МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
"_____"_____ 2022 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Веб-додаток для служб крові на платформі *SPRING*

_____

_____

Виконавець: _____Бондаренко Б.В.

Керівник: _____ к.т.н., доцент Халімон Н. Ф.

Нормоконтролер: _____Тупота Є.В.

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРЖУЮ
Завідувач кафедри

Литвиненко О. Є.

« ____ »_____2022 р.

З А В Д А Н Н Я
на виконання дипломної роботи (проекту)

Бондаренко Богдана Володимировича
(прізвище, ім’я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проекту): Веб-додаток для служб крові на платформі SPRING
затверджена наказом ректора від "15" лютого 2022 року № 251/ст.

2. Термін виконання роботи (проекту): з 16.05.2022 до 19.06.2022

3. Вихідні дані до роботи (проекту): система управління базами даних *PostgreSQL*, інтегроване середовище розробки *Intellij IDEA*, редактор коду *VS Code*, мова *Java*, мова *JS*, мова *SQL*.

4. Зміст пояснювальної записки:
1) Програмне забезпечення служб крові.
2) Проектування веб-додатку для служб крові.
3) Розробка веб-додатку для служб крові.

5. Перелік обов’язкового графічного (ілюстративного) матеріалу:
1) *ER*-діаграма бази даних вікно веб-додатку для служб крові.
2) Схема прикладного потокового інтерфейсу *Stream API*.
3) *UML*-діаграма класів вікно веб-додатку для служб крові.
4) Головне вікно веб-додатку для служб крові.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі дипломного проектування.	17.05.22	
2	Вивчити спеціальну літературу і технічну документацію.	20.05.22	
5	Дослідити існуючі технологічні рішення систем донорства крові.	22.05.22	
6	Написати розділ 1.	24.05.22	
7	Провести проектування веб-додатку для донорства крові.	26.05.22	
8	Написати розділ 2.	30.05.22	
9	Провести розробку веб-додатку для донорства крові.	02.06.22	
10	Написати розділ 3.	05.06.22	
11	Оформити пояснювальну записку	08.06.22	
12	Підготувати графічний демонстраційний матеріал та доповідь.	09.06.22	

7. Дата видачі завдання: “15” лютого 2022 р.

Керівник дипломної роботи (проекту) _____ Халімон Н.Ф.
(підпис керівника (П.І.Б.)

Завдання прийняв до виконання _____ Бондаренко Б.В.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Веб-додаток для служб крові на платформі SPRING»: 50 с., 13 рис., 19 літературних джерел.

POSTGRESQL, *JAVA*, *HIBERNATE*, БАЗА ДАНИХ, *ER*-ДІАГРАМА, ВЕБ-ДОДАТОК СЛУЖБ КРОВІ, ПЛАТФОРМА *SPRING*.

Об’єкт проектування – автоматизовані системи обліку донорства крові.

Предмет проектування – розробка веб-додатку для служб крові на платформі *Spring*.

Мета дипломного проекту – створення веб-додатку для служб крові, який дозволить користувачу здійснювати пошук донорів та реціпієнтів.

Метод проектування – застосування крос-платформного програмного засобу *Spring* для розробки веб-додатку.

Мета дипломного проекту – розробка веб-додатку для служб крові на платформі *Spring*.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СЛУЖБ КРОВІ 9
1.1. Напрямки діяльності служб крові 9
1.2. Програмні засоби обліку донорства крові 10
1.3. Висновки до розділу 18
РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ СЛУЖБ КРОВІ 19
2.1. Платформа *Spring* та технології для створення веб-додатку 19
2.2. Проектування *ER*-моделі 23
2.3. Проектування інтерфейсу веб-додатку 26
2.4. Проектування модулів веб-додатку 28
2.5. Висновки до розділу 32
РОЗДІЛ 3 РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ СЛУЖБ КРОВІ 33
3.1. Особливості використання платформи *Spring* та налаштування середовища розробки 33
3.2. Створення проектів для серверної та клієнтської частини 36
3.3. Розробка серверної частини веб-додатку донорства крові 40
3.4. Розробка клієнтської частини веб-додатку донорства крові 48
3.5. Висновки до розділу 52
ВИСНОВКИ 54
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАННИХ ДЖЕРЕЛ 57

ВСТУП

Проблема донорства крові – одна з найважливіших для здоров'я населення в усьому світі. Практично в кожній державі існує потреба збільшення кількості запасу крові. В рамках проекту необхідно проаналізувати потреби предметної області щодо автоматизації процесів; проаналізувати та визначити сильні та слабкі сторони вже існуючих аналогів; розробити програмний продукт, який би дав можливість вирішення проблеми донорства крові. Для вирішення цього завдання використовуються аналітичні та практичні методи, щоб дослідити предметну область, а за допомогою ітеративного методу розробки, створено програмний продукт.

Служба крові – це структура, яка об'єднує по всій країні медичні установи та їх структурні підрозділи, основним видом діяльності яких є заготівля, переробка, зберігання та забезпечення безпеки донорської крові та її компонентів. В Україні слід звернутися до станції переливання крові або до центру крові, щоб стати донором крові. Першим завданням служби крові є забезпечення медичних закладів компонентами крові. Для цього служба крові організовує роботу з донорами з отримання донорської крові, обстеження, а також організовує роботу з поділу крові на компоненти, зберігання її у спеціальних умовах та транспортування до лікувальних закладів. Ще одним завданням служби крові є розвиток добровільного донорства крові. На практиці розвиток безоплатного та регулярного донорства крові є головною умовою забезпечення максимальної безпеки компонентів крові для реципієнтів та ефективного функціонування служби крові.

Використання автоматизованих систем обліку донорства крові у роботі служб крові значно спрощує ряд робочих процесів та підвищує їх ефективність при управлінні процесами взяття, розподілу донорської крові та її компонентів. Створена автоматизована система може використовуватися автономно, оскільки багатофункціональна медична інформаційна система налаштовується під потреби та особливості функціонування того чи іншого медичного центру.

Технологічні рішення в сфері автоматизованих систем донорства пропонують наступні програмні продукти: ДонорUA, автоматизована

інформаційна система трансфузіології (AICT) та *GiveBlood*. Це програмне забезпечення є найпоширенішими в Україні, Росії та Канаді відповідно.

Можливості програм служби крові дозволяють максимально ефективно управляти пошуком донорів та реципієнтів, процесами збору, перевірки, обробки крові, її компонентів та постачання до медичних центрів. Програмне забезпечення створене відповідно до вимог галузевих норм та стандартів у галузі медицини.

Платформа *Spring* ідеально підходить для розробки веб-додатків з великою кількістю користувачів та модулів. *Spring* разом з системою управління базами даних *PostgreSQL* та фреймворком для клієнтської частини додатку *Vue* надаються потужні інструменти для розробки стабільної, масштабованої, довговічної та зручної системи для донорства крові з сучасним інтерфейсом.

В другому розділі було розглянуто проектування веб-додатку для служб крові. При проектуванні додатку було визначено всі складові частини, функціональні можливості та технології для створення веб-додатку донорства крові. Було використано наступні технології: *JSON Web Token* (*JWT*) для передачі даних аутентифікації у веб-додатку для служб кров; бібліотеку *Vuetify* для створення інтерфейсу для веб-додатку служб крові, *Vuex* як реактивне сховище даних та *Vue Router* як бібліотека для створення роутингу. Для зв'язку клієнтської та серверної частини веб-додатку служб крові було використано архітектурний стиль *Representational State Transfer* (*REST*). *REST* – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

При проектуванні веб-додатку для донорства крові визначено чотири сторінки: сторінка пошуку та створення заявок для донорів і реципієнтів, сторінка реєстрації користувачів, сторінка входу користувачів, сторінка профілю користувача.

В третьому розділі було описано розробку клієнтської частини та серверної частини, особливості платформи *Spring*, основні принципи побудови додатків. Розроблений веб-додаток для служб крові складається з модулів: модуль пошуку та створення заявок донора та реципієнта, модуль аутентифікації, модуль авторизації, модуль перегляду та редагування профілю користувача, асинхронний

модуль очищення прострочених заявок, асинхронний модуль очищення недійсних *JWT* токенів.

Об’єкт проектування – автоматизовані системи обліку донорства крові.

Предмет проектування – розробка веб-додатку для служб крові на платформі *Spring*.

Мета дипломного проекту – створення веб-додатку для служб крові, який дозволить користувачу здійснювати пошук донорів та реципієнтів. Таким чином, мета проекту – розробити універсальний веб-додаток на платформі *Spring*, який збільшить ефективність роботи сервісу та зменшить витрати для служб, що займаються донорством крові.

Донорство крові та її компонентів розглядається як складова частина національної безпеки. Надання людям допомоги визначається не тільки високотехнологічною медичною допомогою, а і засобами, призначеними для комплексної інформатизації процесів донорства, тому тема дипломного проекту «Веб-додаток для служб крові на платформі *SPRING*» є актуальним завданням.

РОЗДІЛ 1
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СЛУЖБ КРОВІ

1.1. Напрямки діяльності служб крові

Служба крові – це структура, що об'єднує по всій країні медичні установи та їх структурні підрозділи, основним видом діяльності яких є заготівля, переробка, зберігання та забезпечення безпеки донорської крові та її компонентів [1]. В Україні, щоб стати донором крові, слід звернутися до станції переливання крові або до центру крові. Донором може бути будь-яка людина, що відповідає таким критеріям:

- дієздатний громадянин України;
- віком від 18 років.
- не має протипоказань і пройшов медичне обстеження.

Одним із завдань служби крові є забезпечення медичних закладів компонентами крові. Для цього служба крові організовує роботу з донорами з отримання донорської крові, обстеження, а також організовує роботу з поділу крові на компоненти, зберігання її у спеціальних умовах та транспортування до лікувальних закладів.

Наступним завданням служби крові є розвиток добровільного донорства крові. Як показує вітчизняна та світова практика, розвиток безоплатного та регулярного донорства крові є головною умовою забезпечення максимальної безпеки компонентів крові для реципієнтів та ефективного функціонування служби крові. Ефективна діяльність служби крові неможлива без участі суспільства загалом, його громадських інститутів, бізнесу, ініціативи приватних осіб.

1.2. Програмні засоби обліку донорства крові

Використання автоматизованих систем обліку донорства крові у роботі центрів трансфузіології (розділ медицини, що вивчає питання переливання біологічних рідин в організмі, зокрема крові та її компонентів) значно спрощує ряд робочих процесів та підвищує їх ефективність при управлінні процесами взяття, розподілу донорської крові та її компонентів [2]. Система може використовуватися автономно, оскільки багатофункціональна медична інформаційна система налаштовується під потреби та особливості функціонування того чи іншого медичного центру.

Основними користувачами інформаційних систем є донор крові та реципієнт. Донор крові та (або) її компонентів – фізична особа, яка добровільно пройшла медичне обстеження та добровільно здає кров та (або) її компоненти. Реципієнт – фізична особа, якій за медичними показаннями потрібна або зроблена трансфузія (переливання) донорської крові та (або) її компонентів [3].

В даний час для крові замінників не існує. Кров або клітини крові, які отримує реципієнт під час переливання, зазвичай здає інша людина. Після здавання крові її перевіряють на належність до певної групи. Також її перевіряють на сифіліс, гепатити B та C, ВІЛ, вірус, пов'язаний із дуже рідкісним видом лейкемії, вірус гарячки Західного Нілу, *Trypanosome cruzi* (паразит, що викликає хворобу Шагаса), вірус Зіка, бактерії (тільки тромбоцити). Якщо за результатами аналізів у ній виявляється будь-яка із цих інфекцій, кров не підлягає подальшому використанню [4].

При реєстрації донор заповнює анкету, в якій вказує необхідні відомості про стан свого здоров'я та спосіб життя. Далі відбувається медичне обстеження. Після цього донор здає кров з пальця в лабораторії, щоб визначити групу крові і резус, а також рівень гемоглобіну донора. Від результатів цього аналізу залежить, чи зможе людина цього дня стати донором. Потім донор відправляється до лікаря, який оглядає його, вивчає анкету, ставить додаткові питання про його здоров'я, спосіб життя та звички. Лікар приймає рішення про допуск до здачі крові. Процедура здачі

крові здійснюється у максимально комфортних для донора умовах, у спеціальному донорському кріслі.

Таким чином, автоматизовані системи рекрутингу донорів повинні мати модулі для знаходження відділень донорства крові, обходу черг на прийом до лікаря, збереження медичної картки та історії донацій. Також програмне забезпечення служб крові має обов'язково містити модуль реєстрації донора, модуль пошуку доступних відділень служби крові та часу їх роботи, модуль збереження та перегляду історії хвороби.

Технологічні рішення в сфері автоматизованих систем донорства пропонують наступні програмні продукти: ДонорUA, автоматизована інформаційна система трансфузіології (АІСТ) та *GiveBlood*. Це програмне забезпечення є найпоширенішими в Україні, Росії та Канаді відповідно.

Веб-додаток ДонорUA – це автоматизована система рекрутингу та управління донорами крові (рис. 1.1) [5]. Проект створено у співпраці ВМГО «Асоціація молодих донорів України» та української ІТ-компанії *DevRain Solutions*. Створений задля розвитку культури донорства крові в Україні. проект створено 13 березня 2014 року. З червня 2020 року МОЗ України почало моніторинг запасів донорської крові на державному рівні, співпрацюючи з ДонорUA. Дані вносяться працівниками закладів. До системи приєдналися 40 закладів з різних регіонів. ДонорUA доступний тільки українською мовою.

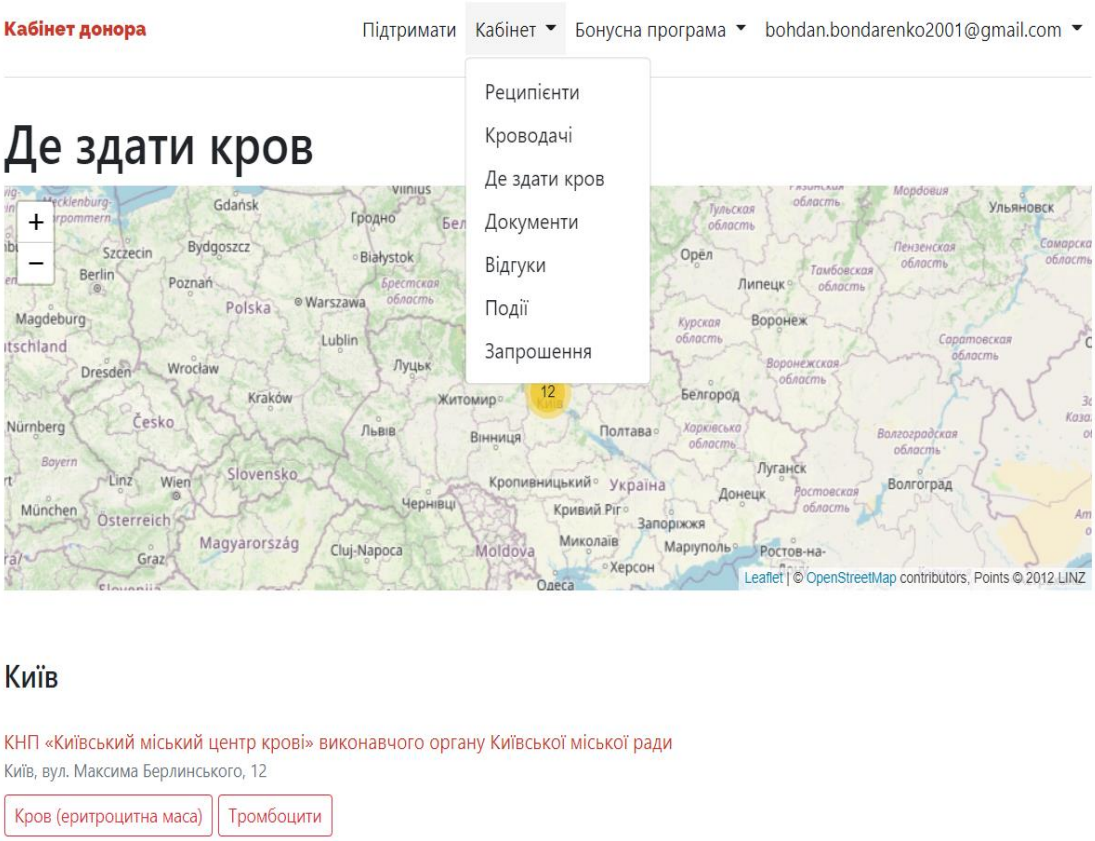

Рис. 1.1. Інтерфейс системи ДонорUA

ДонорUA має модуль реєстрації користувача та його редагування. Таким чином, донор може керувати особистим кабінетом. В кабінеті є можливість змінити прізвище та ім’я, номер телефону, групу крові та резус, адресу.

Для отримання останніх новин пов’язаних з веб-додатком та донорством крові в цілому реалізовано інтеграцію з чат-ботами в месенджері *Telegram* за допомогою *REST API*. Також є можливість підписатися на новини через електронну пошту.

У веб-додатку є модуль пошуку служби крові за містом. Знайти необхідну службу крові можна за допомогою вбудованої карти: на карті нанесені всі доступні міста, де є служби крові. Також є варіант вибору служби за допомогою спеціального вікна пошуку по назві міста. Після вибору відділення забору крові майбутній донор може ознайомитися з графіком його роботи та умовами.

ДонорUA включає в себе також модуль пошуку реципієнтів, яким необхідна допомога якомога швидше. Модуль автоматично підбирає реципієнтів, які сумісні

з донором, та надає необхідну інформацію про них у розділі «Реципієнти». В цьому ж розділі відразу можна зареєструватися для донації в необхідному відділенні на зручний для донора час.

Також веб-додаток дозволяє переглядати історію донацій, додавати документи, що підтверджують особу, писати відгуки про відділення здачі крові, переглядати події, пов’язані з донорством крові.

ДонорUA має базу даних зареєстрованих донорів та потенційних реципієнтів. В системі зберігається інформація про тип крові, документи, імена користувачів. Дані реципієнтів можуть автоматично оновлюватися за допомогою модуля інтеграції з державною базою даних.

Система АІСТ призначена для автоматизації технологічних процесів заготівлі, переробки, зберігання, використання, контролю якості та забезпечення безпеки донорської крові та її компонентів, обробки демографічних показників донорів крові, отримання звітної, аналітичної та статистичної документації [6]. Система задіяна підрозділах забору крові, її зберігання й обробки та використовується для їх інтеграції в єдиний інформаційний простір. Система дозволяє вирішувати завдання лабораторій служби крові: апробацію крові на інфекційну безпеку, серологічне (метод імунодіагностики, що розпізнає головним чином інфекційні захворювання за допомогою реакції «антиген — антитіло») обстеження крові з метою проведення відбору донорів.

АІСТ включає модулі з інформацією про донора, модуль адміністрування, модуль зворотнього зв’язку, що входять до загальної моделі інформаційного обміну (рис. 1.2).

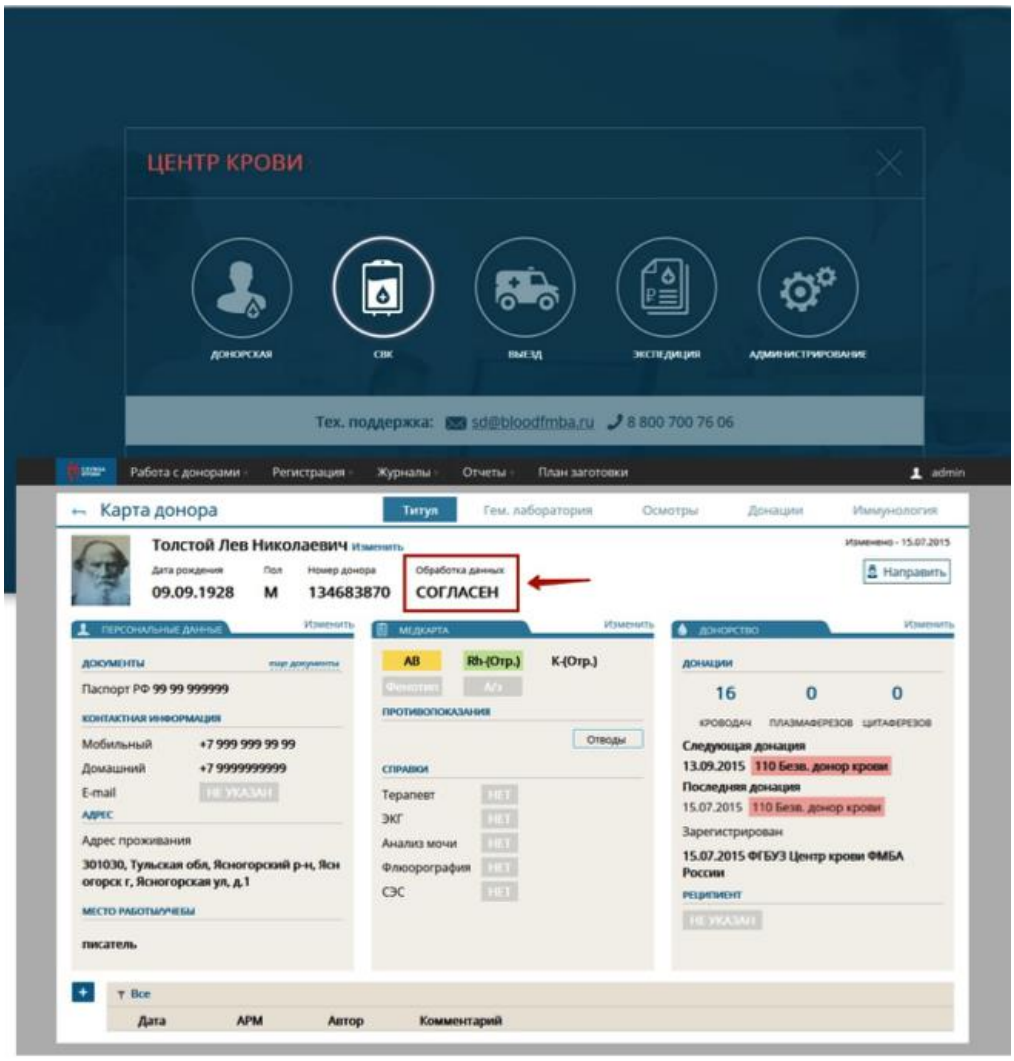

Рис. 1.2. Інтерфейс автоматизованої інформаційної системи трансфузіології

Дана система забезпечує безпеку процесів заготівлі донорської крові завдяки модулям шифрування та обмеження доступу до персональних даних донорів. Також система надає можливість вибору відділення донорства крові для кожного донора, що дозволяє не витрачати даремно контейнери, витратні матеріали, час лікарів, а також гарантує максимальну кількість добровольців, залучених у процес донації крові.

Система має наступні модулі для забезпечення належного виконання медичним персоналом своїх професійних обов’язків:

– модуль виключення помилок із заготівлею, зберіганням та видачею компонентів крові;

- модуль контролю якістю компонентів крові;
- модуль контролю запасів компонентів крові по всіх відділеннях медичної організації;
- модуль перерозподілу запасів компонентів крові між медичними організаціями по запиту.

AICT має базу даних донорів та проведених донацій, тому надає доступ до архіву операцій усіх клієнтів. Також система має такі переваги перед класичним способом збору, управління та зберігання крові: підвищує ефективність та одночасно виключає помилки людського фактору, дозволяє підтримувати миттєвий зв'язок із клінічною трансфузіологією, надає можливість підбору індивідуальних груп донорів для одного реципієнта, дає можливість створення та ведення реєстру фенотипованих (таких, що мають однакові клінічні ознаки, в даному випадку – група крові та резус) донорів, надає можливість створення та ведення реєстру кісткового мозку, дає можливість створення та управління кріобанком, дозволяє автоматизувати всі процеси, починаючи від реєстрації, закінчуючи формуванням звітності, дозволяє формувати єдину базу даних з вичерпною інформацією, забезпечує оперативний доступ до актуальної інформації, забезпечує підбір крові з урахуванням фенотипу, діагнозу, особливостей перебігу захворювання, надає можливість автоматизації обробки запитів від медичних центрів;

Можливості програми служби крові дозволяють максимально ефективно управляти процесами збору, перевірки, обробки крові, її компонентів та постачання до медичних центрів. Система створена відповідно до вимог галузевих норм та стандартів у галузі медицини.

Проте система має недоліки. AICT доступна для ОС *Windows, MacOS, Linux*. Хоча для доступу необхідний лише браузер, даний веб-додаток не має адаптивного макету, тому на мобільних пристроях його використання може бути складним або навіть неможливим.

Програма *GiveBlood* – офіційний мобільний додаток канадської служби крові (рис.1.3). Ця програма дозволяє донорам у Канаді (крім Квебеку) бронювати,

керувати та відстежувати свої пожертвування [7]. Користувачі можуть легко й ефективно знайти найближчі донорські центри та підтвердити призначення пожертвування.

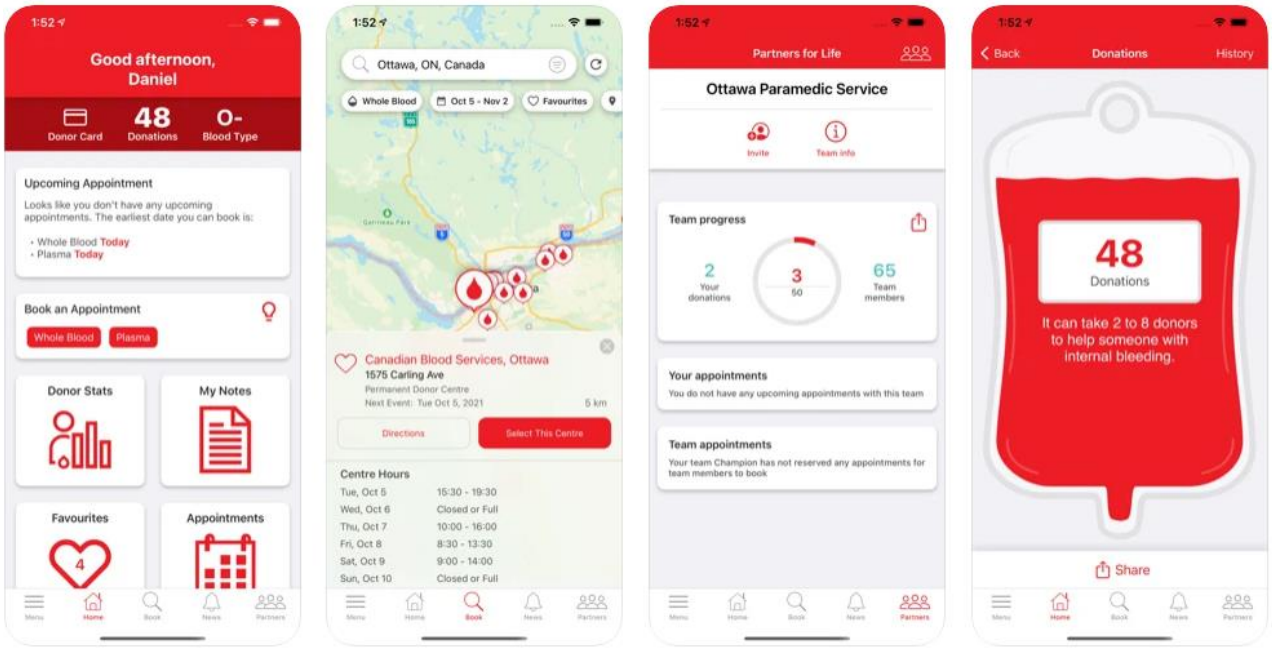

Рис. 1.3. Скріншоти інтерфейсу програми GiveBlood

Програма представлена у вигляді додатку для смартфона на операційних системах *Android* та *iOS*. Такий підхід дозволяє отримати всі переваги мобільного пристрою. Тому даний додаток має модуль визначення поточного місцезнаходження, щоб мати можливість знайти донорські центри поруч та переглянути їхні послуги та зручності.

За допомогою модулю інтеграції з картою є можливість додати свої необхідні точки здачі крові в закладки. Користувач може вибрати пункти прийому крові на карті, переглянути розклад роботи та вихідні дні, і якщо умови влаштовують, забронювати час для донації. Також користувач може переглядати умови, час та місце зустрічі після її бронювання.

За допомогою модулю інтеграції з календарем та соцмережами, можна легко створювати нагадування про найближчі заплановані здачі крові та ділитися своїми

досягненнями з друзями та знайомими. Також можливе легке відслідковування своїх планів донорства у календарі.

Для безпеки даних користувачів реалізований модуль доступу до інформації свого облікового запису. В додатку можна створити свій профіль, переглядати його, та керувати ним. Також можна змінювати налаштування додатку.

Користувачі, які вже здавали кров можуть зберегти та отримати свою картку донора, переглядати історії донацій, ділитися історією пожертв у соціальних мережах. Цю можливість забезпечує модуль інтеграції з соціальними мережами.

В додатку є можливість підписатися на інформацію про обновлення, а також залишити відгук про персонал та відділення здачі крові, яке нещодавно відвідали. Незважаючи на велику кількість модулів, *GiveBlood* доступна тільки англійською та французькою мовами.

Розглянувши існуючі варіанти програмного забезпечення було створено таблицю (див. табл. 1.1) для порівняння їх функцій.

Таблиця 1.1

Порівняння характеристик програмних додатків ДонорUA, автоматизованої інформаційної системи трансфузіології (АІСТ) та *GiveBlood*

Функції	**АІСТ**	***GiveBlood***	**ДонорUA**
Необхідні модулі	1	1	1
Підтримка української мови	0	0	1
Безкоштовність	0	1	1
Реєстрація в системі	1	1	1
Сумарний коефіцієнт	2	3	4

Враховуючи отримані результати порівняння систем, можна сказати, що кожна із них може виконувати свою головну функцію – допомагати службам крові

в забезпеченні збору крові. Але веб-додаток ДонорUA найкращий із них, тому що він безкоштовний, а також підтримує українську мову. Незважаючи на перераховані переваги, ДонорUA має невелику базу даних та менше модулів, ніж АІСТ чи *GiveBlood*.

1.3. Висновки до розділу

Служба крові – це структура, що об'єднує по всій країні медичні установи та їх структурні підрозділи, основним видом діяльності яких є заготівля, переробка, зберігання та забезпечення безпеки донорської крові та її компонентів. Було проаналізовано існуючі технологічні рішення в сфері автоматизованих систем донорства.

Використання автоматизованих систем обліку донорства крові у роботі центрів трансфузіології значно спрощує ряд робочих процесів та підвищує їх ефективність при управлінні процесами взяття, розподілу донорської крові та її компонентів. Автоматизовані системи рекрутингу донорів повинні мати модулі для знаходження відділень донорства крові, обходу черг на прийом до лікаря, збереження медичної картки та історії донацій.

Веб-додаток ДонорUA – це автоматизована система рекрутингу та управління донорами крові. До системи приєдналися 40 закладів з різних регіонів. ДонорUA доступний лише українською мовою. Система АІСТ призначена для автоматизації технологічних процесів заготівлі, переробки, зберігання, використання, контролю якості та забезпечення безпеки донорської крові та її компонентів, обробки демографічних показників донорів крові, отримання звітної, аналітичної та статистичної документації. Програма *GiveBlood* – офіційний мобільний додаток канадської служби крові. Ця програма дозволяє донорам бронювати, керувати та відстежувати свої пожертвування, знаходити найближчі донорські центри та підтвердити призначення пожертвування.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ СЛУЖБ КРОВІ

2.1. Платформа *Spring* та технології для створення веб-додатку

Spring – це програмний каркас (фреймворк) управління для мови Java з відкритим кодом, в основі якого знаходяться контейнери з підтримкою інверсії управління [8]. *Spring* також можна розглядати як набір модулів, які працюють разом для створення основи будь-якого веб-додатку. Основі модулі *Spring*: модуль доступу до даних, модуль для підтримки аспектно-орієнтованого програмування, модуль управління транзакціями, модуль аутентифікації та авторизації, модуль тестування. Фреймворк дозволяє дуже точно конфігурувати веб-додаток донорства крові за допомогою автоконфігурації, *Java* та *XML* коду. Не дивлячись не те, що *Spring* не надає ніякої конкретної моделі програмування, він має велику популярність серед *Java* програмістів і має набагато більшу підтримку, ніж його основний конкурент – *Java Enterprise Edition*. Це зумовлено тим, що Spring має зручні, добре документовані та прості у використанні інструменти для вирішення проблем корпоративного масштабу.

При проектуванні додатку було визначено всі складові частини, функціональні можливості та технології для створення веб-додатку донорства крові. Було використано наступні технології: *JSON Web Token* (*JWT*) для передачі даних аутентифікації у веб-додатку для служб кров; архітектурний стиль *Representational State Transfer* (*REST*) для зв'язку клієнтської та серверної частини веб-додатку служб кров; бібліотеку *Vuetify* для створення інтерфейсу для веб-додатку служб крові, *Vuex* як реактивне сховище даних та *Vue Router* як бібліотека для створення роутингу.

Для передачі даних аутентифікації у веб-додатку для служб крові було використано *JSON Web Token* (*JWT*). *JWT* – це стандарт токена доступу на основі JSON, стандартизованого в *RFC 7519*. Цей стандарт, як правило, використовується при передачі даних для аутентифікації в клієнт-серверних програмах. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи.

JWT вважається одним із найбезпечніших способів передачі інформації між двома учасниками. Для його створення необхідно визначити заголовок (*header*) із загальною інформацією про токен, корисні дані (*payload*), такі як ідентифікатор користувача, його роль і т. д. та підпис (*signature*). Іншими словами, *JWT* – це звичайний рядок у форматі *header.payload.signature*.

Додаток використовує *JWT* для перевірки аутентифікації користувача таким чином:

1. Спочатку користувач заходить на сервер аутентифікації за допомогою аутентифікаційного ключа, у випадку веб-додатку для служб крові це − пара даних з полів логін/пароль.
2. Потім сервер аутентифікації створює *JWT* і відправляє його клієнтській частині користувача.
3. Коли клієнт створює *API* запит, він додає до нього отриманий раніше JWT.
4. Коли користувач виконує запит *API*, додаток служб крові перевіряє переданий *JWT* в запиті і вирішує чи є користувач тим, за кого себе видає.

Заголовок токену містить інформацію про те, як має вираховуватись підпис. Заголовок – це також *JSON* об’єкт, як і сам токен. Заголовок для веб-додатку донорства крові виглядає наступним чином: { “*alg*”: “*HS256*”, “*typ*”: “*JWT*”}, де *alg* – це тип алгоритму шифрування. У додатку донорства крові було використано *HMAC-SHA256*, для роботи якого необхідний лише приватний ключ, а *typ* – це тип токена (*JWT*).

Корисні дані містять додаткові поля для аутентифікації. Ці поля можуть бути використані при створенні *JWT*, але вони не є обов'язковими. Кожне поле в корисних даних токена називається заявкою (*claim*). Існують стандартні заявки, наприклад, iss (додаток, що підписав токен), *exp* (час життя токена). Повний список стандартних заявок знаходиться в специфікації стандарту *RFC 7519*. Також можна використовувати власні заявки, якщо жодна зі стандартних не підійшла. Але варто пам'ятати, що чим більше передається інформації, тим більший вийде сам *JWT*. Зазвичай з цим не буває проблем, але це може негативно позначитися на продуктивності і викликати затримки у взаємодії з сервером. Корисні дані для веб-додатку донорства крові виглядають наступним чином: { "*sub*": "*exmple@mail.com*", "*exp*": "1652619795"}, де *sub* – це стандартна заявка, що містить електронну пошту користувача, а *exp* – це стандартна заявка, що містить строк дії токена (дата у форматі *UNIX*).

Для отримання підпису сервер пропускає необхідні дані у форматі *JSON* та секретний ключ, що зберігається лише на сервері, через хеш-функцію (в додатку для служб крові використовується *HMAC-SHA256*). Результат має такий вигляд: -*xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM*.

Отримані частини токена сервер кодує алгоритмом *BASE64* для скорочення розміру підпису та об'єднує через крапку. Дуже важливо розуміти, що використання JWT не приховує та не маскує дані автоматично. *JWT* використовуються тому, що він гарантує, що надіслані дані були дійсно відправлені авторизованим джерелом. Як було продемонстровано вище, дані всередині *JWT* закодовані та підписані. Мета кодування даних – перетворення структури. Підписані дані дозволяють одержувачу даних перевірити аутентифікацію джерела даних. Таким чином, кодування та підпис не захищають дані.

Для зв'язку клієнтської та серверної частини веб-додатку служб крові було використано архітектурний стиль *Representational State Transfer* (*REST*). *REST* – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів [9]. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу *HTTP*. В основі *REST* закладено принципи функціонування

Всесвітньої павутини і, зокрема, можливості *HTTP*. Філдінг розробив *REST* паралельно з *HTTP 1.1* базуючись на попередньому протоколі *HTTP 1.0*. Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, *HTML, XML, JSON*). Будь-який *REST* протокол (*HTTP* в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформацію про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє їй еволюціонувати з новими вимогами. Антиподом *REST* є підхід, заснований на виклику віддалених процедур *Remote Procedure Call* (*RPC*). Підхід *RPC* дозволяє використовувати невелику кількість мережевих ресурсів з великою кількістю методів і складним протоколом. При підході *REST* кількість методів і складність протоколу суворо обмежені. *REST*, як і кожен архітектурний стиль має ряд архітектурних обмежень. *REST* – це гібридний стиль, який успадковує обмеження з інших архітектурних стилів:

- клієнт-серверна архітектура. Розділення компонентів додатку дозволяє їм еволюціонувати незалежно, що зменшує складність розробки;
- відсутність стану. Кожен запит містить всю необхідну інформацію для його обробки і не покладається на те, що сервер зберігає якусь інформацію з попереднього запиту. Хоча сервер не може запам’ятовувати стани, замість нього цю задачу може виконати клієнт;
- кешування. Система написана за стилем *REST* має підтримувати кешування, тобто всі дані, що повертає сервер повинні містити інформацію про те, чи можна ці дані кешувати, якщо ж можна, то як довго необхідно їх зберігати;
- всі компоненти мають підтримувати однорідний інтерфейс;
- останнім обмеженням для *REST* є поділ на шари абстракції. Кожен компонент потрапляє в якийсь шар і обмінюється даними лише з компонентами в прошарку під ним. Таке обмеження зменшує складність компонентів.

Для створення інтерфейсу для веб-додатку служб крові було використано бібліотеку *Vuetify*. *Vuetify* – це *UI* бібліотека із відкритим вихідним кодом для створення інтерфейсів веб-додатків і мобільних додатків. Ця бібліотека підтримує

технологію *treeshaking*, яка допомагає не включати не використані функції та класи в кінцевий артефакт, як наслідок клієнтська частина веб-додатку для служб крові важить значно менше і завантажується швидше. Всі компоненти бібліотеки *Vuetify* розроблені з використанням стандартів *Google Material Design*.

Для створення глобального реактивного сховища даних на клієнтській частині було використано бібліотеку *Vuex*. *Vuex* гарантує, що дані будуть реактивно зв'язані, тобто при будь-якій зміні даних, шаблон, який пов'язаний з цими даними буде автоматично змінений. *Vuex* – це централізоване сховище даних для всіх компонентів веб-додатку служб крові з правилами доступу, що гарантують передбачувані зміни цих даних. *Vuex* також комплектується вбудованим інструментом відладки та інструментом імпорту/експорту зліпків даних.

Для створення динамічного роутингу на клієнтській частині було використано бібліотеку *Vue Router*. Ця бібліотека дозволяє швидко та точно конфігурувати переходи між сторінками веб-додатку донорства крові.

2.2. Проектування *ER*-моделі

При проектуванні бази даних додатку для донорства крові було використано реляційну систему управління базами даних (СУБД) *PostgreSQL*. *PostgreSQL* – це реляційна система управління базами даних з відкритим вихідним кодом. Вона підтримує більшу частину стандарту *SQL* і пропонує такі сучасні функції: складні запити, зовнішні ключі, тригери, транзакційну цілісність, багатоверсійність. Крім того, користувачі можуть всіляко розширювати можливості *PostgreSQL*, наприклад, створюючи свої: типи даних, функції, оператори, агрегатні функції, методи індексування [10].

Також для проектуванні бази даних було використано бібліотеку *Hibernate*. *Hibernate* – це бібліотека для мови програмування *Java*, призначена для вирішення задач об'єктно-реляційного відображення *Object-Relational Mapping* (*ORM*), найпопулярніша реалізація специфікації *Java Persistence API* (*JPA*). Бібліотека не тільки вирішує завдання зв'язку класів *Java* з таблицями бази даних, але й також

надає засоби для автоматичної генерації та оновлення набору таблиць, побудови запитів та обробки отриманих даних і може значно зменшити час розробки зазвичай витрачається на ручне написання *SQL* і *JDBC*-коду. Відображення (*mapping*, зіставлення, проектування) Java-класів з таблицями бази даних здійснюється за допомогою конфігураційних *XML*-файлів або *Java*-анотацій [11].

При проектуванні бази даних веб-додатку служб крові було спроектовано п’ять таблиць: *users*, *recipient_appication*, *donor_appication*, *jwt_blacklist*, *region*.

Таблиця *users* зберігає інформацію про донора чи реципієнта. Вона має такі поля: ключ (id); електронну пошту користувача (*email*); групу крові користувача (*group_number*); ім’я користувача (*name*); пароль користувача (*password*); мобільний телефон користувача (*phone*); місто, де знаходиться доступна для користувача служба крові (region); резус користувача (*rh*); роль користувача (*role*) – поле для визначення повноважень користувача в систем; стать користувача (*sex*); дату народження користувача (*brith_day*); історію хвороби користувача (*diseases*).

Таблиця *recipient_appication* зберігає інформацію про заявку реципієнта. Вона має ключ (*id*); дату, коли реципієнту необхідно перелити кров (*date*); зовнішній ключ зв’язку з таблицею *users* (*user_id*).

Таблиця *donor_appication* зберігає інформацію про заявку реципієнта. Вона повинна мати ключ (*id*); кінцеву дату, до якої донор може здавати кров (*expires_at*); зовнішній ключ зв’язку з таблицею *users* (*user_id*).

Таблиця *users* має зв’язок один до багатьох з таблицями *recipient_appication* та *donor_appication*.

Таблиця *jwt_blacklist* зберігає у собі *JWT* токени, які не повинні розглядатися як валідні. Наприклад, коли користувач хоче вийти із системи, але термін дії його токену ще не сплив, його токен заноситься в дану таблицю. Тому *jwt_blacklist* має ключ (*id*); дату закінчення строку дії токена (*expires_at*); поле для збереження самого токена (*signature*). Технічно можна обійтися без поля *expires_at*, оскільки дата закінчення строку дій зашифрована у самому токені. Але з метою забезпечення кращої продуктивності було прийнято рішення збереження цієї дати окремо.

Таблиця *region* зберігає інформацію про доступні міста, де є служби крові. Таблиця має лише одне поле – *name*, що використовується для збереження назви міста. Також поле *name* є ключем таблиці. Таке рішення було прийнято, оскільки первинні ключі таблиці індексуються автоматично, і тому таким чином можна досягти кращої продуктивності при виборі запису. Хоча при цьому підході зменшується продуктивність при додаванні та редагуванні запису, цим можна пожертвувати тому, що додавати та редагувати записи в таблиці *region* може лише системний адміністратор. На основі вище перерахованих даних побудовано *ER*-діаграму спроектованої бази даних (рис. 2.1).

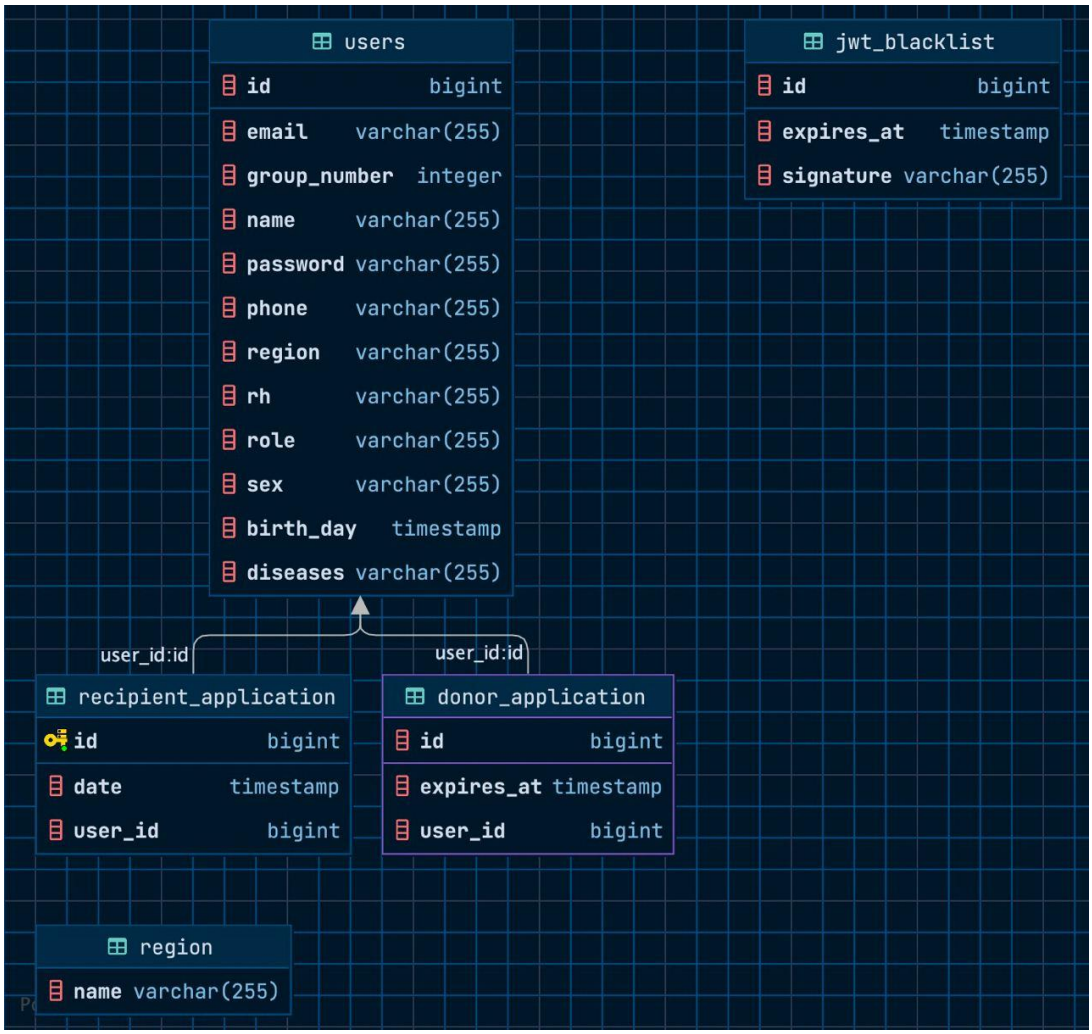

Рис. 2.1. *ER*-діаграма бази даних веб-додатку служб крові

2.3. Проектування інтерфейсу веб-додатку

Веб-додаток для донорства крові має чотири сторінки: сторінка пошуку та створення заявок для донорів і реципієнтів, сторінка реєстрації користувачів, сторінка входу користувачів, сторінка профілю користувача. Кожна сторінка має заголовок однакової структури: зліва логотип з назвою додатку служб крові, кнопки для переходу на сторінки з заявками та профілем (будуть видимі, якщо користувач увійшов), кнопка входу/виходу справа. В додатку для донорства крові було використано чотири кольори: основний (темно-червоний), додатковий (світло-червоний), колір фону (білий), основний колір тексту (чорний). З кольорами важливо визначатися заздалегідь, оскільки вони будуть основою конфігурації *UI* бібліотеки *Vuetify*.

Сторінка пошуку та створення заявок для донорів і реципієнтів ділиться на дві колонки: справа колонка з реципієнтами, зліва – з донорами. Кожну колонку можна поділити на три секції. Секції розташовані зверху вниз: секція з назвою колонки та кнопкою для відкриття модального вікна для створення заявки, секція з фільтрами, секція зі списком заявок.

Модальне вікно в першій секції має календар для вибору дати переливання крові для реципієнта або кінцевої дати здачі крові для донора. Модальне вікно закривається, якщо натиснути поза його межі, або натиснути на кнопку для створення заявки, що знаходиться знизу модального вікна.

Друга секція також ділиться на дві колонки: колонка з фільтрами та колонка з кнопками. Перша колонка має фільтр для пошуку рідкісних груп крові, випадне меню для вибору необхідної служби крові, випадне меню для вибору резусу, випадне меню для вибору групи крові, перемикач для відфільтровування лише своїх заявок. Друга колонка має такі кнопки зверху вниз: кнопку для скидання фільтрів, кнопку для пошуку заявок за вибраними фільтрами, кнопку для підбору відповідних заявок донорів або реципієнтів, за фільтрами заявок реципієнтів або донорів відповідно.

Третя секція має список заявок, які оформлені у вигляді карток зверху вниз. Кожна картка складається з імені донора чи реципієнта, його служби крові, його резусу та групи крові, відповідної дати заявки, та вікна деталей. Вікно деталей розкривається при натиску на кнопку-шеврон навпроти назви вікна. Вікно деталей має інформацію про стать, вік, електронну пошту та телефон користувача. Це вікно можна згорнути, натиснувши на ту ж саму кнопку-шеврон.

Сторінка реєстрації має форму для введення повної інформації про профіль користувача. Елементи вводу інформації розташовуються зверху вниз: елемент вводу електронної пошти з валідацією; елемент вводу телефону з валідацією за форматом +380XXXXXXXXX; елемент вводу імені та прізвища; елемент вводу паролю; елемент вводу міста служби крові з випадним меню з можливими варіантами; елемент вводу резусу крові з випадним меню з можливими варіантами; елемент вводу групи крові з випадним меню з можливими варіантами. Кожне поле форми є обов’язковим. Знизу знаходиться кнопка для відправки форми. При успішному відправленні форми, система має перенаправити користувача на сторінку входу.

Сторінка входу має форму для введення пошти та паролю користувача. Елементи вводу інформації розташовуються зверху вниз. Кожне поле форми є обов’язковим. Знизу знаходиться кнопка для відправки форми. При успішному відправленні форми, система має перенаправити користувача на сторінку з заявками.

Сторінка з профілем призначена для перегляду та редагування персональної інформації користувача. Має такі поля: ім’я та прізвище, електронну пошта, мобільний телефон, стать, місто служби крові, групу крові, резус, дату народження, перенесені хвороби. По замовчуванню всі поля неактивні і не можуть бути відредаговані, вони доступні лише для перегляду. Для того щоб мати змогу відредагувати особисту інформацію, необхідно натиснути кнопку «Редагувати» під полями.

2.4. Проектування модулів веб-додатку

Спроектований веб-додаток для служб крові складається з модулів: модуль пошуку та створення заявок донора та реципієнта, модуль аутентифікації, модуль авторизації, модуль перегляду та редагування профілю користувача, асинхронний модуль очищення прострочених заявок, асинхронний модуль очищення недійсних *JWT* токенів. Завдяки архітектурному стилю *REST* кожен модуль працює незалежно, тому відладка та додавання нових можливостей у модулі є відносно легким та швидким.

Головним модулем веб-додатку для донорства крові є модуль пошуку та створення заявок донорів та реципієнтів. Модуль видаляє заявки з бази даних по їх ідентифікатору, редагувати заявку, знайдену за ідентифікатором. Редагування та видалення заявки донорів та реципієнтів є доступним лише для власника заявки. Модуль також повертає список заявок застосовуючи фільтри, якщо ж фільтрів немає, то має бути повернуто всі наявні заявки. Список заявок донорів та реципієнтів є масивом *JSON* об’єктів, кожен з яких містить детальну інформацію про кожну заявку: ідентифікатор заявки; ідентифікатор донора або реципієнта, що створив заявку; ім’я та прізвище донора або реципієнта, що створив заявку; електронну пошту донора або реципієнта, що створив заявку; групу крові донора або реципієнта, що створив заявку; місто служби крові донора або реципієнта, що створив заявку; стать донора або реципієнта, що створив заявку; мобільний номер телефону донора або реципієнта, що створив заявку; резус донора або реципієнта, що створив заявку; дату, до якої донор може здати кров для заявки донора або дату, коли реципієнт готовий до переливання крові для заявки реципієнта. Модуль може створювати заявки донорів та реципієнтів, для цього йому необхідно передати ідентифікатор донора або реципієнта та дату, до якої донор може здати кров для заявки донора або дату, коли реципієнт готовий до переливання крові для заявки реципієнта.

Модуль аутентифікації дозволяє входити, реєструватися та виходити з додатку. При реєстрації модуль створює новий запис у таблиці *users*, при вході

модуль формує новий *JWT* токен, на основі отриманої електронної пошти та паролю. Якщо дані аутентифікації не співпадають з даними в базі, то вхід буде відхилено. Також модуль аутентифікації дозволяє виходити зі свого облікового запису. При виході з системи модуль заносить токен в чорний список – таблицю *jwt_blacklist*.

Модуль авторизації попередньо опрацьовує кожен запит на сервер, крім запиту на вхід/реєстрацію користувача. Модуль розшифровує JWT токен, який надається разом із запитом. Якщо токен відсутній або не валідний, запит буде відхилено. Запит також буде відхилено, якщо токен знаходиться у чорному списку – таблиці jwt_blacklist. З розшифрованого токену буде отримано електронну пошту користувача, яку модуль використовує для визначення повноти доступу до даних, а також розшифрована електронна пошта може бути передана до кінцевого модулю призначення за необхідності.

Модулі авторизації та аутентифікації обов’язково використовують хешовані паролі. За допомогою одностороннього хешування пароль буде зберігатися у зашифрованому вигляді в базі даних, тому навіть якщо зловмисник перехватить пароль та електронну пошту під час запиту, він не зможе увійти в систему за отриманими даними входу.

Модуль перегляду та редагування профілю користувача дозволяє переглядати та редагувати особисті дані користувача, окрім паролю. Модуль повертає та приймає однаковий набір даних – об’єкт JSON, що містить інформацію про ідентифікатор користувача, його електронну пошту, його ім’я та прізвище, його місто служби крові, його стать, його мобільний номер телефону, його дату народження, його групу крові та резус, а також його історію хвороби. Ідентифікатор користувача не буде відображатися на клієнтській частині, але він необхідний для того, щоб сервер дізнався профіль якого користувача йому треба змінити, якщо той захоче не тільки переглянути свій профіль, а й відредагувати його.

Асинхронний модуль очищення прострочених заявок донорів та реципієнтів знаходить та видаляє заявки, термін дії яких закінчився. Асинхронний модуль

очищення недійсних JWT токенів знаходить та видаляє токени, термін дії яких закінчився.

Вони є асинхронним, тому що не залежать від конкретного запиту клієнта, а запускаються автоматично. Серверна частина веб-додатку служб крові запускає новий програмний потік виконання, в якому буде виконуватися код даних модулів. Після виконання очищення запускається таймер, і модулі засинають на дві години, після чого знову виконують очищення і так далі до зупинки сервера. Це зроблено для оптимізації використання ресурсів сервера. Адже дані, що більше не має сенсу зберігати, просто загромаджують базу даних, а очищувати їх окремим запитом неефективно.

При проектуванні додатку було визначено обробники подій на сторінці пошуку та створення заявок донорів та реципієнтів, на сторінці реєстрації, на сторінці входу та на сторінці профілю. Результатом роботи оброблювачів подій є запит на сервер, ререндер компонента або зміна внутрішнього стану додатку для служб крові.

На сторінці пошуку та створення заявок донорів та реципієнтів знаходяться такі групи оброблювачів подій: оброблювачі подій зміни параметрів фільтрів, оброблювачі подій пошуку заявок, оброблювачі подій видалення та редагування заявки, оброблювачі подій створення заявки. При створенні або редагуванні заявки оброблювач формує запит на сервер з даних користувача (ідентифікатора і *JWT* токена), отриманих з глобального реактивного сховища, та дати, отриманої з модального вікна. При видаленні заявки оброблювач формує запит на сервер з ідентифікатора користувача і *JWT* токена, отриманого з глобального реактивного сховища, та ідентифікатора заявки, отриманого з локального реактивного сховища компонента картки. При виборі фільтрів пошуку заявок оброблювач змінює дані, що зберігаються у локальному реактивному сховищі компонента пошуку (яка група крові та резус необхідна, яке місто служби крові необхідне, чи потрібно знайти лише свої заявки донорів або реципієнтів), які в свою чергу будуть використані обробником події самого пошуку для формування запиту на сервер. При успішному запиті будуть відображені картки із заявками донорів або

реципієнтів, що задовольняють критерії пошуку. В картках буде міститися інформація про саму заявку та її донора або реципієнта: ім’я та прізвище донора або реципієнта, що створив заявку; електронну пошту донора або реципієнта, що створив заявку; групу крові донора або реципієнта, що створив заявку; місто служби крові донора або реципієнта, що створив заявку; стать донора або реципієнта, що створив заявку; мобільний номер телефону донора або реципієнта, що створив заявку; резус донора або реципієнта, що створив заявку; дату, до якої донор може здати кров для заявки донора або дату, коли реципієнт готовий до переливання крові.

В заголовку знаходяться кнопки для переходу між сторінками веб-додатку, тому обробники подій, зареєстровані для цих кнопок, будуть змінювати стан роутера. Такі обробники не роблять ніяких запитів на сервер, вони лише змінюють відображення на клієнті. Також для кнопки виходу створений окремий оброблювач, який видаляє дані з глобального реактивного сховища, а також формує запит, який містить *JWT* токен, на сервер для інвалідації токена.

При реєстрації чи вході обробники подій кнопок відповідних форм відправляють дані цих форм на сервер, якщо дані валідні. При реєстрації на сервер відправляється *JSON* об’єкт, що містить електронну пошту користувача, його ім’я та прізвище, його пароль, його групу крові та резус, його номер мобільного телефону, його стать та його місто служби крові. При вході на сервер відправляється *JSON* об’єкт із паролем та електронною поштою користувача. Якщо запит був успішним, то веб-додаток має перенаправити користувача на сторінку входу чи головну сторінку відповідно.

На сторінці профілю знаходяться всього два обробника подій. Обробник події натиску на кнопку редагування профілю робить поля з даними профілю доступними для зміни. Обробник події обновлення даних профілю відправляє відредаговані дані у формі *JSON* об’єкта на сервер, а також робить поля з даними профілю недоступними для зміни. *JSON* об’єкт містить такі дані: інформацію про ідентифікатор користувача, його електронну пошту, його ім’я та прізвище, його

місто служби крові, його стать, його мобільний номер телефону, його дату народження, його групу крові та резус, а також його історію хвороби.

При завантаженні сторінок профілю, реєстрації та головної сторінки відбувається запит на сервер для отримання списку доступних служб крові. Цей список надається у вигляді *JSON* масиву рядків. Також при завантаженні головної сторінки веб-додатку служб крові всі заявки донорів та реципієнтів завантажуються з сервера.

2.5. Висновки до розділу

При проектуванні додатку було визначено всі складові частини, функціональні можливості та технології для створення веб-додатку донорства крові. Було використано наступні технології: *JSON Web Token* (*JWT*) для передачі даних аутентифікації у веб-додатку для служб кров; архітектурний стиль *Representational State Transfer* (*REST*) для зв'язку клієнтської та серверної частини веб-додатку служб кров; бібліотеку *Vuetify* для створення інтерфейсу для веб-додатку служб крові, *Vuex* як реактивне сховище даних та *Vue Router* як бібліотека для створення роутингу.

При проектуванні бази даних додатку для донорства крові було використано реляційну систему управління базами даних (СУБД) *PostgreSQL*. При проектуванні бази даних веб-додатку служб крові було спроектовано п'ять таблиць: *users* з інформацією про користувача, *recipient_appication* з інформацією про реципієнта, *donor_appication* з інформацією про донора, *jwt_blacklist* з невалідними токенами доступу, *region* для збереження інформації про міста служб крові.

Веб-додаток для донорства крові має чотири сторінки: сторінка пошуку та створення заявок для донорів і реципієнтів, сторінка реєстрації користувачів, сторінка входу користувачів, сторінка профілю користувача. Кожна сторінка має заголовок однакової структури: зліва логотип з назвою додатку служб крові, кнопки для переходу на сторінки з заявками та профілем (будуть видимі, якщо користувач увійшов), кнопка входу/виходу справа.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ СЛУЖБ КРОВІ

3.1. Особливості використання платформи *Spring* та налаштування середовища розробки

Платформа *Spring* ідеально підходить для розробки додатків з великою кількістю модулів та додатків, що потребують високої надійності. Незважаючи не серйозні вимоги до додатків на *Spring*, фреймворк пропонує відносно прості та добре документовані інструменти розробки як середніх за розміром програм, так і програм із сотнями класів. Ці властивості зумовлені трьома ключовими поняттями, на яких базується *Spring*: впровадження залежностей – *Dependency injection* (*DI*), інверсія керування – *Inversion of control* (*IoC*), аспектно-орієнтоване програмування – *Aspect-oriented programming* (*AOP*).

Інверсія керування – це принцип побудови програми, при якому її частини отримують потік керування (викликаються) із загальної бібліотеки спільного користування. Однією з реалізацій *IoC* є впровадження залежностей, що використовується в багатьох фреймворках, в тому числі і в *Spring*, ці залежності називаються *IoC* контейнери. Програміст не повинен самостійно управляти залежностями, натомість *Spring* як система управління контейнерами самостійно вибирає необхідні контейнери та впроваджує їх туди, де вони необхідні. Такий підхід дозволяє значно знизити зв’язність коду, що полегшує його написання та відладку. Чим більший додаток, тим більше відчуваються переваги *Ioc* та *DI*.

Аспектно-орієнтоване програмування – парадигма програмування, яка дозволяє виокремити перехресну (наскрізну) функціональність. Аспектно-орієнтований підхід розглядає програмну систему як набір модулів, кожен з яких виражає особливість функціонування системи.

При розробці системи програміст вибирає модулі так, щоб кожен із них реалізовував певну функціональну вимогу. Натомість в рамках об’єктно-орієнтованого підходу реалізація деяких вимог до програми часто не може бути локалізована в окремому модулі, в результаті чого код, що відображає такі функціональні завдання, буде знаходитись у різних модулях (наприклад, код ведення журналу подій). Аспектно-орієнтований підхід зменшує складність розроблюваного коду. Аспектно-орієнтоване програмування часто використовується для створення журналів подій.

Для розробки веб-додатку для служб крові було використано інтегроване середовище розробки *Intellij IDEA* та редактор коду *VS Code*. Все програмне забезпечення запускалось під управлінням операційної системи *MacOS*. Також було використано пакетні менеджери *NPM* і *Homebrew*. Оскільки було використано фреймворк *Spring* як платформу для створення веб-додатку служб крові, то було встановлено інтегроване середовище розробки *Intellij IDEA*.

Для установки інтегрованого середовища розробки *Intellij IDEA* було перейдено на офіційний сайт компанії *Jetbrains*, потім вибрано *Intellij IDEA* версію *Ultimate* (версія *Ultimate* має переваги перед звичайною, такі як зручна інтеграція з СУБД, вбудований *HTTP*-клієнт для відладки, повна підтримка фреймворку *Spring*), і в кінці вибрано операційну систему *MacOS* та завантажено образ віртуального диску. Після завантаження образу диску його було змонтовано та відкрито установлене інтегроване середовище розробки *Intellij IDEA*.

Оскільки було вибрано версію *Ultimate*, необхідно було отримати ліцензію для подальшого користування середовищем. *Jetbrains* надає ліцензію безкоштовно на рік з можливістю її продовження для студентів

Щоб отримати таку ліцензію було створено обліковий запис на сайті *Jetbrains* використовуючи корпоративну пошту, після створення облікового запису та прийняття всіх умов користування студентською ліценцією на пошту прийшов лист з підтвердженням оформлення ліцензії. Отримавши ліцензію необхідно авторизуватися в установленому інтегрованому середовищі розробки Intellij IDEA.

Intellij IDEA по замовчуванню встановлює всі необхідні плагіни для розробки додатку донорства крові. Наприклад, *Spring Plugin, Hibernate Plugin, Maven Plugin, Lombok. Lombok* – це плагін для однойменного препроцесора для компілятора коду *Java*. За допомогою цього плагіна середовище розробки зможе розпізнавати нові синтаксичні конструкції, що вводить препроцесор. Сам препроцесор розпізнає спеціальні анотації, які конвертуються в код. За допомогою одної анотації можна замінити спростити або ж повністю прибрати шаблонні конструктори, мутатори та аксесори, перевизначені методи класу *Object* і навіть архітектурні патерни, наприклад, патерн *Builder*.

Для швидкої та зручної установки СУБД *PostgreSQL* та редактору коду *VS Code* було завантажено пакетний менеджер *Homebrew*. Для цього відкрито термінал та виконано наступну команду: «*/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"*». Після завантаження *Homebrew* було виконано такі дві команди: «*brew install --cask visual-studio-code*» та «*brew install postgresql*», що автоматично установлять останні версії *VS Code* та *PostgreSQL* відповідно.

Після завантаження редактора коду *VS Code* необхідно його налаштувати. По замовчуванню *VS Code* – це просто потужний редактор тексту. Щоб перетворити його на справжній редактор коду треба установити необхідні плагіни. Оскільки *VS Code* буде використовуватися для написання коду клієнта на *Vue 2* для додатку служб крові, то потрібен плагін *Vetur*. *Vetur* надає можливість зручного створення та редагування компонентів *Vue*, підказки для основних синтаксичних конструкцій *Vue*, виправлення помилок на етапі написання коду, вбудований сервер для розробки з технологією гарячої заміни модулів.

Для розробки клієнтської частини знадобиться пакетний менеджер *NPM*, що дозволяє установлювати пакети *JS* знаючи їх назву з віддаленого репозиторію. Також для зручного та швидкого створення проекту *Vue* знадобиться розширення для терміналу *Vue CLI*. Виконавши в терміналі команди «*brew install node*» та «*brew install vue-cli*» було установлено *NPM* та *Vue CLI* відповідно.

3.2. Створення проектів для серверної та клієнтської частини

При розробці веб-додатку для донорства крові було створено два проекти: проект для серверної частини під управлінням фреймворку *Spring* та проект для клієнтської частини під управлінням фреймворку *Vue*. Також було запущено СУБД *PostgreSQL* та під'єднано її до інтегрованого середовища розробки *Intellij IDEA* для подальшого управління та моніторингу даних.

Для створення проекту для серверної частини в головному меню *Intellij IDEA* було вибрано опцію створення нового проекту за допомогою *Spring Initializr* (веб-інструмент для швидкої генерації проекту на платформі Spring). У вікні, що відкрилося в полі вводу *Project name* було вказано ім'я проекту (*blood-donation-app-server*), в полі *Location* було указано його розташування у файловій системі (по замовчуванню), в полі *Group* було указано групу для артефакта, отриманого після збору (*com.bondarenko*), в полі *Artifact* було указано ідентифікатор для артефакта отриманого після збору (*blood-donation-app-server*), в полі *Project SDK* було указано версію *JDK* (1.8), в полі *Packaging* було указано варіант пакування (*WAR*, оскільки створюється веб-додаток). Далі було вибрано необхідні пакети для створення серверної частини веб-додатку служб крові: *Lombok*, *Spring Web*, *Spring Data JPA*, *PostgreSQL Driver*, *Hibernate*, *Spring Security*; і натиснуто кнопку «*Finish*», далі після декількох хвилин очікування, *Intellij IDEA* завантажив указані вище пакети з центрального або віддаленого репозиторію та закінчив конфігурування проекту.

Останніми кроками створення проекту серверної частини є конфігурування *Spring*, *Maven* та підключення бази даних до інтегрованого середовища розробки. Для підключення бібліотеки для *JWT* було зайдено на сайт віддаленого репозиторію *Maven*, знайдено там пакет *com.auth0.java-jwt*, скопійовано код конфігурації та вставлено його в файл *pom.xml* в секцію *dependencies*. Після чого натиснуто кнопку для обновлення конфігурації *Maven* в боковому меню справа.

Після успішного завантаження пакету *com.auth0.java-jwt* було виконано підключення *PostgreSQL*. Для цього було створено новий системний термінал в

Intellij IDEA, названо *PostgreSQL terminal* та виконано команду «*postgres -D /opt/homebrew/var/postgres*», що запустить локальний сервер *PostgreSQL* зі стандартною конфігурацією. Далі перейдено в бокове меню справа під назвою «*Database*», вибрано *PostgreSQL* як джерело даних, та здійснено конфігурування: хост – *localhost*, порт – 5432, користувач – *postgres*, пароль – *postgres*, і натиснуто кнопку «*Apply*». Тепер в боковому меню «*Database*» повинна з'явитися база даних з назвою *postgres@localhost*. Далі в контекстному меню бази даних *postgres@localhost* викликано консоль і створено нову схему для веб-додатку донорства крові за допомогою команди «*CREATE SCHEMA blood_donation_app*».

В директорії *resources* створено файл *application.properties*, де буде зберігатися базова конфігурація *Spring* (рис. 3.1). В ньому указано, на якому порту має запуститись сервер, як під'єднатись до бази даних, та параметри налаштування *JPA/Hibernate* (режим моніторингу *SQL* запитів, режим *DDL* операцій, діалект *SQL* та схема бази даних).

```
server.port=${PORT:8080}

# DATABASE CONNECTION

spring.main.banner-mode=off
spring.datasource.initialization-mode=always
spring.datasource.platform=postgres
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=postgres

# JPA / HIBERNATE

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.default_schema=blood_donation_app
```

Рис. 3.1. Базова конфігурація серверної частини веб-додатку служб крові

В директорії *com.bondarenko.blooddonationapp* створено додаткові директорії для структуризації коду (рис. 3.2). В директорії *configs* знаходяться класи конфігурації бази даних, конфігурації безпеки та конфігурації *JSON* парсера. В директорії *constants* знаходяться класи з константами, які використовуються по всьому проекту. В директорії *security* знаходяться класи, що відповідають за безпеку, але не є класами конфігурації та компонентами *Spring*. В директорії *util* знаходяться утилітарні класи зі статичними службовими методами, що можуть бути використані будь-де в проекті. Оскільки головними архітектурними принципами побудови додатків на платформі *Spring* є багатошаровість та модель *MVC*, створено директорії для контролерів (*controllers*), сервісів (*services*), репозиторіїв доступу до бази даних (*repositories*), доменної області (*domain*). В доменній області знаходяться моделі БД та об'єкти переносу даних – *Data Transfer Objects* (*DTOs*).

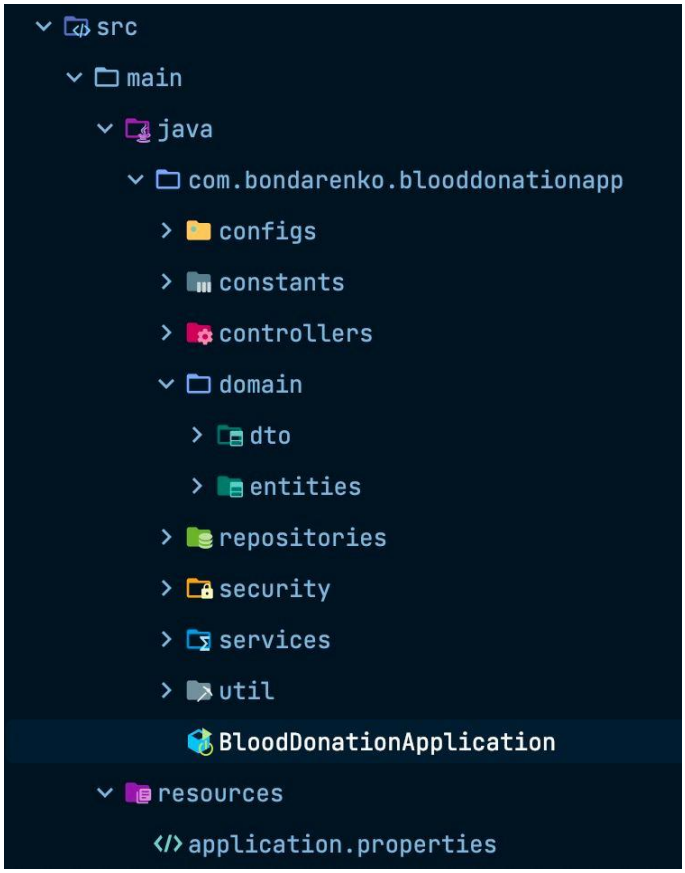

Рис. 3.2. Структура серверної частини веб-додатку служб крові

Тобто при створенні проекту серверної частини веб-додатку служб крові було виконано наступні етапи:

1. Завантажено інтегроване середовище розробки *Intellij IDEA*.
2. Завантажено пакетний менеджер *Homebrew*.
3. Завантажено СУБД *PostgreSQL*.
4. Створено проект.
5. Налаштовано проект.
6. Підключено БД до *Intellij IDEA*.

При створення проекту для клієнтської частини веб-додатку служб крові було створено нову директорію та відкрито в ній *VS Code*, далі відкрито термінал *VS Code* та запущено команду *Vue CLI* «*vue create blood-donation-app-client*». У відкритому меню в терміналі вибрано профіль установки по замовчуванню. За допомогою *Vue CLI* було автоматично установлено всі необхідні модулі для клієнтської частини веб-додатку для донорства крові.

Vue CLI створив такі директорії та файли: *node_modules* – директорію зі всіма модулями, що були завантажені *NPM* для веб-додатку служб крові; *public* – директорію зі статичними ресурсами (іконка та *HTML* файл куди буде змонтовано компоненти *Vue*); *src* – директорію з компонентами *Vue*, серед них є стандартний компонент *App.vue*, який буде змонтований до *HTML* файлу та до якого будуть змонтовані створені компоненти; *babel.config.js* – файл з конфігурацією транспайлера (транслятор, який приймає вихідний код програми та змінює його на іншу версію. Це зроблено для підтримки старих браузерів); *package.json* – файл, в якому описано всі модулі *NPM* необхідні для веб-додатку донорства крові; *package-lock.json* – файл зі згенерованими залежностями для системи збору. В директорії *src* було створено директорії: *api* – директорія із модулем *REST* запитів до серверу; *assets* – директорія зі статичними ресурсами для компонентів; components – директорія з компонентами; *plugins* – директорія з модулем підключення бібліотек як плагінів (*Vuetify*, *Axios*, *Vuex*, *Vue* Router); *router* – директорія з модулем

роутингу; *store* – директорія з модулем глобального реактивного сховища; *views* – директорія з компонентами сторінок (рис. 3.3).

Тобто при створенні проекту клієнтської частини веб-додатку служб крові було виконано наступні етапи:

1. Завантажено редактор коду *VS Code*.
2. Завантажено пакетний менеджер *NPM*.
3. Завантажено розширення для терміналу *Vue CLI*.
4. Створено проект.
5. Налаштовано проект.

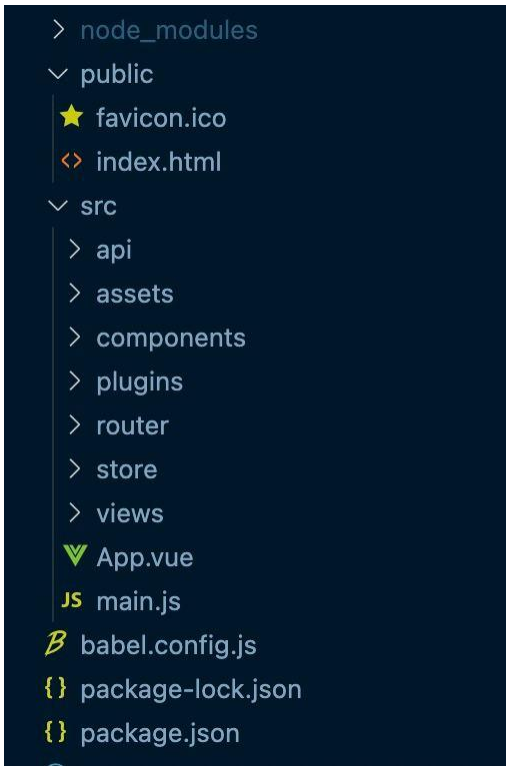

Рис. 3.3. Структура клієнтської частини веб-додатку служб крові

3.3. Розробка серверної частини веб-додатку донорства крові

Спочатку було створено п’ять класів, які є відображенням таблиць в базі даних: клас *User*, що відображає таблицю *users*; клас *DonorApplication, що* відображає таблицю *donor_application*; клас *RecipientApplication*, що відображає таблицю *recipient_application*; клас *Region*, що відображає таблицю *region*; клас

JwtBlacklist, що відображає таблицю *jwt_blacklist*. Ці класи було збережено в директорію *domain.entities*. У всіх класах-сутностях окрім *Region* ключ – це поле *private Long id* з анотацією *@Id*, що позначає поле як ключ, та анотацією *@GeneratedValue*, що визначає алгоритм генерації ключа.

Клас *User* має такі анотації: *@Data* – анотація *Lombok*, що генерує мутатори та аксесори, перевизначає методи *equals* та *hashcode*; *@Entity* – анотація *JPA*, що визначає клас як сутність бази даних; *@NoArgsConstructor* – анотація *Lombok*, що генерує конструктор без параметрів; *@AllArgsConstructor* – анотація *Lombok*, що генерує конструктор з параметрами для всіх полів класу; *@Builder* – анотація *Lombok*, що застосовує патерн *Builer*. Клас має такі поля: *private String email* (електронна пошта), *private String password* (пароль), *private String name* (ім'я та прізвище), *private Role role* (роль), *private String region* (місто служби крові), *private String sex* (стать), *private String phone* (мобільний телефон), *private Integer groupNumber* (група крові), *private String rh* (резус), *private String diseases* (історія хвороби), *private Date birthday* (день народження). Також клас має конструктор, що приймає об'єкт *DTO – UserDto*, про який буде іти мова далі.

Клас *DonorApplication* відображає таблицю *donor_application*. Клас має анотації *@Entity*, *@NoArgsConstructor*, *@Data*. Клас має такі поля: *private User user*, що має анотацію *@ManyToOne* для визначення типу зв'язку та анотацію *@JoinColumn* для визначення назви поля із зовнішнім ключем; *private Date expiresAt*, що визначає дату строку дії заявки. Також клас має конструктор, що приймає об'єкт класу *User* та дату строку дії заявки.

Клас *RecipientApplication* має анотації *@Entity*, *@NoArgsConstructor*, *@Data*. Клас має такі поля: *private User user* з анотаціями *@ManyToOne* та *@JoinColumn*; *private Date date*, що визначає дату строку дії заявки. Також клас має конструктор, що приймає об'єкт класу *User* та дату строку дії заявки.

Клас *Region* має анотації *@Entity*, *@NoArgsConstructor*, *@AllArgsConstructor*, *@Data*, *@Builder*. Клас має одне поле – *@Id private String name*.

Клас *JwtBlacklist* має анотації *@Entity*, *@NoArgsConstructor*, *@AllArgsConstructor*, *@Data*, *@Builder*. Клас має поля: *private String signature* (*JWT* токен), *private LocalDateTime expiresAt* (дата закінчення дії токена).

Далі було створено три класи *DTO* (*UserDto*, *TokenWithUserWrapper*, *ApplicationInfoDto*) та збережено їх в директорію *domain.dto*. Клас *UserDto* має анотації *@Data*, *@NoArgsConstructor*, *@AllArgsConstructor*, *@JsonIgnoreProperties(ignoreUnknown = true)* (ігнорує поля *JSON* об'єкта, що не вказані в класі) та має такі ж поля як і клас *User*. Також клас *UserDto* має конструктор, що приймає об'єкт класу *User*. Клас *TokenWithUserWrapper* має анотації *@Data*, *@NoArgsConstructor*, *@AllArgsConstructor*, *@Builder* та має такі ж поля *private String token* (*JWT* токен), *private String email* (електронна пошта), *private String id* (ідентифікатор користувача). Клас *ApplicationInfoDto* має анотації *@Data*, *@NoArgsConstructor*, *@AllArgsConstructor* та має такі ж поля як і класи *DonorApplication* та *RecipientApplication*.

Для кожного класу сутності було створено інтерфейс-репозиторій: *UserRepository*, *DonorApplicationRepository*, *RecipientApplication*, *RegionRepository*, *JwtBlacklistRepository*. Репозиторій це об'єкт доступу до даних – *Data Access Object* (*DAO*). Під час компіляції коду *Spring* автоматично створює реалізацію інтерфейс-репозиторію, що створена на основі назви методів (*Spring* аналізує назву методу і на основі отриманих даних генерує його реалізацію, тобто назви методів для репозиторіїв повинні підпорядковуватися певним правилам, описаним у документації *JPA*). Репозиторій обов'язково повинен мати анотацію *@Repository*, що має семантичне значення, а також реєструє його як компонент *Spring*. Також репозиторій має бути унаслідуваним від інтерфейсу *Repository* або від будь-якого іншого стандартного інтерфейсу, який наслідується від інтерфейсу *Repository*.

Всі *DAO* у веб-додатку для донорства крові наслідуються від інтерфейсу *JpaRepository<T, ID>*, де *T* – це сутність, для якої буде створено *DAO*; *ID* – це тип даних ключа сутності. Інтерфейс *JpaRepository* має вже готові методи для перегляду, створення, зміни та видалення сутностей. Для сутності *User* було створено репозиторій *UserRepository*, що містить створений метод *findByEmail*,

який повертає об’єкт класу *User*, знайдений за полем *email*. Для сутності *DonorApplication* було створено репозиторій *DonorApplicationRepository*, що містить метод *findAllByExpiresDateBefore*, який повертає список об’єктів класу *DonorApplication*, знайдених за полем *expiresAt*, та *findByIdAndUserId*, який повертає об’єкт класу *DonorApplication*, знайдений за ідентифікатором сутності *User* та *DonorApplication*. Для сутності *RecipientApplication* було створено репозиторій *RecipientApplicationRepository*, що містить створений метод *findAllByExpiresDateBefore*, який повертає список об’єктів класу *RecipientApplication*, знайдених за полем *expiresAt*, та *findByIdAndUserId*, який повертає об’єкт класу *RecipientApplication*, знайдений за ідентифікатором сутності *User* та *RecipientApplication*. Для сутності *Region* було створено репозиторій *RegionRepository*. Для сутності *JwtBlacklist* було створено репозиторій *JwtBlacklistRepository*, що містить створений метод *existsBySignature*, який перевіряє чи існує сутність, знайдена за полем *signature*, та *findAllByExpiresDateBefore*, який повертає список об’єктів класу *JwtBlacklist*, знайдених за полем *expiresAt*.

Після прошарку *DAO*, було створено прошарок сервісів. Для веб-додатку служб крові було створено такі сервіси: *AuthService* для авторизації та аутентифікації, *ApplicationService* для операцій із заявками, *RegionService* для операцій із містами служб крові, *UserService* для операцій із профілем користувача, *ApplicationCleanerService* для видалення прострочених заявок, *JwtBlacklistCleanerService* для видалення прострочених заявок, *UserDetailService* для надання доступу до даних користувача модулю *Spring Security*.

Кожен клас-сервіс має набір методів з бізнес-логікою, об’єднаних за певною ознакою. Також кожен сервісний клас повинен мати анотацію *@Services*, що має семантичне значення, а також реєструє його як компонент *Spring*, а також анотацію *@RequiredArgsConstructor*, що створює конструктор для *final* полів класу. За допомогою згенерованого конструктора *Spring* автоматично впроваджує залежності, якими в контексті сервісу є інші сервіси, об’єкти конфігурації та *DAO*. Кожен сервіс має виділений інтерфейс, оскільки *Spring* використовує різні

механізми впровадження залежностей для класів та інтерфейсів, і останній працює набагато швидше.

Для сервісу *AuthService* фреймворк *Spring* впроваджує такі залежності: *UserRepository*; *JwtBlacklistRepository*; *PasswordEncoder*, про конфігурацію якого мова буде іти пізніше. Також сервіс *AuthService* має такі методи: *saveDto*, що приймає об'єкт *UserDto* та створює користувача в системі описаного цим об'єктом; *logout*, що приймає *JWT* токен та заносить його в чорний список; *canAuthorize*, що приймає *JWT* токен та перевіряє чи знаходиться цей токен в чорному списку.

Для сервісу *UserService* фреймворк *Spring* впроваджує залежність *UserRepository*. Також сервіс *UserService* має такі методи: *getUserInfo*, що повертає об'єкт класу *UserDto* знайдений за його ідентифікатором; *saveUser*, що приймає об'єкт *UserDto* та обновляє інформацію про користувача в системі даними, що інкапсульовані в цьому об'єкті.

Для сервісу *RegionService* фреймворк *Spring* впроваджує залежність *RegionRepository*. Також сервіс *RegionService* має метод *getAllRegions*, що повертає список назв всіх міст служб крові.

Для сервісу *ApplicationService* фреймворк *Spring* впроваджує такі залежності: *UserRepository*, *DonorApplicationRepository*, *RecipientApplicationRepository*. Також сервіс *UserRepository* має такі методи: *findDonorApplication*, *findRecipientApplication*, *addDonorApplication*, *addRecipientApplication*, *editDonorApplication*, *editRecipientApplication*, *deleteDonorApplication*, *deleteRecipientApplication*. Методи *findDonorApplication* та *findRecipientApplication* повертають список об'єктів класу *ApplicationInfoDto*, що створений за допомогою *Stream API*. *Stream API* – це програмний інтерфейс, що входить в склад *JDK 8+*, який дозволяє проводити операції над списками у функціональному стилі (рис. 3.4). Використовуючи *Stream API*, було відфільтровано всі знайдені заявки за параметрами (якщо вони не дорівнюють *null*): *Long userId* (ідентифікатор користувача), *String rh* (резус), *Integer groupNumber* (група крові), *String region* (місто служби крові) [12].

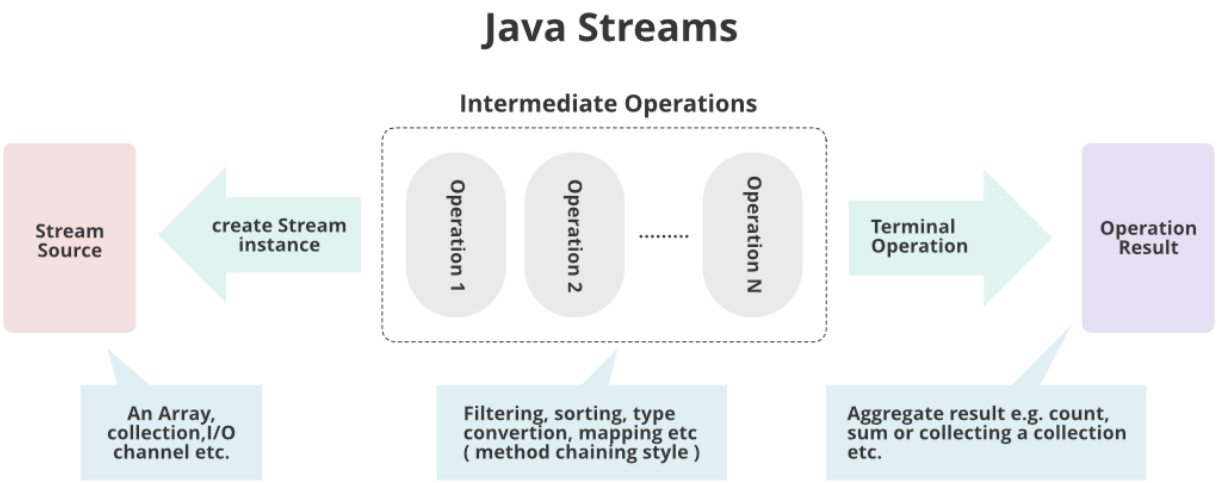

Рис. 3.4. Схема прикладного потокового інтерфейсу *Stream API*

Методи *addDonorApplication* та *addRecipientApplication* створюють нові заявки донорів та реципієнтів відповідно за переданими ідентифікатором користувача та датою. Методи *editDonorApplication* та *editRecipientApplication* редагують заявки донорів та реципієнтів відповідно датою, що була передана як параметр цих методів. Заявки, що будуть редагуватися були знайдені за ідентифікатором користувача та ідентифікатором заявки, що були передані як параметри методів. Методи *deleteDonorApplication* та *deleteRecipientApplication* видаляють заявки донорів та реципієнтів відповідно, що знайдені за переданими ідентифікатором користувача та ідентифікатором заявки.

Для сервісу *ApplicationCleanerService* фреймворк *Spring* впроваджує такі залежності: *DonorApplicationRepository, RecipientApplicationRepository*. Сервіс *ApplicationCleanerService* у своєму конструкторі створює два окремих потоки за допомогою класу *ScheduledExecutorService*, які кожні дві години шукають прострочені заявки донорів та реципієнтів та видаляють їх.

Для сервісу *JwtBlacklistCleanerService* фреймворк *Spring* впроваджує залежність *JwtBlacklistRepository*. Сервіс *JwtBlacklistCleanerService* у своєму конструкторі створює окремий потік за допомогою класу *ScheduledExecutorService*, який кожні дві години шукає недійсні *JWT* токени та видаляє їх.

Після прошарку сервісів було створено прошарок контролерів. Кожен клас-контролер описує *REST* інтерфейс та створює об'єкти з даних, що отримані із запиту, для виклику методів впроваджених сервісів. Також кожен клас-контролер повинен мати анотацію *@RestController*, що реєструє його як компонент *Spring*, а саме як *REST* контролер; анотацію *@RequiredArgsConstructor*, що створює конструктор для *final* полів класу; анотацію *@ReguestMapping*, в якій указано відносний шлях до всіх запитів, які містяться в контролері. Всього було розроблено два контролера: *AuthController*, що відповідає за аутентифікацію та авторизацію; *RequestController*, що відповідає за операції над заявками та профілем.

Для контролеру *AuthController* фреймворк *Spring* впроваджує залежність *AuthService*. Контролер *AuthController* оголошує два ендпоінти: «/auth/register» типу *POST*, що проводить реєстрацію нового користувача, за даними, що передані як тіло запиту у форматі *JSON*; *«/auth/logout*» типу *POST*, що проводить вихід користувача із системи, який ідентифікується за *JWT* токеном, що був переданий в *HTTP* заголовку *authorization*.

Для контролеру *RequestController* фреймворк *Spring* впроваджує залежності *UserService, ApplicationService, RegionService*. Контролер *AuthController* оголошує такі ендпоінти:

- *«/api/findDonorApplications*» типу *GET*. Повертає масив *JSON* об'єктів з інформацією про заявки донорів, що міститься в об'єктах класу *ApplicationInfoDto*. Приймає параметри *userId, rh, groupNumber, region* як елементи рядку запиту.
- *«/api/findRecipientApplications*» типу *GET*. Повертає масив *JSON* об'єктів з інформацією про заявки реципієнтів, що міститься в об'єктах класу *ApplicationInfoDto*. Приймає параметри *userId, rh, groupNumber, region* як елементи рядку запиту.
- *«/api/addDonorApplications*» типу *POST*. Приймає *JSON* об'єкт з інформацією про заявку донора як тіло запиту та створює відповідний запис в базі даних.

– *«/api/addRecipientApplications»* типу *POST*. Приймає *JSON* об’єкт з інформацією про заявку реципієнта як тіло запиту та створює відповідний запис в базі даних.

– *«/api/deleteDonorApplications»* типу *DELETE*. Приймає *JSON* об’єкт з ідентифікатором заявки донора як тіло запиту та видаляє відповідний запис в базі даних.

– *«/api/deleteRecipientApplications»* типу *DELETE*. Приймає *JSON* об’єкт з ідентифікатором заявки реципієнта як тіло запиту та видаляє відповідний запис в базі даних.

– *«/api/editDonorApplications»* типу *POST*. Приймає *JSON* об’єкт з інформацією про заявку донора як тіло запиту та редагує відповідний запис в базі даних.

– *«/api/editRecipientApplications»* типу *POST*. Приймає *JSON* об’єкт з інформацією про заявку реципієнта як тіло запиту та редагує відповідний запис в базі даних.

– *«/api/user/{id}»* типу *GET*, де *id* – це ідентифікатор користувача. Повертає *JSON* об’єкт з інформацією про користувача знайденого за переданим ідентифікатором.

– *«/api/user»* типу *POST*. Приймає *JSON* об’єкт з інформацією про користувача та створює відповідний запис в базі даних.

– *«/api/regions»* типу *GET*. Повертає *JSON* масив рядків з назвою усіх можливих міст служб крові.

Було створено два фільтри: *JWTAuthenticationFilter* та *JWTAuthorizationFilter*, що реалізують стандартні інтерфейси *UsernamePasswordAuthenticationFilter* та *BasicAuthorizationFilter* (рис. 3.5). В контексті *Spring* фільтр – це певний клас, код якого виконується до і (або) після виконання коду ендпоінту. У веб-додатку служб крові це зроблено для аутентифікації та авторизації відповідно. Також було створено такі класи конфігурації: *DbSecurityConfig* (конфігурація алгоритму шифрування), *DataConfig* (конфігурація підключення до БД), *JsonConfig*

(конфігурація *JSON* парсера для перетворення *DTO* в *JSON* об'єкти і навпаки), *WebSecurityConfig* (конфігурація аутентифікації та авторизації).

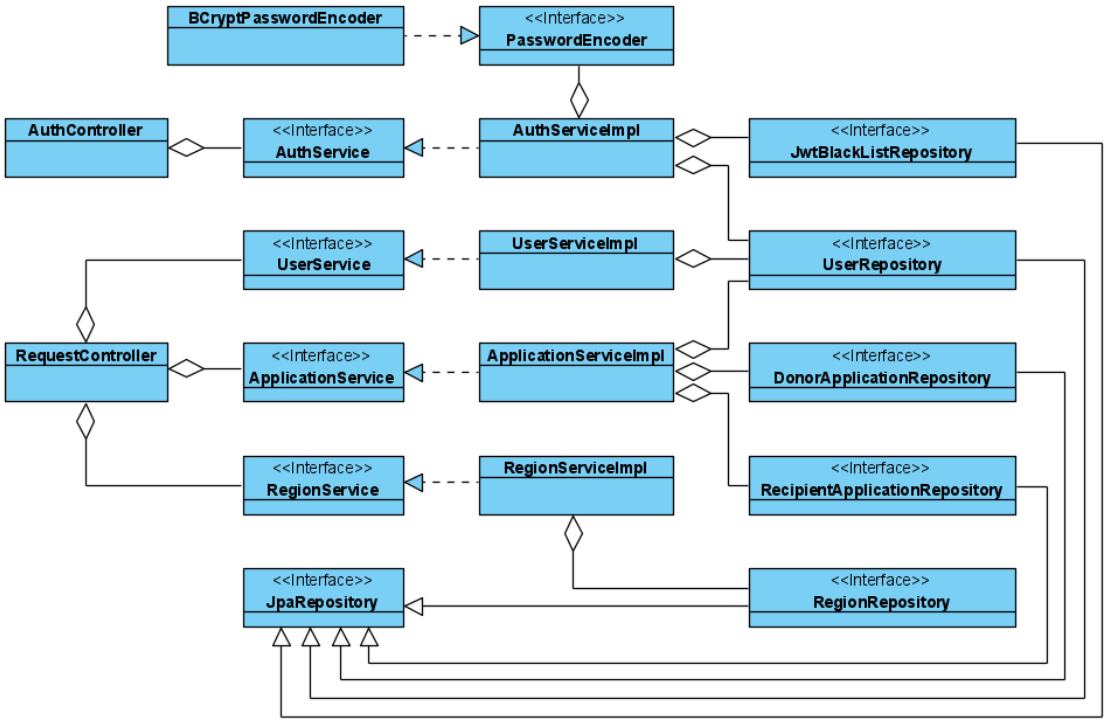

Рис. 3.5. *UML* діаграма основних класів веб-додатку донорства крові

3.4. Розробка клієнтської частини веб-додатку донорства крові

При розробці клієнтської частини веб-додатку донорства крові було сконфігуровано бібліотеку *Vuetify* як плагін в директорії */src/plugins*, було указано основні кольори. Також було сконфігуровано бібліотеку *Axios* в директорії */src/api* так, щоб вона надсилала запити на сервер веб-додатку для служб крові. Потім було зареєстровано у файлі *main.js* всі бібліотеки, які будуть доступні для використання через змінні контексту *Vue*: *Vuex*, *Vue Router*, *Vuetify* [13]. Також у файлі *main.js* було змонтовано головний компонент – *App*, про який буде іти мова нижче.

Після створення базової конфігурації клієнтської частини було створено основні компоненти: *Card*, *CreateAppModal*, *EditAppModal*, *SearchBar*. Кожен компонент має *HTML* шаблон [14], який зв'язаний з обробниками подій та реактивними змінними, *JS* скрипт [15] та необов'язковий тег з *CSS* кодом [16].

Компоненти можуть використовувати екземпляри один-одного в шаблоні для забезпечення простоти коду.

Компонент *Card* відображає інформацію про одну заявку донора або реципієнта. Інформація про заявку передається з компонента, що стоїть вище по ієрархії. Якщо заявка була створена поточним користувачем (відбувається перевірка ідентифікатора збереженого у *Vuex* та ідентифікатора користувача в заявці), то відображаються кнопки для видалення та редагування заявки із зареєстрованими обробниками подій відповідно.

Компонент *CreateAppModal* створює модальне вікно з компонентом вибору дати *Vuetify* для створення заявки донора або реципієнта. При виборі дати кнопка «Створити» стає активною і якщо користувач натисне на неї, то функція-обробник сформує запит для створення заявки донора або реципієнта на сервер з вибраної дати, ідентифікатора користувача і *JWT* токена.

Компонент *EditAppModal* викликає модальне вікно з компонентом вибору дати *Vuetify* для редагування заявки донора або реципієнта. При виборі дати кнопка «Редагувати» стає активною і якщо користувач натисне на неї, то функція-обробник сформує запит для редагування заявки донора або реципієнта на сервер з вибраної дати, ідентифікатора користувача, ідентифікатора заявки і *JWT* токена.

Компонент *SearchBar* містить фільтри та кнопки для управління відфільтрованою вибіркою заявок донорів або реципієнтів. При завантаженні компонента, автоматично завантажується список міст служб крові для однойменного фільтру. При виборі опції фільтра вибраний варіант записується в локальне сховище компонента. При натисканні на кнопку «Пошук» формується запит на сервер для отримання відфільтрованого списку заявок, при натисканні на кнопку «Скинути» формується запит на сервер для отримання повного списку заявок, а також очищуються значення фільтрів у локальному сховищі компонента.

Після розробки основних компонентів було створено компоненти-сторінки: *Home*, *Profile*, *SignIn*, *SignUp*. Кожен компонент-сторінка має ті ж властивості, що й звичайний компонент, крім цього компонент-сторінка повинен бути

зареєстрований в роутері (файл *router/index.js*) для забезпечення правильного переходу між сторінками [17].

Сторінка *Home* відображає головне вікно веб-додатку для донорства крові. В шаблон сторінки *Home* включено по два компоненти *Card* та *SearchBar*, по одному для заявок донорів та заявок реципієнтів. При завантаженні компонента, автоматично завантажується список заявок донорів та реципієнтів, далі створюється відповідна кількість компонентів *Card* з переданою інформацією про донора або реципієнта. За допомогою обробки подій пошуку з фільтром компонента *SearchBar*, дані що передаються разом з подіями перехвачуються на сторінці *Home* та запускають рендер компонентів *Card* з новими даними.

Сторінка *Profile* відображає інформацію про донора чи реципієнта, сесія якого активна на даний момент. При завантаженні сторінки створюється запит на сервер для отримання даних користувача. Сторінка *SignUp* має компоненти вводу *Vuetify* з вбудованою валідацією. При введенні валідних персональних даних надсилається запит на сервер для створення профілю користувача і якщо запит успішний користувача буде перенаправлено на сторінку входу *SignIn*. На сторінці *SignIn* знаходяться два компоненти вводу *Vuetify* з вбудованою валідацією для паролю та електронної пошти, при успішній реєстрації сервер поверне *JWT* токен та ідентифікатор користувача, які будуть збережені в сховищі *Vuex* для того, що бути доступними в будь-якому компоненті *Vue*.

В результаті розробки веб-додатку для служб крові було отримано такі сторінки: головна сторінка зі списком донорів та реципієнтів (рис. 3.6), сторінка профілю користувача (рис. 3.7), сторінка входу (рис. 3.8) , сторінка реєстрації (рис. 3.9). Між сторінками створено зручні переходи за допомогою елементів інтерфейсу.

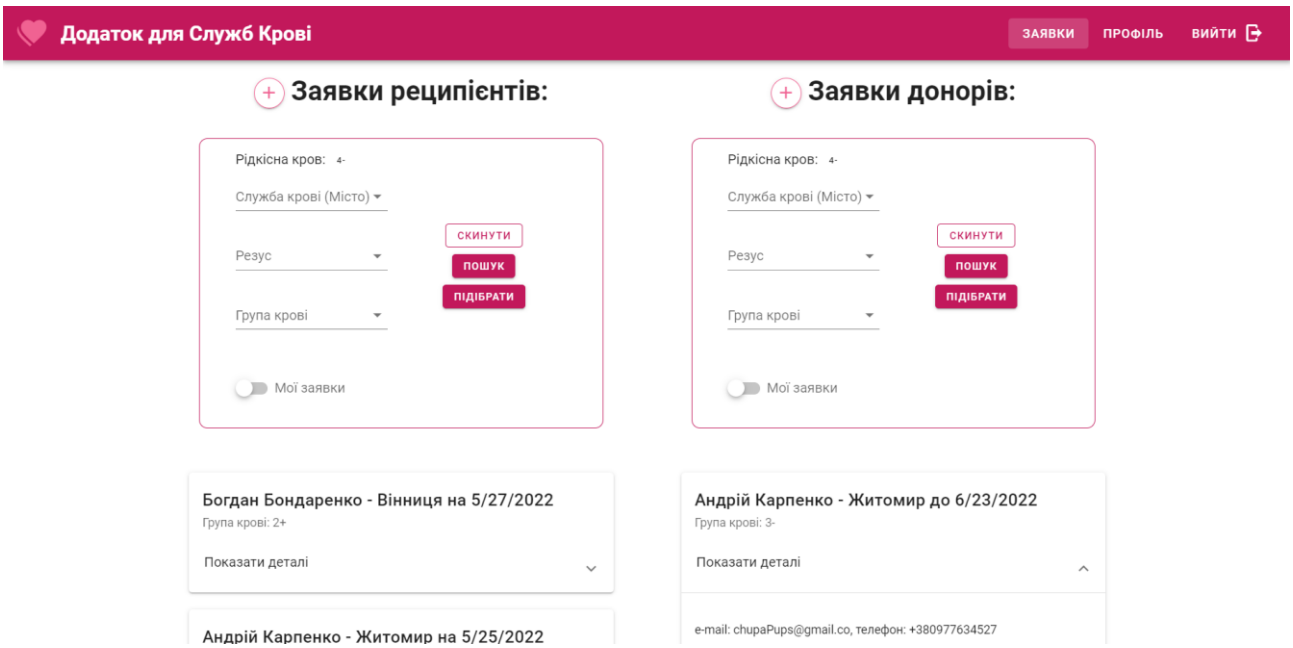

Рис. 3.6. Головна сторінка веб-додатку служб крові

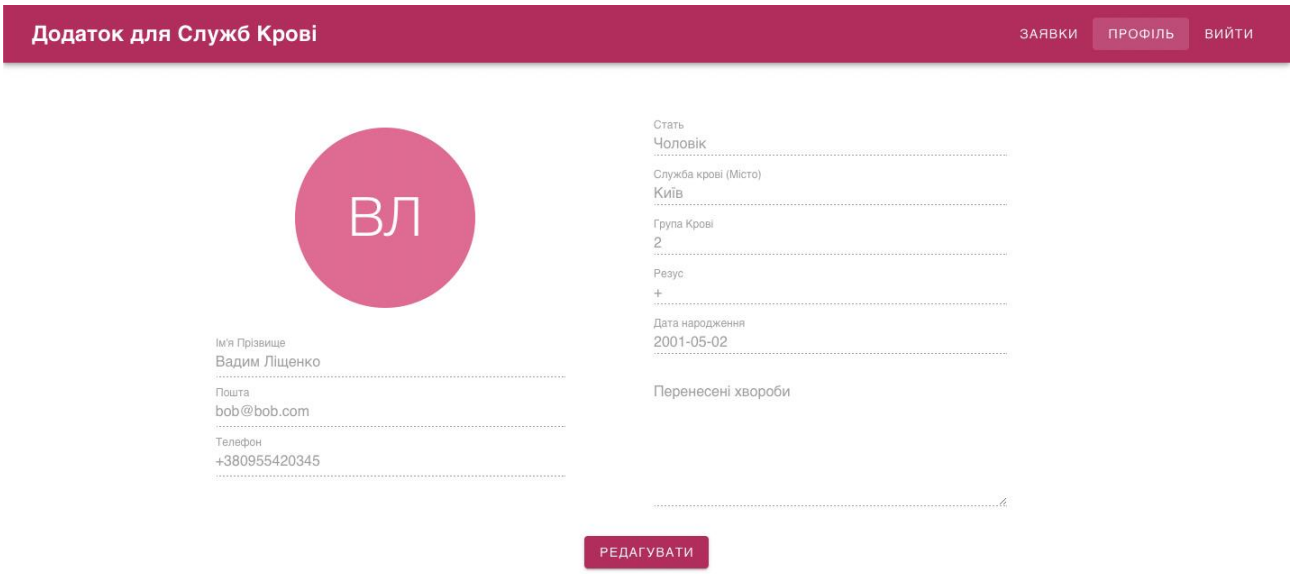

Рис. 3.7. Сторінка профілю користувача веб-додатку служб крові

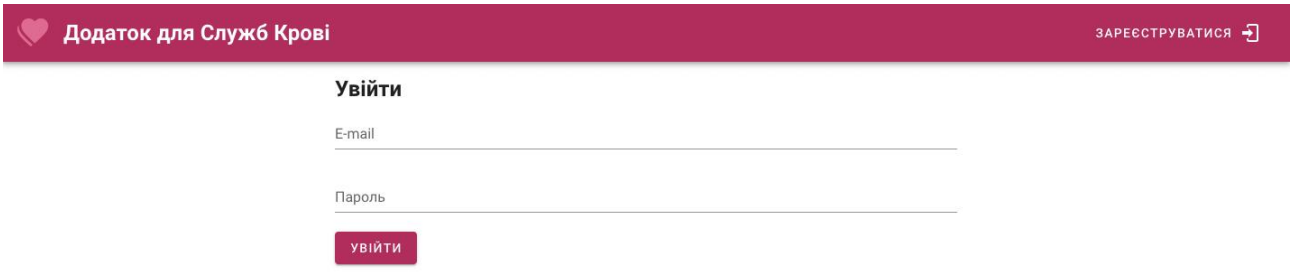

Рис. 3.8. Сторінка входу у веб-додатку служб крові

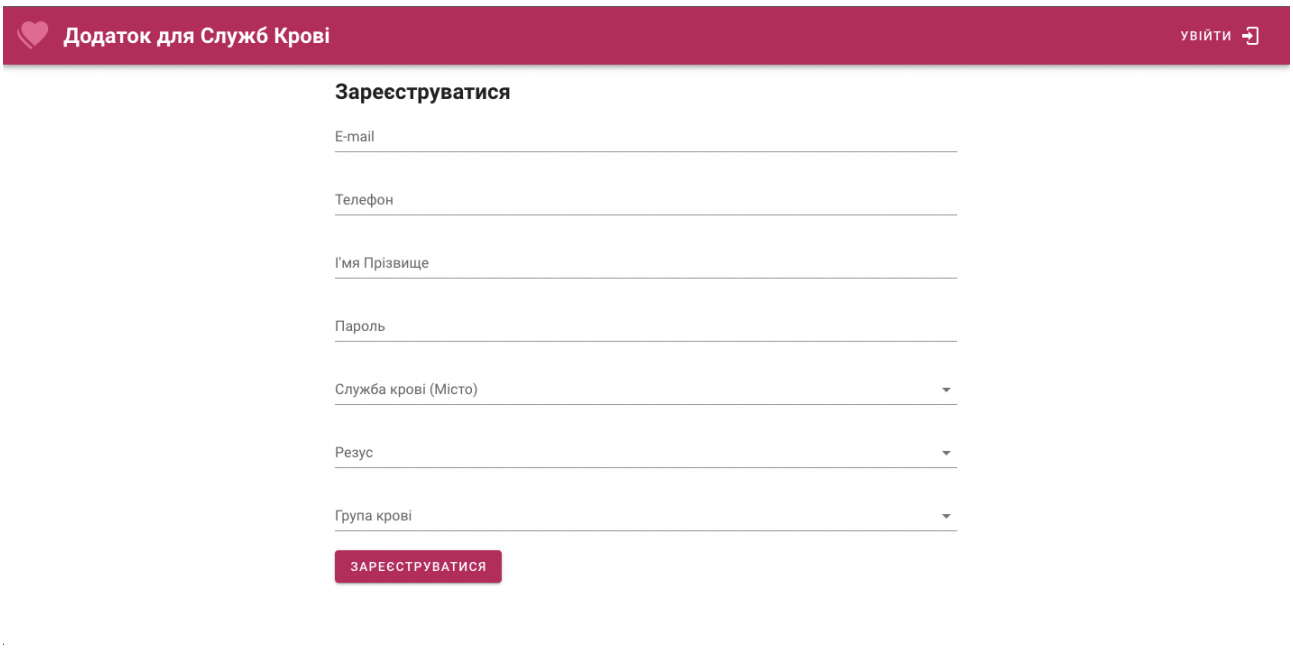

Рис. 3.9. Сторінка реєстрації у веб-додатку служб крові

3.5. Висновки до розділу

Платформа *Spring* ідеально підходить для розробки додатків з великою кількістю модулів та додатків, що потребують високої надійності. Незважаючи не серйозні вимоги до додатків на *Spring*, фреймворк пропонує відносно прості та добре документовані інструменти розробки як середніх за розміром програм, так і програм із сотнями класів. Ці властивості зумовлені трьома ключовими поняттями, на яких базується *Spring*: впровадження залежностей – *Dependency injection* (*DI*), інверсія керування – *Inversion of control* (*IoC*), аспектно-орієнтоване програмування – *Aspect-oriented programming* (*AOP*).

Було завантажено необхідні пакетні менеджери (*Homebrew*, *NPM*), що необхідні для установки бібліотек, пакетів та модулів для розробки веб-додатку для донорства крові. Також було завантажено редактор коду *VS Code* та інтегроване середовище розробки *Intellij IDEA* для розробки клієнтської та серверної частини веб-додатку для служб крові. При розробці веб-додатку для донорства крові було створено два проекти: проект для серверної частини під управлінням фреймворку *Spring* та проект для клієнтської частини під управлінням фреймворку *Vue*. Також

було запущено СУБД *PostgreSQL* та під'єднано її до інтегрованого середовища розробки *Intellij IDEA* для подальшого управління та моніторингу даних.

При розробці серверної частини було створено класи-сутності, що відображають таблиці, які знаходяться в базі даних; для кожного класу сутності було створено інтерфейс-репозиторій, який є об'єктом доступу до даних. Інтерфейси-репозиторії використовуються в розроблених класах з бізнес-логікою – класах-сервісах. Кожен клас-сервіс має набір методів, об'єднаних за певною ознакою. Клас-сервіс має виділений інтерфейс, оскільки *Spring* використовує різні механізми впровадження залежностей для класів та інтерфейсів, і останній працює набагато швидше. Після сервісів було створено контролери. Кожен клас-контролер описує *REST* інтерфейс та створює об'єкти з даних, що отримані із запиту, для виклику методів впроваджених сервісів.

При розробці клієнтської частини веб-додатку донорства крові було сконфігуровано бібліотеку *Vuetify* як плагін в директорії */src/plugins*, було указано основні кольори. Також було сконфігуровано бібліотеку *Axios* в директорії */src/api* так, щоб вона надсилала запити на сервер веб-додатку для служб крові. Потім було зареєстровано у файлі *main.js* всі бібліотеки, які будуть доступні для використання через змінні контексту *Vue*: *Vuex, Vue Router, Vuetify*.

Після створення базової конфігурації клієнтської частини було створено основні компоненти: *Card, CreateAppModal, EditAppModal, SearchBar*. Кожен компонент має *HTML* шаблон, який зв'язаний з обробниками подій та реактивними змінними, *JS* скрипт та необов'язковий тег з *CSS* кодом. В результаті розробки веб-додатку для служб крові було отримано такі сторінки: головна сторінка зі списком донорів та реципієнтів, сторінка профілю користувача, сторінка входу, сторінка реєстрації. Між сторінками створено зручні переходи за допомогою елементів інтерфейсу.

ВИСНОВКИ

Служба крові – це структура, що об'єднує по всій країні медичні установи та їх структурні підрозділи, основним видом діяльності яких є заготівля, переробка, зберігання та забезпечення безпеки донорської крові та її компонентів. Було проаналізовано існуючі технологічні рішення в сфері автоматизованих систем донорства. Було проаналізовано напрямки діяльності служб крові. Було визначено основні завдання служб крові: забезпечення медичних закладів компонентами крові та розвиток добровільного донорства крові. Для цього служба крові організовує роботу з донорами з отримання донорської крові, її зберігання та видачу. Визначено, що ефективна діяльність служби крові неможлива без участі суспільства загалом, його громадських інститутів, бізнесу, ініціативи приватних осіб.

Використання автоматизованих систем обліку донорства крові у роботі центрів трансфузіології значно спрощує ряд робочих процесів та підвищує їх ефективність при управлінні процесами взяття, розподілу донорської крові та її компонентів. Автоматизовані системи рекрутингу донорів повинні мати модулі для знаходження відділень донорства крові, обходу черг на прийом до лікаря, збереження медичної картки та історії донацій.

Було визначено основні технологічні рішення в сфері автоматизованих систем донорства, такі як ДонорUA, автоматизована інформаційна система трансфузіології (АІСТ) та *GiveBlood*. Ці програмні забезпечення є найпоширенішими в Україні, Росії та Канаді відповідно. Кожна із вище перерахованих систем виконує свою головну функцію – допомагає службам крові в забезпеченні збору крові. Але веб-додаток ДонорUA має значну перевагу, тому що він безкоштовний, а також підтримує українську мову, незважаючи на це, ДонорUA має невелику базу даних та менше модулів, ніж АІСТ чи *GiveBlood*.

Далі було визначено технології для проектування веб-додатку служб крові: *JSON Web Token* (*JWT*) для передачі даних аутентифікації у веб-додатку для служб кров; архітектурний стиль *Representational State Transfer* (*REST*) для зв’язку клієнтської та серверної частини веб-додатку служб кров; бібліотеку *Vuetify* для

створення інтерфейсу для веб-додатку служб крові, *Vuex* як реактивне сховище даних та *Vue Router* як бібліотека для створення роутингу і основна платформа для спроектованого веб-додатку – *Spring* для створення серверної частини. Було спроектовано *ER*-модель бази даних та визначено необхідні таблиці: таблиця для збереження інформації про користувача, таблиця для збереження заявок донорів, таблиця для збереження заявок реципієнтів, таблиця для збереження не валідних токенів доступу, таблиця для збереження міст служб крові. Було спроектовано інтерфейс веб-додатку служб крові на основі інструментарію, що пропонує *UI* бібліотека *Vuetify* та фреймворк *Vue*. Далі було спроектовано модулі веб-додатку, серед них модуль пошуку заявок донора та реципієнта, модуль реєстрації та аутентифікації, модуль перегляду профілю.

Було завантажено необхідні пакетні менеджери (*Homebrew, NPM*), що необхідні для установки бібліотек, пакетів та модулів для розробки веб-додатку для донорства крові. Також було завантажено редактор коду *VS Code* та інтегроване середовище розробки *Intellij IDEA* для розробки клієнтської та серверної частини веб-додатку для служб крові. Створено та налаштовано проекти для серверної та клієнтської частини веб-додатку служб крові на *Spring* та *Vue*, що використані як основні технології відповідно. При створенні проекту серверної частини веб-додатку служб крові було виконано наступні етапи: , завантажено пакетний менеджер *Homebrew*, завантажено інтегроване середовище розробки *Intellij IDEA*, завантажено СУБД *PostgreSQL*, створено сам проект, налаштовано проект, підключено БД до *Intellij IDEA*. При створенні проекту клієнтської частини веб-додатку служб крові було виконано наступні етапи: завантажено пакетний менеджер *NPM*, завантажено редактор коду *VS Code*, завантажено пакетний менеджер *NPM*, завантажено розширення для терміналу *Vue CLI*, створено сам проект, налаштовано проект.

При розробці серверної частини було створено класи-сутності, що відображають таблиці, які знаходяться в базі даних; для кожного класу сутності було створено інтерфейс-репозиторій, який є об’єктом доступу до даних. Інтерфейси-репозиторії використовуються в розроблених класах з бізнес-логікою

– класах-сервісах. Кожен клас-сервіс має набір методів, об’єднаних за певною ознакою. Клас-сервіс має виділений інтерфейс, оскільки *Spring* використовує різні механізми впровадження залежностей для класів та інтерфейсів, і останній працює набагато швидше. Після сервісів було створено контролери. Кожен клас-контролер описує *REST* інтерфейс та створює об’єкти з даних, що отримані із запиту, для виклику методів впроваджених сервісів.

При розробці клієнтської частини було сконфігуровано бібліотеки, які є необхідними для створення сучасного та зрозумілого інтерфейсу користувача. Також було створено основні компоненти *Vue*, та на їх основі – сторінки *Vue*.

В результаті розробки веб-додатку для служб крові було отримано такі сторінки: головна сторінка зі списком донорів та реципієнтів, сторінка профілю користувача, сторінка входу, сторінка реєстрації. Між сторінками створено зручні переходи за допомогою елементів інтерфейсу. Після виконання роботи було наведено скріншоти роботи веб-додатку донорства крові.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Напрямки діяльності служб крові [Електронний ресурс]. – Режим доступу: https://www.apteka.ua/article/384608 (дата звернення 19.05.2022)
2. Інформаційні системи обліку донорства крові [Електронний ресурс]. – Режим доступу: https://bloodservice.org.ua/naukovi-statti/vprovadzhennya-avtomatizovanih-informatsijnih-tehnologij-v-zakladi-sluzhbi-krovi.html (дата звернення 19.05.2022)
3. Процедура здачі крові [Електронний ресурс]. – Режим доступу: https://cardio.org.ua/dovidnik-donora-krovi/ (дата звернення 19.05.2022)
4. Клінічні аспекти трансфузіології: навч. посіб. / О. О. Потапов, М. М. Рубанець, О. П. Кмита. – Суми: Сумський державний університет, 2019. – 397 с.
5. ДонорUA [Електронний ресурс]. – Режим доступу: https://www.donor.ua
6. Служба крові AICT [Електронний ресурс]. – Режим доступу: https://zdrav.expert/index.php/Проект:Служба_крови (дата звернення 19.05.2022)
7. GiveBlood [Електронний ресурс]. – Режим доступу: https://apps.apple.com/ca/app/giveblood/id804765636 (дата звернення 19.05.2022)
8. Крейг Уоллс. Spring в действии. – Третє. – М.: «Manning», 2014. – 624 с.
9. MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org (дата звернення 22.05.2022)
10. Організація баз даних: практичний курс: Навч. посіб. для студ. / А. Ю. Берко, О. М. Верес; Нац. ун-т «Львів. політехніка». – Л., 2003. – 149 с. – Бібліогр.: 8 назв.
11. К. Бауэр, Г. Грегори, Г. Кинг. Java Persistence API и Hibernate. – Друге. – ДМК Пресс, 2017. – 632 с.
12. Герберт Шилдт. Java. Полное руководство. – Десяте. – М.: «Диалектика», 2018. – 1488 с.
13. Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org (дата звернення 22.05.2022)
14. Фримен Эрик, Фримен Элизабет. Изучаем HTML, XHTML и CSS. – Перше. – М.: «Питер», 2010. – 656 с.

15. Мартін Фаулер. Рефакторинг кода на JavaScript: улучшение проекта существующего кода. – Друге. – М.: «Диалектика», 2019. – 464 с.

16. Дэвид Сойер Макфарланд. Новая большая книга CSS. – Санкт-Петербург: Питер, 2017. – 720 с.

17. Callum Macrae. Vue.js: Up and Running. – O'Reilly, 2017. – 219 с.

18. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86с.

19. Слободян О. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: Видавництво НАУ, 2017. – 63с.