------------------КЛАСИЧНИЙ ПРИВАТНИЙ УНІВЕРСИТЕТ-------------------

(повне найменування вищого навчального закладу)

-------------------------------ІНСТИТУТ УПРАВЛІННЯ------------------------------  
(повне найменування інституту, назва факультету (відділення))

--------КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ-------

(повна назва кафедри (предметної, циклової комісії))

ЗВІТ

З ПЕРЕДДИПЛОМНОЇ ПРАКТИКИ

Студент групи МІ-122---------------------

спеціальності 121 Інженерія програмного забезпечення-------------------------------------

освітньої програми Інженерія програмного забезпечення--------------------------------------

Рябчук Євгеній Андрійович---------------

(прізвище та ініціали)

Керівник

Проф. Кафедри інформаційних технологій  
та дизайну Хрипко С. Л. -----------------------

(посада, науковий ступінь, прізвище та ініціали)

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

Кількість балів: \_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_

Члени комісії

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

**Зміст**

[Вступ 3](#_Toc148936391)

* 1. Основна [суть та мета розробки 3](#_Toc148936392)
  2. [Модель ПЗ та технології реалізації 3](#_Toc148936393)
  3. [Вимоги до розробки 4](#_Toc148936394)

[Опис серверної частини та технологія реалізації (backend) 5](#_Toc148936395)

* 1. [Рівень зберігання даних 5](#_Toc148936396)
  2. [Рівень взаємодії із даними 8](#_Toc148936397)
  3. [Рівень бізнес-логіки 9](#_Toc148936398)
  4. [Рівень API 11](#_Toc148936399)
  5. [Додаткові технології 13](#_Toc148936400)

[Структура та процес розробки клієнтського додатку (frontend) 16](#_Toc148936401)

* 1. [Планування інтерфейсу 16](#_Toc148936402)
  2. [Структура Angular додатку 16](#_Toc148936403)
  3. [Тестування і дослідження швидкодії 18](#_Toc148936404)

# Вступ

## Основна суть та мета розробки

Проект був створений як наглядний приклад демонстрації досліджень проведених для написання магістерської роботи.

Кінцевим продуктом праці є створена повнофункціональна веб-платформа для публікації професійних фотографій різних авторів з метою заробітку на своїй творчості. Найчастіший приклад використання подібних сервісів поширення стокових фотографій, це бажання комерційних компаній швидко та легально знайти матеріал для своїх рекламних компаній.

Мій проект використовує підписну модель з різними рівнями доступу та ліцензіями на використання авторського матеріалу. Монетизаційна схема проекта така: кліент платить за підписку – ці гроші виплачуються авторам в залежності від кількості завантажень їх зображень (ліміт завантажень кліента в місяць залежить від його рівня підписки) – фіксований процент комісійних отримує платформа.

Кожен автор має свій профіль де має змогу заповнити інформацію про себе, оцінити свою статистику (кількість підписників, завантажень, переглядів та відміток), створити підбірку фото, завантажити зображення та перегляну всю свою галерею. Кліент в свою чергу має змогу оцінити роботу автора, завантажити її чи підписатися на нього. Всі користовачі мають свою роль та можливості.

Програмне забезпечення такого роду працює з великою кількістю відносно великих даних (зображення в форматі raw, dng, tiff чи png, jpeg великих розмірів), тому це ідеальний об’єкт для досліджень.

Отже, тема розробки, це створення "природнього" середовища для тестування різних сучасних методів та програмних засобів стиснення зображення для інтернет простору.

Основними показниками для вимірювання ефективності стиснення є швидкість завантаження веб-сторінок в різних розділах сайта, швидкість кодуваня та декодування зображень, об’єм займаного місця на сервері (розмір зображення), якість зображення в порівнянні з іншими методами.

## 

## Модель ПЗ та технології реалізації

Серверна сторона моделі буде представлена ASP.NET Core Web API сервером із REST архітектурою. Структура Web API додатку зазвичай містить чотири основні рівні логіки (рис. 1.1), що реалізують принцип розділення відповідальності.

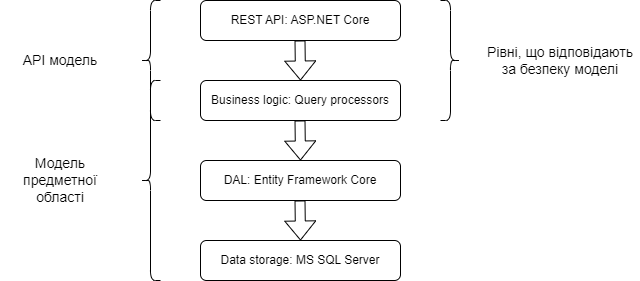


Рис. 1.1. Структура Web API додатку

Клієнтська сторона буде представлена веб сайтом, створеним за допомогою фреймворку Angular. Angular дозволяє будувати складні односторінкові додатки із простою модульною структурою. На рис. 1.2 зображена спрощена схема архітектури Angular додатку.

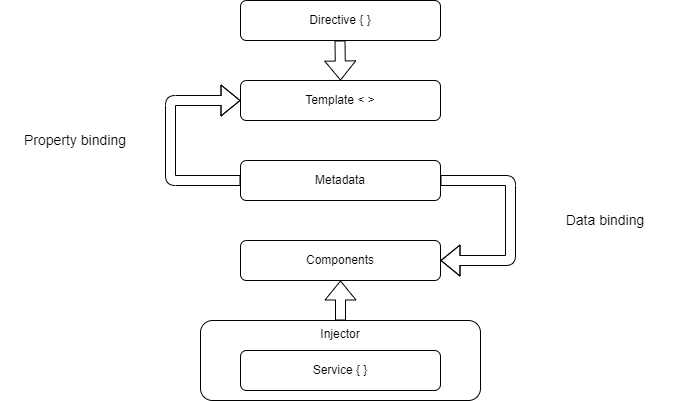


Рис. 1.2. Структура Angular додатку

## Вимоги до розробки

Основною вимогою для кінцевого користувача є можливість швидко та зручно опрацьовувати (переглядати, сортувати, проводити пошук, вносити зміни) велику кількість записів із бази, враховуючи, що дані містять зображення великих розмірів.

Для розробника – модульна модель розробки, що дозволить легко замінювати окремі елементи, можливість швидкого та наочного тестування.

# Структура та процес розробки серверного API

## Рівень зберігання даних

Розглянемо структуру моделі, зображену на рис. 1.1 більш детально. На рівні зберігання даних знаходиться база даних. Для початку, я обрала вже знайому мені СУБД MS SQL SERVER, але при проведенні подальших досліджень планую спробувати інші більш популярні бази, для порівняння впливу обраної СУБД на швидкодію кінцевої розробки. Варто зауважити, що для можливості швидкої заміни СУБД в проекті, необхідно обрати технологію взаємодії із даними, яка підтримуватиме різні СУБД. Такою технологією є Entity Framework Core.

Entity Framework Core – це набір технологій в ADO.NET, що підтримують розробку програмно-орієнтованих на дані програмних додатків. EF Code First – один із інструментів EF, що дозволяє створити БД автоматично, спираючись на моделі даних (Entities) та контекст бази (DbContext), описані в коді. Всього мій проект нараховує 10 класів типу Entity. Кожен з цих класів описує окрему таблицю та зв'язки між ними у майбутній БД. Приклад цих класів та класу контексту можна побачити на рис. 2.1 та рис. 2.2.





Рис. 2.1. Клас DbContext



Рис. 2.2. Клас Part – відповідає таблиці Parts у базі

Після завершення написання всіх класів та запуску двух коротких команд, маємо готову базу, схему якої можна перевірити у SQL Server Management Studio (рис. 2.3). Як можна помітити, кількість таблиць у БД відповідає кількості класів типу Entity у проекті.

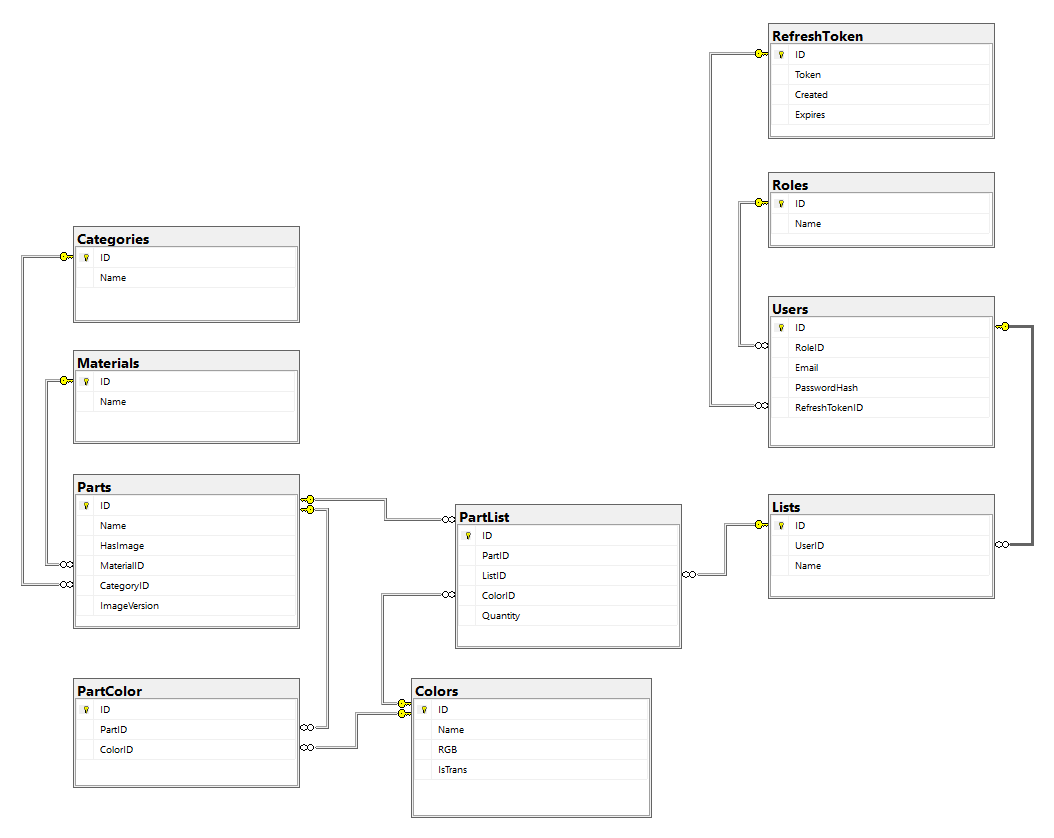


Рис. 2.3. Схема отриманої бази даних

## Рівень взаємодії із даними

Класи групи DataLayer містять відокремлену логіку CRUD (Create, Read, Update, Delete) операцій для взаємодії із базою. Класи групи DataLayer використовують створений раніше класс DbContext як об`єкт підключення до БД та функції, що містяться у бібліотеці Microsoft.EntityFrameworkCore.Query для створення запитів до бази. На рис 2.4 наведено деякі функції класу Parts\_DAL, що працює із таблицею Parts.

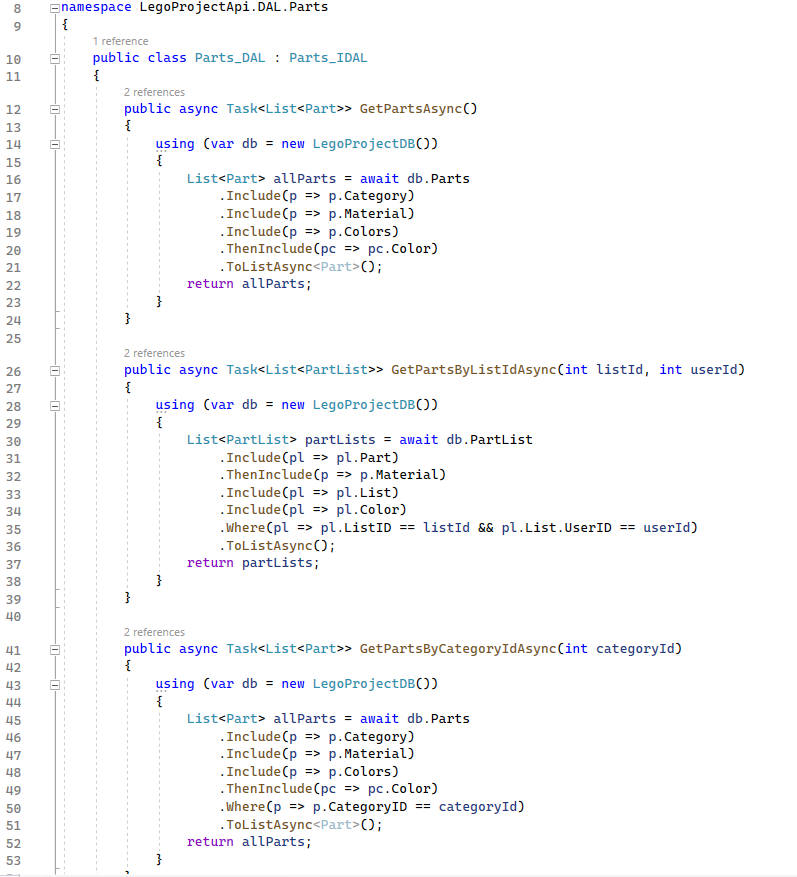


Рис. 2.4. Клас Parts\_DAL

## Рівень бізнес-логіки

На рівні бізнес-логіки знаходяться класи-сервіси, що описують логіку обробки даних між контролерами та рівнем доступу до даних. Кожному класу типу DAL відповідає окремий сервіс. Загалом структура сервісу повторює структуру DAL класу, проте CRUD функції у сервісах містять додаткову логіку обробки даних, що має відбуватися на серверній стороні. В сервісах також знаходиться логіка, що відповідає за безпеку даних. На рис 2.5 зображено деякі функції класу BrickService, що відповідає таблиці Parts.



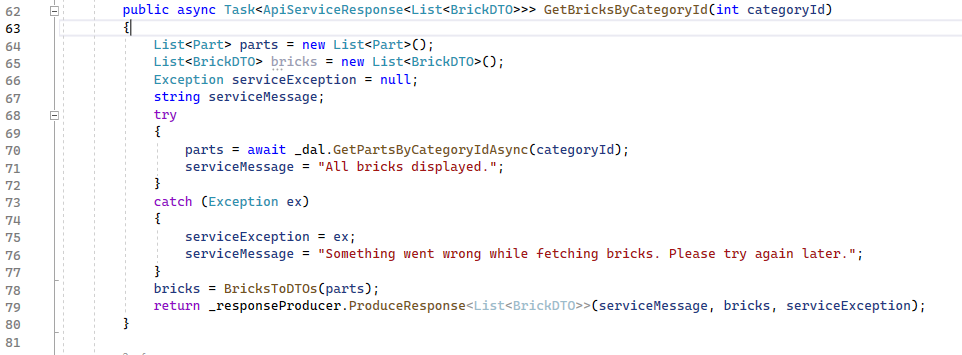
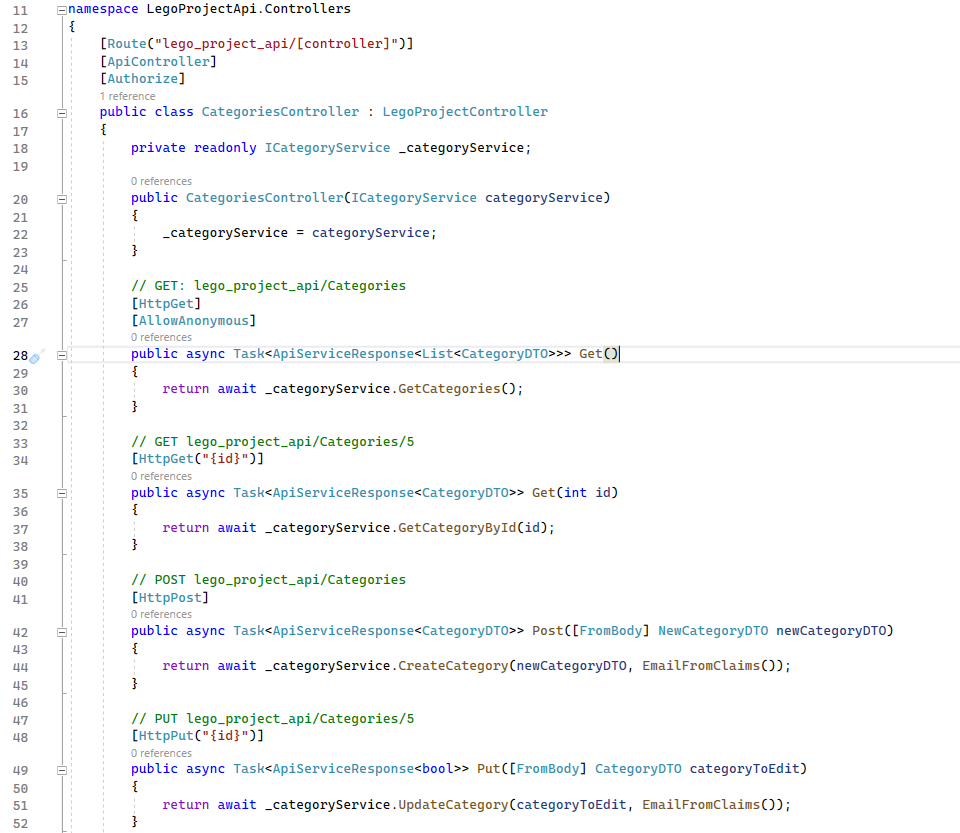


Рис. 2.5. Клас BrickService

## Рівень API

Класи-контролери у Web API відповідають за обробку Http - запитів. Запит приймається у вигляді об'єкту HttpRequestMessage, оброблюється за допомогою методів-сервісів, після чого генерується відповідь, що відправляється на клієнт у вигляді об'єкту HttpResponseMessage. На рис. 2.6 зображено контролер, що опрацьовує всі запити, пов'язані із категоріями деталей (таблиця Categories у БД).



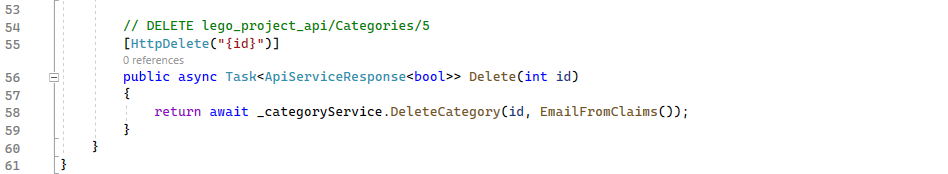


Рис. 2.6. Клас CategoriesController

Data Transfer Object – один із шаблонів проектування, що використовується для передачі даних між підсистемами проекту (між сервером і клієнтом). Загалом класи DTO повторюють структуру DbEntities, проте вони можуть містити додаткові поля, що не зберігаються у базі, або містити лише частину полей, що необхідні для конкретного запиту. На рис. 2.7 зображено клас типу DTO, що містить скорочений перелік полей класу Brick, які необхідно відобразити, коли користувач бажає переглянути деталі, що знаходяться в окремому інвентарному списку.

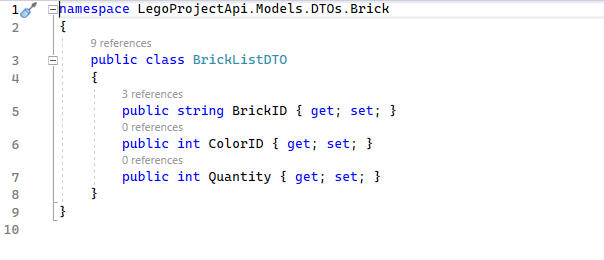


Рис. 2.7. Клас BrickListDTO

На цьому етапі основна структура і функціонал серверного додатку повністю завершені. На рис 2.8 зображена файлова структура проекту, що відповідає схемі представленій у першому розділі на рис 1.1.

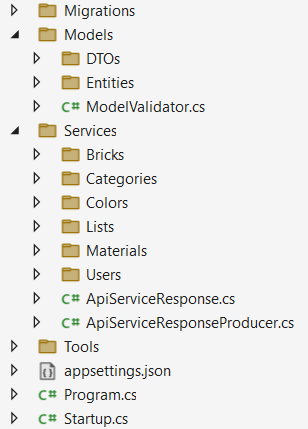
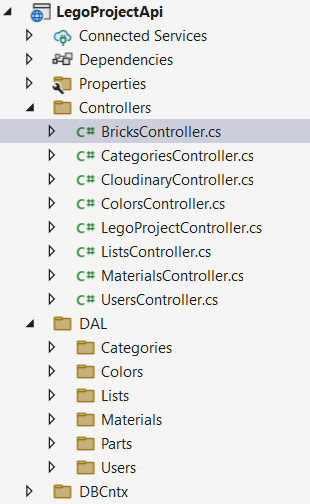


Рис. 2.8. Файлова структура проекту серверного додатку

## Додаткові технології

Окремої уваги потребує транспортування конфіденційних даних користувача (логін та пароль) між клієнтом і сервером. JSON Web Token – один із поширених стандартів створення токенів доступу, що використовуються для безпечної передачі даних при аутентифікації у клієнтсько-серверних додатках.

Для простого тестування серверного додатку використаємо Swagger API – фреймворк, що дозволяє автоматично створювати документацію та тестовий інтерфейс (рис. 2.9) для розробника для Web API додатків на основі коду.

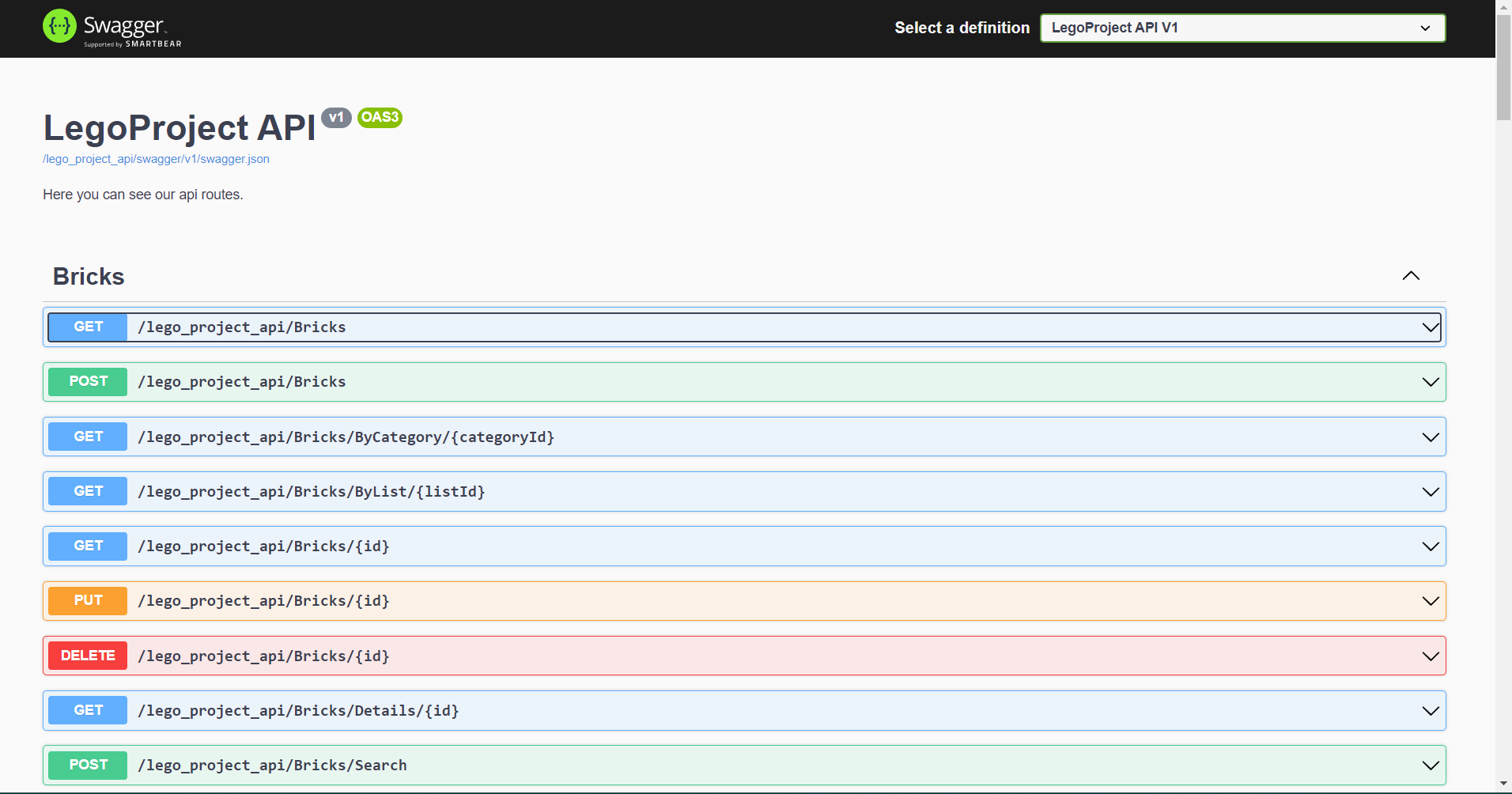


Рис. 2.9. Інтерфейс, створений за допомоги фреймворку Swagger

Використання хмарних ресурсів для зберігання зображення і відео дозволяє зменшити як навантаження на власну БД так і вимоги до машини, на якій буде розгорнуто сервер. Високі обчислювальні можливості хмарних серверів дозволяють не лише швидко надавати доступ до ресурсів, а і проводити різноманітні маніпуляції із ними (змінювати розмір і формат зображень, застосовувати графічні ефекти тощо.), не зберігаючи при цьому їх копії. Одним із таких ресурсів є Cloudinary.com – хмарний хостинг зображення та відео. Cloudinary надає API та зручну документацію для роботи із хмарним сховищем у бекенд і фронтенд додатках на всіх поширених мовах програмування. На рис 2.10 – 2.11 зображені функції для роботи із хмарним сховищем на бекенді та фронтенді. На рис 2.12 – інтерфейс хмарного сховища.



Рис. 2.10. Функція для запиту зображення із клієнтського додатку (Angular, TypeScript)

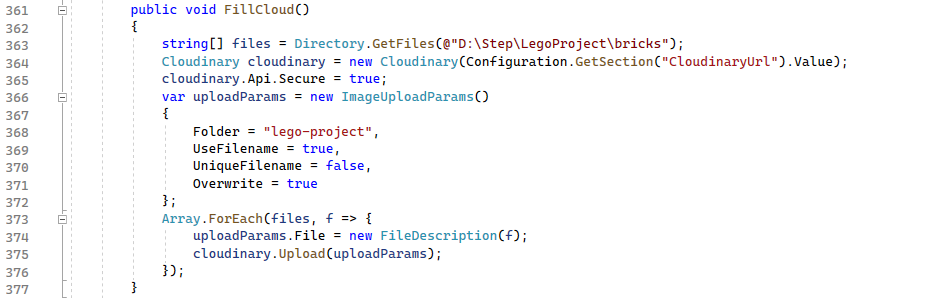


Рис. 2.11. Функція для завантаження зображень із файлової системи у хмарне сховище (.NET, C#)

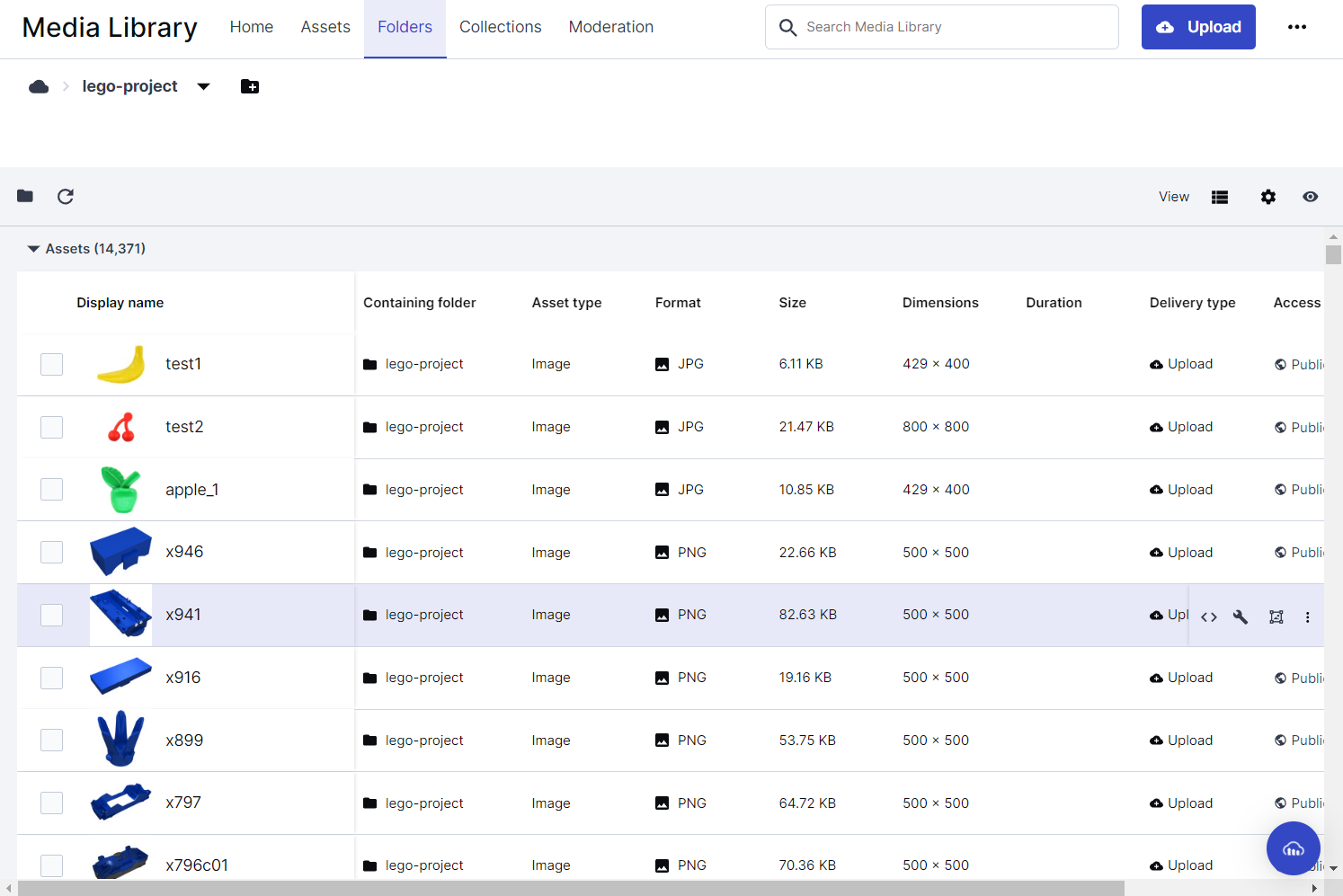


Рис. 2.12. Зображення у папці проекту на хмарному сервері

# Структура та процес розробки клієнтського додатку

## Планування інтерфейсу

Під час розробки інтерфейсу для додатку заданої тематики необхідно враховувати, що основне призначення додатку – відображення великої кількості даних на екрані. Відображені дані мають легко зчитуватися, а допоміжні елементи інтерфейсу не заважати роботі з ними. На рис. 3.1 зображено спрощену схему інтерфейсу, де більшу частину простору займає таблиця із даними, а весь функціонал скомпоновано у боковій панені меню, яку можна згорнути, для більш комфортної роботи з таблицею.

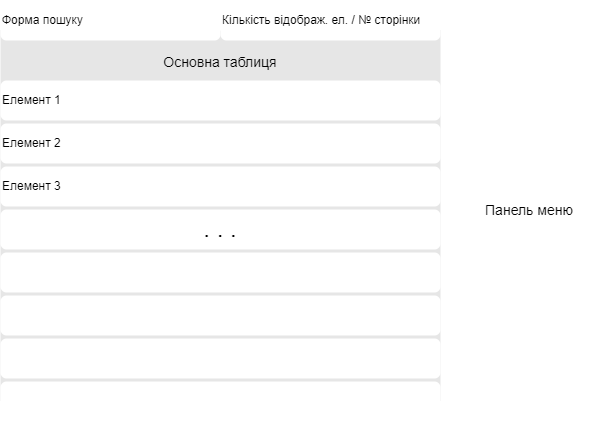


Рис. 3.1. Макет інтерфейсу

## Структура Angular додатку

Клієнтський додаток на Angular складається із декількох модулів, що утилізують функцію динамічного завантаження, зменшуючі таким чином час очікування завантаження для кінцевого користувача. Мій проект містить п`ять основних модулів: Login, Register, About, Main та Dev (інструменти розробника для тестування). Завантаженням модулів керує роутер (рис. 3.2).

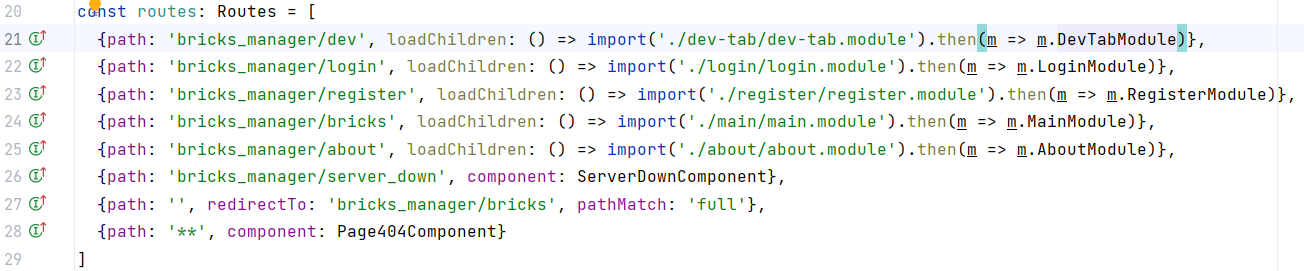


Рис. 3.2. Роутинг клієнтського додатку

Крім основних модулів клієнтський додаток містить окремі компоненти, що використовуються в цих модулях. Такі компоненти описують зовнішній вигляд та поведінку таких елементів меню як: діалогові вікна, списки, меню, складні елементи таблиці тощо.

Angular надає велику кількість готових елементів інтерфейсу із можливістю налаштувати їх зовнішній вигляд та поведінку. Деякі комплексні елементи, як наприклад списки та таблиці, дозволяють автоматично виводити елементи колекцій за заданим шаблоном та мають готову логіку сортування, фільтрації, пагінації тощо.

У файловій структурі модуль відображається наступним чином (рис. 3.3): файл module.ts містить використані залежності та внутрішній роутинг модуля, файл component.ts – бізнес логіку модуля, component.html – html розмітку сторінки, component.scss – застосовані стилі.



Рис. 3.3. Модуль у файловій структурі

На рис 3.4 зображено готовий головний модуль додатку із таблицею і відкритим меню.

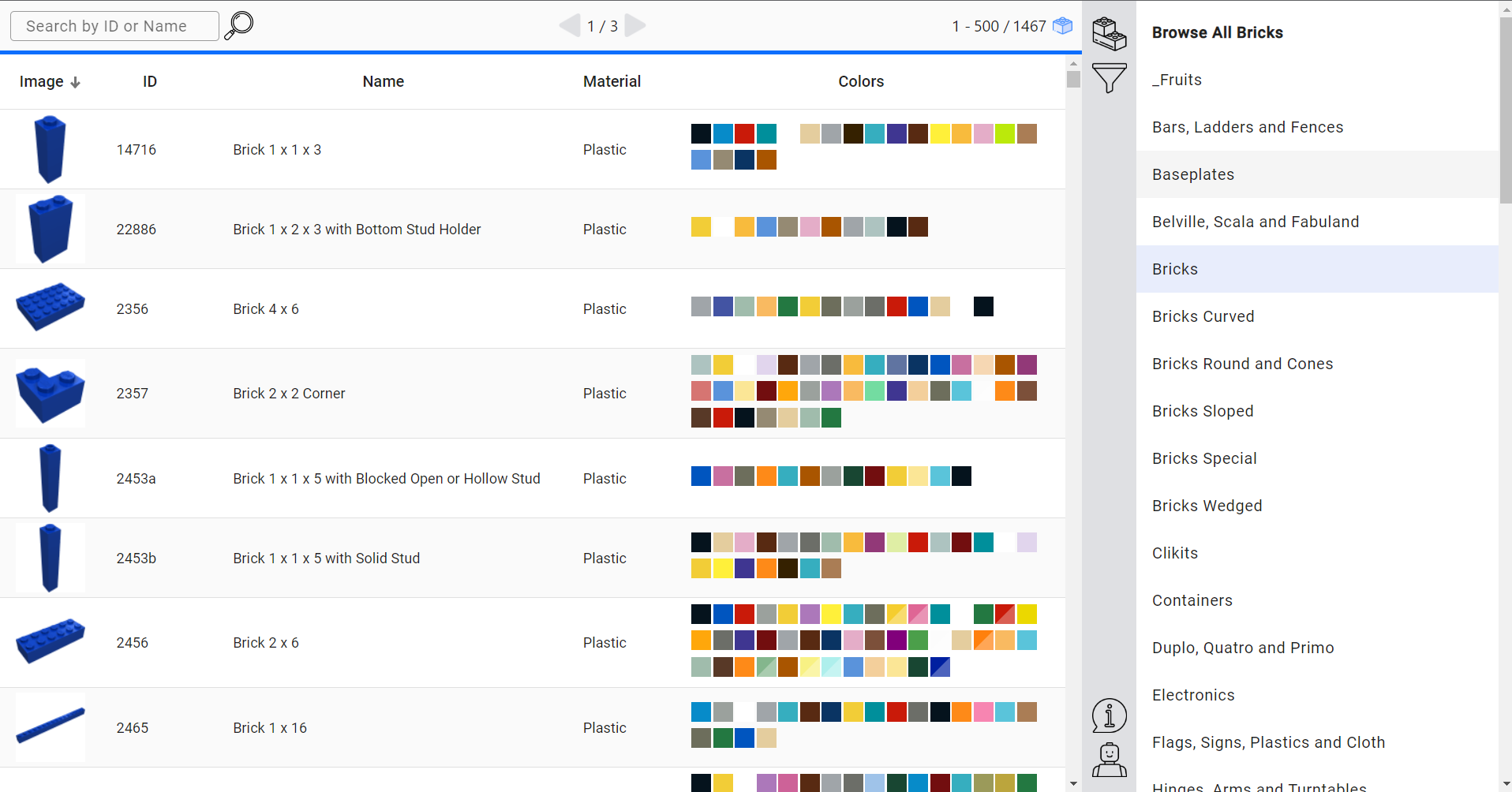


Рис. 3.4. Клієнтський додаток у браузері

## Тестування і дослідження швидкодії

Для подальшої роботи над даним проектом і підготовки до захисту дипломної роботи, необхідно додати можливість тестування впливу змін у коді на швидкість обробки запитів. Використовуючи інтерфейс HttpInterceptor можна виміряти час від надсилання http запиту до отримання відповіді. Побудувати графіки для візуалізації часових вимірів можна за допомогою бібліотеки NGX-CHARTS (рис. 3.5).

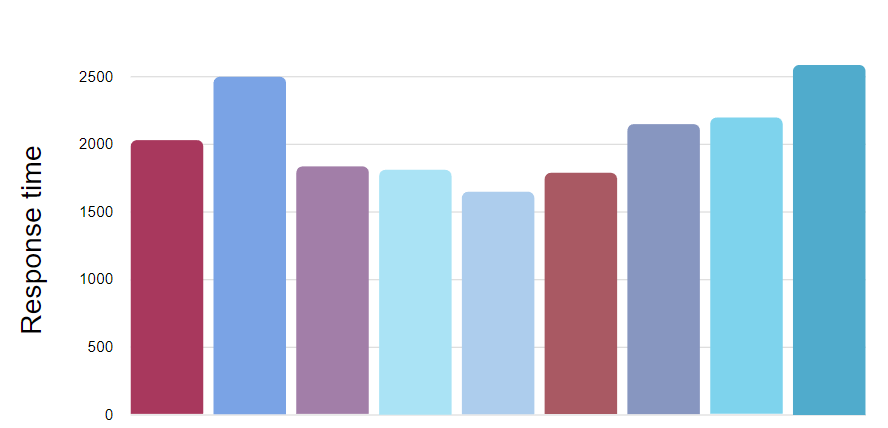


Рис. 3.5. Приклад роботи бібліотеки NGX-CHARTS