Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: "Розроблення програмного забезпечення мобільного застосунку "Нумізмат" з використанням фреймворку Іоніс"

Виконав: студент 6 курсу групи КН(м)
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Пилипців Б.М.
(прізвище та ініціали)

Керівник Карашецький В.П.
(прізвище та ініціали)

Рецензент Пірко І. Б.
(прізвище та ініціали)

Львів – 2024 року

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ
Завідувач кафедри
Борецька І.Б.
"05" січня 2024 року

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

Пилипців Богдан Михайлович
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи Розроблення програмного забезпечення мобільного застосунку "Нумізмат" з використанням фреймворку Ionic

керівник роботи Карашецький Володимир Петрович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "13" 02 2023 року № С-49
2. Термін подання студентом проєкту (роботи) 05 січня 2024 року
3. Вихідні дані до проєкту (роботи) Розроблення програмного забезпечення мобільного застосунку "Нумізмат" з використанням фреймворку Ionic.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Розділ 1. Стан проблемної області.
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Дата видачі завдання 15 лютого 2023 року

2

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних та інших джерел згідно досліджуваної теми.	1.03.23 – 27.03.23	виконано
2	Аналіз досліджуваної теми та вибір відповідних варіантів її розробки.	4.04.23 – 24.04.23	виконано
3	Постановка задачі та її формалізація.	25.04.23 – 1.05.23	виконано
4	Вибір та обгрунтування методів і засобів проведення дослідження.	2.05.23 – 22.05.23	виконано
5	Розроблення концептуальної схеми реалізації завдання.	23.05.23 – 29.05.23	виконано
6	Програмна реалізація завдання.	30.05.23 – 30.08.23	виконано
7	Тестування програмного продукту та отриманих результатів	31.08.23 – 30.10.23	виконано
8	Розробка пояснювальної записки та презентації дипломної роботи	31.10.23 – 30.12.23	виконано

Студент _____ Пилипців Б.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карашецький В.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська дипломна робота складається із 85 стор., 21 рис., 3 табл., 2 додатки, 2 джерел.

В даному дипломному проєкті розроблено мобільний застосунок “Нумізмат”, який дозволяє користувачам створювати та редагувати власні акаунти колекцій монет або валют, а також залишати свої коментарі до певної колекції. Розроблений застосунок надає можливість адміністратору застосунку редагувати або видаляти коментарі будь-якого користувача. Виконано тестування програмного забезпечення застосунку.

Ключові слова: мобільний застосунок, нумізмат, Ionic, Android, Angular, Node.js, MongoDB.

ABSTRACT

Master's thesis consists of 85 pages, 21 figures, 3 tables, 2 appendices, 25 sources.

In this diploma project, the mobile application “Numizmat” was designed, which allows users to create and edit their own accounts of coins or currencies, as well as leave their comments on a certain collection. The developed application should allow the application administrator to edit or delete any user's comments. Application software testing is done.

Keywords: mobile application, numismatist, Ionic, Android, Angular, Node.js, MongoDB.

ТЕХНІЧНЕ ЗАВДАННЯ

Спроєктувати мобільний застосунок “Нумізмат”, який дозволяє користувачам створювати та редагувати власні акаунти колекцій монет або валют, а також залишати свої коментарі до певної колекції. Розроблений застосунок повинен надавати можливість адміністратору редагувати або видаляти коментарі будь-якого користувача.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ОС – операційна система;

ПЗ – програмне забезпечення;

СКБД – система керування базами даних;

Android – операційна система для смартфонів;

API – Application Programming Interface;

VS Code – середовище розробки Visual Studio Code;

UI – User Interface;

UX – User Experience;

NoSQL – Not Only SQL;

SQL – Structured Query Language;

CLI – Command Line Interface;

JSON – JavaScript Object Notation;

JWT – JSON Web Token.

ВСТУП

Мобільні застосунки мають безліч переваг для користувачів у різних сферах життя. Вони забезпечують зручний доступ до інформації, спрощують організацію та планування завдань, полегшують комунікацію та спілкування з іншими людьми, а також надають доступ до різноманітних функцій і сервісів, які поліпшують повсякденне життя.

Загалом, мобільні застосунки мають значний вплив на покращення ефективності, зручності та комунікації людей у цифрову епоху.

Об'єкт дослідження – можливості фреймворку Ionic для розробки високоякісних мобільних застосунків.

Предмет дослідження – мобільний застосунок розроблений з використанням фреймворку Ionic.

Новизна роботи полягає в проведенні дослідження можливостей платформи Ionic та Node.js для спрощення розробки мобільних застосунків.

Практична значимість. Розроблене програмне забезпечення демонструє основні переваги використання фреймворку Ionic для створення високоякісних мобільних застосунків.

Спроєктований застосунок дозволяє розробнику наповнювати і доповнювати базу даних інформацією у реальному часі.

Актуальність роботи. Розроблений мобільний застосунок має безліч переваг для колекціонерів і нумізматів. Він надає зручний доступ до інформації про монети, їх характеристики та історію. Застосунок дозволяє організувати колекцію, вести облік монет, фотографувати їх і зберігати докладну інформацію. а також спілкуватися з іншими колекціонерами, обмінюватися досвідом і знаходити нових колекціонерів для покупки та обміну монет. Загалом, мобільний застосунок стане незамінним інструментом для сучасних колекціонерів і нумізматів.

Мета дослідження – створення зручного мобільного застосунку для нумізматів та колекціонерів.

Для досягнення поставленої мети передбачено розв’язання таких завдань:

- провести аналіз можливостей, що надає фреймворк Ionic для розробки мобільних застосунків;
- спроєктувати та розробити застосунок “Нумізмат” для демонстрації використаних можливостей фреймворку Ionic;
- розроблення API з використанням платформи Node.js та зберіганням даних у БД з використанням СКБД MongoDB;
- додати можливість реєстрації та авторизації користувачам, а також редагування акаунта;
- виконати тестування спроєктованого застосунку;
- проаналізувати отримані результати та зробити висновки.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Актуальність предметної області

У сучасному цифровому світі мобільні застосунки стали невід'ємною частиною нашого повсякденного життя. Вони надають нам доступ до різноманітних сервісів, інформації та можливостей прямо зі смартфона чи планшета. В цьому контексті розробка мобільного застосунку для нумізматів і колекціонерів є актуальною і важливою задачею. Нижче приведено декілька аргументів, що підкреслюють актуальність цієї предметної області:

1. *Зручність та доступність* застосунки – Мобільні застосунки надають користувачам зручний спосіб управління колекцією нумізматичних або колекційних предметів. Завдяки ним, користувачі можуть легко переглядати, додавати, редагувати та оновлювати інформацію про предмети колекції, додавати фотографії, відстежувати статус та вартість своїх колекцій.

2. *Організація та каталогізація* – Мобільні застосунки можуть надати структуровану систему організації та каталогізації нумізматичних або колекційних предметів. Користувачі можуть створювати категорії, теги, мітки або групи для своїх предметів, що спрощує пошук і навігацію в колекції.

3. Зберігання інформації – Мобільні застосунки надають можливість зберігати детальну інформацію про кожен предмет колекції. Вони дозволяють записувати дату, місце придбання, історію власності, опис, стан, рідкісність та інші характеристики предмета. Це допомагає нумізматам та колекціонерам відстежувати історію своїх предметів та надає зручну базу даних (БД) для подальшого дослідження.

4. *Взаємодія зі спільнотою* – Мобільні застосунки можуть забезпечити можливість взаємодії зі спільнотою нумізматів та колекціонерів. Вони можуть включати функції обговорень, коментування, обміну досвідом та знаннями з іншими користувачами. Це дозволяє створити спільноту однодумців та знайти нових друзів та колег у своїй галузі.

5. *Тренд цифровізації* – За останні роки спостерігається загальний тренд цифровізації різних сфер життя [6]. Використання мобільного застосунку для управління колекцією відображає цей тренд і дозволяє нумізматам та колекціонерам крокувати з часом та використовувати сучасні технології для оптимізації та поліпшення своїх процесів.

Зважаючи на ці фактори, розробка мобільного застосунку для нумізматів і колекціонерів є актуальною темою. Це не лише сприятиме вдосконаленню процесів управління колекцією, але й дозволить продемонструвати свої навички розробки мобільних застосунків та розуміння потреб цільової аудиторії.

1.2 Аналіз предметної області мобільного застосунку

У рамках магістерської дипломної роботи, аналіз предметної області мобільного застосунку для нумізматів і колекціонерів є ключовим етапом для розуміння потреб та вимог користувачів. Нижче наведено декілька аспектів, які варто врахувати при проведенні аналізу даної предметної області:

1. *Цільова аудиторія* – Важливо визначити цільову аудиторію застосунку, а саме нумізматів і колекціонерів. Аналізуючи цю групу користувачів, необхідно зрозуміти їх потреби, очікування та проблеми, з якими вони зіштовхуються при управлінні своїми колекціями.

2. *Основні функціональні вимоги* – Визначення основних функціональних вимог застосунку є важливим кроком. Це можуть бути такі функції, як додавання і каталогізація нових предметів, відстеження деталей про кожен предмет, оцінка вартості, створення колекцій з фотографіями предметів, обмін інформацією з іншими користувачами, аналіз статистики і т.д.

3. *Інтерфейс користувача* – Зручний та привабливий інтерфейс користувача є важливим елементом мобільного застосунку. Важливо дослідити та проаналізувати стандарти дизайну мобільних застосунків, рекомендації щодо розміщення елементів, навігацію, використання кольорів та шрифтів, щоб забезпечити зручність використання та задоволення користувачів.

4. *Інтеграція з іншими сервісами* – Можливості інтеграції застосунку з іншими сервісами, такими як онлайн-аукціони, бази даних нумізматичних даних, інформаційні ресурси, соціальні мережі тощо. Це може полегшити доступ до додаткової інформації та розширити можливості взаємодії користувачів.

5. *Безпека та конфіденційність* – Забезпечення безпеки та конфіденційності інформації користувачів є критичним аспектом [2]. Потрібно аналізувати можливості шифрування даних, автентифікації користувачів, забезпечення резервного копіювання даних та інших механізмів, щоб гарантувати захист даних користувачів.

6. *Технологічні аспекти* – Технологічні аспекти розробки мобільного застосунку, такі як вибір платформи (iOS, Android або обидві), використання фреймворків (React Native, Flutter, тощо), інтеграція з БД та серверними API [9], адаптивний дизайн для різних пристроїв та інші технічні вимоги.

Аналіз предметної області мобільного застосунку для нумізматів і колекціонерів допоможе визначити основні вимоги та напрямки розробки застосунку. Важливо враховувати потреби цільової аудиторії та надати їм зручні, функціональні та безпечні рішення для управління їх колекціями.

1.3 Застосунки конкурентів

На ринку існує декілька мобільних застосунків для нумізматів, які пропонують різні функціональні можливості. Ось декілька популярних мобільних застосунків для нумізматів:

1. Застосунок “Numista” – переваги: велика база даних монет, можливість додавання та керування колекцією, обмін інформацією з іншими нумізматами, функція відстеження прогресу колекціонування; недоліки: деякі користувачі зазначають, що інтерфейс може бути складним для новачків, а також бажання більшої актуалізації деяких даних (рис. 1.1).

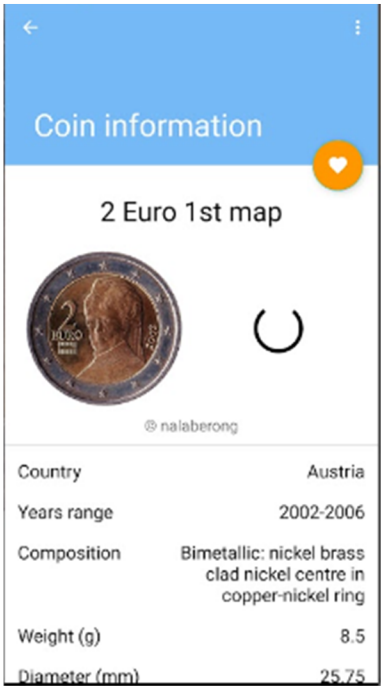


Рис. 1.1. Інтерфейс застосунку “Numista”

2. Застосунок “Coin Catalog” – переваги: велика база даних монет, можливість створювати власні каталоги, функція пошуку та фільтрації, зручний інтерфейс; недоліки: деякі користувачі вказують на недостатню актуалізацію даних та можливість відсутності деяких менш популярних монет (рис. 1.2).

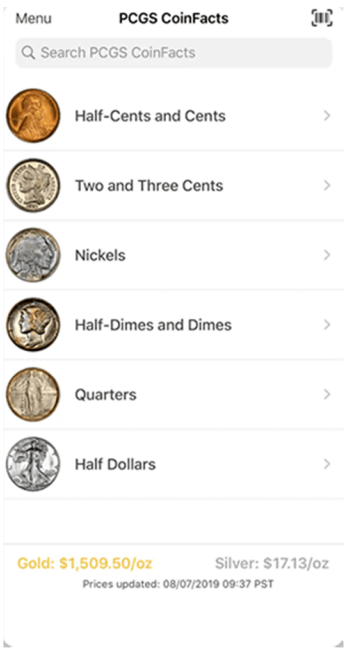


Рис. 1.2. Інтерфейс застосунку “Coin Catalog”

3. Застосунок “Coinoscope Pro” – переваги: функція розпізнавання монет за фотографією, доступ до інформації про монети, можливість каталогізувати колекцію; недоліки: обмежена база даних, не завжди точне розпізнавання монет, можливість пропущення деяких рідкісних монет (рис. 1.3).

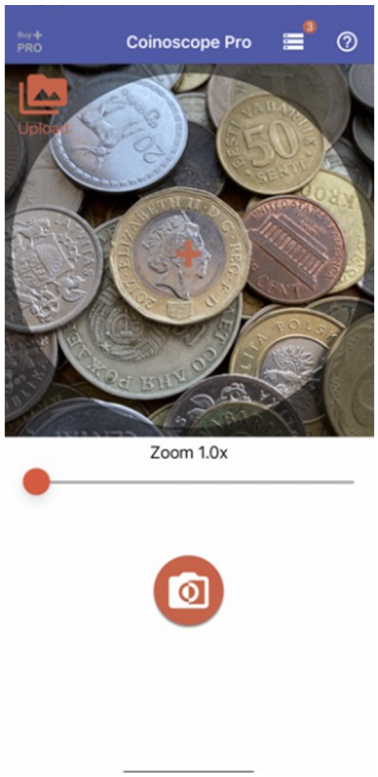


Рис. 1.3. Інтерфейс застосунку “Coinoscope Pro”

1.4. Вибір операційної системи для розробки застосунку

Операційна система (ОС) – це програмне забезпечення, яке управляє роботою комп’ютера або мобільного пристрою. Вона забезпечує взаємодію між апаратними компонентами пристрою та програмними застосунками, дозволяючи їм працювати разом [5].

ОС Android стала дуже популярною і має велику базу користувачів та розробників. Вона надає широкі можливості для розробки застосунків та персоналізації пристроїв, що робить її привабливим вибором для багатьох користувачів та розробників [24]. Android – це операційна система, розроблена компанією Google, спеціально для мобільних пристроїв, таких як смартфони та

планшети. ОС Android є відкритою платформою, що означає, що розробники можуть змінювати та адаптувати її під свої потреби.

Основні характеристики системи Android:

Відкритість – ОС Android базується на відкритих стандартах і забезпечує розширені можливості для розробників. Це означає, що розробники можуть створювати різноманітні застосунки та модифікації для покращення функціональності пристроїв.

Широка платформа – Android працює на різних пристроях, включаючи смартфони, планшети, телевізори, годинники і навіть автомобільні системи. Це дозволяє розробникам створювати застосунки, які працюють на багатьох різних пристроях.

Багата екосистема застосунків – Google Play Store, офіційний магазин застосунків для Android, має велику їх кількість у різних категоріях, що дає користувачам широкі можливості для розширення функціональності своїх пристроїв [7].

Персоналізація – Android надає користувачам широкі можливості налаштування свого пристрою, включаючи зміну розміщення елементів на екрані, вибір різних тем і налаштування сповіщень.

Висновки. В першому розділі було проаналізовано стан проблемної області та існуючі застосунки, їх переваги і недоліки, які буде враховано при розробленні мого власного рішення.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вибір середовища для розробки

Вибір середовища для розробки на Angular та Node.js може бути питанням особистих вподобань та командних потреб. Вибір текстового редактора Visual Studio Code (VS Code) для розробки на Angular та Node.js визначається кількома ключовими факторами, які роблять його найкращим вибором серед інших існуючих засобів розроблення.

Visual Studio Code надає широкий спектр розширень, які дозволяють розробникам налаштовувати своє робоче середовище відповідно до їх конкретних потреб [21]. Сюди входять розширення для підтримки певних фреймворків, інструменти для роботи з БД та інтеграції з іншими службами. Гнучкість у виборі розширень робить VS Code інструментом, який може адаптуватися до різних завдань розробки.

VS Code також пропонує вбудовану підтримку TypeScript та JavaScript, включаючи потужний аналіз коду та автоматичні доповнення. Це підвищує продуктивність розробки на Angular і допомагає уникнути потенційних помилок у коді. Інтеграція VS Code з Angular CLI дозволяє розробникам ефективно взаємодіяти з проєктами Angular. Команди редактора та користувальницького інтерфейсу (рис. 2.1) полегшують створення, редагування та управління проєктами Angular, забезпечуючи зручний робочий процес.

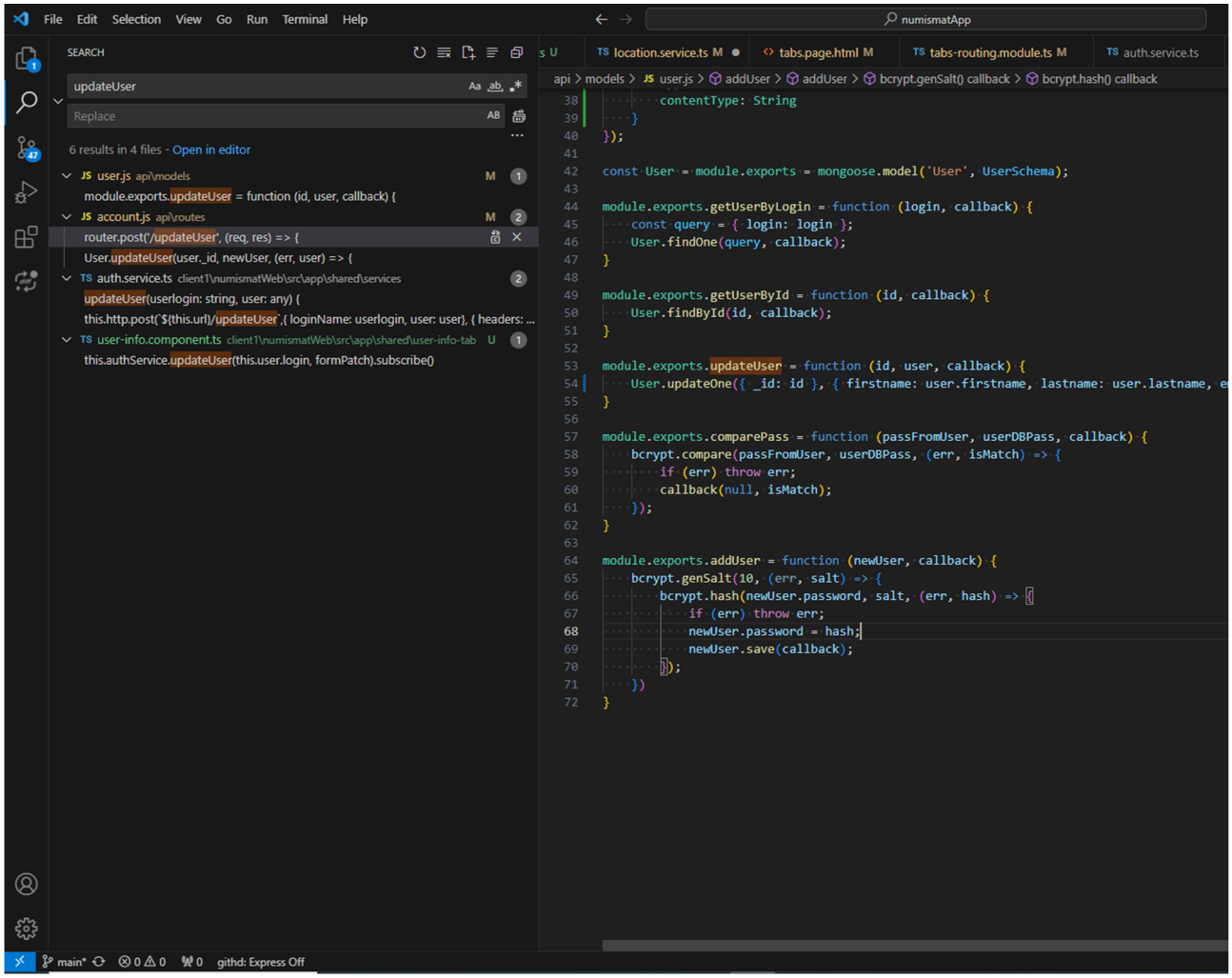


Рис. 2.1. Інтерфейс редактора VS Code

VS Code підтримує використання локального сервера для автоматичного оновлення змін у режимі реального часу (live server). Це особливо важливо при розробці інтерфейсу, оскільки можна бачити зміни в режимі реального часу без перезавантаження сторінки. Це полегшує налагодження та керування інтерфейсом. Завдяки вбудованій підтримці Git в VS Code можна зручно управляти системою управління версіями, не виходячи з редактора [17]. Це включає можливість перегляду стану змін, комітів, гілок та конфліктів злиття.

Даний редактор включає вбудований термінал, який дозволяє виконувати команди безпосередньо з редактора. Це може допомогти виконувати різні операції, такі як запуск сервера, встановлення залежностей та виконання інших команд. Вбудований налагоджувач для Node.js у VS Code дозволяє ефективно

налагоджувати серверний код. Надає можливість встановлювати точки зупинки, аналізувати стек викликів і переглядати значення змінних під час налагодження.

Однією з головних переваг Visual Studio Code є наявність великої та активної спільноти розробників, що забезпечує швидке виявлення та виправлення помилок, а також регулярні оновлення з новими функціями та вдосконаленнями. Це підвищує довіру до редактора та його довгострокову актуальність, що важливо для успішної розробки на Angular та використання функцій у Node.js.

2.2 Вибір мови програмування та фреймворків

Ionic з Angular – це комбінація фреймворків для розробки гібридних мобільних застосунків з використанням мови TypeScript [25].

TypeScript – це мова програмування, яка є розширенням JavaScript, розроблена компанією Microsoft. TypeScript надає статичну типізацію, що дозволяє визначати типи для змінних, аргументів функцій та інших елементів коду під час його написання [16]. Це допомагає виявляти помилки на етапі розробки та полегшує рефакторинг коду. TypeScript підтримує сучасні стандарти ECMAScript та надає можливість використання нововведень JavaScript, таких як стрілкові функції, async/await, і т.д.

Поєднання Angular та Ionic дає потужну комбінацію для розробки застосунків та вебслужб. Angular служить основою для створення бізнес-логіки та структури застосунків, тоді як Ionic надає готові мобільні компоненти та інструменти для створення користувальницьких інтерфейсів та взаємодії з пристроями. Як Ionic, так і Angular роблять процес розробки гібридних мобільних застосунків більш продуктивним і дієвим, забезпечуючи швидку розробку і підтримку для різних платформ.

Вибір фреймворків Ionic з Angular для розробки клієнтської частини Android застосунку та Node.js з Express.js для створення його серверної частини може бути дуже обгрунтованим з кількох причин.

Використання Ionic з Angular для клієнтської частини забезпечує:

1. *Кросплатформність*. Ionic побудований на базі веб-технологій, що дозволяє створювати кросплатформні застосунки, які працюють як на Android, так і на iOS [3]. Використання Angular у поєднанні з Ionic робить можливим швидке створення мобільних застосунків з загальним кодом.

2. *Сучасний та стильний дизайн*. Ionic пропонує елементи дизайну, які відповідають матеріальному дизайну Google, що дозволяє створювати стильні та сучасні дки для Android. Це сприяє поліпшенню користувацького досвіду.

3. *Широкий вибір плагінів*. Ionic має розширювану систему плагінів, яка дозволяє розробникам легко інтегрувати різні функції, такі як камера, інформація про місце знаходження та push-сповіщення, що полегшує розробку функціональних застосунків.

4. *Оптимізація для мобільних пристроїв*. Ionic оптимізований для мобільних пристроїв, забезпечуючи високий рівень продуктивності та ефективність роботи на Android-платформі [10].

Node.js з Express.js для серверної частини забезпечує:

1. *Швидкість та легкість розробки*. Node.js це – платформа, яка побудована на асинхронному програмуванні, що забезпечує швидку та ефективну взаємодію між клієнтською та серверною частинами програми. Проста структура Express.js робить розробку серверної частини простою і ефективною.

2. *RESTful API*. Express.js полегшує створення архітектури RESTful API, яка є ефективною та простою у взаємодії з клієнтськими програмами. Це забезпечує ефективний обмін даними між клієнтом і сервером [20].

3. *Широкі можливості розширення*. Node.js має велику кількість модулів і бібліотек, що дозволяють розробникам швидко розширювати функціональність серверної частини в міру необхідності [18].

4. *Real-time взаємодія*. Використання Node.js та Express.js дозволяє легко реалізувати взаємодію в режимі реального часу і особливо корисно для

застосунків, які потребують негайної передачі даних, таких як чати та оновлення в режимі реального часу.

Загальні переваги використання Angular з Ionic та Node.js з Express:

1. *Однорідні технології.* Використання Angular разом з Ionic на клієнтській стороні і Node.js з Express.js на сервері дозволяє створити застосунок, використовуючи однорідні технології, що полегшує командну роботу та утримання кодової бази.

2. *Підтримка та документація*. Як Angular, так і Node.js мають широку та активну спільноту розробників, а також детальну та актуальну документацію, що сприяє ефективній розробці та вирішенню проблем.

3. *Масштабованість*. Node.js і Express.js легко масштабуються, що дозволяє розробникам збільшувати ресурси та виробничі можливості застосунку при необхідності [15].

Об'єднуючи Ionic з Angular та Node.js з Express.js, ми отримуємо зручний та ефективний стек технологій для створення мобільних застосунків для Android з покращеним користувацьким досвідом та ефективною серверною частиною.

2.3 Вибір СКБД бази MongoDB

MongoDB – це документно-орієнтована система керування базами даних типу NoSQL [19], яка стала широко відомою серед розробників своєю гнучкістю та ефективністю. Ось основні причини, чому MongoDB підходить для мого проєкту:

Документо-орієнтована модель даних MongoDB відображає об'єкти застосунку більш природним чином, оскільки дані зберігаються у вигляді документів у форматі BSON (бінарна серіалізація JSON) [22]. MongoDB легко масштабується горизонтально, що дозволяє додавати нові сервери для обробки більших обсягів даних та трафіку. Це особливо важливо для проєктів, які потребують високої доступності та масштабованості. Мова запитів MongoDB

схожа на SQL, що полегшує взаємодію з базою даних. Можна використовувати різноманітні запити для отримання необхідної інформації з БД.

MongoDB характеризується гнучкістю схеми даних. Документи з різною структурою можуть зберігатися в одній колекції, що полегшує зміну схеми даних у міру зміни вимог проєкту [8].

MongoDB має потужні можливості агрегування, які дозволяють ефективно обробляти та аналізувати дані в БД. Це корисно при виконанні різних операцій з обробки та аналізу даних. Вона також має підтримку великої та активної спільноти розробників, що робить її надійним вибором для проєктів, які потребують підтримки та розвитку.

Також MongoDB можна використовувти разом з графічним клієнтом MongoDB Compass [11]. MongoDB Compass – це офіційний графічний інтерфейс для роботи з СКБД MongoDB (рис. 2.2). Він призначений для полегшення роботи з БД, їх візуалізації та адміністрування. Загальна зручність використання MongoDB Compass полягає в тому, що він надає інструменти для роботи з MongoDB без необхідності написання складних команд або використання консолі. Це особливо корисно для розробників та адміністраторів БД, які шукають зручний та візуальний інтерфейс для взаємодії з MongoDB.

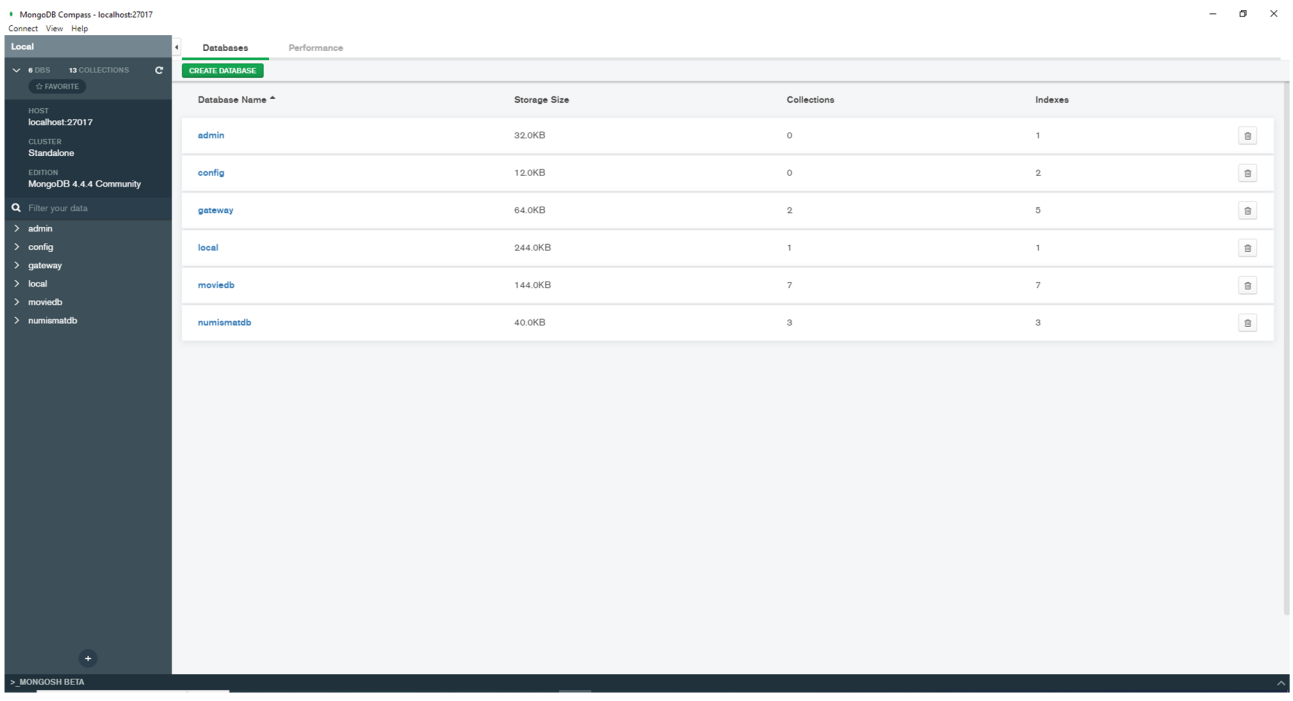


Рис. 2.2. Інтерфейс MongoDB Compass

MongoDB Compass дозволяє візуально працювати з даними колекцій, можливість переглядати та редагувати документи за допомогою інтуїтивно зрозумілого інтерфейсу та легко взаємодіяти зі своїми даними, не використовуючи команди MongoDB безпосередньо [4]. Він також відображає структуру БД, що містить інформацію про її індекси та полегшує аналіз та оптимізацію запитів. Також існує можливість додавати нові індекси, використовуючи простий інтерфейс [12].

Висновки. В даному розділі проаналізовано інструменти для розробки застосунків, в результаті чого обрано мову TypeScript з використанням фреймворків Ionic та Angular для клієнтської частини застосунку, а також Node.js та Express.js для написання його серверної частини. В якості середовища для розробки застосунку та написання коду обрано редактор Visual Studio Code, який надає зручний інструментарій для написання та відлагодження коду.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Шифрування паролів з використанням Bcrypt

BCrypt (Blowfish Crypt) – хеш-функція, яку зазвичай використовують для хешування (шифрування) паролів. BCrypt ґрунтується на алгоритмі шифрування Blowfish і використовує робочий фактор Work Factor для ускладнення обчислень і зниження швидкості атак методом перебору [21].

Основні особливості BCrypt:

1. *Сіль (salt).* BCrypt містить автоматично згенеровану "сіль" (випадково згенерований рядок), яку додають до пароля перед його хешуванням. Це ускладнює атаки з використанням Rainbow Table [13].

2. *Робочий фактор (Work Factor)* – це параметр, що визначає кількість обчислень необхідних для генерації хешу. Зазвичай обирається значення, що забезпечує прийнятний рівень безпеки та продуктивності.

3. *Секретність алгоритму.* Алгоритм BCrypt розроблено таким чином, щоб бути стійким до різних типів атак, включно з атаками, спрямованими на прискорення обчислень, наприклад, з використанням спеціалізованих обчислювальних пристроїв.

4. *Криптографічна стійкість.* Важливою частиною математичного забезпечення є криптографічна стійкість алгоритму. Це означає, що якщо при атаці зловмисникові навіть відомі всі параметри хешу ("сіль", робочий фактор), знаходження вихідних даних (пароля) повинно бути обчислювально складним завданням [1].

BCrypt заснована на хеш-алгоритмі і Blowfish-блоковому шифрі. Використовуючи математичні операції, як-от заміна і перестановка бітів, Blowfish перетворює вхідні дані (пароль і "сіль") на хеш-значення фіксованої довжини (рис. 3.1).

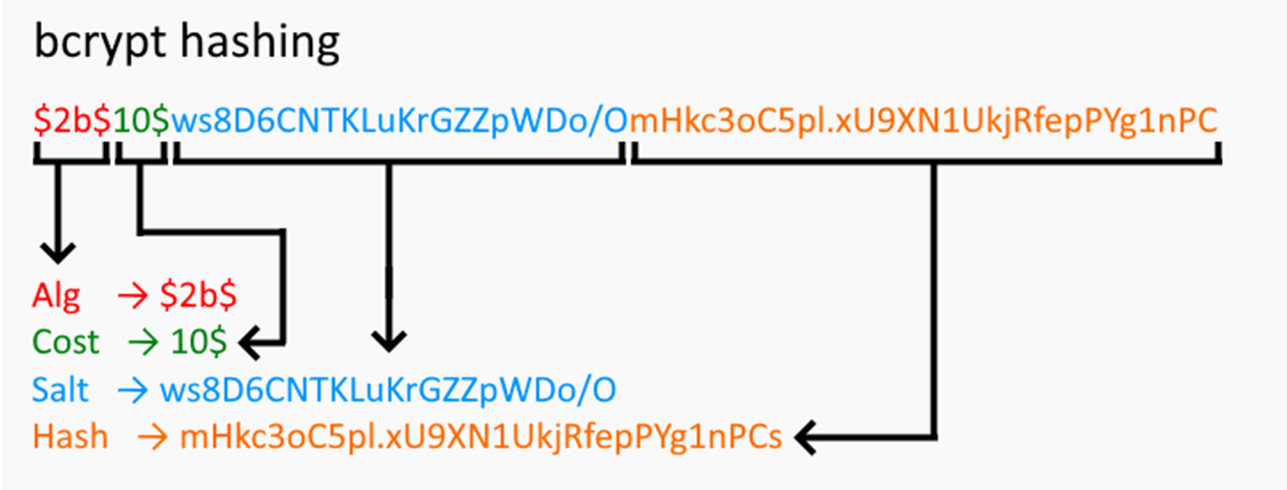


Рис. 3.1. Структура хешування

Важливим аспектом хешування є використання “солі” – випадково згенерованої послідовності бітів. Це додає ентропію в процес хешування і ускладнює атаки з використанням заздалегідь обчислених таблиць (rainbow tables) [14]. Робочий фактор визначає вартість генерації хеша. Це поняття використовується для ускладнення обчислень і уповільнення атак на паролі методом перебору. Збільшення робочого фактора ускладнює використання спеціалізованих обчислювальних пристроїв для атаки на паролі.

BCrypt розроблено з використанням алгоритму налаштування ключів Eksblowfish, який забезпечує високий ступінь безпеки зберігання паролів. Використання BCrypt дає змогу уникнути багатьох типів атак, включно з атаками методом перебору атаками за заздалегідь обчисленими таблицями.

3.2 Шифрування даних за допомогою bcrypt.js

Bcrypt.js – це бібліотека для шифрування паролів у Node.js, що використовує алгоритм bcrypt. Цей алгоритм використовується для зберігання паролів у вигляді надійного хешу, що робить їх важко доступними для взлому. Встановлюється bcrypt командою npm install bcrypt. Дана бібліотека дозволяє легко створювати хеші паролів і порівнювати їх.

Створення хеша пароля виглядає наступним чином:

```
const bcrypt = require('bcrypt');
```

```
const plainTextPassword = 'mySecurePassword';
// Генеруємо “сіль” (salt)
const saltRounds = 10;
bcrypt.genSalt(saltRounds, function(err, salt) {
  if (err) {
    // Обробка помилки
    return console.error(err);
  }
  // Хешуємо пароль за допомогою “солі”
  bcrypt.hash(plainTextPassword, salt, function(err, hash) {
    if (err) {
      // Обробка помилки
      return console.error(err);
    }
    // Тепер 'hash' містить захешований пароль
    console.log('Hashed Password:', hash);
  });
});
```

Для генерації “солі” використовуємо функцію bcrypt.genSalt(). “Сіль” додається до пароля перед хешуванням, що робить атаки “пошукового простору” більш складними. Генерація випадкової “солі” є важливим елементом безпеки. “Сіль” ускладнює роботу атакам злому, оскільки однакові паролі матимуть різні хеші.

Для хешування пароля за допомогою отриманої солі використовуємо функцію bcrypt.hash(). Результат (хеш) можна зберегти у базі даних.

Кількість раундів (saltRounds) визначає, наскільки складно буде зламати хеш. Використання більшої кількості раундів робить процес більш витратним для обчислення, забезпечуючи вищий рівень безпеки.

Перевірка пароля здійснюється наступним чином:

```
const hashedPassword =
'$2b$10$S5CJ8OwKBMvl9AjAqkF5pOzH3gXNd2m0PVFDBVi9dFZ1bjRtIeKlG';
    bcrypt.compare('mySecurePassword', hashedPassword, function(err, result) {
      if (err) {
        // Обробка помилки порівняння
        return console.error('Error comparing passwords:', err);
      }
      if (result) {
        console.log('Password is correct!');
      } else {
        console.log('Password is incorrect!');
      }
    });
```

Функція bcrypt.compare() використовується для порівняння введеного користувачем пароля із збереженим хешем у БД. Якщо паролі співпадають, результат буде true, в іншому випадку – false. Якщо при порівнянні виникне помилка, її також необхідно обробляти, щоб забезпечити якісну практику управління помилками.

Висновки. У цьому розділі подано математичне забезпечення для надійного шифрування паролів. Подано фрагменти кодів, згідно яких шифруються дані та паролі на основі хеш-функції Bcrypt та бібліотеки bcrypt.js з урахуванням загальних принципів для забезпечення високої ступені безпеки.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Структура бази даних системи

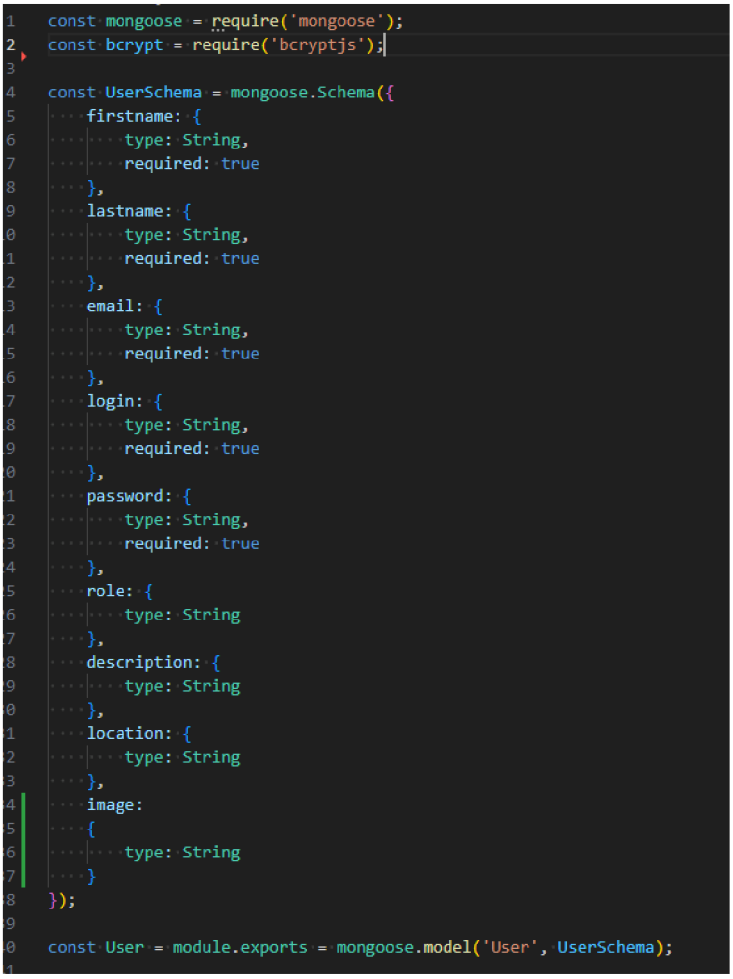
База даних (БД) необхідна для зберігання інформації протягом тривалого періоду часу, незалежно від сеансу роботи користувача. В БД зберігається інформація про зареєстрованих користувачів, їхні колекції, чати, коментарі та вподобання. Для цього потрібно створити таблиці (колекції) для кожного елементу системи у СКБД MongoDB за допомогою бібліотеки mongoose для Node.js. Для цього використовуємо схеми (Schema) та моделі (Model). Для коментарів було створено CommentsSchema та Comments модель (рис. 4.1), для таблиці з користувачами – UserSchema та User модель (рис. 4.2), для чату – ChatSchema та Chat модель (рис. 4.3), для колекцій – CollectionSchema та Collections модель (рис. 4.4).

```
const mongoose = require('mongoose');

const CommentsSchema = mongoose.Schema({
    comment: {
        type: String,
        required: true
    },
    user: {
        type: String,
        required: true
    },
    collectionId: {
        type: String,
        required: true
    },
    date: {
        type: String,
        required: true
    },
    likes: {
        type: Array,
        required: true
    }
});

const Comments = module.exports = mongoose.model('Comments', CommentsSchema);
```

Рис. 4.1. Фрагмент коду CommentsSchema



```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = mongoose.Schema({
    firstname: {
        type: String,
        required: true
    },
    lastname: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    login: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    role: {
        type: String
    },
    description: {
        type: String
    },
    location: {
        type: String
    },
    image:
    {
        type: String
    }
});

const User = module.exports = mongoose.model('User', UserSchema);
```

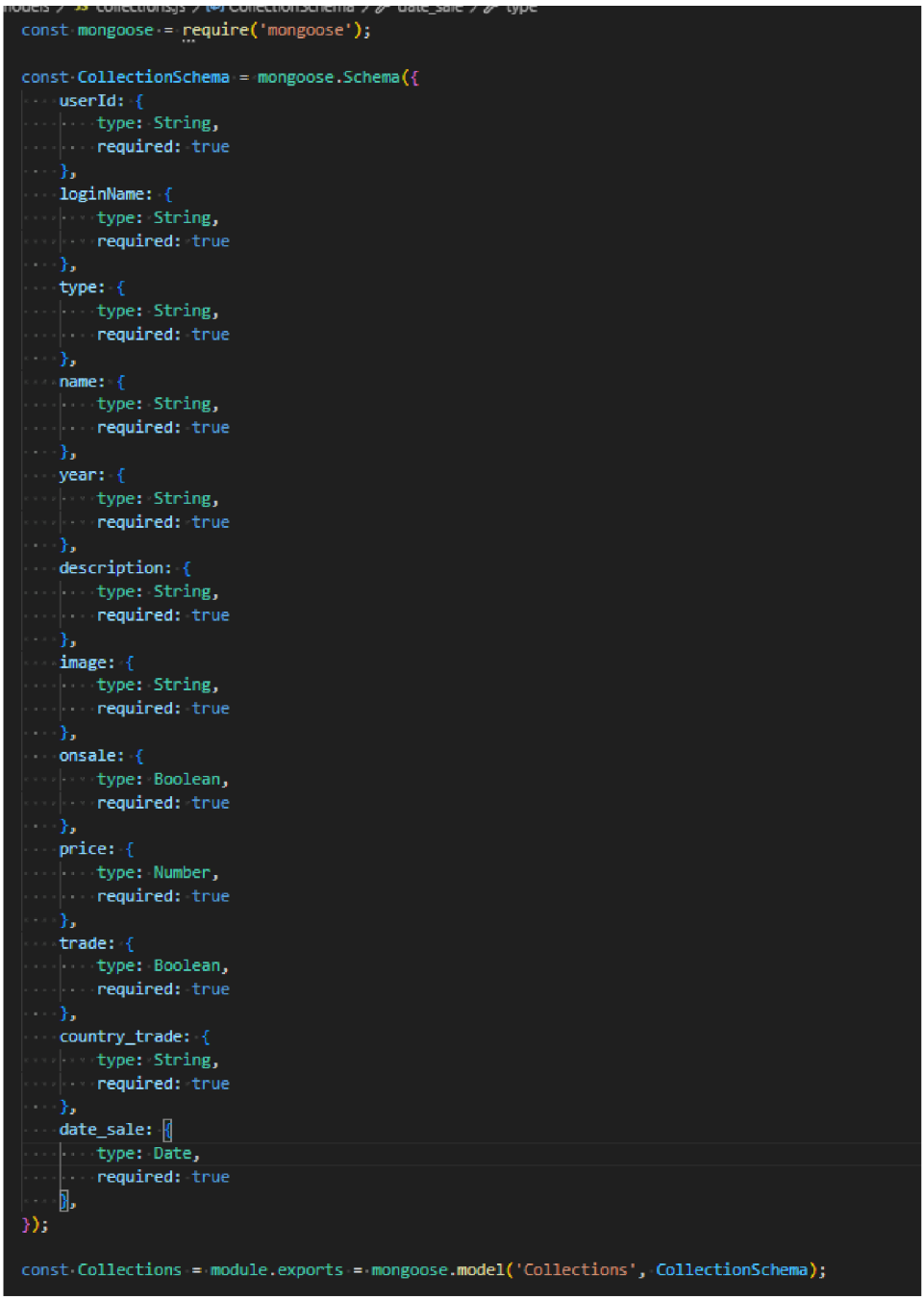
Рис. 4.2. Фрагмент коду UserSchema

```
const mongoose = require('mongoose');

const ChatSchema = mongoose.Schema({
    comment: {
        type: String,
        required: true
    },
    chatId: {
        type: Number,
        required: true
    },
    collectionId: {
        type: Number,
        required: true
    },
    userId: {
        type: Number,
        required: true
    },
    date: {
        type: Date,
        required: true
    }
});

const Chat = module.exports = mongoose.model('Chat', ChatSchema);
```

Рис. 4.3. Фрагмент коду ChatSchema



```
const mongoose = require('mongoose');

const CollectionSchema = mongoose.Schema({
    userId: {
        type: String,
        required: true
    },
    loginName: {
        type: String,
        required: true
    },
    type: {
        type: String,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    year: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    image: {
        type: String,
        required: true
    },
    onsale: {
        type: Boolean,
        required: true
    },
    price: {
        type: Number,
        required: true
    },
    trade: {
        type: Boolean,
        required: true
    },
    country_trade: {
        type: String,
        required: true
    },
    date_sale: {
        type: Date,
        required: true
    },
});

const Collections = module.exports = mongoose.model('Collections', CollectionSchema);
```

Рис. 4.4. Фрагмент коду CollectionSchema

Для того щоб переглянути вміст усіх колекцій БД (рис. 4.6), потрібно спочатку зайти в MongoDb Compass та під’єднатись до сервера (рис. 4.5).

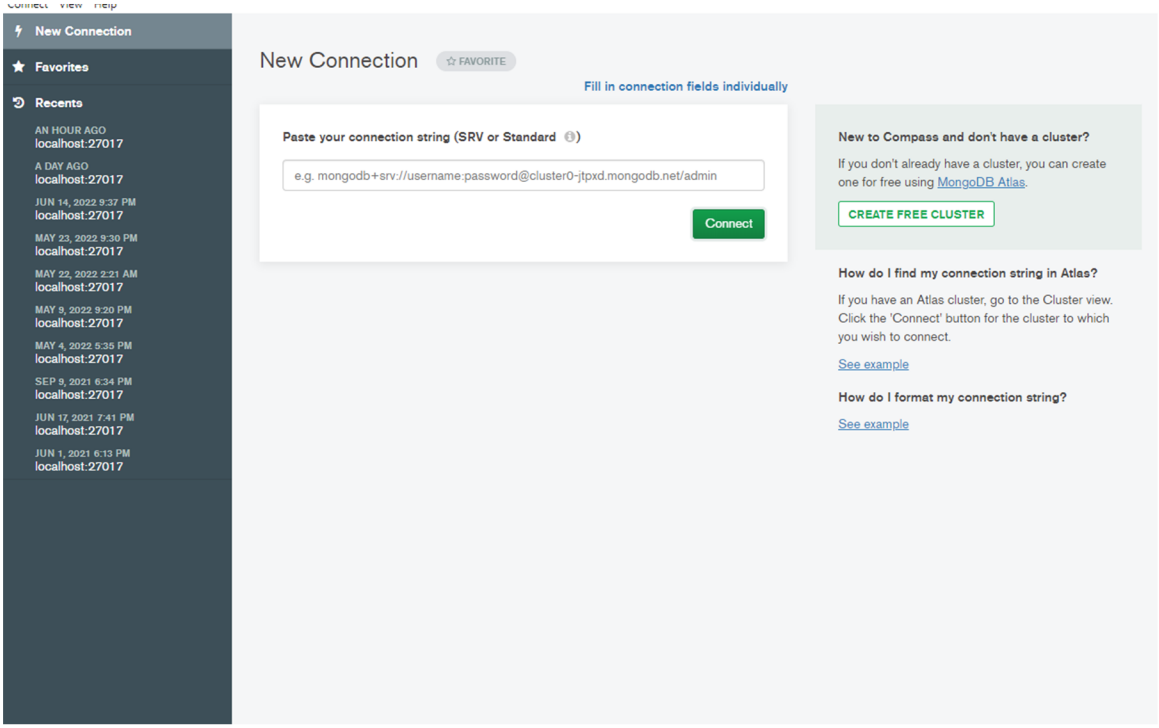


Рис. 4.5. Вікно під’єднання до сервера

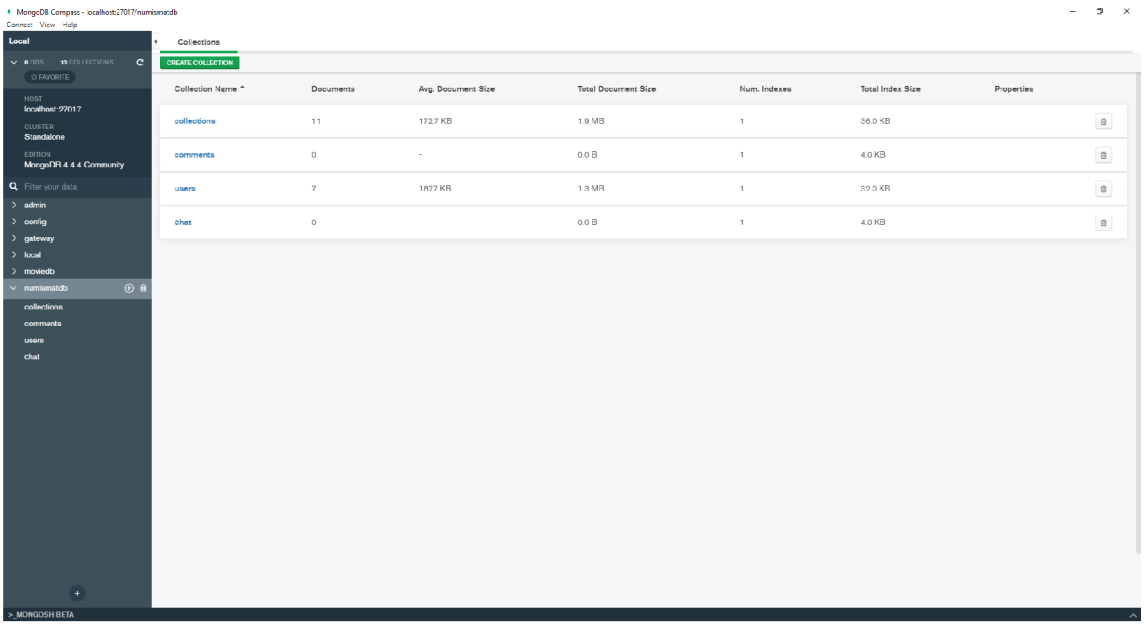


Рис. 4.6. Вікно колекцій БД

Після створення моделей і схем в БД numismatdb було додано наступні колекції:

- comments – для коментарів;
- collections – для одиниць колекції;
- users – для інформації про користувачів;

● chat – для чату.

На рисунку 4.7 представлено діаграму колекцій БД numismatdb.

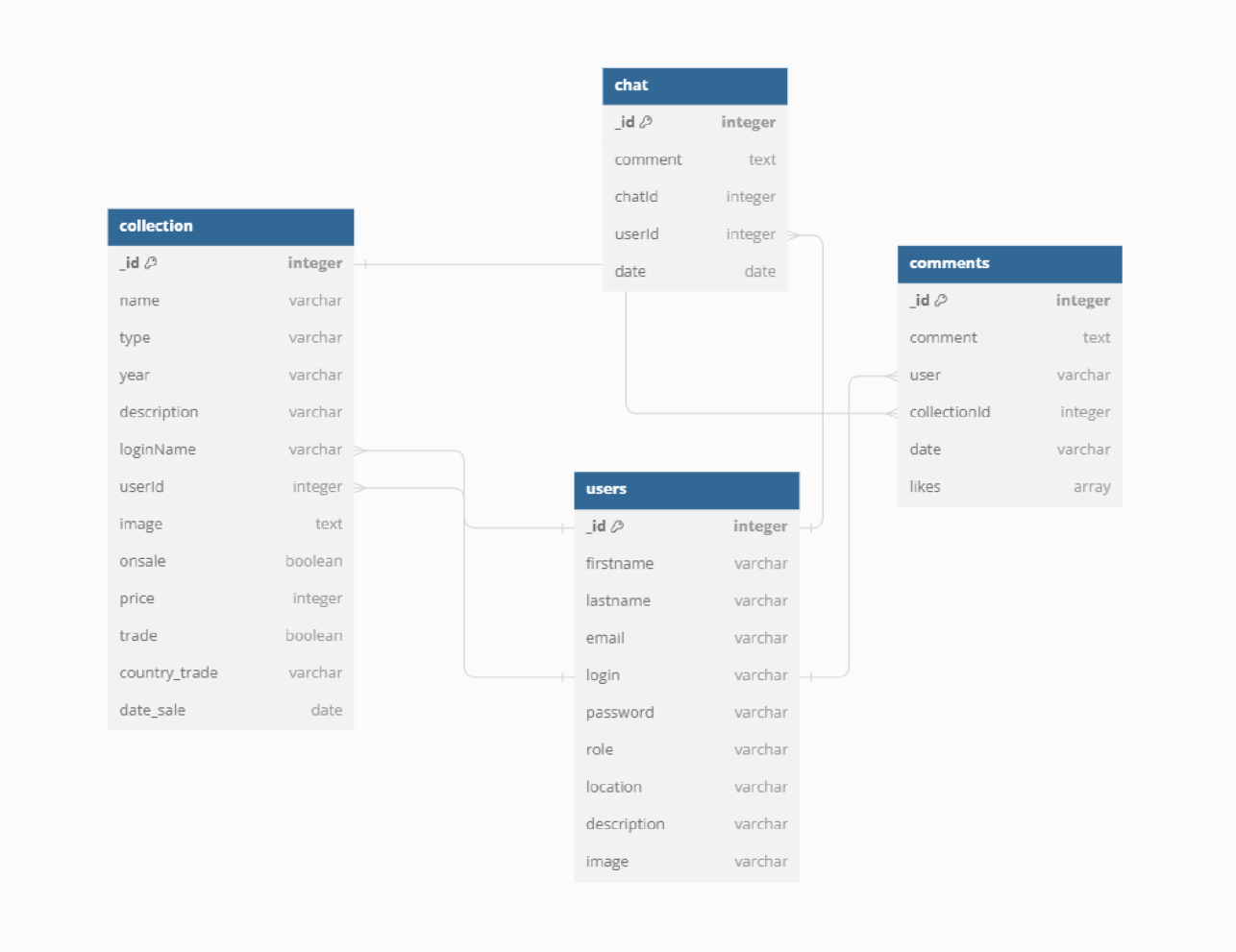


Рис. 4.7. Діаграма колекцій БД numismatdb

4.2 Розроблення основної логіки застосунку

Основна логіка застосунка – це набір інструкцій, правил та операцій, які визначають, як застосунок повинен функціонувати. Це визначено в коді програмного забезпечення та врегулювано всі аспекти взаємодії з користувачем та обробкою даних. Зазвичай, основна логіка застосунку будується на основі бізнес-вимог та функціональних вимог до продукту. Оскільки я використовую Angular з Ionic, бізнес-логіка має бути описана в модулях, компонентах і сервісах. Перелік основних модулів представлено на рисунку 4.8.

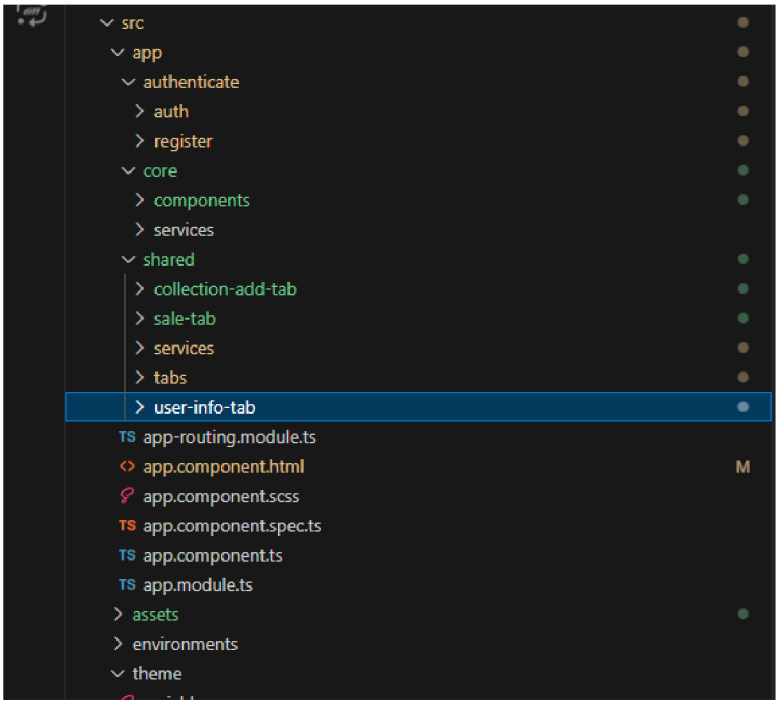


Рис. 4.8. Перелік основних модулів проєкту

Призначення кожного модуля на фронтенд частині є наступним:

1. AppModule – є кореневим модулем застосунку. Angular використовує AppModule як точку входу та розглядає дерево компонентів та залежностей, описане в цьому модулі, для побудови та ініціалізації застосунку;

2. AuthModule – відповідає за авторизацію користувачів;

3. RegisterModule – відповідає за регістрацію нових користувачів;

4. TabsModule – відповідає за вкладки застосунку та переходи між ними;

5. UserInfoTabModule – відповідає за вкладку профілю користувача та його колекцію;

6. CollectionAddTabModule – відповідає за вкладу додавання нової одиниці в колекцію;

7. SaleTabModule – відповідає за вкладку продажів і обмінів колекції.

На бекенді основна логіка часто розділена між моделями (Models) і роутами (Routes). Це дозволяє організувати код застосунку, полегшити його розширення та підтримку. Загальна структура бекенду застосунку представлена на рисунку 4.9.

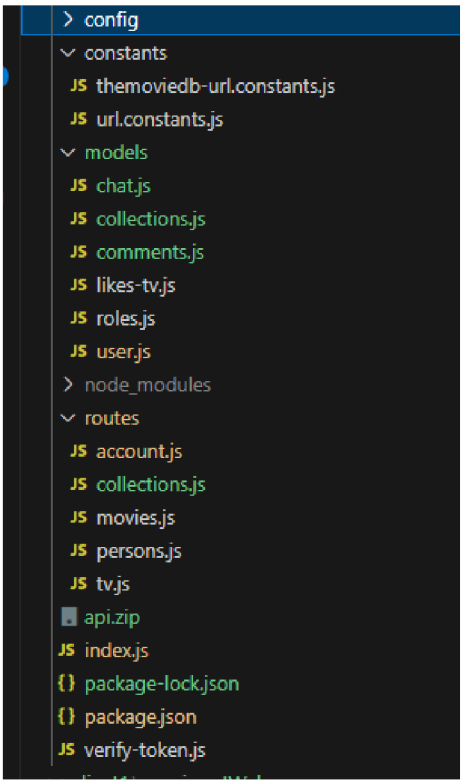


Рис. 4.9. Загальна структура бекенду застосунку

Призначення моделей є наступним:

1. *chat.js* – набір методів і модель для роботи чату;

2. *collections.js* – набір методів і модель для роботи з колецією і обробкою кожної одиниці;

3. *comments.js* – набір методів і модель для обробки коментарів користувачів;

4. *user.js* – набір методів і модель для реєстації і авторизації користувачів.

В user.js важливими методами є updateUserPassword() і comparePass():

```
module.exports.updateUserPassword = function (id, passwords, callback) {
    bcrypt.genSalt(10, (err, salt) => {
        bcrypt.hash(passwords.newPassword, salt, (err, hash) => {
            if (err) throw err;
```

```
            passwords.newPassword = hash;
            User.updateOne({ _id: id }, { password: passwords.newPassword },
callback);
        });
    });
}
module.exports.comparePass = function (passFromUser, userDBPass,
callback) {
    bcrypt.compare(passFromUser, userDBPass, (err, isMatch) => {
        if (err) throw err;
        callback(null, isMatch);
    });
}
```

Метод updateUserPassword() створенний для оновлення пароля користувача в БД. “Сіль” генерується за допомогою методу genSalt(). Метод hash() хешує новий пароль з використанням цієї “солі”. В методі updateOne(), який оновлює пароль користувача в БД, в якості параметрів використовується: id – ідентифікатор користувача, чий пароль потрібно оновити; passwords – об’єкт, що містить новий пароль (newPassword); callback – функція зворотнього виклику, яка викликається після успішного оновлення пароля або виникнення помилки.

Метод comparePass() призначений для порівняння пароля, введеного користувачем, із збереженим хешем пароля в БД. В ньому використовується для порівняння хешів паролів метод compare(), параметрами якого є: passFromUser – пароль, введений користувачем; userDBPass – хеш пароля, який зберігається в БД; callback – функція зворотнього виклику, яка повертає результат порівняння у вигляді булевого значення.

4.3 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача (UI) – це важлива частина процесу

створення програмного забезпечення, включаючи мобільні застосунки, вебсайти, адміністративні панелі тощо. Інтерфейс користувача повинен бути зручним та естетично приємним для користувача. Розробка інтерфейсу в Ionic включає в себе використання HTML, CSS і TypeScript для створення компонентів та розміщення їх у структурі застосунку. Ionic базується на веб-технологіях і використовує Angular для створення масштабованих та легко управляючих користувацьких інтерфейсів і має багато готових компонентів. Розглянемо це на прикладі фрагмента коду register.component.html (рис 4.10).

```
                label="Ім'я"
                labelPlacement="stacked"
                [clearInput]="true"
                name="firstname"
                [(ngModel)]="firstname"
                placeholder="Введіть ім'я">
            </ion-input>
        </ion-item>
        <ion-item>
            <ion-input class="input-box"
                label="Прізвище"
                labelPlacement="stacked"
                [clearInput]="true"
                name="lastname"
                [(ngModel)]="lastname"
                placeholder="Введіть прізвище">
            </ion-input>
        </ion-item>
        <ion-item>
            <ion-input class="input-box"
                label="Емейл"
                labelPlacement="stacked"
                [clearInput]="true"
                name="email"
                [(ngModel)]="email"
                placeholder="Введіть емейл">
            </ion-input>
        </ion-item>
        <ion-item>
            <ion-input class="input-box"
                label="Пароль"
                labelPlacement="stacked"
                [clearInput]="true"
                name="password"
                type="password"
                [(ngModel)]="password"
                placeholder="Введіть пароль">
            </ion-input>
        </ion-item>
        <ion-button class="login-button"
                type="submit"
                expand="block">Створити профіль</ion-button>
        <ion-button routerLink="/login"
                fill="clear">На сторінку авторизації</ion-button>
    </ion-list>
</form>
<ion-toast [isOpen]="isToastOpen"
            [message]="notifitcationText"
            [duration]="5000"
            (didDismiss)="setOpen(false)"></ion-toast>
</div>
```

Рис. 4.10. Фрагмент коду register.component.html

Основними компонентами та їх призначеннями є:

- <ion-list> – контейнер для групи елементів, які відображаються в списку;

- <ion-item> – компонент Ionic для створення елемента списку;
- <ion-input> – компонент для введення тексту;
- <ion-button> – кнопка для взаємодії користувача;
- <ion-toast> – компонент Toast для відображення повідомлень;
- (submit)=”userRegisterClick()” – обробник подій, який викликає функцію userRegisterClick() при відправці форми;
- [(ngModel)] – двонапрямковий зв’язок для зв’язування даних між HTML та TypeScript (Angular).

Ці компоненти є основою для форми реєстрації в Ionic. Для повного функціоналу також потрібно реалізувати логіку обробки подій та взаємодії з сервером. Дотримано також принципів дизайну для забезпечення кращого користувацького досвіду (рис 4.11).

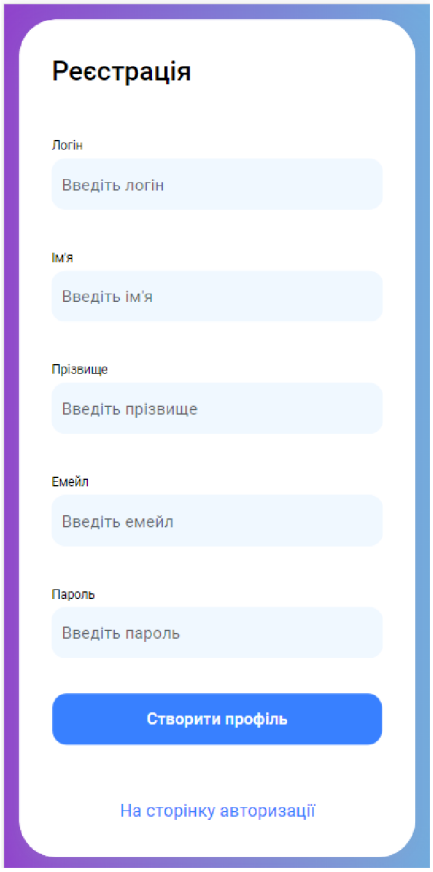


Рис. 4.11. Екран реєстрації

Розглянемо тепер HTML-код та TypeScript-код, які представляють собою функціонал для пошуку користувачем свого розташування за допомогою OpenStreetMap Nominatim API та його відображення в інтерфейсі користувача:

HTML-код:

```
<ion-item>
  <ion-input type="text"
             label="Локація"
             labelPlacement="stacked"
             class="input-box"
             placeholder="Локація"
             (ionInput)="onInput($event)"
             formControlName="location">
  </ion-input>
</ion-item>
<ion-card *ngIf="isOpenLocation">
  <ion-list [inset]="true">
    <ion-item *ngFor="let location of value"
(click)="selectLocation(location.display_name)">
      <ion-label>{{ location.display_name }}</ion-label>
    </ion-item>
  </ion-list>
</ion-card>
```

TypeScript-код LocationService:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { map } from 'rxjs/operators';
import { FormControl } from '@angular/forms';
@Injectable({
  providedIn: 'root'
})
export class LocationService {
  url: string = `https://nominatim.openstreetmap.org/search`;
```

```
    constructor(private http: HttpClient) { }
    searchUser(params: string) {
      return
this.http.get(`${this.url}?q=${params}&format=json&addressdetails=3&limit=5&pol
ygon_svg=1`).pipe(map((res: any) => res));
    }
  }
```

В інтерфейсі користувача використовується компонент ion-input, який дозволяє вводити текст для пошуку своєї локації користувачем. За допомогою (ionInput)=”onInput($event)” викликається метод onInput() при введенні тексту користувачем.

Якщо isOpenLocation дорівнює true, тоді виводиться блок ion-card, який містить список локацій, отриманих з сервісу. За допомогою директиви *ngFor елементи представляються в ion-list у вигляді ion-item, та виводиться display_name локації. При клацанні на ion-item викликається функція selectLocation(), яка обробляє вибір конкретної локації.

Сервіс LocationService використовує Angular HttpClient для здійснення HTTP-запитів до OpenStreetMap Nominatim API. Метод searchUser() отримує дані за допомогою GET-запиту з параметрами для пошуку локацій. Дані отримуються у форматі JSON та обробляються оператором map перед поверненням.

Загальний функціонал цього коду полягає у введенні користувачем свого розташування, виклику HTTP-запиту до Nominatim API для пошуку розташування та відображенні результатів у вигляді списку (рис 4.12), з яким користувач може взаємодіяти.

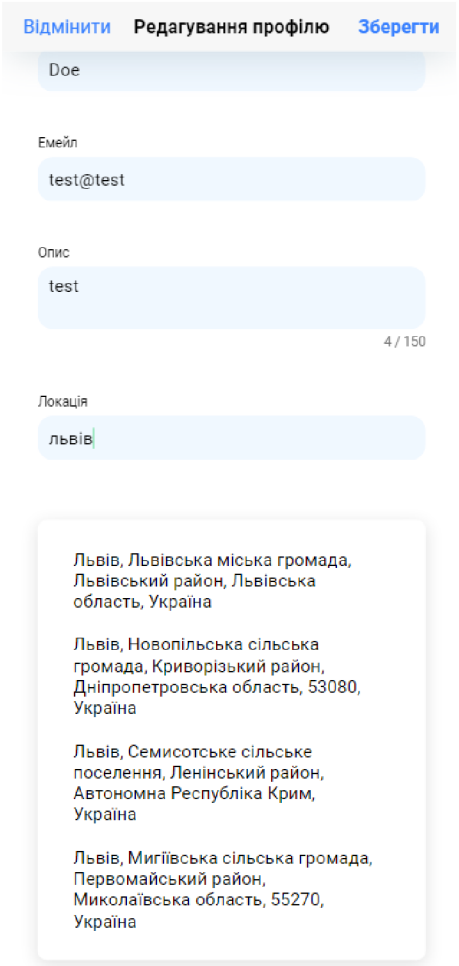


Рис. 4.12. Список розташуваннь

4.4 Формування валідаторів та тестування

Валідація даних є критичним елементом для забезпечення безвідмовності та безпеки програмного забезпечення. Існують два основних типи валідації: на стороні клієнта та на стороні сервера. Обидва підходи грають важливу роль в запобіганні помилок та недоліків в системі.

Валідація на стороні клієнта зазвичай відбувається в браузері користувача і призначена для негайного фільтрування та перевірки даних перед їх відправкою на сервер. Це дозволяє заощадити час та ресурси, зменшити кількість неправильно введених даних та покращити користувацький досвід. В Angular використовується Angular Reactive Forms для створення форм та

валідації введених даних. Ось приклад використання Reactive Forms для додавання до колекції:

```
export class CollectionAddBuilder {
  constructor(private formBuilder: FormBuilder) {}
  getForm(): FormGroup {
    return this.formBuilder.group({
      name: ['', Validators.compose([Validators.required])],
      type: ['',  Validators.compose([Validators.required])],
      year: ['',  Validators.compose([Validators.required])],
      description:  ['',  Validators.compose([Validators.required])],
    })
  }
}
```

Для відображення повідомлень про помилки коли спрацьовує валідатор використовується errorText для полів заповнення інформації:

```
<ion-textarea label="Опис"
                labelPlacement="stacked"
                placeholder="Опис"
                class="textarea-box"
                maxlength="150"
                [counter]="true"
                [autoGrow]="true"
                errorText="Введіть опис"
                ngModel
                formControlName="description">
        </ion-textarea>
```

Після спрацьовування одного з валідаторів на екрані буде висвітлений

текст підказки червоного кольору, який ми додали в errorText, а також максимальна допустима довжина тексту, введена у maxlength (рис 4.13).

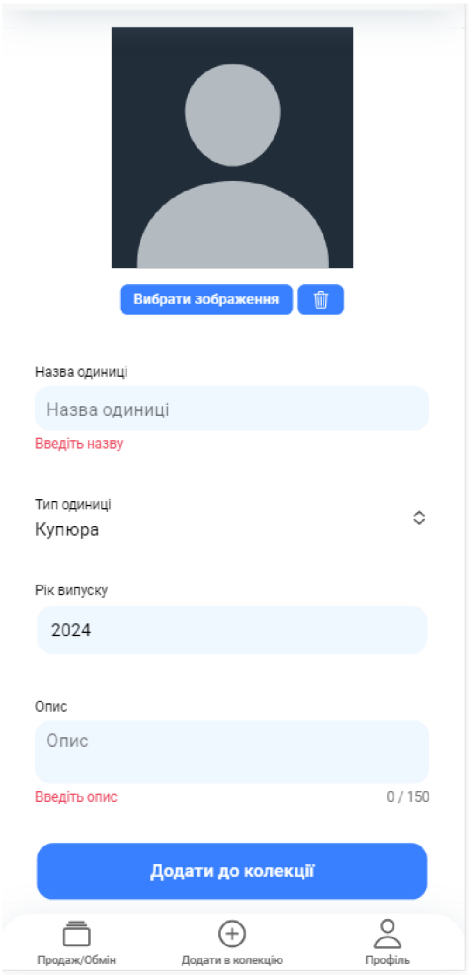


Рис. 4.13. Повідомлення про некоректні дані

Валідація на стороні сервера виконується після того, як дані були відправлені з клієнта та перед тим, як вони потраплять до БД чи іншого сховища на сервері. Це є критичний етап для перевірки даних на коректність та безпеку. Розглянемо таку валідацію на прикладі роуту для авторизації користувачів (рис 4.14).

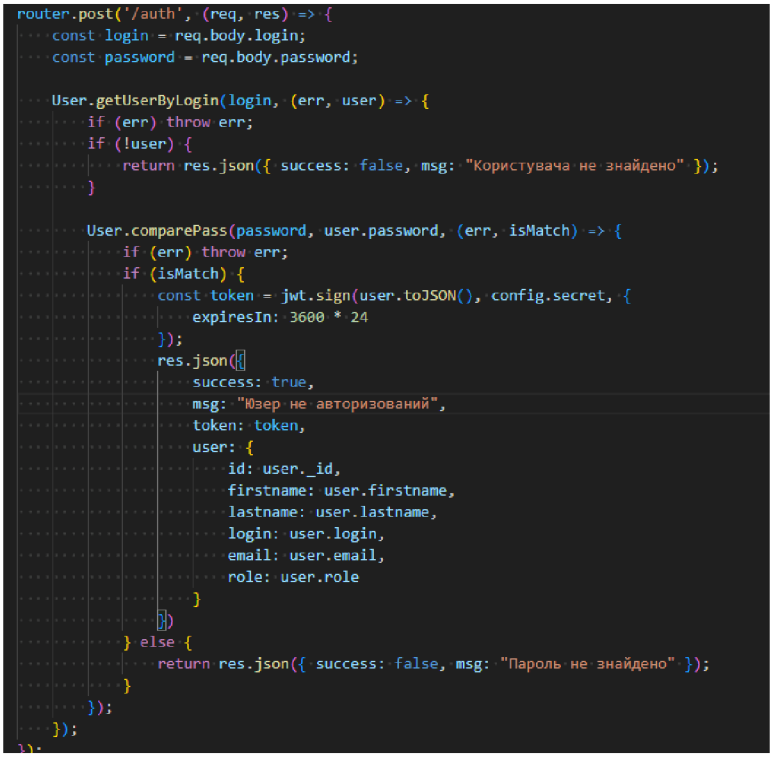


Рис. 4.14. Фрагмент коду для авторизації користувачів

Цей код представляє процес аутентифікації користувача на сервері за допомогою логіна та пароля. Розглянемо кожен етап валідації та виведення повідомлень при введенні помилкових даних.

Спочатку викликається метод getUserByLogin(), який шукає у БД користувача за введеним логіном. Якщо такий користувач не знайдений, відправляється відповідь з помилкою та повідомленням "Користувача не знайдено". Потім викликається метод comparePass(), який порівнює введений пароль з паролем користувача у БД. Якщо паролі не співпадають, відправляється відповідь з помилкою та повідомленням "Пароль не знайдено". Якщо аутентифікація успішна, генерується JWT токен та відправляється відповідь із токеном та інформацією про користувача на фронтенд.

Таким чином, валідація проводиться шляхом перевірки наявності користувача та порівняння паролів. При кожній помилці відправляється відповідь із відповідним повідомленням про помилку і це повідомлення обробляється на фронтенді і відображається за допомогою ion-toast (рис 4.15).

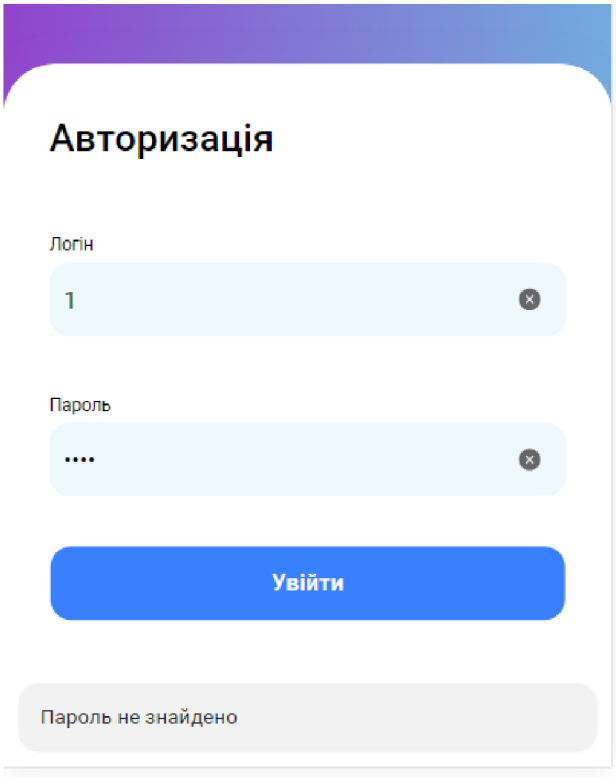


Рис. 4.14. Оброблене повідомлення про помилку

4.5 Створення .apk файлу для системи Android

Для того щоб створити APK файл для Android системи, потрібно у проєкт додати Capacitor. Capacitor – це інструмент для розробки мобільних застосунків, який дозволяє використовувати веб-технології для створення нативних застосунків для iOS, Android та інших мобільних платформ. Цей інструмент був розроблений командою Ionic та призначений для заміни Cordova.

Основні функції Capacitor:

● веб-технології – Capacitor дозволяє розробникам використовувати веб-технології (HTML, CSS, JavaScript) для побудови нативних мобільних застосунків;

● нативні плагіни – Capacitor підтримує використання нативних плагінів, які дозволяють взаємодіяти з функціоналом пристрою (камерою, геолокацією, файловою системою, тощо) з вебзастосунок;

● оптимізація продуктивності – Capacitor дозволяє розробникам використовувати особливості та API, які доступні на конкретних платформах, для оптимізації продуктивності;

● можливості розширення – Capacitor дозволяє легко додавати підтримку для нових платформ та використовувати функції Cordova, якщо це необхідно;

● сумісність з іншими фреймворками – Capacitor може бути використаний з будь-яким фреймворком для розробки мобільних застосунків, не лише з Ionic.

Створення APK файлу для Ionic з використанням Capacitor включає декілька етапів:

● Додавання Capacitor до проєкту – ionic integrations enable capacitor.

● Додавання платформи Android – npx cap add android.

● Компіляція Ionic проєкту – ionic build. Ця команда збирає Ionic проєкт та копіює зібрані файли в папку www і capacitor буде використовувати ці файли під час розгортання на мобільні платформи).

● Оновлення проєкту Capacitor – npx cap copy.

● Збірка Android проєкту – npx cap open android. Ця команда відкриває проєкт в Android Studio, в якому можна використовувати інструменти Android Studio для компіляції та пакування застосунку Android. В меню Android Studio вибераємо Build та Build Bundle(s) / APK(s), щоб створити APK файл.

● знаходження та використання APK файл. Після завершення збірки можна знайти APK файл у папці android/app/build/outputs/apk/debug/app-debug.apk.

Після успішного створення APK файлу можемо завантажити його на Android смартфон і виконати тестування застосунку.

Висновки. В четвертому розділі розглянуто структуру БД, процес її створення, здійснено реалізацію необхідних модулів, сервісів, моделей та роутів, розроблено графічний інтерфейс користувача, а також прописано валідатори.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Таблиця 5.1. Інформаційна карта проєкту

Назва номінації	Мобільний застосунок
Назва проєкту	Застосунок для нумізматів “Нумізмат”
Назва ВНЗ, спеціальності	НЛТУ, кафедра комп’ютерних наук, “Комп’ютерні науки”
Прізвище, ім’я, по-батькові автора	Пилипців Богдан Михайлович
Цілі і задачі проєкту	Спроєктувати мобільний застосунок “Нумізмат”, який дозволяє створювати колекції монет або валют
Короткий зміст проєкту	Застосунок дозволяє користувачам створювати власні акаунти та редагувати їх, створювати колекції монет або валют, а також залишати свої коментарі до певної колекції. Розроблений застосунок повинен надавати можливість адміністратору редагувати або видаляти коментарі будь-якого користувача.
Цільова аудиторія	Нумізмати, колекціонери
Терміни виконання проєкту	3 місяці
Бюджет проєкту	104 000 грн.

Суть проєкту полягає в тому, щоб розробити мобільний застосунок, який дає користувачу можливість з легкістю створити акаунт та додавати колекції монет і купюр, комунікувати з іншими колекціонерами та реагувати коментарями на їхні колекції, виставляти на обмін або продаж кожну одиницю, а адміністратору видаляти коментарі користувачів. Отримання такого застосунку і є метою даного проєкту.

5.2 Розроблення ринкової стратегія проєкту

Розроблення ринкової стратегії передбачає визначення стратегії охоплення ринку (табл. 5.2).

Таблиця 5.2. Опис цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції на ринку	Простота виходу на ринок
1.	Нумізмати	Висока	Високий	Конкуренція на ринку помірна, проте внесення унікальних рішень допоможе отримати місце на ринку	Розробка застосунку вимагає незначних витрат, обмежуючись витратами на працю програміста, оплату хостингу для бекенду та акаунт розробника у Google Play Market. Ймовірність успішного використання серед цільової аудиторії є високою.
2.	Колекціонери	Висока	Високий		
3.	Торговці монетами	Висока	Високий		
4.	Любителі історії та культури	Середня	Середній		

Протягом роботи застосунку користувач познайомиться з основними його аспектами, кожен з яких спрямований на певні групи споживачів. Проте можливість легко та ефективно вести облік колекцій своїх монет, включаючи додавання нових екземплярів, опис та історію кожної монети дозволяють користувачу задовільнити свої основні потреби у даній сфері.

5.3 Розробка маркетингової програми стартап-проєкту

Таблиця 5.3. Визначення ключових переваг концепції потенційного продукту.

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Наявність доступу до застосунку	Єдиний код написаний на фреймоврку Ionic дозволяє одночасно запускати застосунок у браузері та на мобільних платформах Android і IOS	Кросплатформність
2	Дружній інтерфейс	Відсутність надлишкових елементів та сучасного наявність UI/UX дизайну	Заохочує нових користувачів продовжувати використання платформ Android і IOS
3	Глибоке розуміння потреб та очікувань нумізматів щодо управління своїми колекціями	Зручність в управлінні колекцією, а також можливості продажу та обміну	Інструменти для продажу та обміну одиниць своїх колекцій

Функції для легкого проведення продажу та обміну монетами дозволяє зробити застосунок зручним для активної участі в нумізматичній спільноті.

Таблиця 5.4. Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на послугу
Невідомо	Невідомо	600 $ +	100-300$

Висновки. Розроблений застосунок має сприятливі прогнози на ринку. Він задовольняє потреби певного кола користувачів: нумізматів, колекціонерів, та любителів історії. Зручний доступ до інформації про монети, їх характеристики, історію і легкість в управлінні колекцією робить його конкурентоспроможним на ринку застосунків даного сегменту.

ВИСНОВКИ

В результаті виконання дипломної роботи було розроблено мобільний застосунок для нумізматів та колекціонерів, який дозволяє користувачам створювати власні акаунти, колекції монет і купюр, виставляти на продаж або обмін будь-який екземпляр з колекцій а також реагувати коментарями і лайками на колекції інших користувачів.

Мобільний застосунок спроєктовано з використанням мови програмування TypeScript, сімейства фреймворків Angular, Ionic та Express.js, платформи Node.js, методів шифрування bcrypt, інтегрованого середовища розробки Visual Studio Code та не реляційної СКБД MongoDB. Даний застосунок дозволяє адміністратору видаляти коментарі інших користувачів. Виконано тестування розробленого програмного забезпечення застосунку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Are Your Passwords Safe: Energy-Efficient Bcrypt Cracking with Low-Cost Parallel Hardware [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.usenix.org/system/files/conference/woot14/woot14-malvoni.pdf (дата звернення 16.11.2023). – Назва з екрана.

2. Confidentiality, availability and integrity of information [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://stboinf.wordpress.com/ 2013/03/12/кофіденційністьдоступнійсть-і-цілі (дата звернення 16.11.2023). – Назва з екрана.

3. Cross Platform [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://ionicframework.com/docs/core-concepts/cross-platform (дата звернення 16.11.2023). – Назва з екрана.

4. Database Commands [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.mongodb.com/docs/manual/reference/command/ (дата звернення 16.11.2023). – Назва з екрана.

5. DEFINITION operating system (OS) [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.techtarget.com/whatis/definition/operating-system-OS (дата звернення 16.11.2023). – Назва з екрана.

6. Digitalization as a Global Trend and Growth Factor of the Modern Economy [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://ceur-ws.org/Vol-2422/paper35.pdf (дата звернення 16.11.2023). – Назва з екрана.

7. Everything You Need To Know About Mobile App Ecosystem [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https:// www.webskitters.com/blog/everything-you-need-to-know-about-mobile-app-ecosystem (дата звернення 16.11.2023). – Назва з екрана.

8. Flexibility of Schema with MongoDB [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.oodlestechnologies.com/blogs/flexibility-of-schema-with-mongodb/ (дата звернення 16.11.2023). – Назва з екрана.

9. From API to Database: A Step-by-Step Guide on Efficient Data Integration [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://airbyte.com/data-engineering-resources/api-to-database (дата звернення 16.11.2023). – Назва з екрана.
10. Mobile Optimization: What It Is & How to Do It Successfully [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://emarsys.com/learn/blog/mobile-optimization-strategies/ (дата звернення 16.11.2023). – Назва з екрана.
11. MongoDB Compass [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.javatpoint.com/mongodb-compass (дата звернення 16.11.2023). – Назва з екрана.
12. MongoDB Indexes: Creating, Finding & Dropping Top Index Types [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.bmc.com/blogs/mongodb-indexes/ (дата звернення 16.11.2023). – Назва з екрана.
13. Rainbow Tables & Why To Add Salt [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://dev.to/salothom/rainbow-tables-why-to-add-salt-45l9 (дата звернення 16.11.2023). – Назва з екрана.
14. Salting and Hashing Passwords with bcrypt.js: A Comprehensive Guide [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://medium.com/@arunchaitanya/salting-and-hashing-passwords-with-bcrypt-js-a-comprehensive-guide-f5e31de3c40c (дата звернення 16.11.2023). – Назва з екрана.
15. Scaling Node.js Applications [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.freecodecamp.org/news/scaling-node-js-applications-8492bd8afadc/ (дата звернення 16.11.2023). – Назва з екрана.
16. Static Typing in TypeScript: Typescript Basics [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://patrickkarsh.medium.com/static-typing-in-typescript-typescript-basics-9dbf527f1ef9 (дата звернення 16.11.2023). – Назва з екрана.

17. Using Git source control in VS Code [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://code.visualstudio.com/docs/sourcecontrol/overview#:~:text=VS%20Code's%20built%2Din%20Git,ID%20and%20Copy%20Commit%20Message. (дата звернення 16.11.2023). – Назва з екрана.
18. What are the best libraries for using Node.js with a command-line interface? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://reintech.io/blog/best-libraries-for-using-node-js-with-a-command-line-interface (дата звернення 16.11.2023). – Назва з екрана.
19. What is a NoSQL database? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.mongodb.com/nosql-explained#:~:text=When%20people%20use%20the%20term,format%20other%20than%20relational%20tables. (дата звернення 16.11.2023). – Назва з екрана.
20. What is a REST API? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата звернення 16.11.2023). – Назва з екрана.
21. What is Blowfish encryption, and how does it work? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://nordvpn.com/uk/blog/what-is-blowfish-encryption/ (дата звернення 16.11.2023). – Назва з екрана.
22. What is BSON? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.mongodb.com/basics/bson#:~:text=BSON%20stands%20for%20Binary%20Javascript,binary%20formats%2C%20like%20Protocol%20Buffers (дата звернення 16.11.2023). – Назва з екрана.
23. What is Visual Studio Code? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.educative.io/answers/what-is-visual-studio-code (дата звернення 16.11.2023). – Назва з екрана.
24. Why Android OS is More Popular Among Developers? [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.codesquadz.com/blog/why-android-os-is-more-popular-among-developers/ (дата звернення 16.11.2023). – Назва з екрана.

25. Your First Ionic App: Angular [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://ionicframework.com/docs/angular/your-first-app (дата звернення 16.11.2023). – Назва з екрана.

ДОДАТКИ

ДОДАТОК А

Passport.js

```
const config  = require('./db');
const User  = require('../models/user');

var JwtStrategy = require('passport-jwt').Strategy, ExtractJwt = require('passport-jwt').ExtractJwt;

module.exports = function(passport) {
var opts = {}
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = config.secret;
passport.use(new JwtStrategy(opts, function(jwt_payload, done) {
   User.findOne({id: jwt_payload.sub}, function(err, user) {
      if (err) {
         return done(err, false);
      }
      if (user) {
         return done(null, user);
      } else {
         return done(null, false);
         // or you could create a new account
      }
   });
}));
}
```

User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = mongoose.Schema({
   firstname: {
      type: String,
      required: true
```

```
    },
    lastname: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    login: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    role: {
        type: String
    },
    description: {
        type: String
    },
    location: {
        type: String
    },
    image:
    {
        type: String
    }
});

const User = module.exports = mongoose.model('User', UserSchema);

module.exports.getUserByLogin = function (login, callback) {
    const query = { login: login };
    User.findOne(query, callback);
}
```

```
module.exports.getUserById = function (id, callback) {
  User.findById(id, callback);
}

module.exports.updateUser = function (id, user, callback) {
  User.updateOne({ _id: id }, { firstname: user.firstname, lastname: user.lastname, email: user.email,
description: user.description, location: user.location, image: user.image }, callback);
}

module.exports.updateUserPassword = function (id, passwords, callback) {
  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(passwords.newPassword, salt, (err, hash) => {
      if (err) throw err;
      passwords.newPassword = hash;
      User.updateOne({ _id: id }, { password: passwords.newPassword }, callback);
    });
  });
}

module.exports.comparePass = function (passFromUser, userDBPass, callback) {
  bcrypt.compare(passFromUser, userDBPass, (err, isMatch) => {
    if (err) throw err;
    callback(null, isMatch);
  });
}

module.exports.addUser = function (newUser, callback) {
  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(newUser.password, salt, (err, hash) => {
      if (err) throw err;
      newUser.password = hash;
      newUser.save(callback);
    });
  })
}
```

Comments.js

```
const mongoose = require('mongoose');
const CommentsSchema = mongoose.Schema({
    comment: {
        type: String,
        required: true
    },
    user: {
        type: String,
        required: true
    },
    collectionId: {
        type: String,
        required: true
    },
    date: {
        type: String,
        required: true
    },
    likes: {
        type: Array,
        required: true
    }
});
const Comments = module.exports = mongoose.model('Comments', CommentsSchema);
```

collections.js

```
const mongoose = require('mongoose');


const CollectionSchema = mongoose.Schema({
    userId: {
        type: String,
        required: true
    },
    loginName: {
        type: String,
        required: true
```

```
    },
    type: {
        type: String,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    year: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    image: {
        type: String,
        required: true
    },
});
const Collections = module.exports = mongoose.model('Collections', CollectionSchema);
module.exports.addCollection = function (collection, callback) {
    collection.save(callback);
}
module.exports.getCollectionsByLogin = function (loginName, callback) {
    const query = { loginName: loginName };
    Collections.find(query, callback);
}
module.exports.deleteCollectionById = function (id, callback) {
    Collections.deleteOne({ _id: id }, callback);
}
```

Chat.js

```
const mongoose = require('mongoose');
```

```
const ChatSchema = mongoose.Schema({
    comment: {
        type: String,
        required: true
    },
    chatId: {
        type: Number,
        required: true
    },
    collectionId: {
        type: Number,
        required: true
    },
    userId: {
        type: Number,
        required: true
    },
    date: {
        type: Date,
        required: true
    }
});
const Chat = module.exports = mongoose.model('Chat', ChatSchema);
```

Account.js

```
const express = require('express');
const router = express.Router();
const User = require('../models/user');
const passport = require('passport');
const jwt = require('jsonwebtoken');
const config = require('../config/db');


router.get('/reg', (req, res) => {
    res.send('Registration page')
});
```

```
router.post('/reg', async (req, res) => {
    let newUser = new User({
        firstname: req.body.firstname,
        lastname: req.body.lastname,
        email: req.body.email,
        login: req.body.login,
        password: req.body.password,
        location: '',
        description: '',
        role: 'user',
        image: ''
    });
    const usernameExists = await User.findOne({ login: newUser.login });
    const emailExists = await User.findOne({ email: newUser.email });
    if (usernameExists && emailExists) {
        res.status(400);
        res.json({ success: false, msg: "Username and email already exists" })
    } else if (usernameExists) {
        res.status(400);
        res.json({ success: false, msg: "Username already exists" })
    } else if (emailExists) {
        res.status(400);
        res.json({ success: false, msg: "Email already exists" })
    } else {
        User.addUser(newUser, (err, user) => {
            if (err) {
                res.json({ success: false, msg: "User doesn't added to db" });
            } else {
                res.json({ success: true, msg: "User added to db" });
            }
        });
    }
});
router.post('/updateUser', (req, res) => {
    try {
        const loginName = req.body.loginName;
        const user = req.body.user;
        let newUser = new User({
```

```
        ...user,
        role: 'user'
      });
      User.getUserByLogin(loginName, async (err, user) => {
        if (!user) {
          return res.json({ success: false, msg: "User not found" });
        } else {
          User.updateUser(user._id, newUser, (err, user) => {

            if (err) {
              return res.json({ success: false, msg: "User doesn't updated" });
            }
            return res.json({ success: true, msg: "User updated successfully" });
          })
        }
      });
    } catch (err) {
      return res.json({ success: false, msg: err });
    }
});
router.post('/updateUserPassword', (req, res) => {
    try {
      const loginName = req.body.loginName;
      const passwords = req.body.passwords;
      User.getUserByLogin(loginName, async (err, user) => {
        if (!user) {
          return res.json({ success: false, msg: "User not found" });
        } else {
          User.comparePass(passwords.oldPassword, user.password, (err, isMatch) => {
            if (err) throw err;
            if (isMatch) {
              User.updateUserPassword(user._id, passwords, (err, user) => {
                if (err) {
                  return res.json({ success: false, msg: "User password doesn't updated" });
                }
                return res.json({ success: true, msg: "User password updated successfully" });
              });
            } else {
```

```
                return res.json({ success: false, msg: "Password not found" });
            }
          });
        }
      });
    } catch (err) {
      return res.json({ success: false, msg: err });
    }
});
router.get('/getCurrentUser', (req, res) => {
    try {
      const {
        loginName,
      } = req.query;
      const login = loginName;
      if (!login) {
        return res.json({ success: false, msg: "User not found" });
      } else {
        User.getUserByLogin(login, async (err, user) => {
          if (!user) {
            return res.json({ success: false, msg: "User not found" });
          } else {
            if (err) {
              return res.json({ success: false, msg: "User doesn't exist or other error" });
            }
            else {
              res.json({
                success: true, msg: "User found",
                user: {
                  id: user._id,
                  firstname: user.firstname,
                  lastname: user.lastname,
                  email: user.email,
                  login: user.login,
                  role: user.role,
                  description: user.description,
                  location: user.location,
                  image: user.image
```

```
                    }
                });
            }
        }
    });
  }
 } catch {
  return res.json({ msg: 'Error getCurrentUser' });
 }
});
router.post('/auth', (req, res) => {
 const login = req.body.login;
 const password = req.body.password;

 User.getUserByLogin(login, (err, user) => {
  if (err) throw err;
  if (!user) {
   return res.json({ success: false, msg: "Користувача не знайдено" });
  }
  User.comparePass(password, user.password, (err, isMatch) => {
   if (err) throw err;
   if (isMatch) {
    const token = jwt.sign(user.toJSON(), config.secret, {
     expiresIn: 3600 * 24
    });
    res.json({
     success: true,
     msg: "Юзер не авторизований",
     token: token,
     user: {
      id: user._id,
      firstname: user.firstname,
      lastname: user.lastname,
      login: user.login,
      email: user.email,
      role: user.role
     }
    })
```

```
        } else {
            return res.json({ success: false, msg: "Пароль не знайдено" });
        }
    });
  });
});
router.get('/dashboard', passport.authenticate('jwt', { session: false }), (req, res) => {
  res.send('Dashboard page')
});
module.exports = router;
```

Collections.js

```
const express = require('express');
const router = express.Router();
const Collections = require('../models/collections');
const User = require('../models/user');
const verifyToken = require('../verify-token');

router.post('/addCollection', async (req, res) => {
  let newCollection = new Collections({
    ...req.body.collection
  });
  Collections.addCollection(newCollection, (err, user) => {
    if (err) {
      console.log(err)
      res.json({ success: false, msg: "Collection doesn't added to db" });
    } else {
      res.json({ success: true, msg: "Collection added to db" });
    }
  });
});
router.get('/getUserCollections', (req, res) => {
  try {
    const login = req.query.loginName;
    if (!login) {
      return res.json({ success: false, msg: "Collections not found" });
    } else {
      Collections.getCollectionsByLogin(login, async (err, collections) => {
```

```
            if (!collections) {
              return res.json({ success: false, msg: "Collections not found" });
            } else {
              if (err) {
                return res.json({ success: false, msg: "Collections doesn't exist or other error" });
              }
              else {
                res.json({
                  success: true, msg: "Collections found",
                  collections: collections
                });
              }
            }
          });
        }
      } catch {
        return res.json({ msg: 'Error getUserCollections' });
      }
});
router.delete(`/deleteUserCollection`, (req, res) => {
  try {
    const login = req.body.loginName;
    const id = req.body.id;
    const collectionOwner = req.body.collectionOwner;
    if (login === null) {
      res.status(500).send('unauthorized');
    }
    User.getUserByLogin(login, async (err, user) => {
      if (!user) {
        return res.json({ success: false, msg: "User not found" });
      } else {
        if (err) {
          return res.json({ success: false, msg: "User doesn't exist or other error" });
        }
        else {
          if (user.role === 'admin') {
            Collections.deleteCollectionById(id, (err, user) => {
              if (err) {
```

```
                    res.status(500).json({ success: false, msg: "Collection wasn't removed from db" });
                  } else {
                    res.json({ success: true, msg: "Collection was removed from db" });
                  }
                });
              } else {
                if (collectionOwner === user.login) {
                  Collections.deleteCollectionById(id, (err, user) => {
                    if (err) {
                      res.status(500).json({ success: false, msg: "Collection wasn't removed from db" });
                    } else {
                      res.json({ success: true, msg: "Collection was removed from db" });
                    }
                  });
                } else {
                  res.status(500).send('unauthorized');
                }
              }
            }
          }
        });
    } catch (e) {
      res.status(500).send(e);
    }
});
module.exports = router;
```

ДОДАТОК Б

Collections-details.component.html

```
<ion-header class="header-container">
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-button class="input-box"
                (click)="cancel()">Повернутись</ion-button>
    </ion-buttons>
    <ion-title>Деталі</ion-title>
    <!-- <ion-buttons slot="end">
      <ion-button [strong]="true">Зберегти</ion-button>
    </ion-buttons> -->
  </ion-toolbar>
</ion-header>
<ion-content class="ion-padding">
  <ion-list lines="none"
            class="item-list">
    <ion-item>
      <ion-img [src]="collection.image"></ion-img>
    </ion-item>
    <ion-item class="item">
      <b>Назва: </b>
      <div> &#8203; {{collection.name}}</div>
    </ion-item>
    <ion-item class="item">
      <b>Тип: </b>
      <div> &#8203; {{collection.type === "coin" ? "Монета" : "Куп'юра"}}</div>
    </ion-item>
    <ion-item class="item">
      <b>Рік випуску: </b>
      <div> &#8203; {{collection.year | date:'yyyy' }}</div>
    </ion-item>
    <ion-item class="item">
      <b>Опис: </b>
      <div>&#8203; {{collection.description}}</div>
    </ion-item>
```

```
    <ion-item>
      <ion-button expand="block"
                  id="present-alert"><ion-icon name="trash-outline"></ion-icon></ion-button>
      <ion-alert trigger="present-alert"
                 header="Ви точно пітверджуєте видалення?"
                 [buttons]="alertButtons"
                 (didDismiss)="setResult($event)">
      </ion-alert>
    </ion-item>
    <ion-item>
      <ion-textarea type="text"
                    label="Коментар"
                    labelPlacement="stacked"
                    class="textarea-box"
                    placeholder="Додати коментар"></ion-textarea>
      <ion-button class="margin-top"
                  expand="block">
        <ion-icon name="send-outline"></ion-icon>
      </ion-button>
    </ion-item>
  </ion-list>
</ion-content>
```

collection-details-modal.component.scss

```
.textarea-box::ng-deep {
  .textarea-wrapper {
    flex-direction: column !important;

    .native-wrapper {
      background: aliceblue;
      border-radius: 14px;
      padding: 0 10px;
      margin-top: 10px;
    }
  }
  .textarea-bottom {
    border: none;
  }
```

```
}
```

collection-details-modal.component.spec.ts

```
import { ComponentFixture, TestBed, waitForAsync } from '@angular/core/testing';
import { IonicModule } from '@ionic/angular';
import { CollectionDetailsModalComponent } from './collection-details-modal.component';
describe('CollectionDetailsModalComponent', () => {
  let component: CollectionDetailsModalComponent;
  let fixture: ComponentFixture<CollectionDetailsModalComponent>;
  beforeEach(waitForAsync(() => {
    TestBed.configureTestingModule({
      declarations: [ CollectionDetailsModalComponent ],
      imports: [IonicModule.forRoot()]
    }).compileComponents();
    fixture = TestBed.createComponent(CollectionDetailsModalComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  }));
  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

collection-details-modal.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ModalController } from '@ionic/angular';
@Component({
  selector: 'num-collection-details-modal',
  templateUrl: './collection-details-modal.component.html',
  styleUrls: ['./collection-details-modal.component.scss'],
})
export class CollectionDetailsModalComponent  implements OnInit {
  name: string;
  collection: any;
  alertButtons = [
    {
      text: 'Cancel',
      role: 'cancel',
```

```
      handler: () => {
        console.log('Alert canceled');
      },
    },
    {
      text: 'OK',
      role: 'confirm',
      handler: () => {
        console.log('Alert confirmed');
      },
    },
  ];
  constructor(private modalCtrl: ModalController) { }
  ngOnInit() {
    console.log(this.collection)
  }
  cancel() {
    return this.modalCtrl.dismiss(null, 'cancel');
  }
  confirm() {
    return this.modalCtrl.dismiss(this.name, 'confirm');
  }
  setResult(ev: any) {
   if(ev.detail.role === 'confirm') {

   }
  }
}
```

Auth.component.html

```
<div class="page-wrapper">
  <div class="section-container">
    <div class="img-container">
      <img class="img-treasure"
          src="../../assets/treasure.svg">
    </div>
  </div>
```

```
<form (submit)="userLoginClick()"
    class="form-wrapper">
 <ion-list lines="none"
        class="item-list">
  <ion-item class="title-item">
   <ion-text>
    <h1>Авторизація</h1>
   </ion-text>
  </ion-item>
  <ion-item>
   <ion-input class="input-box"
           label="Логін"
           labelPlacement="stacked"
           [(ngModel)]="login"
           name="login"
           [clearInput]="true"
           placeholder="Введіть логін">
   </ion-input>
  </ion-item>
  <ion-item>
   <ion-input class="input-box"
           label="Пароль"
           labelPlacement="stacked"
           [(ngModel)]="password"
           name="password"
           type="password"
           password
           ngModel
           [clearInput]="true"
           placeholder="Введіть пароль">
   </ion-input>
  </ion-item>
  <ion-button class="login-button"
           expand="block"
           type="submit">Увійти</ion-button>
  <ion-button routerLink="/register"
           fill="clear">Досі немає профілю? Реєстрація</ion-button>
 </ion-list>
```

```
  </form>
  <ion-toast [isOpen]="isToastOpen"
          [message]="notifitcationText"
          [duration]="5000"
          (didDismiss)="setOpen(false)"></ion-toast>
</div>
```

auth.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from 'src/app/shared/services/auth.service';
import { FormService } from 'src/app/shared/services/form.service';
@Component({
  selector: 'num-auth',
  templateUrl: './auth.component.html',
  styleUrls: ['./auth.component.scss'],
})
export class AuthComponent {
  login: string = '';
  password: string = '';
  notifitcationText: string= '';
  isToastOpen = false;

  constructor(private formService: FormService,
    private authService: AuthService,
    private router: Router) {}
  ngOnInit(): void {
  }
  setOpen(isOpen: boolean) {
    this.isToastOpen = isOpen;
  }

  userLoginClick(): void {
    const user = {
      login: this.login,
      password: this.password
    }
```

```
    if (!this.formService.checkLogin(user.login)) {
      this.notifitcationText = "Login does not entered";
      this.setOpen(true);
    } else if (!this.formService.checkPassword(user.password)) {
      this.notifitcationText = "Password does not entered";
      this.setOpen(true);
    }
    this.authService.authUser(user).subscribe(data => {
      console.log(data)
      // debugger
      this.notifitcationText = data.msg;
      if (!data.success) {
        this.setOpen(true);
        this.router.navigate(['/login']);
      } else {
        this.authService.storeUser(data.token, data.user);
        this.setOpen(true);
        setTimeout(() => window.location.reload(), 500);
        const user = this.authService.getUser();
        this.router.navigate(['/tabs/user-info-tab', user.login]);
      }
    });
  }
}
```

Register.component.ts

```
<div class="page-wrapper">
  <form (submit)="userRegisterClick()"
        class="form-wrapper">
    <ion-list lines="none"
              class="item-list">
      <ion-item class="title-item">
        <ion-text>
          <h1>Реєстрація</h1>
        </ion-text>
      </ion-item>
```

```
<ion-item>
  <ion-input class="input-box"
        label="Логін"
        labelPlacement="stacked"
        name="login"
        [(ngModel)]="login"
        [clearInput]="true"
        placeholder="Введіть логін">
  </ion-input>
</ion-item>
<ion-item>
  <ion-input class="input-box"
        label="Ім'я"
        labelPlacement="stacked"
        [clearInput]="true"
        name="firstname"
        [(ngModel)]="firstname"
        placeholder="Введіть ім'я">
  </ion-input>
</ion-item>
<ion-item>
  <ion-input class="input-box"
        label="Прізвище"
        labelPlacement="stacked"
        [clearInput]="true"
        name="lastname"
        [(ngModel)]="lastname"
        placeholder="Введіть прізвище">
  </ion-input>
</ion-item>
<ion-item>
  <ion-input class="input-box"
        label="Емейл"
        labelPlacement="stacked"
        [clearInput]="true"
        name="email"
        [(ngModel)]="email"
        placeholder="Введіть емейл">
```

```
      </ion-input>
    </ion-item>
    <ion-item>
      <ion-input class="input-box"
                 label="Пароль"
                 labelPlacement="stacked"
                 [clearInput]="true"
                 name="password"
                 type="password"
                 [(ngModel)]="password"
                 placeholder="Введіть пароль">
      </ion-input>
    </ion-item>
    <ion-button class="login-button"
                type="submit"
                expand="block">Створити профіль</ion-button>
    <ion-button routerLink="/login"
                fill="clear">На сторінку авторизації</ion-button>
  </ion-list>
 </form>
 <ion-toast [isOpen]="isToastOpen"
            [message]="notifitcationText"
            [duration]="5000"
            (didDismiss)="setOpen(false)"></ion-toast>
</div>
```

register.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from 'src/app/shared/services/auth.service';
import { FormService } from 'src/app/shared/services/form.service';
@Component({
 selector: 'num-register',
 templateUrl: './register.component.html',
 styleUrls: ['./register.component.scss'],
})
export class RegisterComponent {
```

```
email: string = '';
login: string = '';
firstname: string = '';
lastname: string = '';
password: string = '';
notifitcationText: string = '';
isToastOpen = false;
constructor(private formService: FormService,
  private authService: AuthService,
  private router: Router) { }
ngOnInit(): void {
}
setOpen(isOpen: boolean) {
  this.isToastOpen = isOpen;
}
userRegisterClick() {
  const user = {
    firstname: this.firstname,
    lastname: this.lastname,
    email: this.email,
    login: this.login,
    password: this.password
  }
  if (!this.formService.checkName(user.firstname)) {
    this.notifitcationText = "Firstname does not entered";
    this.setOpen(true);
  } else if (!this.formService.checkLogin(user.lastname)) {
    this.notifitcationText = "Lastname does not entered";
    this.setOpen(true);
  }
  else if (!this.formService.checkLogin(user.login)) {
    this.notifitcationText = "Login does not entered";
    this.setOpen(true);

  } else if (!this.formService.checkEmail(user.email)) {
    this.notifitcationText = "Email does not entered";
    this.setOpen(true);
```

```
    } else if (!this.formService.checkPassword(user.password)) {
      this.notifitcationText = "Password does not entered";
      this.setOpen(true);
    }
    this.authService.registerUser(user).subscribe(data => {
      this.notifitcationText = data.msg;
      if (!data.success) {
        this.setOpen(true);
        this.router.navigate(['/reg']);
      } else {
        this.notifitcationText = "User created successfully";
        this.setOpen(true);
        this.router.navigate(['/login']);
      }
    });
  }
}
```

user-info.component.html

```
<ion-menu content-id="main-content">
  <ion-content>
    <div class="img-info-content">
      <ion-avatar class="avatar-size ion-padding">
        <img alt="person"
            imageViewer
            [src]="baseImg" />
      </ion-avatar>
      <div class="ion-padding info-box">
        <div class="info-container">
          <h3>{{currentUserInfo?.firstname}} {{currentUserInfo?.lastname}}</h3>
          <span>{{currentUserInfo?.email}}</span>
        </div>
      </div>
    </div>
    <ion-list lines="full">
      <ion-item id="open-settings-modal">
        <ion-icon name="settings-outline"></ion-icon>
```

```
      <ion-label>Налаштування</ion-label>
    </ion-item>
    <ion-item (click)="logOut()">
      <ion-icon name="log-out-outline"></ion-icon>
      <ion-label>Вийти з аккаунту</ion-label>
    </ion-item>
  </ion-list>
 </ion-content>
</ion-menu>
<div class="ion-page"
   id="main-content">
 <ion-header class="header-container"
          [translucent]="true">
  <ion-toolbar>
   <ion-title>
    Інформація про користувача
   </ion-title>
   <ion-buttons slot="start">
    <ion-menu-toggle>
     <ion-menu-button menu="side"
              color="lili"
              auto-hide="false"></ion-menu-button>
    </ion-menu-toggle>
   </ion-buttons>
   <ion-buttons slot="end">
    <ion-icon name="log-out-outline"
          color="primary"
          (click)="logOut()"
          size="large">
    </ion-icon>
   </ion-buttons>
  </ion-toolbar>
 </ion-header>
 <ion-content class="light-bg"
          [fullscreen]="true">
  <div class="d-flex ion-justify-content-center content-margin">
   <ion-avatar class="avatar-size">
    <img alt="person"
```

```
      imageViewer
      [src]="baseImg" />
</ion-avatar>
<div class="info-container">
 <h3>{{currentUserInfo?.firstname}} {{currentUserInfo?.lastname}}</h3>
 <span>{{currentUserInfo?.email}}</span>
 <span><ion-icon name="location-outline"></ion-icon>{{currentUserInfo?.location}}</span>
 <div class="description">
  {{currentUserInfo?.description}}
 </div>
</div>
<ion-button id="open-modal"
          *ngIf="isCurrentUser"
          size="small"
          class="ion-margin-top"
          expand="block">Редагувати профіль</ion-button>
<form [formGroup]="userDetails">
 <ion-modal trigger="open-modal"
           (willDismiss)="onEditWillDismiss($event)">
  <ng-template>
   <ion-header class="header-container">
    <ion-toolbar>
     <ion-buttons slot="start">
      <ion-button class="input-box"
                (click)="cancelEdit()">Відмінити</ion-button>
     </ion-buttons>
     <ion-title>Редагування профілю</ion-title>
     <ion-buttons slot="end">
      <ion-button (click)="confirmEdit()"
                [strong]="true">Зберегти</ion-button>
     </ion-buttons>
    </ion-toolbar>
   </ion-header>
   <ion-content class="ion-padding">
    <ion-list lines="none"
            class="item-list">
     <ion-row class="ion-justify-content-center">
      <ion-item>
```

```
    <div class="img-avatar-container">
      <ion-avatar class="avatar-size"
                  slot="end">
        <img alt="person"
             [src]="baseImg"
             imageViewer />
      </ion-avatar>
      <div class="d-flex">
        <ion-button expand="block"
                    (click)="pickImages()">Вибрати зображення</ion-button>
        <ion-button expand="block"
                    (click)="deleteImage()"><ion-icon name="trash-outline"></ion-icon></ion-button>
      </div>
    </div>
  </ion-item>
</ion-row>
<ion-item>
  <ion-input type="text"
             label="Ім'я"
             labelPlacement="stacked"
             class="input-box"
             placeholder="Ім'я"
             formControlName="firstname"></ion-input>
</ion-item>
<ion-item>
  <ion-input type="text"
             label="Прізвище"
             labelPlacement="stacked"
             class="input-box"
             placeholder="Прізвище"
             formControlName="lastname"></ion-input>
</ion-item>
<ion-item>
  <ion-input type="text"
             label="Емейл"
             labelPlacement="stacked"
             class="input-box"
             placeholder="Емейл"
```

```
                formControlName="email"></ion-input>
        </ion-item>
        <ion-item>
          <ion-textarea label="Опис"
                  labelPlacement="stacked"
                  placeholder="Опис"
                  class="textarea-box"
                  maxlength="150"
                  [counter]="true"
                  [autoGrow]="true"
                  formControlName="description">
          </ion-textarea>
        </ion-item>
        <ion-item>
          <ion-input type="text"
                  label="Локація"
                  labelPlacement="stacked"
                  class="input-box"
                  placeholder="Локація"
                  (ionInput)="onInput($event)"
                  formControlName="location">
          </ion-input>
        </ion-item>
        <ion-card *ngIf="isOpenLocation">
          <ion-list [inset]="true">
            <ion-item *ngFor="let locations of value"
                   (click)="selectLocation(locations.display_name)">
              <ion-label>{{locations.display_name}}</ion-label>
            </ion-item>
          </ion-list>
        </ion-card>
      </ion-list>
    </ion-content>
  </ng-template>
 </ion-modal>
</form>
<num-settings-modal></num-settings-modal>
</div>
```

```
    <num-collections></num-collections>
  </ion-content>
</div>
```

user-info.component.ts

```
import { Component, ViewChild } from '@angular/core';
import { FormGroup } from '@angular/forms';
import { DomSanitizer, SafeUrl } from '@angular/platform-browser';
import { ActivatedRoute, Router } from '@angular/router';
import { FilePicker, PickedFile } from '@capawesome/capacitor-file-picker';
import { IonModal, MenuController } from '@ionic/angular';
import { OverlayEventDetail } from '@ionic/core/components';
import { AuthService } from '../services/auth.service';
import { LocationService } from '../services/location.service';
import { UserInfoBuilder } from './user-info-builder';
@Component({
  selector: 'num-user-info',
  templateUrl: 'user-info.component.html',
  styleUrls: ['user-info.component.scss'],
})
export class UserInfoComponent {
  user: any;
  message = 'This modal example uses triggers to automatically open a modal when the button is clicked.';
  name: string = '';
  value: any;
  userDetails: FormGroup;
  currentUserInfo: any;
  userImg: any = '';
  @ViewChild('popover') popover: any;
  files: PickedFile[] = [];
  userImageURL: SafeUrl;
  isOpenLocation = false;
  objectURL: any;
  baseImg: string;
  isCurrentUser: boolean;
```

```
@ViewChild(IonModal) modal: IonModal;
constructor(private authService: AuthService,
        private locationService: LocationService,
        private router: Router,
        private userInfoBuilder: UserInfoBuilder,
        private route: ActivatedRoute,
        private sanitizer: DomSanitizer,
        public menuCtrl: MenuController) { }
ngOnInit() {
  const id = this.route.snapshot.paramMap.get('id');
  this.user = this.authService.getUser();
  this.userDetails = this.userInfoBuilder.getForm();
  this.getCurrentUser();
  this.isCurrentUser = this.user.login === id;
}
logOut() {
  this.authService.logOut();
  this.router.navigate(['login']);
}
cancelEdit() {
  this.modal.dismiss(null, 'cancel');
}
confirmEdit() {
  this.userDetails.markAllAsTouched();
  if (this.userDetails.valid) {
  const formPatch = {
    firstname: this.userDetails.controls['firstname'].value,
    lastname: this.userDetails.controls['lastname'].value,
    email: this.userDetails.controls['email'].value,
    description: this.userDetails.controls['description'].value,
    location: this.userDetails.controls['location'].value,
    image: this.baseImg
  }
  this.authService.updateUser(this.user.login, formPatch).subscribe(value => {
    this.getCurrentUser();
    if(value.success) {
      this.modal.dismiss(this.name, 'confirm');
    }
```

```
  })
 }
 }
 selectLocation(location: string) {
  this.userDetails.controls['location'].setValue(location);
  this.isOpenLocation = false;
 }
 getCurrentUser() {
  const id = this.route.snapshot.paramMap.get('id');
  this.authService.getCurrentUser(id as string).subscribe(value => {
   this.currentUserInfo = value.user;
   this.userDetails.patchValue({
    firstname: this.currentUserInfo.firstname,
    lastname: this.currentUserInfo.lastname,
    email: this.currentUserInfo.email,
    description: this.currentUserInfo.description,
    location: this.currentUserInfo.location
   })
   this.baseImg = this.currentUserInfo?.image?.length > 1 ? this.currentUserInfo?.image :
"https://ionicframework.com/docs/img/demos/avatar.svg";
  })
 }
 async pickImages(): Promise<void> {
  const { files } = await FilePicker.pickImages({
   multiple: false,
  });
  this.objectURL = URL.createObjectURL(files[0].blob as Blob);
  this.sanitizer.bypassSecurityTrustUrl(this.objectURL);
  this.files = files;
  this.blobToBase64(files[0].blob as Blob).then((data: any) => this.baseImg = data);
 }
 imgConverter(image: string) {
  return this.sanitizer.bypassSecurityTrustUrl(image);
 }
 deleteImage() {
  this.baseImg = "https://ionicframework.com/docs/img/demos/avatar.svg";
 }
```

```
  blobToBase64(blob: Blob): Promise<any> {
    return new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onerror = reject;
      reader.onabort = reject;
      reader.onload = () => resolve(reader.result as string);
      reader.readAsDataURL(blob);
    })
  }
  onEditWillDismiss(event: Event) {
    const ev = event as CustomEvent<OverlayEventDetail<string>>;
    if (ev.detail.role === 'confirm') {
      this.message = `Hello, ${ev.detail.data}!`;
    } else {
      this.baseImg = this.currentUserInfo?.image;
    }
  }
  onInput(event: any) {
    this.locationService.searchUser(event.detail.value).subscribe(value => {
      this.value = value;
    });
    this.isOpenLocation = true;
  }
}
```