

**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** — відповідає за підтримку працездатності, змінює ролі, слідкує за базою, курєє співробітниками та їх розкладом;
- **Доктор** — приймає записи пацієнтів, виписує рецепти, результати прийому. Оновлює мед картку пацієнта;
- **Пацієнт** — перегляд розкладу лікарів, запис до лікаря, робить запити 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 прийому Час прийому	Прийом

Задача	Вхідна інформація	Вихідна інформація
2. 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 прийому
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
3. 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	Медична картка пацієнта

	Номер телефону Ідентифікатор користувача	
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 **трюхланкова 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.

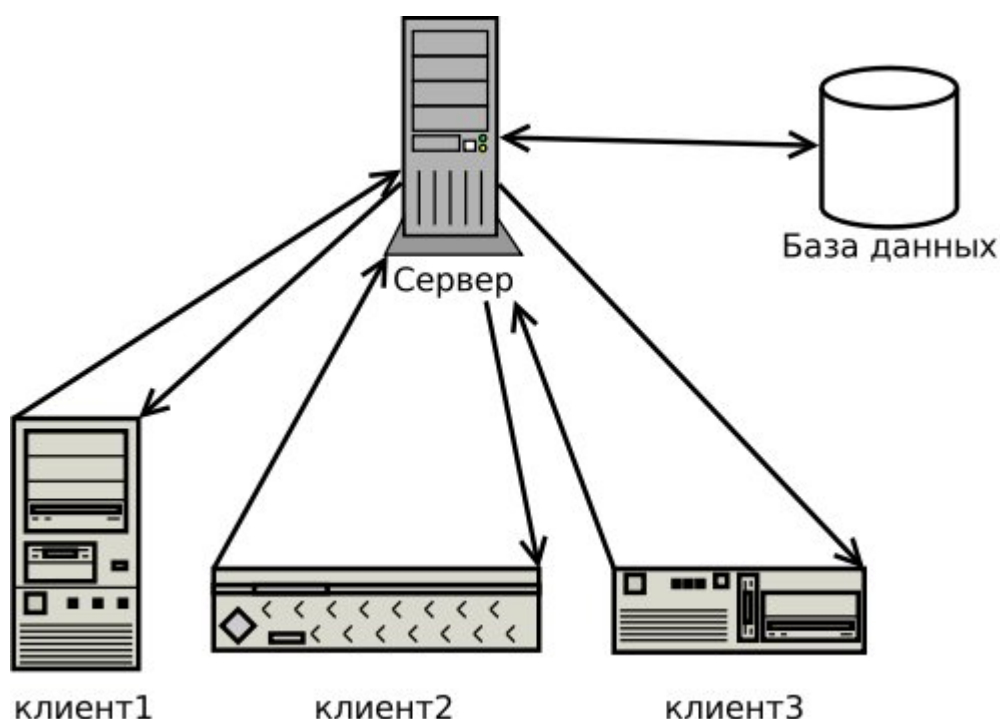


Рисунок 2.1 — схема трюхланкової архітектури.

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 гнучко та підтриємаємо.

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

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

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

Схема архітектурного шаблону наведена на рисунку 2.2.

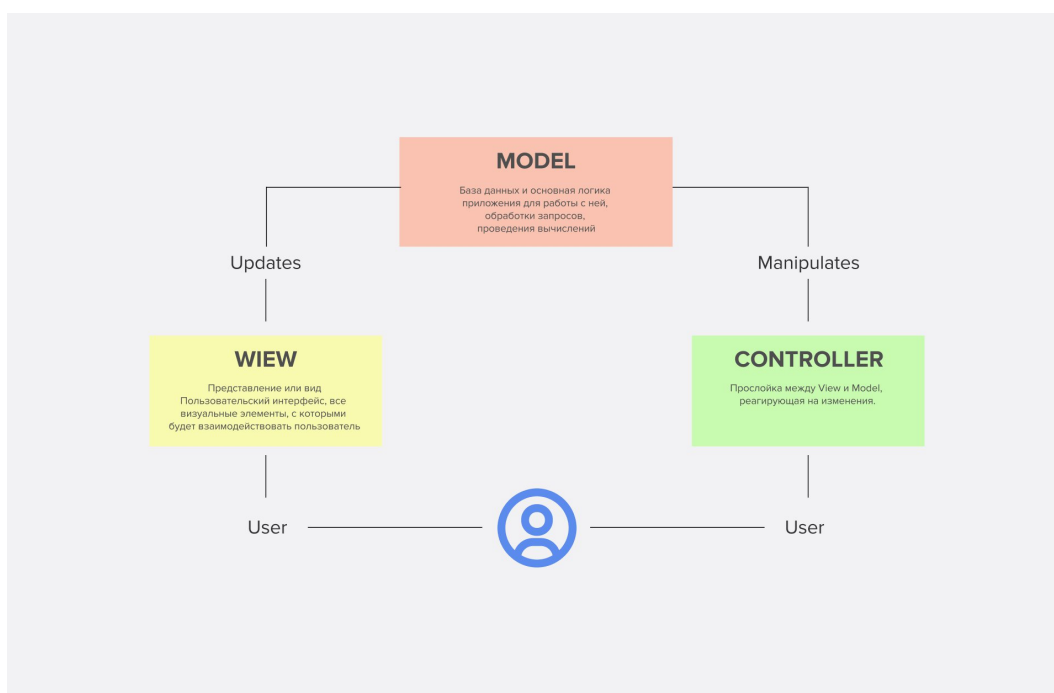


Рисунок 2.2 — схема архітектурного патерна MVC.

Преимущества MVC:

- Розділення обов'язків — розділення на три компоненти робить код більш організованим та підтримуємим.
- Модульність — усі компоненти незалежні та можуть бути замінені.

- Тестованість — усі компоненти 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.

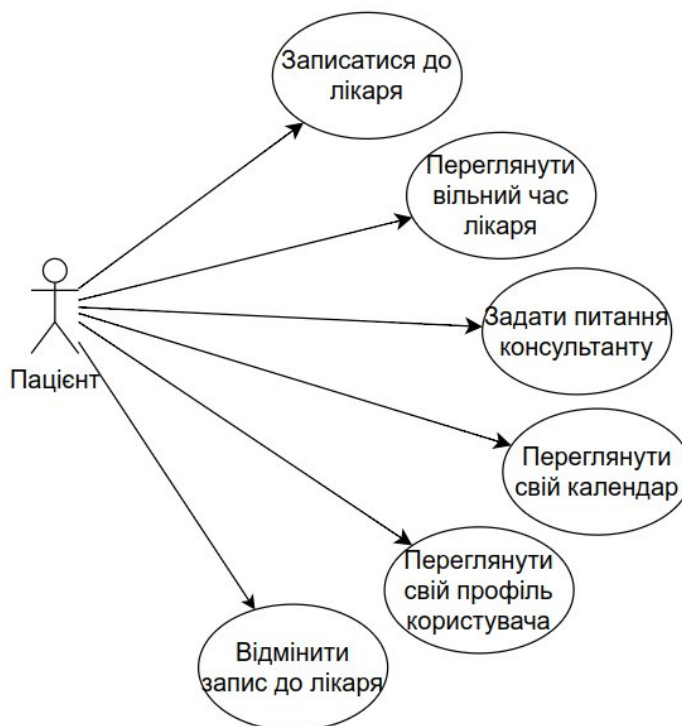


Рисунок 3.1 – діаграма прецедентів для пацієнта

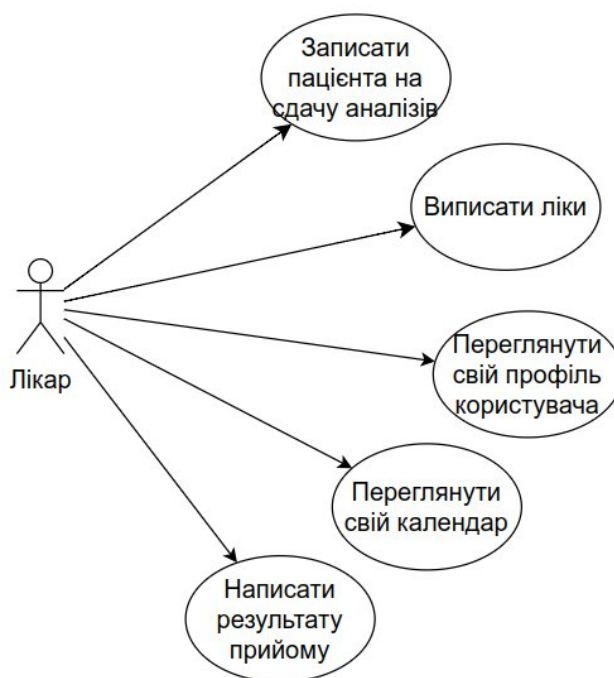


Рисунок 3.2 – діаграма прецедентів для лікаря

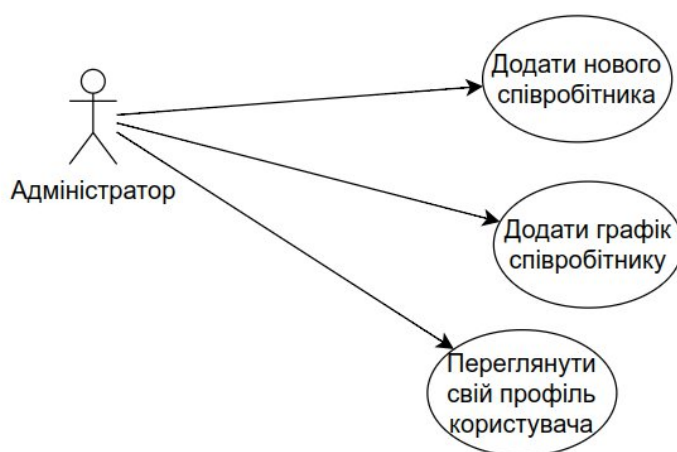


Рисунок 3.3 – діаграма прецедентів для адміністратора

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.

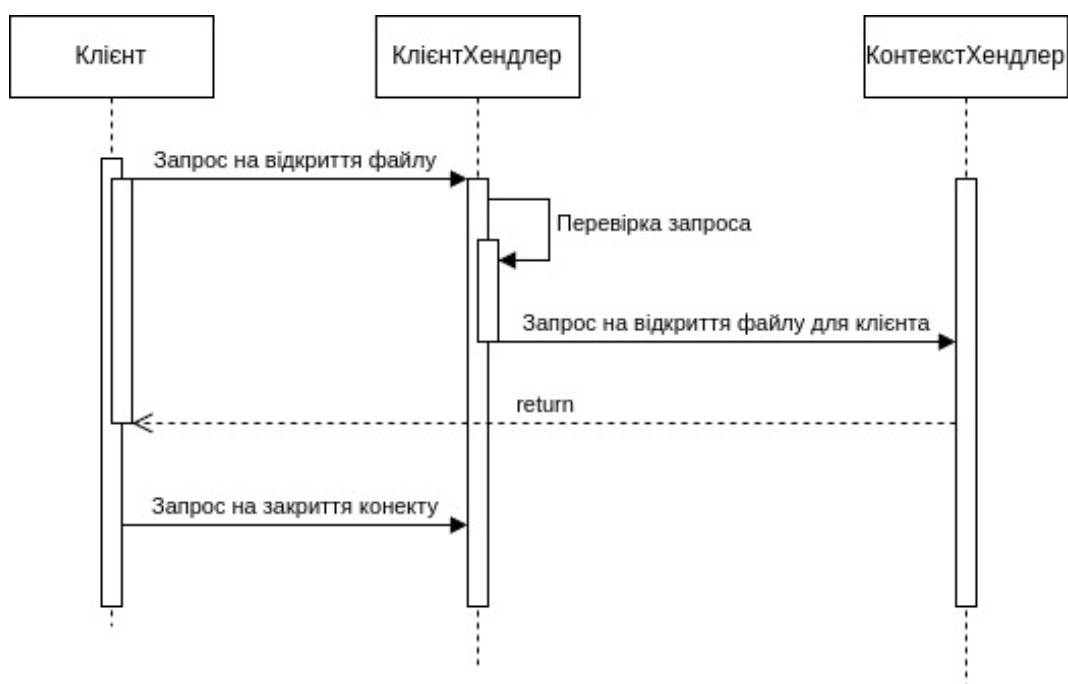


Рисунок 4.1 – діаграма послідовності для відкриття файлу

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/РоліКористувачів		
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	TIMESTAMP TZ, 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/Виписані Ліки		
id	Ідентифікатор	PRIMARY KEY (PK)
drug_id	Ідентифікатор препарату	FOREIGN KEY (FK)
doctor_appointment_id	Ідентифікатор прийому лікаря	FOREIGN KEY (FK)

Між сутностями наявні 2 типи зв'язків:

- 1:N — “один-до-багатьох”. 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 зовнішнього 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**, **gtest**, **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**, **gtest**, **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.
- **gtest:** 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.

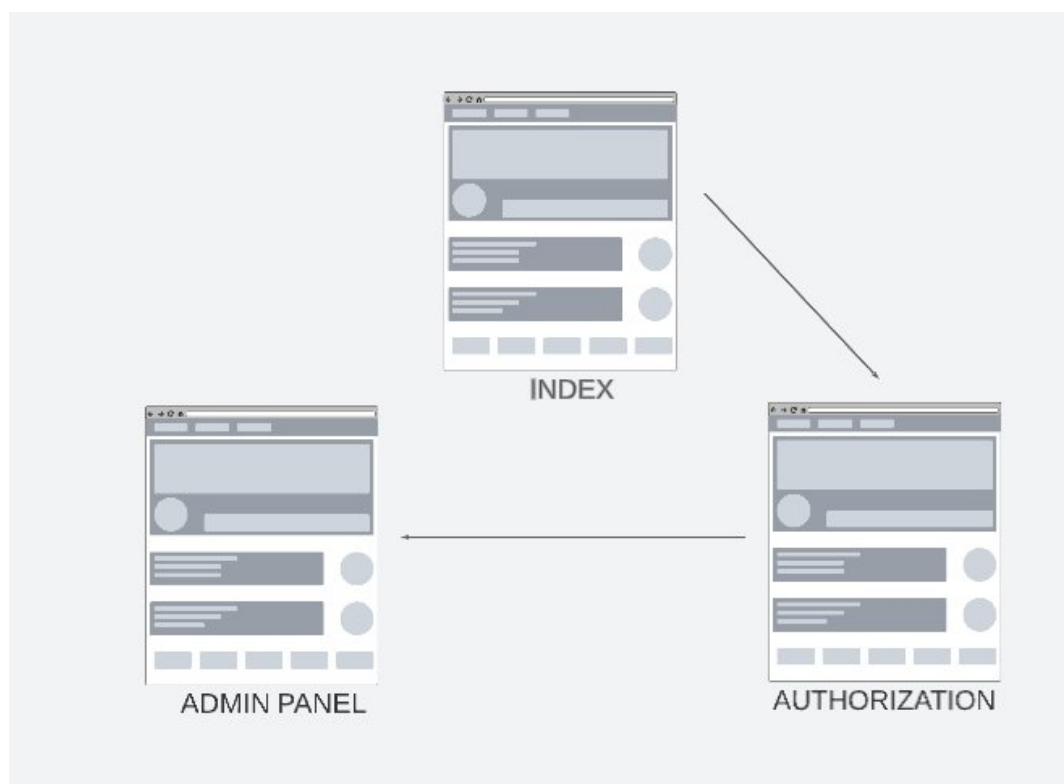


Рисунок 7.1 – ієрархія сторінок адміністратора

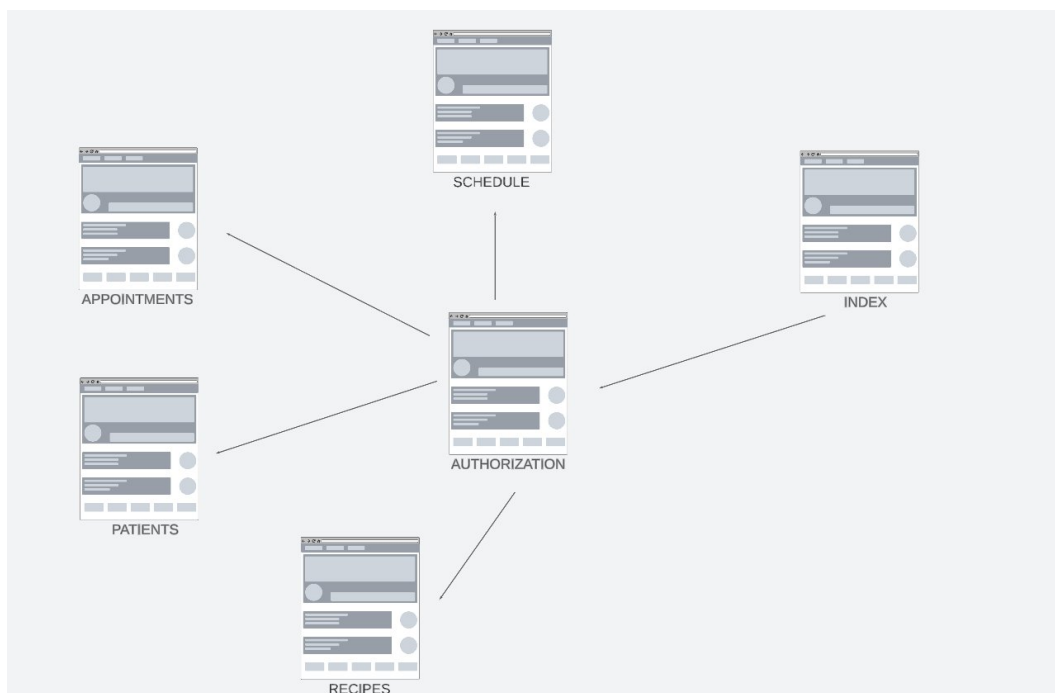


Рисунок 7.2 – ієрархія сторінок лікаря

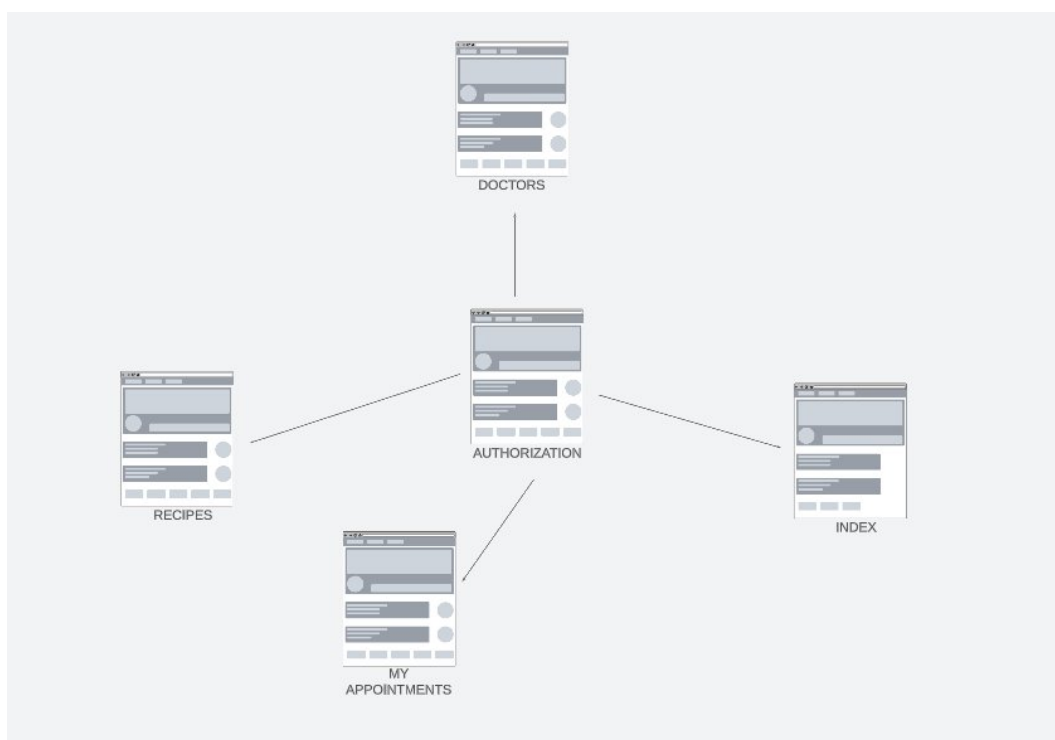


Рисунок 7.3 – ієрархія сторінок пацієнта

Шаблон MVC 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.

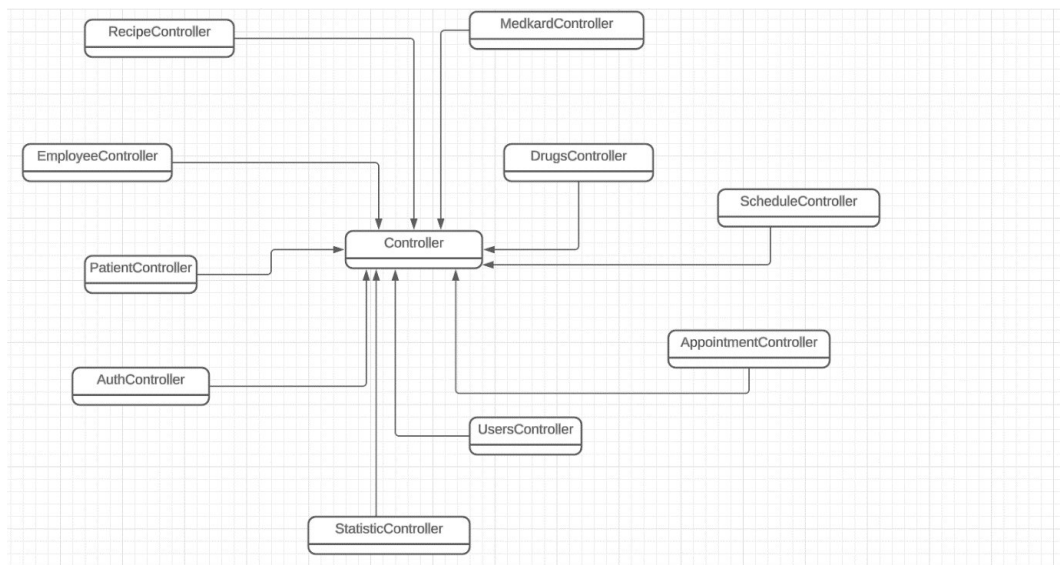


Рисунок 7.4 – Ієрархія класів – прикладного компоненту

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.

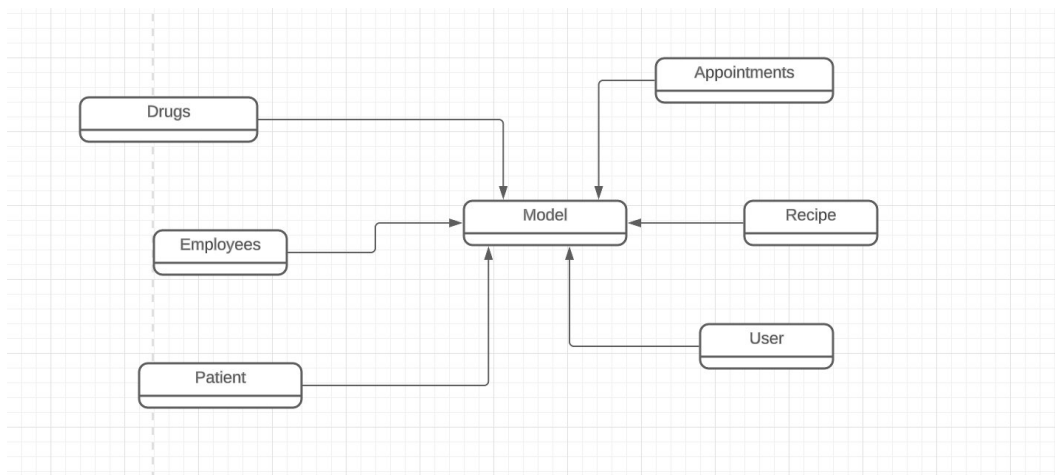


Рисунок 7.5 – Ієрархія класів – рівень управління ресурсами

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 прикладі “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;
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">
  
  <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">
    
    <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 виводу 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. Рисунок 10.1

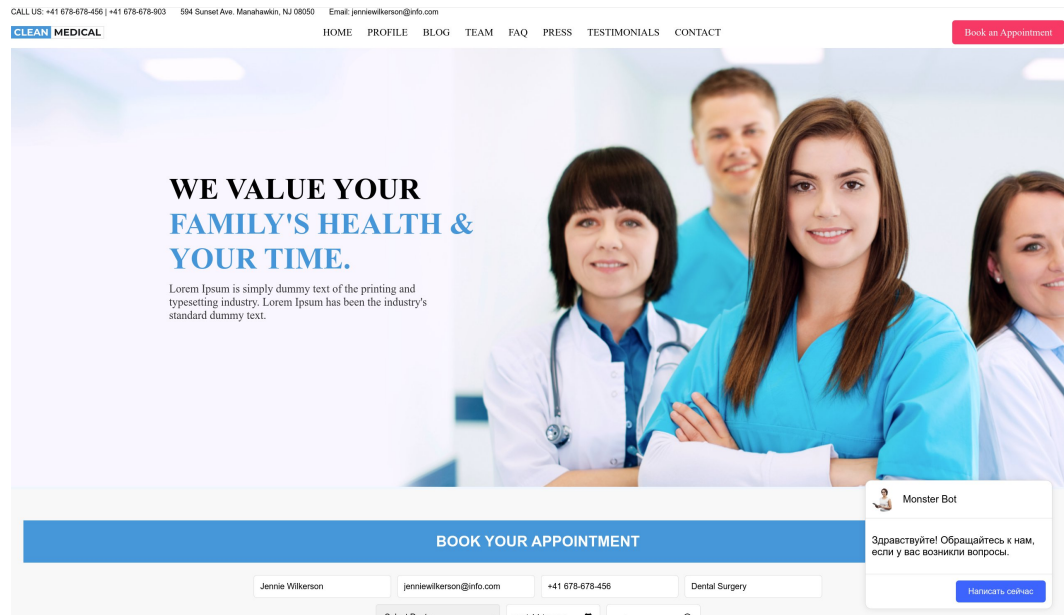
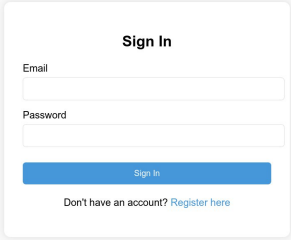


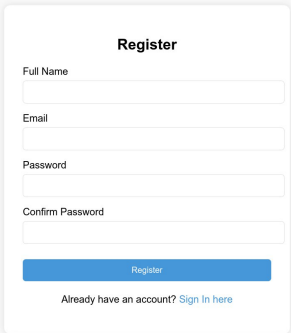
Рисунок 10.1 – головна сторінка веб-сайту

Щоб увійти у систему потрібно ввести дані користувача такі як пошта та пароль (рис 10.2). Якщо ж цих даних ще немає треба зареєструватися (рис 10.3). Після реєстрації вас перекидує на сторінку профіля користувача. Там вказані основні дані, такі як ПІБ, номер телефону, електронна 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



The image shows a 'Sign In' form centered on a light gray background. The form is a white rounded rectangle with a blue border. It has a title 'Sign In' at the top. Below the title are two input fields: 'Email' and 'Password'. Below these fields is a blue button with the text 'Sign In'. At the bottom of the form, there is a link that says 'Don't have an account? Register here'.

Рисунок 10.2 – вікно входу в профіль



The image shows a 'Register' form centered on a light gray background. The form is a white rounded rectangle with a blue border. It has a title 'Register' at the top. Below the title are four input fields: 'Full Name', 'Email', 'Password', and 'Confirm Password'. Below these fields is a blue button with the text 'Register'. At the bottom of the form, there is a link that says 'Already have an account? Sign In here'.

Рисунок 10.3 – вікно реєстрації

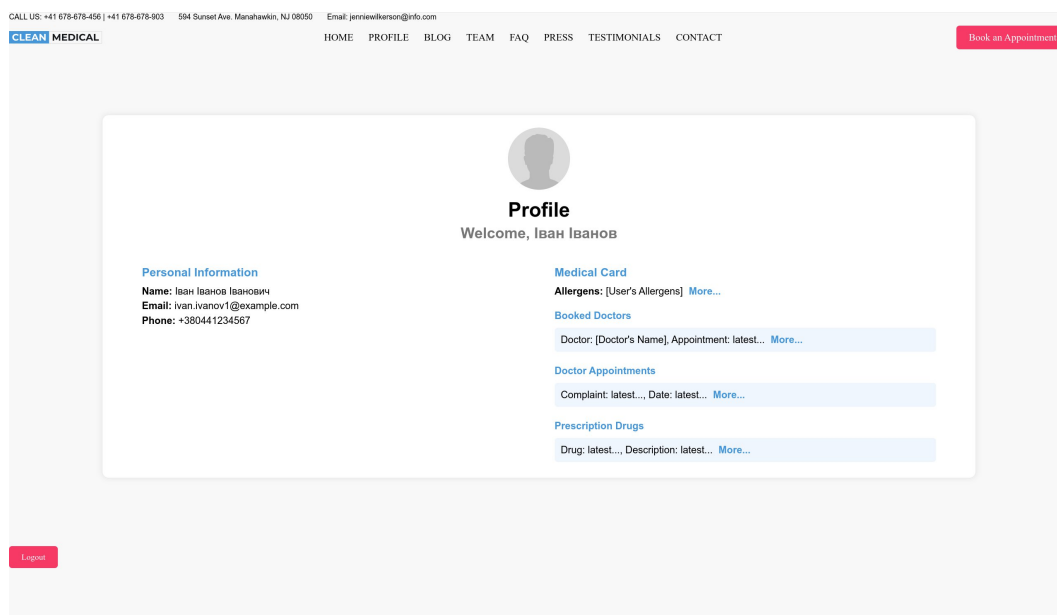


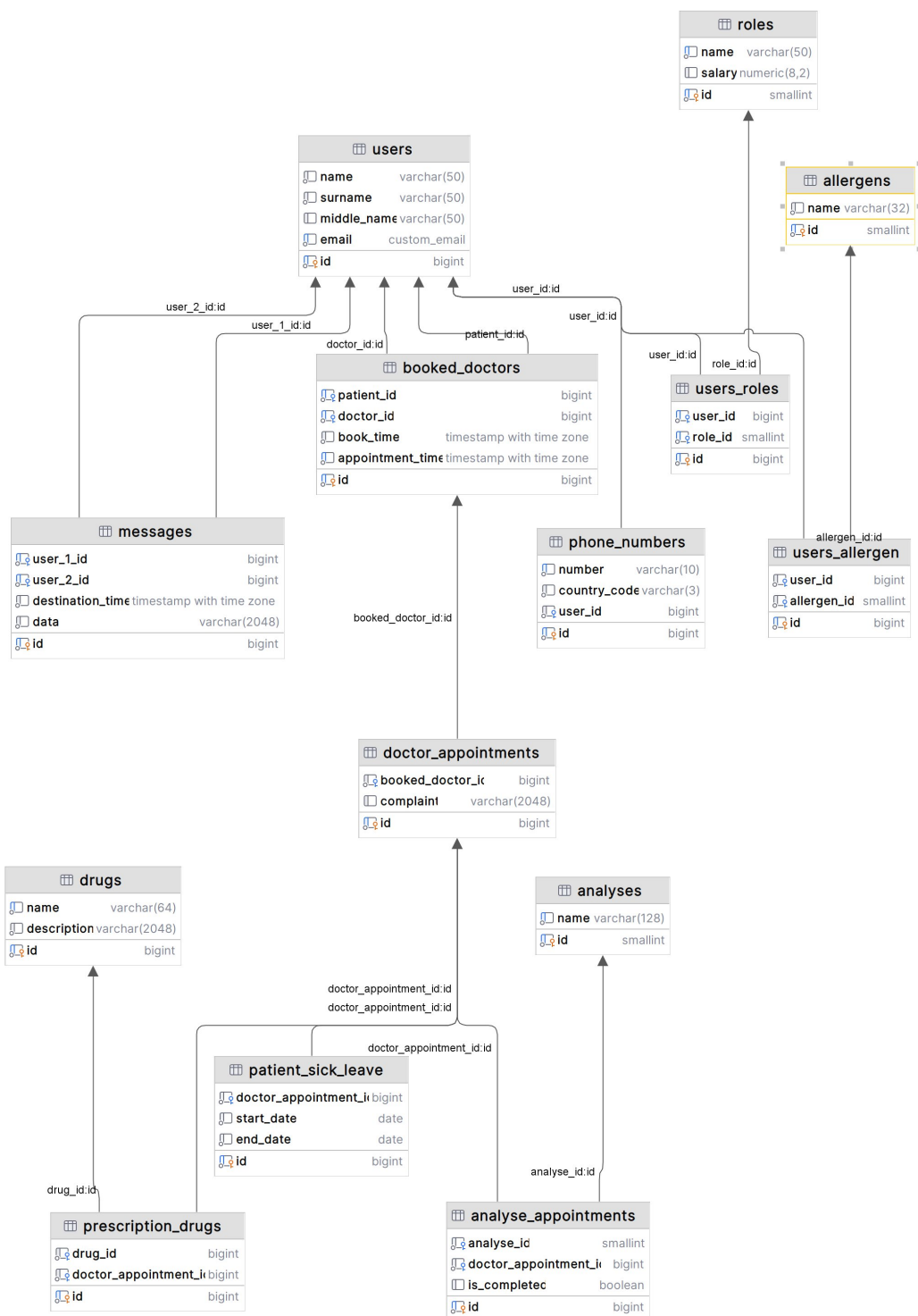
Рисунок 10.4 – профіль користувача

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/autol/index.html>
- PostgreSQL : Документація: 9.6: CREATE TRIGGER [Електронний ресурс] – Режим доступу: <https://postgrespro.ru/docs/postgresql/9.6/sql-createtrigger>
- PostgreSQL: Документація: CREATE PROCEDURE [Електронний 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 [Електронний ресурс]-Режим доступу:
<https://wiki.merionet.ru/servernye-resheniya/100/chto-takoe-arhitektura-mvc-model-view-controller/>

Додаток А

СХЕМА БАЗИ ДАНИХ



```
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 after 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)
);

```

```

/* Descripton:

```

```

* 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 - required 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 '%cepu%';
```

```
/* 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;

```

```

        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">
    
</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>
        </div>
    </form>
</div>

```

```

        </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">
        

        <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">
  
  <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">

      <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
EnumStringConverter::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("woff") != 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 modifier) {
    std::int32_t ret = -1;

    m_file_fd = ::open(file_path.c_str(), static_cast<std::int32_t>(modifier));

    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 EXTENSION_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 modifier);
    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_HP
P;#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::mo
dels::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::IClientHandlerMode
l> 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::IClientHand
lerModel> 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::istreamstream 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_CLIENHANDLER_CONTROLLERS_CLI
ENHANDLERCONTROLLER_HPP
#define
POLYCLINIC_SERVER_SERVER_CLIENHANDLER_CONTROLLERS_CLI
ENHANDLERCONTROLLER_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_CLIENTH
ANDLERMODEL_HPP
#define
POLYCLINIC_SERVER_SERVER_CLIENTHANDLER_MODELS_CLIENTH
ANDLERMODEL_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_CLIENTHAN
DLERINTERFACE_HPP
#define
POLYCLINIC_SERVER_SERVER_CLIENTHANDLER_VIEW_CLIENTHAN
DLERINTERFACE_HPP

#include "IClientHandlerInterface.hpp"

namespace server::client_handler::view {

class ClientHandlerInterface : public IClientHandlerInterface {

};

} // !server::client_handler::view;

#endif

// !POLYCLINIC_SERVER_SERVER_CLIENTHANDLER_VIEW_CLIENTH
ANDLERINTERFACE_HPP;#ifndef
POLYCLINIC_SERVER_SERVER_CLIENTHANDLER_VIEW_ICLIENTHAN
DLERINTERFACE_HPP
#define
POLYCLINIC_SERVER_SERVER_CLIENTHANDLER_VIEW_ICLIENTHAN
DLERINTERFACE_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_ICLIEN
HANDLERINTERFACE_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::ICon
textHandlerModel> 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::EXTENSION_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
#ifdef
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_CONTROLLERS_C
ONTEXTHANDLERCONTROLLER_HPP
#define
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_CONTROLLERS_C
ONTEXTHANDLERCONTROLLER_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_CONTROLLER

```

```
S_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_CONTEXT
HANDLERMODEL_HPP
#define
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_MODEL_CONTEXT
HANDLERMODEL_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_CON
TEXTHANDLERMODEL_HPP;#ifndef
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_MODEL_ICONTE
XHANDLERMODEL_HPP
#define
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_MODEL_ICONTE
XHANDLERMODEL_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_ICON
TEXTHANDLERMODEL_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_ICONTEXT
HANDLERINTERFACE_HPP
#define
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_VIEW_ICONTEXT
HANDLERINTERFACE_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_ICONT
EXTHANDLERINTERFACE_HPP;#ifndef
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_VIEW_CONTEXT
HANDLERINTERFACE_HPP

```

```

#define
POLYCLINIC_SERVER_SERVER_CONTEXTHANDLER_VIEW_CONTEXT
HANDLERINTERFACE_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_CONTE
XHANDLERINTERFACE_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::IServerSta
rterModel> 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_SER
VERSTARTERCONTROLLER_HPP
#define
POLYCLINIC_SERVER_SERVER_SERVERSTARTER_CONTROLLERS_SER
VERSTARTERCONTROLLER_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_SERVERS
TARTERMODEL_HPP
#define
POLYCLINIC_SERVER_SERVER_SERVERSTARTER_MODELS_SERVERS
TARTERMODEL_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_SERVE
RSTARTERMODEL_HPP;#ifndef

```

```

POLYCLINIC_SERVER_SERVER_SERVERSTARTER_MODELS_ISERVERS
TARTERMODEL_HPP

#define
POLYCLINIC_SERVER_SERVER_SERVERSTARTER_MODELS_ISERVERS
TARTERMODEL_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_ISERV
ERSTARTERMODEL_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 <ctime>
#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::ServerStarterContr
oller::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;
}

```