MODELS\_ICLIENTHANDLERMODEL\_HPP;#include "ClientHandlerInterface.hpp"

namespace server::client\_handler::view {

} // !server::client\_handler::view;#ifndef POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_VIEW\_CLIENTHANDLERINTERFACE\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_VIEW\_CLIENTHANDLERINTERFACE\_HPP

#include "IClientHandlerInterface.hpp"

namespace server::client\_handler::view {

class ClientHandlerInterface : public IClientHandlerInterface {

};

} // !server::client\_handler::view;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_VIEW\_CLIENTHANDLERINTERFACE\_HPP;#ifndef POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_VIEW\_ICLIENTHANDLERINTERFACE\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_VIEW\_ICLIENTHANDLERINTERFACE\_HPP

#include "Signal"

#include "Socket.hpp"

namespace server::client\_handler::view {

class IClientHandlerInterface {

public:

common::Signal<common::Socket> on\_client\_disconnected;

};

} // !server::client\_handler::view;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CLIENTHANDLER\_VIEW\_ICLIENTHANDLERINTERFACE\_HPP;#include "ContextHandlerController.hpp"

#include "Logger.hpp"

#include "config.hpp"

#include "HttpHeaders.hpp"

#include "File.hpp"

#include <sys/stat.h>

#include <unistd.h>

#include <sys/sendfile.h>

#include <sstream>

#include <fcntl.h>

#include <cstring>

namespace server::context\_handler::controllers {

ContextHandlerController::ContextHandlerController(std::weak\_ptr<models::IContextHandlerModel> model,

std::weak\_ptr<view::IContextHandlerInterface> view)

: m\_context\_handler\_model{model.lock()}

, m\_context\_handler\_interface{view.lock()}

, on\_open\_uri\_slot{std::bind(&ContextHandlerController::on\_open\_uri, this, std::placeholders::\_1, std::placeholders::\_2)}

{

BDECLARE\_TAG\_SCOPE("ContextHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("Constructor called");

init();

}

void ContextHandlerController::init() {

common::connect(&m\_context\_handler\_interface->open\_uri, &on\_open\_uri\_slot);

}

void ContextHandlerController::on\_open\_uri(std::string uri, common::Socket socket) {

BDECLARE\_TAG\_SCOPE("ContextHandlerController", \_\_FUNCTION\_\_);

BLOG\_INFO("uri = ", uri, "; socket = ", socket.to\_string());

m\_context\_handler\_model->set\_current\_uri(uri);

std::stringstream request;

/\*

[] -> [ ]

0x04234324 - memory where std::string::char\_ptr;

size = 234;

buf\_size = 280;

std::string obj

obj += "" // 100 bytesl

234 + 100 - 280 < 0; // not enough memory;

tmp\_char\_ptr = new char (size + 100 + 20) // allocate new buf;

strcat(char\_ptr, tmp\_char\_ptr); // tmp\_char\_ptr = char\_ptr;

strcat(new\_additional\_data, tmp\_char\_ptr + size + 1);

delete char\_ptr;

char\_ptr = tmp\_char\_ptr;

\*/

request << "HTTP/1.1 200 OK\r\n";

std::string file\_path = "./frontend\_web";

std::int32\_t extension\_index\_pos = uri.find\_last\_of(common::config::EXTENSTION\_SEPARATOR);

std::string extension = "html";

if (extension\_index\_pos != std::string::npos && extension\_index\_pos < uri.size()) {

extension = uri.substr(extension\_index\_pos + 1);

}

common::HttpHeaders http\_header;

request << http\_header.extension\_to\_content\_type(extension);

file\_path += uri;

if (extension == "html" && extension\_index\_pos == std::string::npos) {

file\_path += common::config::DEFAULT\_HTML\_FILE;

}

BLOG\_DEBUG("request = ", request.str(), "; file\_path = ", file\_path);

common::File requested\_file;

if (requested\_file.open(file\_path, common::File::OpenTypeEnum::READ\_ONLY) < 0) {

BLOG\_ERROR("Cannot open file path: ", file\_path, ". Error: ", requested\_file.latest\_error());

std::string error\_response = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n\r\n<h1>404 Not Found</h1>";

socket.send(error\_response);

requested\_file.close();

return;

}

request << "Content-Length: " << requested\_file.size() << "\r\n";

request << "Connection: close\r\n\r\n";

std::string response = request.str();

if (socket.send(response) < 0) {

BLOG\_ERROR("Failed to write HTTP headers: ", strerror(errno));

requested\_file.close();

return;

}

auto sent\_bytes = socket.sendfile(requested\_file);

if (sent\_bytes == requested\_file.size()) {

BLOG\_DEBUG("Sent file: ", file\_path, " happened successfull. Was sent ", sent\_bytes, " bytes.");

} else {

BLOG\_ERROR("Error sending file. Bytes sent = ", sent\_bytes, "; total size = ", requested\_file.size(), ". Error ", socket.latest\_error());

}

requested\_file.close();

m\_context\_handler\_interface->page\_updated.emit(uri, socket);

}

} // namespace server::context\_handler::controllers

#ifndef POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_CONTROLLERS\_CONTEXTHANDLERCONTROLLER\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_CONTROLLERS\_CONTEXTHANDLERCONTROLLER\_HPP

#include "IContextHandlerModel.hpp"

#include "IContextHandlerInterface.hpp"

#include <memory>

#include <string>

namespace server::context\_handler::controllers {

class ContextHandlerController {

public:

ContextHandlerController(std::weak\_ptr<models::IContextHandlerModel> model,

std::weak\_ptr<view::IContextHandlerInterface> view

);

void init();

void open\_default\_page\_by\_uri(const std::string& uri);

/\*

\* Slots starts;

\*/

public:

common::Slot<std::string, common::Socket> on\_open\_uri\_slot;

void on\_open\_uri(std::string address, common::Socket socket);

/\*

\* Slots ends;

\*/

private:

std::shared\_ptr<models::IContextHandlerModel> m\_context\_handler\_model;

std::shared\_ptr<view::IContextHandlerInterface> m\_context\_handler\_interface;

};

} // !server::context\_handler::controllers;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_CONTROLLERS\_CONTEXTHANDLERCONTROLLER\_HPP;#include "ContextHandlerModel.hpp"

namespace server::context\_handler::models {

std::string ContextHandlerModel::current\_uri(void) const {

std::lock\_guard guard(m\_current\_uri\_mutex);

return m\_current\_uri;

}

void ContextHandlerModel::set\_current\_uri(const std::string& val) {

std::lock\_guard guard(m\_current\_uri\_mutex);

m\_current\_uri = val;

}

void ContextHandlerModel::set\_current\_uri(std::string&& val) {

std::lock\_guard guard(m\_current\_uri\_mutex);

m\_current\_uri = std::move(val);

}

std::string ContextHandlerModel::current\_page\_file(void) const {

std::lock\_guard guard(m\_current\_page\_file\_mutex);

return m\_current\_page\_file;

}

void ContextHandlerModel::set\_current\_page\_file(const std::string& val) {

std::lock\_guard guard(m\_current\_page\_file\_mutex);

m\_current\_page\_file = val;

}

void ContextHandlerModel::set\_current\_page\_file(std::string&& val) {

std::lock\_guard guard(m\_current\_page\_file\_mutex);

m\_current\_page\_file = std::move(val);

}

} // !server::context\_handler::models;#ifndef POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_MODEL\_CONTEXTHANDLERMODEL\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_MODEL\_CONTEXTHANDLERMODEL\_HPP

#include "IContextHandlerModel.hpp"

#include <string>

#include <mutex>

namespace server::context\_handler::models {

class ContextHandlerModel : public IContextHandlerModel {

public:

std::string current\_uri(void) const;

void set\_current\_uri(const std::string& val);

void set\_current\_uri(std::string&& val);

std::string current\_page\_file(void) const override;

void set\_current\_page\_file(const std::string& val) override;

void set\_current\_page\_file(std::string&& val) override;

private:

mutable std::mutex m\_current\_uri\_mutex;

std::string m\_current\_uri;

mutable std::mutex m\_current\_page\_file\_mutex;

std::string m\_current\_page\_file;

};

} // !server::context\_handler::models;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_MODEL\_CONTEXTHANDLERMODEL\_HPP;#ifndef POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_MODEL\_ICONTEXTHANDLERMODEL\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_MODEL\_ICONTEXTHANDLERMODEL\_HPP

#include "IContextHandlerModel.hpp"

#include <string>

namespace server::context\_handler::models {

class IContextHandlerModel {

public:

virtual std::string current\_uri(void) const = 0;

virtual void set\_current\_uri(const std::string& val) = 0;

virtual void set\_current\_uri(std::string&& val) = 0;

virtual std::string current\_page\_file(void) const = 0;

virtual void set\_current\_page\_file(const std::string& val) = 0;

virtual void set\_current\_page\_file(std::string&& val) = 0;

};

} // !server::context\_handler::models;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_MODEL\_ICONTEXTHANDLERMODEL\_HPP;#include "ContextHandlerInterface.hpp"

#include "Logger.hpp"

namespace server::context\_handler::view {

void ContextHandlerInterface::request\_to\_open\_uri(std::string uri, common::Socket socket) {

BDECLARE\_TAG\_SCOPE("ContextHandlerInterface", \_\_FUNCTION\_\_);

BLOG\_VERBOSE("uri = ", uri);

open\_uri.emit(uri, socket);

}

} // !server::context\_handler::view;#ifndef POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_VIEW\_ICONTEXTHANDLERINTERFACE\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_VIEW\_ICONTEXTHANDLERINTERFACE\_HPP

#include "Signal"

#include "Socket.hpp"

#include <string>

namespace server::context\_handler::view {

class IContextHandlerInterface {

public:

virtual void request\_to\_open\_uri(std::string uri, common::Socket socket) = 0;

/\*

\* Signals starts;

\*/

public:

common::Signal<std::string, common::Socket> open\_uri;

common::Signal<std::string, common::Socket> page\_updated;

/\*

\* Signals ends;

\*/

};

} // !server::context\_handler::view;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_VIEW\_ICONTEXTHANDLERINTERFACE\_HPP;#ifndef POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_VIEW\_CONTEXTHANDLERINTERFACE\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_VIEW\_CONTEXTHANDLERINTERFACE\_HPP

#include "IContextHandlerInterface.hpp"

#include "Signal"

namespace server::context\_handler::view {

class ContextHandlerInterface : public IContextHandlerInterface {

public:

void request\_to\_open\_uri(std::string uri, common::Socket socket) override;

private:

};

} // !server::context\_handler::view;

#endif // !POLYCLINIC\_SERVER\_SERVER\_CONTEXTHANDLER\_VIEW\_CONTEXTHANDLERINTERFACE\_HPP;#include "ServerStarterController.hpp"

#include "Logger.hpp"

#include "config.hpp"

#include <thread>

#include <algorithm>

// #include <unistd.h>

namespace server::serverstarter::controllers {

ServerStarterController::ServerStarterController(std::weak\_ptr<models::IServerStarterModel> model,

std::weak\_ptr<context\_handler::view::IContextHandlerInterface> context\_handler\_interface,

std::weak\_ptr<client\_handler::view::IClientHandlerInterface> client\_handler\_interface

) : m\_server\_starter\_model{model.lock()}, m\_context\_handler\_interface{context\_handler\_interface.lock()},

m\_client\_handler\_interface{client\_handler\_interface.lock()}

{

BDECLARE\_TAG\_SCOPE("ServerStarterController", \_\_FUNCTION\_\_);

BLOG\_INFO("constructor called on thread #", std::this\_thread::get\_id());

m\_client\_handler\_controller = std::make\_shared<client\_handler::controllers::ClientHandlerController>(m\_server\_starter\_model,

m\_client\_handler\_interface, m\_context\_handler\_interface

);

}

ServerStarterController::~ServerStarterController(void) {

BDECLARE\_TAG\_SCOPE("ServerStarterController", \_\_FUNCTION\_\_);

BLOG\_INFO("destructor called on thread #", std::this\_thread::get\_id());

close();

}

void ServerStarterController::start(void) {

BDECLARE\_TAG\_SCOPE("ServerStarterController", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

run\_server();

std::thread(&client\_handler::controllers::ClientHandlerController::start, m\_client\_handler\_controller.get()).join();

}

void ServerStarterController::close(void) {

BDECLARE\_TAG\_SCOPE("ServerStarterController", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

std::int32\_t err = 0;

auto server\_socket = m\_server\_starter\_model->socket();

if (!server\_socket.is\_valid()) {

BLOG\_WARNING("Server Socket is not valid");

}

if (server\_socket.close()) {

BLOG\_ERROR("impossible to close the server ", server\_socket.to\_string(), ". ", server\_socket.latest\_error());

err = 0;

} else {

m\_server\_starter\_model->set\_socket(server\_socket);

}

}

void ServerStarterController::run\_server() {

BDECLARE\_TAG\_SCOPE("ServerStarterController", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

auto server\_socket = m\_server\_starter\_model->socket();

server\_socket.set\_port(common::config::SERVER\_PORT);

if (server\_socket.init()) {

BLOG\_FATAL("Server socket create error. ", server\_socket.latest\_error());

abort();

}

if (server\_socket.bind()) {

BLOG\_ERROR("impossible to bind", server\_socket.latest\_error());

}

if (server\_socket.listen(common::config::SERVER\_LISTEN\_SIZE)) {

BLOG\_ERROR("impossible to listen", server\_socket.latest\_error());

}

m\_server\_starter\_model->set\_socket(std::move(server\_socket));

}

} // !server::serverstarter::controllers;#ifndef POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_CONTROLLERS\_SERVERSTARTERCONTROLLER\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_CONTROLLERS\_SERVERSTARTERCONTROLLER\_HPP

#include "IServerStarterModel.hpp"

#include "IClientHandlerInterface.hpp"

#include "ClientHandlerController.hpp"

#include <memory>

namespace server::serverstarter::controllers {

class ServerStarterController {

public:

ServerStarterController(std::weak\_ptr<models::IServerStarterModel> model,

std::weak\_ptr<context\_handler::view::IContextHandlerInterface> context\_handler\_interface,

std::weak\_ptr<client\_handler::view::IClientHandlerInterface> client\_handler\_interface

);

~ServerStarterController(void);

void start(void);

void close(void);

private:

void run\_server();

private:

std::shared\_ptr<context\_handler::view::IContextHandlerInterface> m\_context\_handler\_interface;

std::shared\_ptr<client\_handler::view::IClientHandlerInterface> m\_client\_handler\_interface;

std::shared\_ptr<models::IServerStarterModel> m\_server\_starter\_model;

// controllers;

std::shared\_ptr<client\_handler::controllers::ClientHandlerController> m\_client\_handler\_controller;

};

} // !server::serverstarter::controllers;

#endif // !POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_CONTROLLERS\_SERVERSTARTERCONTROLLER\_HPP;#include "ServerStarterModel.hpp"

namespace server::serverstarter::models {

common::Socket ServerStarterModel::socket() const {

std::lock\_guard guard(m\_socket\_mutex);

return m\_socket;

}

void ServerStarterModel::set\_socket(const common::Socket& val) {

std::lock\_guard guard(m\_socket\_mutex);

m\_socket = val;

}

void ServerStarterModel::set\_socket(common::Socket&& val) {

std::lock\_guard guard(m\_socket\_mutex);

m\_socket = std::move(val);

}

} // !server::serverstarter::models;#ifndef POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_MODELS\_SERVERSTARTERMODEL\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_MODELS\_SERVERSTARTERMODEL\_HPP

#include "IServerStarterModel.hpp"

#include <mutex>

namespace server::serverstarter::models {

class ServerStarterModel : public IServerStarterModel {

public:

common::Socket socket(void) const override;

void set\_socket(const common::Socket& val) override;

void set\_socket(common::Socket&& val) override;

private:

mutable std::mutex m\_socket\_mutex;

common::Socket m\_socket;

};

} // !server::serverstarter::models;

#endif // !POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_MODELS\_SERVERSTARTERMODEL\_HPP;#ifndef POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_MODELS\_ISERVERSTARTERMODEL\_HPP

#define POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_MODELS\_ISERVERSTARTERMODEL\_HPP

#include "Socket.hpp"

namespace server::serverstarter::models {

class IServerStarterModel {

public:

virtual common::Socket socket(void) const = 0;

virtual void set\_socket(const common::Socket& val) = 0;

virtual void set\_socket(common::Socket&& val) = 0;

};

} // !server::serverstarter::models;

#endif // !POLYCLINIC\_SERVER\_SERVER\_SERVERSTARTER\_MODELS\_ISERVERSTARTERMODEL\_HPP;#include "config.hpp"

#include "DBQuery.hpp"

#include "Logger.hpp"

#include "ServerStarterController.hpp"

#include "ContextHandlerController.hpp"

#include "ServerStarterModel.hpp"

#include "ContextHandlerModel.hpp"

#include "ContextHandlerInterface.hpp"

#include "ClientHandlerInterface.hpp"

#include <cinttypes>

#include <chrono>

#include <ctime>

#include <iomanip>

#include <sstream>

#include <thread>

#include <memory>

#include <queue>

#include <sys/stat.h> // mkdir;

namespace db = server::db;

std::string current\_date\_time\_as\_string() {

// Get the current time using std::chrono

auto now = std::chrono::system\_clock::now();

// Convert the current time to a time\_t object

std::time\_t now\_c = std::chrono::system\_clock::to\_time\_t(now);

// Convert time\_t to a struct tm (broken down time)

std::tm tm\_struct = \*std::localtime(&now\_c);

// Format the date-time as a string

std::stringstream ss;

ss << std::put\_time(&tm\_struct, "%Y.%m.%d\_%H.%M.%S");

return ss.str();

}

std::string generate\_log\_file\_name(const std::string &first\_name\_part) {

std::stringstream file\_name\_stream;

file\_name\_stream << first\_name\_part << "\_" << current\_date\_time\_as\_string() << ".log";

return std::move(file\_name\_stream.str());

}

int main(int argc, char \*\*argv) {

const std::string LOGS\_FOLDER\_DIR\_NAME = "logs";

mkdir(LOGS\_FOLDER\_DIR\_NAME.c\_str(), 0755);

BLOG\_INIT(std::move(generate\_log\_file\_name(LOGS\_FOLDER\_DIR\_NAME + "/server\_logs.txt")), true);

BDECLARE\_TAG\_SCOPE("", \_\_FUNCTION\_\_);

std::queue<std::thread> thread\_pull;

auto context\_handler\_interface = std::make\_shared<server::context\_handler::view::ContextHandlerInterface>();

auto context\_handler\_model = std::make\_shared<server::context\_handler::models::ContextHandlerModel>();

auto context\_handler\_controller = std::make\_shared<server::context\_handler::controllers::ContextHandlerController>(context\_handler\_model, context\_handler\_interface);

auto server\_starter\_model = std::make\_shared<server::serverstarter::models::ServerStarterModel>();

auto client\_handler\_interface = std::make\_shared<server::client\_handler::view::ClientHandlerInterface>();

server::serverstarter::controllers::ServerStarterController server\_starter\_controller(server\_starter\_model,

context\_handler\_interface, client\_handler\_interface

);

thread\_pull.push(std::thread(&server::serverstarter::controllers::ServerStarterController::start, &server\_starter\_controller));

// dbquery.output\_all\_users();

while (!thread\_pull.empty()) {

if (thread\_pull.front().joinable()) {

thread\_pull.front().join();

}

thread\_pull.pop();

}

return 0;

}