МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра інформаційних систем

Дипломна робота

Розробка веб-додатку для обміну та збереження фотографій
на мікросервісній архітектурі

Виконав: студент групи ПМі-44
спеціальності
122 «Комп'ютерні науки та інформаційні технології»

Кізло Т. М.

Керівник Бернакевич І. Є.

Рецензент

Львів – 2020

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *інформаційних систем*

Освітньо-кваліфікаційний рівень *бакалавр*

Напрям підготовки *6.050101*

Спеціальність *122 «Комп'ютерні науки та інформаційні технології»*

«ЗАТВЕРДЖУЮ»

Зав. кафедрою
проф.Шинкаренко Г.А.

« 9 » *вересня* 2019 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ (КВАЛІФІКАЦІЙНУ) РОБОТУ СТУДЕНТА

Кізло Тараса Михайловича
(прізвище, ім'я, по батькові)

1. Тема роботи
Розробка веб-додатку для обміну та збереження фотографій на мікросервісній архітектурі
керівник роботи *доц. Бернакевич І.Є.*
затверджені Вченою радою факультету від «_» ___ 20*19* р., №

2. Строк подання студентом роботи *10.06.2020*

3. Вихідні дані до роботи
Література та інтернет-ресурси за тематикою роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд існуючих систем та технологій
2. Постановка задач
3. Проєктування доменної моделі
4. Розробка архітектури аплікації
5. Програмна реалізація
6. Апробація
7. Оформлення роботи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація дипломної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *25.09.19*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд існуючих систем та технологій*	*вересень 2019*	
2.	*Постановка задач*	*жовтень 2019*	
3.	*Проєктування доменної моделі*	*листопад 2019*	
4.	*Розробка архітектури аплікації*	*грудень 2019*	
5.	*Програмна реалізація*	*січень 2020 – березень 2020*	
6.	*Апробація*	*квітень 2020*	
7.	*Оформлення роботи*	*травень 2020*	

Студент _____ *Кізло Т. М.*
підпис

Керівник роботи _____ *доц Бернакевич І.Є.*
підпис

ЗМІСТ

ВСТУП

У сучасному світі інформація являється одним із найважливіших соціальних ресурсів. Вона здатна допомогти людині адаптуватися у житті в умовах невизначеності, пристосуватися до постійних змін, виробити нові стереотипи поведінки, що відповідають новим обставинам. Інформація являється неоціненим те необмеженим джерелом знань, яке дозволяє нам освоїти нові науки, отримати натхнення та розв'язати, до того не осяжні, завдання.

Однак інформація втрачає свою цінність без можливості обміну. Люди являються соціальними істотами. Нам важко прожити без можливості проявити свою унікальність. Взаємодія з іншими людьми дозволяє нам обмінюватись інформацією, набувати досвіду минулих поколінь та перевершувати їх у навичках.

Сьогодні обмін інформації та знань один із найлегших процесів, який таким не був ще століття чи два назад. А все завдяки інтернету, соціальним мережам та сховищам даних. Більше не мають значення географічна віддаленість, соціальний розвиток чи економічне становлення людей.

На цей час одним із найпопулярнішим способом обміну інформації являються соціальні мережі. Вони становлять важливу частину життя будь-якої людини. В них зареєстрований практично кожен. Причому дані ресурси зуміли підкорити користувачів будь-якого віку, від малого до великого. Різноманітних соціальних мереж стає все більше. Вони відрізняються функціоналом чи способом спілкування. Але мета їх однакова — об'єднати людей із однаковими чи не дуже інтересами.

Серед соціальних мереж найбільшими фаворитами вважають наступні:

а) Lenkedln. Сюди входять в основному представники бізнес-структур. Та зазвичай обмінюються науковими працями та діляться новинами;

б) Tumblr. Улюблене місце блогерів. Вони ведуть свої блоги, коментують і читають чужі;

в) Instagram. Досить швидко зайняв провідне місце серед соціальних мереж. Користувачам сподобалася можливість ділитися відео та фотознімками;

г) Pinterest. Тут можна обмінюватися ідеями на різні теми, рецептами, проєктами у вигляді графічних файлів;

д) Flikr. Даний проєкт дозволяє пересилати велику кількість фотографій;

е) MySpace. Використовується для: обміну фото, відео, повідомленнями;

ж) Scribd. В основному використовується для обміну книгами;

Можна побачити, що всі соціальні мережі тою чи іншою мірою дозволяють обмінюватись інформацію. Вони використовують свої внутрішні сховища даних, до яких користувач немає доступу. Іншим недоліком є те що, перераховані ресурси розпоряджаються файлами своїх клієнтів на власний розсуд. Таким чином у разі досягнення дозволеного ліміту, старі файли можуть стиратись.

У такій ситуації на допомогу приходять хмарні сховища зберігання даних. Попри те що технології невпинно розвиваються, у нас все одно виникають проблеми із максимальним обсягом віртуальної пам’яті, яку ми можемо використати у наших цілях. Саме цю проблему і намагаються вирішити хмарні сховища. Вони дозволяють зберігати ресурси не на власному пристрої, а в спеціальному сховищі, яке можна досягнути при потребі за допомогою інтернету.

Хмарні сховища даних надають особливу гнучкість в доступі до інформації, через те, що ми не обмежені використанням лише одного пристрою, а можемо отримати файли за допомогою комп’ютера чи телефона, тощо. У нас більше немає потреба переживати про надійність зберігання, оскільки вся інформація зберігається у віддаленому сховищі, яке містить декілька копій та при втраті однієї копію завжди є можливість використати іншу. Але основна можливість — це обмін ресурсами з іншими користувачами.

Розглянемо приклади різноманітних сховищ даних, та їх особливості:

а) Dropbox. Надає широке різноманіття тарифних планів в залежності від обсягу та надійності сховища;

б) Google Drive. Дозволяє мати необмежену кількість файлів створених за допомогою вбудованих сервісів сховища;

в) Mega. Містить покращений алгоритм шифрування даних;

г) Google Photo. Необмежене сховище для зберігання фотографій;

д) OneDrive. Продукт компанії Microsoft, який має вбудовану синхронізацію з іншими сервісами цієї компанії;

е) Youtube. Надає можливості збереження та перегляду відеоматеріалів;

Можна побачити, що великої популярності набувають ті ресурси, які спеціалізуються на одному конкретному форматі файлів, та розвивають свої технології саме у цій сфері. Це стосується як соціальних мереж так і хмарних сховищ. І це не дивно, адже конкуренція у цих сферах зазвичай менша, а можливості розширення функціоналу більші.

Так, наприклад, соціальні мережі, що свідомо вирішили відмовитись від стандартних засобів комунікації (розмова, листування тощо) та перейти до більш екстравагантних, таких як обмін картинками, відео, аудіо тощо, все більш популярні у сучасному суспільстві.

Instagram — соціальна мережа, що базується на обміні фотографіями, дозволяє користувачам робити фотографії, застосовувати до них фільтри, а також поширювати їх через свій сервіс і низку інших соціальних мереж.

Подібний функціонал має Flickr — вебсайт для розміщення фотографій та відеоматеріалів, їх перегляду, обговорення, оцінки та архівування. Flickr популярний завдяки зручній та простій системі завантаження та пошуку фотографій. Дозволяє спілкуватися та створювати тематичні групи, соціальні мережі.

Власник фотографій має можливість:

а) обмежити доступ до своїх фото;

б) встановити умови використання фото;

в) супроводжувати свої фото коментарями;

г) дозволяти коментувати свої фото;

Чи Pinterest — соціальний фотосервіс. Його основний функціонал полягає в завантажені фото на певну тематику, та знаходження подібних за допомогою алгоритмів програми. Після того, як зображення завантажені на Pinterest, вони називаються пінами, а колекції, до яких вони належать, — дошки. Місія сайту звучить, як «об’єднати весь світ за допомогою речей, які їм цікаві».

Серед хмарних сховищ особливо популярним являється Google Photo. Він надає великі можливості з пошуку та сортування фотографій. Також користувачі цього сервісі можуть зберігати необмежену кількість фотографій у високій якості. Є можливість видалити фотографії та відновити їх протягом певного періоду часу. Головна особливість — це вбудований редактор фотографій. Він дозволяє змінити яскравість чи контрастність фото, застосувати різноманітні фільтри, тощо.

При цьому хмарні сховища надають можливість обмінюватись файлами. Робити до них замітки. Бачити реакції інших користувачів, тощо. Таким чином, на цей час важко визначити межу між соціальними мережами та хмарними сховищами, оскільки їх функціонал частково, або повністю повторюється.

Не дивно, що найпопулярніші сервіси орієнтовані на фотографії. Фотографії настільки увійшли в наше буденне життя, що сьогодні ми ледве усвідомлюємо їх істинне значення. Ми фіксуємо моменти життя, щоб зберегти їх у спогадах, бо наша пам'ять мінлива. Ми відчуваємо необхідність в них самі того не усвідомлюючи.

Як бачимо, все більше сервісів не безпричинно фокусуються на фотографіях. Саме на їх основі і виникла ідея створення веб додатку для обміну та збереження фотографій. Цей сервіс повинний об'єднувати переваги даних мереж:

а) Instagram — можливість обміну фото;

б) Flickr — розміщення фотографій та відеоматеріалів, їх перегляду, обговорення, оцінки та архівування;

в) Pinterest — розширений пошук фото на спільну тематику, оптимізаційні способи збереження мінімальної фізичної кількості фото, унормована політика завантаження та фільтрації фото;
г) Google Photo — необмеженість пам’яті та вбудований фоторедактор.

Та не буде містити їх недоліків:

а) Instagram — рекламні профілі, погані алгоритми пошуку фото;
б) Flickr — обмежений об’єм пам'яті для збереження фото;
в) Pinterest — специфікації на обмежену групу аудиторії, спам та реклама у деяких фото;
г) Google Photo — втручання у приватні дані користувачів з метою формування таргетної реклами;

СУТЬ ДИПЛОМНОЇ РОБОТИ

Суть даного завдання полягає у створенні веб додатку для обміну та збереження фотографій та дослідженні мікросервісної архітектури. Також будуть реалізований функціонал групування фотографій в альбоми, можливість коментувати фотографії, поміщати їх в кошик, отримувати сповіщення про зміни в даних, та основний акцент буде зроблено алгоритмах пошуку фотографій.

Користувач зможе авторизуватись у системі за допомогою соціальних мереж, таких як Google, Facebook та Twitter. У випадках коли цих опцій недостатньо чи вони не задовільняють потреби, для користувача буде надана можливість створити профіль у системі за допомогою власної поштової адреси та пароля. При потребі ці дані можна буде відновити, за допомогою повідомлення, що прийде на зареєстровану пошту.

Після реєстрації користувач отримує доступ до свого профілю, де зможе переглянути введені дані. Профіль являється приватним, та інші користувачі не мають доступу до нього.

Щоб мати змогу працювати із системою необхідно завантажити фотографії. Додавання фото буде здійснене за допомогою вікна перетягування чи спеціальної кнопки. Таким чином користувач з легкістю може обрати декілька фотографій та перетягнути їх на інтерфейс програми. Перед збереженням фотографії, її можна вилучити зі списку чи змінити опис.

Після завантаження фотографій у користувача буде можливість вибрати декілька фото, для того щоб видалити їх чи завантажити із хмарного сховища на свій пристрій.

Завантажені фотографії можна переглядати у повноекранному режимі, змінювати їм назву та опис, додавати коментарі, ділитись з іншими користувачами за допомогою електронної адреси та змінювати їх у спеціальному фоторедакторі.

Фоторедактор надає наступні операції:

а) Зміна розміру зображення;

б) Поворот та віддзеркалення фото;

в) Зміна яскравості чи контрасту;

г) Застосування одного із багатьох фільтрів;

Завантажені фотографії можна буде групувати по альбомах. Для самих альбомів передбачені операції створення, редагування та видалення.

Користувач зможе ділитись фотографіями з іншими користувачами за умови, що знає їхню електронну адресу. Якщо хтось надав доступ до своєї фотографії користувач повинен отримати про це спеціальне сповіщення.

Для пошуку фотографій має бути спрощений функціонал, який вимагатиме від користувача лише введення тексту, а сам пошук буде наскрізний — тобто такий, що ігноруватиме помилки в тексті та видаватиме результати по кращому збігу. Основну увагу сайту буде звернено саме на можливість поглибленого пошуку використовуючи такі функції як повнотекстовий пошук, що дасть змогу ефективно і навіть не знаючи всієї інформації шукати потрібні матеріали, оскільки повнотекстовий пошук дає змогу здійснювати пошук за ключовими словами в будь-якому порядку.

При пошуку фотографії будуть враховуватись наступні поля:

а) назва фото;

б) опис фото;

в) коментарі під фото;

г) назва альбому, в якому знаходиться фотографії;

д) пошук по фотографіях інших користувачів, до яких є доступ;

При переміщенні фотографії в кошик, користувач матиме можливість відновити фотографію протягом певного періоду часу. В іншому випадку система видалить файл автоматично. По завершенні цього терміну із системи будуть видалені не лише дані фотографії, а також коментарі, інформація в альбомах тощо.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Фізична постановка задачі

Для користувачів такої інформаційної системи будуть надані наступні можливості:

а) створення персонального профілю. Користувач вводить лише електронну адресу, який перевіряється на валідність та унікальність, а також пароль;

б) додавання багатьох фото в одному вікні та їх перегляд;

в) завантаження збережених фото із сховища на власний пристрій

г) можливість видалення та відновлення фото;

д) перегляд фото у розширеному форматі із їхніми даними (опис, дата завантаження, коментарі тощо);

е) редагування фото. Цей функціонал передбачає можливість зміни кольорової схеми зображення: на світлішу чи темнішу, вибір наперед виготовлених кольорових тем: сепія, чорно-біла тема, тощо. Користувач також зможе повертати фото чи відзеркалити, щоб встановити їх правильну позицію, та обрізати фото до необхідного розміру;

ж) при наведенні на фото користувача буде бачити коротку інформацію, кількість коментарів, підпис до фото та її назву;

з) додавання та видалення коментарів під своїм фото, або видалення своїх коментарів під чужим фото. Коментар користувача, буде містити не лише фото, ім’я та текст користувача, а також автоматично згенеровану дату коментаря;

и) групування фотографій відповідно до тематики по альбомах;

к) можливість додати/ видалити фотографію з альбому, але не з профілю в цілому;

л) отримувати сповіщення про фотографії, до яких надали доступ інші користувачі;

1.2 Програмна постановка задачі

Як можна побачити дана програма передбачає велику кількість можливостей для звичайного користувача. Тепер коли ми визначились, які вимоги накладаються на нашу програму, залишилось вирішити за допомогою якої мови програмування буде побудована ця інформаційна система. Також варто вибрати технології розробки та платформу на якій буде розгортатись програма.

Веб аплікації передбачають поділ системи на клієнта (частину із якою взаємодії користувач) та сервер (важко навантажений модуль, що здійснює всі обчислення та відповідає із взаємодію з клієнтами), ***рисунок 1.1***.

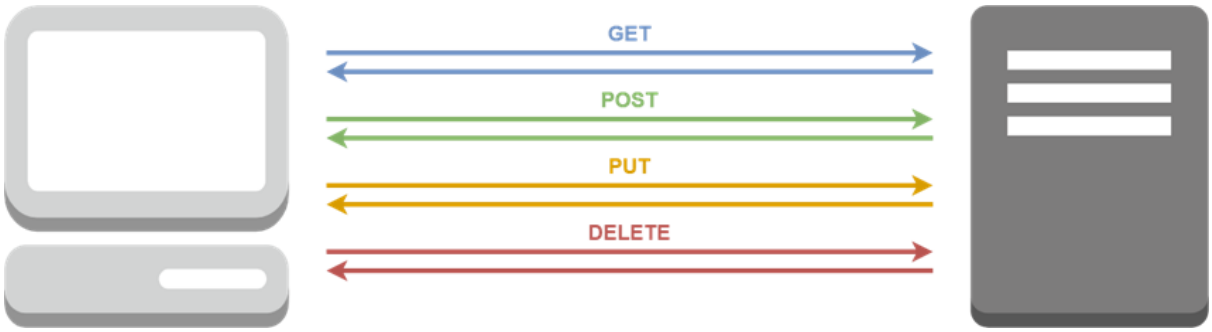

Рисунок 1.1 — Модель взаємодії клієнт-сервер

При цьому доступ до інформаційних ресурсів буде здійснюватись за допомогою методів HTTP запиту, згідно до ***таблиці 1.1***. Таким чином кожна із дій користувача відповідатиме наступному методу:

Таблиця 1.1 — Приклад відповідності дій користувача HTTP методам

Дія	HTTP метод
Читання даних	Get
Створення даних	Post
Редагування даних	Put
Видалення даних	Delete

Основними підходами при розробці веб аплікації являються односторінкові застосунок та багатосторінковий. Розглянемо їх переваги та недоліки, згідно ***таблиці 1.2***:

Таблиця 1.2 — Порівняння односторінкових та багатосторінкових застосунків.

	Односторінковий застосунок (SPA)	Багатосторінковий застосунок (MPA)
переваги	– кращий досвід у використанні для користувача, оскільки не відбувається перезавантаження сторінки; – швидкодія; – незалежність розробки клієнтської аплікації та сервера;	– простота; – Оптимізація для пошукових систем; – надійніший захист;
недоліки	– складність; – навантаження на клієнта;	– оновлення сторінки на кожну дію;

У багатосторінкових застосунках сервер присилає повноцінну сторінку клієнту на кожний запит, ***рисунок 1.2***. Що приводить до оновлення сторінки. Даний підхід популярний серед багатьох сайтів, наприклад, Habr чи DOU.

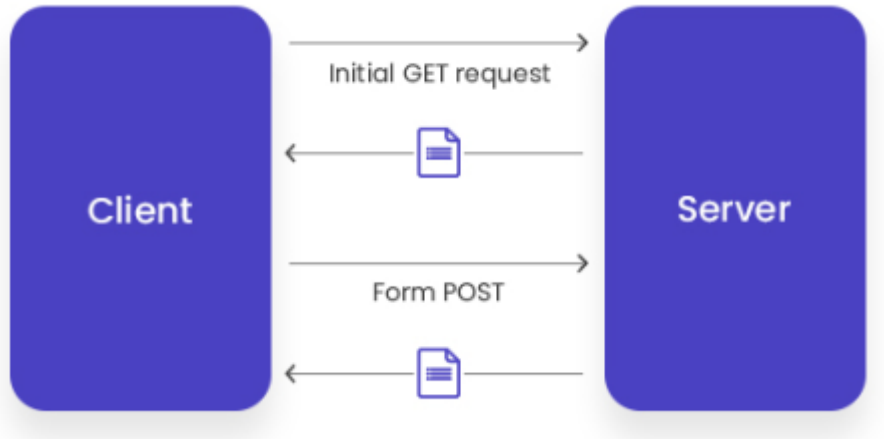

Рисунок 1.2 — Приклад взаємодії між клієнтом та сервером для багаторінкових застосунків.

В той час як в односторінкових застосунках сервер присилає лише необхідні дані у визначеному наперед форматі, ***рисунок 1.3***. Це зменшує розмір передачі даних та призводить до оновлення не всієї сторінки, а лише необхідного компонента. Типові представники Youtube, Google Mail чи Twitter.

Рисунок 1.3 — Приклад взаємодії між клієнтом та сервером для односторінкових застосунків.

Обидва підходи, актуальні на цей момент, але оскільки багато операції в системі передбачені на стороні клієнта їх буде зручніше реалізовувати в односторінковій аплікації.

Для розробки клієнтської частини я буду використовувати такі основні технології як HTML, CSS та JavaScript. Та вони надають лише основний каркас. Щоб спростити розробку та уникнути недоліків цих технологій я також використаю наступні засоби.

а) Bulma — фреймворк для швидкої побудови адаптивної розмітки сайту;
б) Less — це динамічна мова стилів, яка інтерпретується в каскадні таблиці стилів;
в) TypeScript — мова програмування, представлена Microsoft восени 2012, що розширює можливості JavaScript, а саме основний недолік — відсутність типізації;

Для побудови клієнта я обрав фреймворк Angular, який розробляється компанією Google. Його основні переваги:

а) модульність;
б) поділ на компоненти;

в) динамічне завантаження модулів;

г) асинхронність;

д) велика інфраструктура із багатьма вбудованими інструментами;

Для роботи із даними я обрав бібліотеку RXJS, яка надає можливість реактивного програмування. А для управління станом програми — Redux. Композиції цих бібліотек дає NGRX — один із популярних підходів для керуваннями даними в односторінкових застосунках із використанням Angular.
Для обробки серверної частини аплікації, я обрав C#. C# — об'єктноорієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft).

Серед переваг мови C# можна визначити наступні:

а) безпечний ООП підхід, звичайне наслідування, множинне наслідування реалізується за допомогою інтерфейсів;

б) не потрібно використовувати вказівники;

в) автоматичне управління пам’яттю за допомогою збирання сміття. Враховуючи це, ключове слово delete в C# не зарезервоване;

г) формальні синтаксичні конструкції для класів, інтерфейсів, структур, переліку і делегатів;

д) перевантаження операцій для спеціальних типів без зайвої складності;

е) підтримка узагальненого програмування;

ж) підтримка анонімних методів;

з) підтримка суворо типізованих запитів LINQ;

и) безпека типів;

к) підтримка всіх можливостей середовища та сучасних технологій;

Проєкт буде створений на платформі .NET Core— це програмна платформа для побудови програм на базі сімейств операційних систем Windows Linux, macOS, а також інших систем.

Мова програмування C# надає великі можливості та незліченну кількість фреймворків для розробки веб аплікацій. Для розробки серверної частини

аплікації, я буду використовувати Active Server Pages . Це одна із провідних технологій розробки програмного забезпечення, яка отримала свою популярність, завдяки простоті та активній підтримці.

Оскільки дана програма оперує великою кількістю даних, то варто їх зберігати в Базі даних. Серед різноманітних систем керування базами даних, я використовую MS SQL Server. Вона являється однієї із найбільш популярних СКБД та чудово підходить для різноманітних аплікацій: від невеликих додатків, до великих, багато навантажених проєктів.

Його основні особливості:

а) Продуктивність та швидкодія при великих навантаженнях;

б) Надійність і безпека, завдяки можливості шифруванню даних;

в) Простота та зручний користувацький інтерфейс для адміністраторів;

г) Реалізації реляційної моделі Едагара Кодда, яка на сьогодні являється стандартом для організації баз даних;

Для спрощення роботи з базою даних використаємо модуль Entity Framework, який реалізовує об'єктноорієнтований підхід для доступу до бази даних. Використання цього модуля допоможе робити проєкцію даних на колекції C# та пришвидшуватиме роботу програми.

Цей фреймворк пропонує зручну роботу із класами сутностями, які описують моделі таблиць, у вигляді класів. Робота із базою даних, виглядає подібно до роботи зі звичайною колекцією мови C#.

Важливою частиною системи являються фотографії. У систем контролю баз даних, є спеціальний тип даних, призначений, в першу чергу, для зберігання зображень, аудіо та відео. Однак підтримка базами даних великих двійкових об'єктів не є універсальною. Саме тому збереження файлів відбуватиметься на спеціальному сховищі — Blob Storage. Дане сховище надає зручний інтерфейс, та дозволяє за іменем файлу отримати фізичний об’єкт.

Іншою вагомою особливістю аплікації являється повнотекстовий пошук. Стандартні системи контролю версій мають обмежений набір команд пошуку. У такому разі я вирішив зберігати дані про фото за допомогою ElasticSearch.

Elasticsearch — вільне програмне забезпечення, пошуковий сервер, розроблений на базі Lucene. Він надає розподілений, повнотекстовий пошуковий рушій з HTTP вебінтерфейс і підтримкою без схемних JSON документів.

Окрім цього, щоб пришвидшити роботу деяких запитів, мені потрібне спеціальне сховище для кешування даних. Я обрав Redis. Redis — це розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання за бажанням користувача. На відміну від кешування в пам’яті, Redis забезпечує постійне зберігання даних на диску і гарантує збереження даних у разі аварійного завершення роботи.

Також мені необхідний спосіб взаємодії між сервісами. Однією із платформ, що реалізує систему обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP являється RabbitMq. Також як альтернативу я буду використовувати Azure Service Bus.

По завершені роботи над аплікацією, аби всі користувачі мали до неї доступ мені варто її запустити на сервері. З цією метою, я використаю Microsoft Azure — це хмарна платформа та інфраструктура корпорації Microsoft, призначена для розробників застосунків хмарних обчислень і покликана спростити процес створення онлайнових додатків. Але Azure надасть мені лише місце де буде розміщуватись моя аплікація. Для того щоб спростити сам процес її розгортання я використовуватиму Docker. Він надає ряд наступних можливостей:

а) розміщення в ізольованому оточенні виконуваних файлів, бібліотек, файлів конфігурації, скриптів тощо;
б) підтримка роботи на комп'ютерах із різною архітектурою;
в) ізоляція на рівні мережі;

2 ПРОЄКТУВАННЯ

Щоб спростити процес розробки та уникнути багатьох типових помилок варто виділити час на проєктування. Спробуємо як найкраще розбити систему на частини, описати їх взаємодію одна з одною, те як між ними передається інформація. Розглянемо як ці частини розвиваються поодинці та опишемо все використовуючи формальну чи неформальну нотацію.

Розглянемо такі аспекти розробки, як:

а) створення об'єктноорієнтованої моделі предметної області;

б) розбиття програми на окремі проєкт, розробка яких буде вестись незалежно один від одного;

в) архітектура програми із використанням мікросервісного підходу;

г) розробка інтерфейсу користувача з розділенням моделі і представлення;

д) розробка компонентів для побудови інтерфейсу користувача;

2.1 Доменна область

Основними сутностями в даній системі являються користувачі та фотографії. До фотографій можна також додавати коментарі. Самі ж фотографії можна групувати в альбоми. Оскільки основні задачі належати до фотографій виділимо їх в окремий сервіс — PhotoAPI. Тоді решту доменної моделі помістимо в — LamaAPI. Таким чином, архітектура системи, на даному етапі має наступний вигляд, ***рисунок 2.1***:

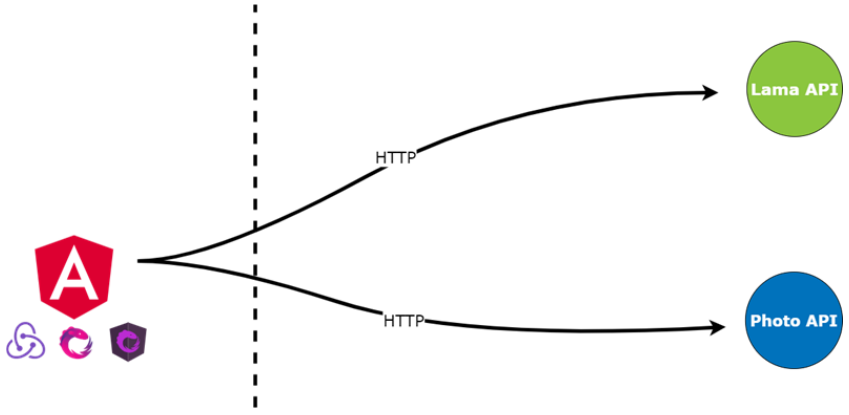

Рисунок 2.1 — Архітектура системи.

2.2 Модель даних

Для розв'язання поставленої задачі, сформуємо базу даних. Основною сутністю буде користувач (таблиця User) він може виставляти фотографії (таблиця Photo) та писати коментарі (таблиця Comment). Оскільки є можливість ділитись фотографіями потрібна проміжна таблиця, яка встановлюватиме зв’язок між користувачами та фото. Такою таблицею буде — SharedPhoto. Вона також містить допоміжну інформацію про те коли саме фотографія була додана у публічний доступ. Також варто виділити таблицю, що буде містити дані про альбоми — Album, та зв’язок із фотографіями — PhotoAlbum. При пошуку фотографій користувач також матиме підказки у вигляді списку із декількох останніх пошукових запитів. Щоб реалізувати цей функціонал варто зберігати інформацію про історію пошуку — SearchHistory. Для того щоб показувати сповіщення виділимо окрему сутність — Notifications. ER-діаграму бази даних можна побачити на ***рисунку 2.2***.

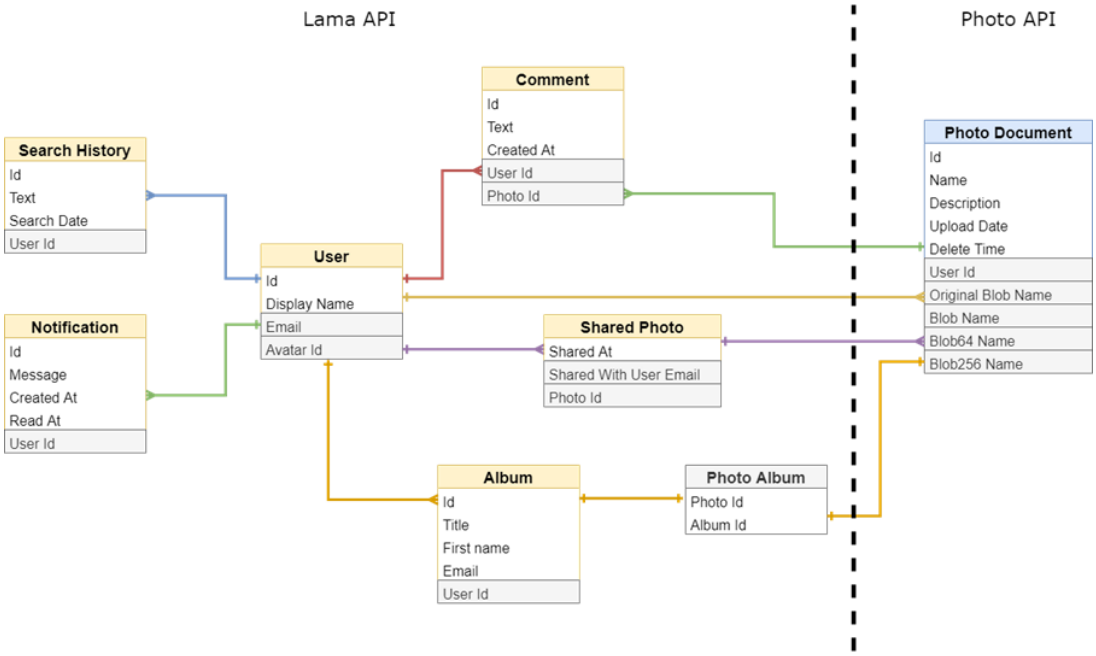

Рисунок 2.2 — ER-діаграма бази даних.

Дана модель не є фінальною, але вона становить гарний початок. Для того, щоб не втратити даних, але мати можливість змінити структуру бази даних, ми використаємо механізм міграцій. Таким чином ми матимемо історію того як змінювалась структура моделей та із легкістю зможемо перейти від однієї версії до іншої.

Варто зазначити, що дані мають зв’язки між собою. Та з точки зору сервісно-орієнтованої архітектури одним із важливих моментів являється децентралізації. Децентралізоване збереження передбачає, що кожний сервіс має лише свою базу даних та не взаємодіє із даними інших сервісів, приклад взаємодії на ***рисуноку 2.3***.

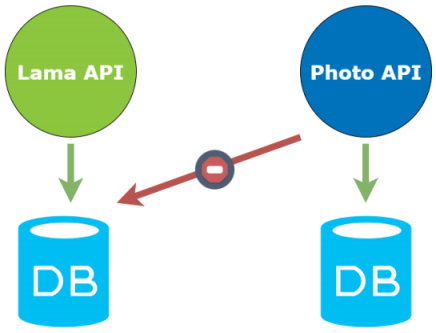

Рисунок 2.3 — Приклад взаємодії сервісів із сховищем даних.

Якщо в одного сервісу виникає потреба отримати дані іншого то обмін відбувається між сервісами напряму.

2.3 Вибір способу збереження даних

На цей момент система містить взаємопов'язані сутності такі як User, Album, Comments тощо. А також сутність для фотографій та самі фізичні файли фотографій.

2.3.1 Збереження доменної моделі

На домену модель не накладається жодних додаткових правил. Для збереження оберемо MsSqlServer та отримуватимемо дані за допомогою декларативної мови запитів SQL. Таким чином ми досягнемо незалежності від системи контролю даних та отримаємо декларативний підхід, який дозволить нам із легкістю будувати складні запити з великою кількістю зв’язків. Складні операції можна здійснювати за допомогою транзакцій забезпечуючи при цьому надійну роботу бази даних та цілісність даних.

2.3.2 Збереження інформації про фотографії

На фотографії накладаються ряд обмежень. Основна функціональна властивість полягає у забезпечені повнотекстового пошуку та показі

результатів по кращому збігу. У цьому випадку реляційні бази даних надають обмежені можливості пошуку, а також програють в продуктивності не реляційним, особливо у випадку коли варто шукати по багатьох полях.

Альтернативою реляційним базам даних являються розподілені. Їх доволі багато і всі вони спроєктований для різних цілей. Так наприклад MongoDb — використовує документно-орієнтований підхід в той час як ArangoDb базується на графах. Однак жодна з них не забезпечує пошуку і сортування за кращим результатом. В такому випадку доведеться або реалізовувати власний алгоритм чи знайти відповідний інструмент, який вже вміє це робити. Таким інструментом являється Elasticsearch. Це документно-орієнтована не реляційна база даних спроєктований для підтримки повнотекстового пошуку із можливістю сортування даних в залежності від заданих конфігурацій.

2.3.3 Збереження фотографій

Також варто визначити один із підходів до збереження фізичних файлів. Я розглядав наступні варіанти та виділив їх основні переваги та недоліки, ***таблиця 2.1:***

Таблиця 2.1 — Порівняння способів збереження фізичних файлів.

	Власний сервер	Сторонній сервер	Спеціалізоване сховище
Переваги	– простота у використанні; – велика кодова база прикладів застосування;	– надійність. Наявність копій; – зручний інтерфейс;	– ефективний доступ до файлів;
Недоліки	– ручна робота із файлами; – варто запам’ятовувати шляхи до файлів; – сервер нагромаджується файлами клієнтів;	– обмеження по розміру/кількості файлів; – залежність від стороннього сервісу. При його недоступності не можна працювати із системою;	– окремий сервіс призначений лише для файлів;

На перший погляд може здатись, що найкращим варіантом являється використання стороннього сервісу, через те, що він містить оптимальне відношення переваг та недоліків. Та моя система не передбачає обмеженості у

кількості файлів, тому кращим варіантом являється спеціалізоване сховище. Таким чином ми уникаємо ручної роботи із файлами та при цьому виділяємо окремий сервіс для їх збереження.

2.4 Огляд отриманої структури

Таким чином наша аплікації розділена на клієнта та сервера.

На частині клієнта ми маємо Angular аплікацію, яка взаємодії із сервером за допомогою HTTP протоколу.

На частині сервера знаходиться два незалежні сервіси — LamaAPI та PhotoAPI. LamaAPI призначений для зберігання основної предметної області та зберігає дані у реляційній базі. В той час як PhotoAPI містить алгоритми для взаємодії із фотографіями. Він зберігає дані про фотографії в Elasticsearch, а самі фото у спеціальному сховищі, ***рисунок 2.4***.

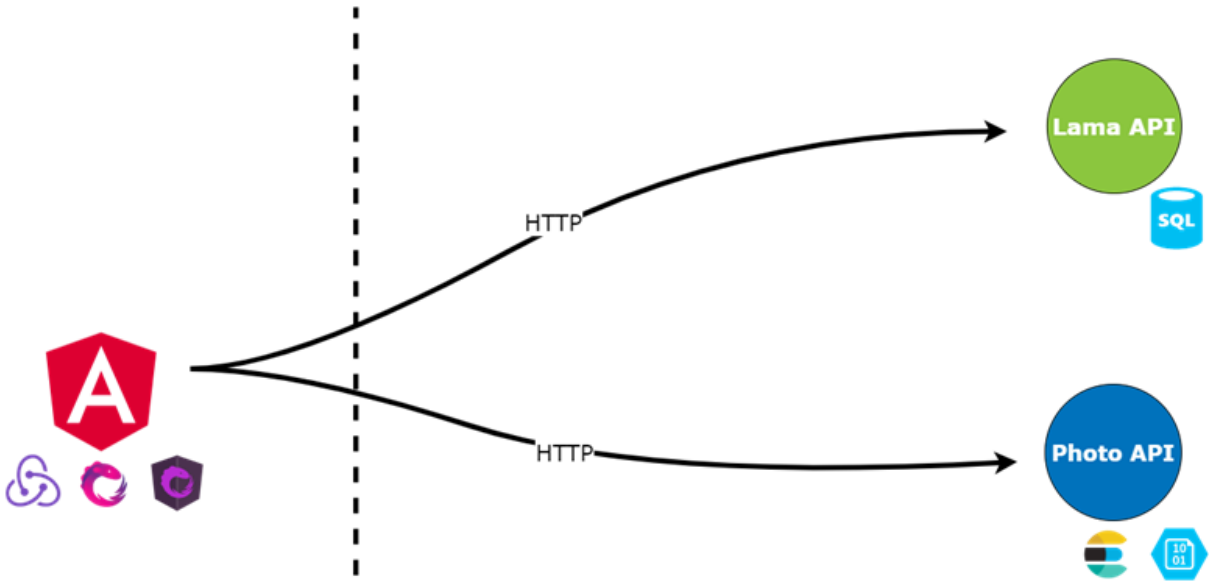

Рисунок 2.4 — Архітектура системи.

2.5 Організація процесу роботи

Варто також зазначити сам процес розробки. У випадку коли проєкт може зазнавати змін у графічному інтерфейсі чи інших компонентах, а також можуть змінюватися вимоги або технічне завдання потрібно скористатись методикою неперервної інтеграції. Таким чином, розробка не відбуватиметься над одним компонентом та по завершені розпочинатись над іншим, а одночасно над усіма компонентами. Це робиться для того щоб завжди мати робочу версію програми

— Minimal Valuable Product. Таким чином ми завжди можемо переглянути наш додаток, показати його майбутньому користувачу, виявити недоліки в моделі та інтерфейсі на ранніх стадіях розробки.

Цикл розробки при неперервній інтеграції виглядає так:

а) розробка моделі;

б) розробка інтерфейсу;

в) тестування;

г) рефакторинг;

Отже, розробка полягає в постійному внесені невеликих змін в модель та інтерфейс. Це все потрібно, оскільки набагато краще мати легку і примітивну робочу програму, аніж складну і не робочу. Якщо в кінці розробки виявиться, що ми взагалі невірно зрозуміли програмну область, тоді переробляти невелику програму буде значно простішою, аніж велику. Інтерфейс не обов’язково буде розроблятись швидше моделі, але якщо так станеться, то частина його можливостей буде замінена фальшивими компонентами.

3 АРХІТЕКТУРА

3.1 Архітектура додатку клієнта

При розробці архітектури додатку клієнта варто враховувати технологію на якій відбувається розробка та її можливості. Як вже зазначалось раніше для побудови додатку використовуватиметься фреймворк Angular із додатковими бібліотеками для керування даними — RxJs, Redux, NgRx.

Основними підходами при розробці архітектури для клієнта виділяють наступні: feature-based та component-based. Розглянемо їх переваги та недоліки, а також спосіб реалізації.

При реалізації component-based підходу файли поміщаються в директорії в залежності від їх функціональних обов'язків. Таким чином усі сервіси знаходяться у директорії із назвою "сервіси", усі компоненти — у директорії "компоненти" і так далі.

При реалізації feature-based — файли поміщаються у директорії, що реалізовують незалежну логіку аплікації. В середині директорії використовується те саме розміщення, що і component-based.

Розглянемо переваги та недоліки кожного із підходів, ***таблиця 3.1***.

Таблиця 3.1 — Порівняння feature-based та component-based архітектур.

	feature-based	component-based
Переваги	– легкий у підтримці; – просто розширювати функціонал;	– простий в реалізації; – зрозумілий у використанні;
Недоліки	– важкий у реалізації; – не очевидний при використанні;	– важкий у підтримці; – важко додавати новий функціонал;

Як бачимо один підхід містить переваги та недоліки іншого. Оскільки до нашої аплікації ставиться вимога по можливості її розширення, а також я хочу забезпечити легку підтримку компонентів, то використаємо feature-based підхід.

Кожна логічна одиниця аплікації знаходиться у спеціальному модулі, ***рисунок 3.1***. Модулі є незалежними та не взаємодіють один з одним. Таким чином вони забезпечують принцип розділення обов'язків.

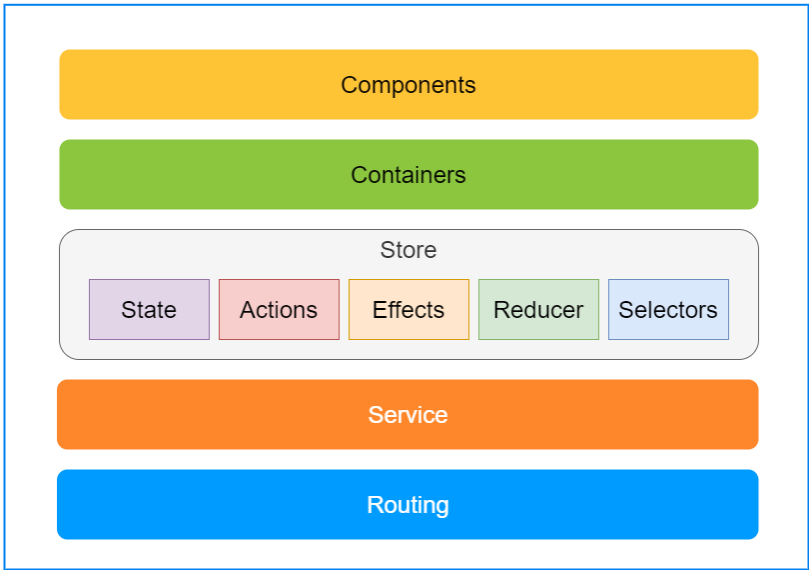

Рисунок 3.1 — Приклад структури модуля.

Розглянемо структуру модуля:

а) Components — об’єкти, які відповідають за представлення користувачу даних із допомогою html та css;

б) Containers — містять логіку модуля, та відповідають про те як саме варто показувати components;

в) Store:

1) State — містить приховані внутрішні дані модуля;
2) Actions — представляють механізм зміни даних;
3) Effects — містять логіку аплікації, яка виконується у залежності від виконаних Actions;
4) Reducer — містять алгоритми зміни даних в залежності від Actions;
5) Selectors — містять функції, які дозволяють ефективно та безпечно отримувати дані із State;

г) Services — класи, які містять логіку взаємодію із сервером для даного модуля;

д) Routing — не обов’язковий компонент модуля. Необхідний в тому випадку, якщо передбачена зміна модуля в залежності від адреси веб додатку;

Як приклад модуля на сторінці може виступати — логотип, заголовок сторінки, бокова панель, основний контент сторінки, тощо, ***рисунок 3.2***.

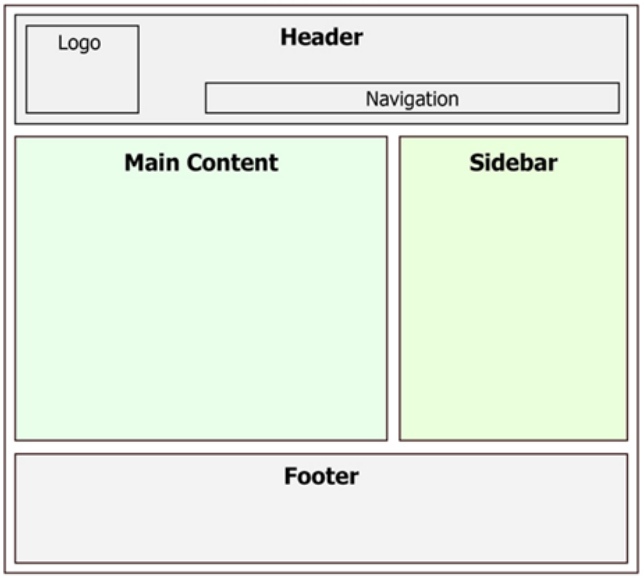

Рисунок 3.2 — Приклад компонентів-модулів.

Окрім основних модулів, варто виділити, ті що не вписуються у жодну із цих характеристик чи містять спільну логіку для багатьох компонентів:

а) Core — забезпечує компоненти комунікації та функціонування інших модулів;

б) Shared — містить спільну для багатьох модулів логіку;

3.2 Архітектура сервера

При розробці серверної частини додатку, варто зазначити, що вона має надавати інтерфейс для взаємодії різноманітних клієнтів, а також повинна забезпечувати високу навантаженість на систему. Саме для цього найкраще підходить мікросервісний підхід, де кожний із сервісів виконує лише свою логіку та може перетягнути половину із запитів на себе.

Сервіси не залежать один від одного та не обов’язково повинні бути побудовані за однаковими правила чи із використанням однієї технології. Головне це забезпечити однаковий спосіб передачі даних. Щоб це продемонструвати для кожного із сервісів будуть використані різні підходи побудови.

3.2.1 Архітектура основного сервісу

Для основного сервісу використаємо onion-архітектуру, ***рисунок 3.3***. Згідно з цим підходом в основі нашої аплікації знаходиться доменна модель, навколо якої надбудовуються додаткові рівні логіки. На одному рівні може знаходитись декілька незалежних шарів. При цьому важливо, що рівні мають доступ до *всіх* рівнів нижче. Зв'язок із нижчих шарів до вищих не дозволяється.

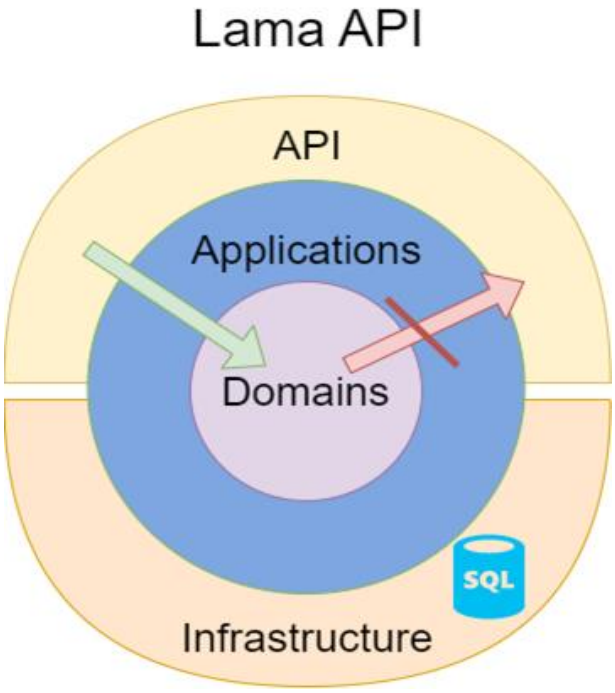

Рисунок 3.3 — Реалізація Onion-архітектури.

Розглянемо рівні детальніше:

а) Domains — містить класи доменної моделі. Основні об'єкти системи, які зберігаються в базу даних;

б) Applications — цей рівень містить логіку нашого сервісу, а також інтерфейси, які описують взаємодію із нашою моделлю не спираючись на реалізацію;

в) Infrastructure — на цьому рівні знаходиться реалізації структури необхідної Application рівню. Реалізовує доступ до даних, зміну файлів, звернення до сторонніх ресурсів та логіку, яка прив'язана до конкретних технологій;

г) API — рівень інтерфейсу та взаємодії із сервісом. Представлений у вигляді RESTful API;

Також варто зазначити, що для підтримки масштабування та високого навантаження сервісу використовується CQRS підхід.

CQRS (command-query responsibility segregation) — це підхід, при якому дії із даними розділяються на команди та запити. Команди відповідальні за зміну, видалення чи редагування ресурсу, тобто будь-яку операцію, яка передбачає зміну стану об'єкту. В той час, як запити виконують лише читання даних. Таким чином ми зможемо розділити логіку аплікації та зменшити навантаження з інфраструктури. Сама логіка реалізована за допомогою класів-команд, а їх взаємодія регулюється класом-посередником.

3.2.2 Архітектура сервісу із фотографіями

Щодо сервісу із фотографіями то в його основі лежить багаторівнева архітектура, ***рисунок 3.4***. При такому підході кожний із рівнів має доступ лише до шару нижче. Сам рівень оголошує інтерфейс взаємодії із ним та містить необхідну реалізацію. При потребі можемо розширювати кожний із рівнів незалежно від інших.

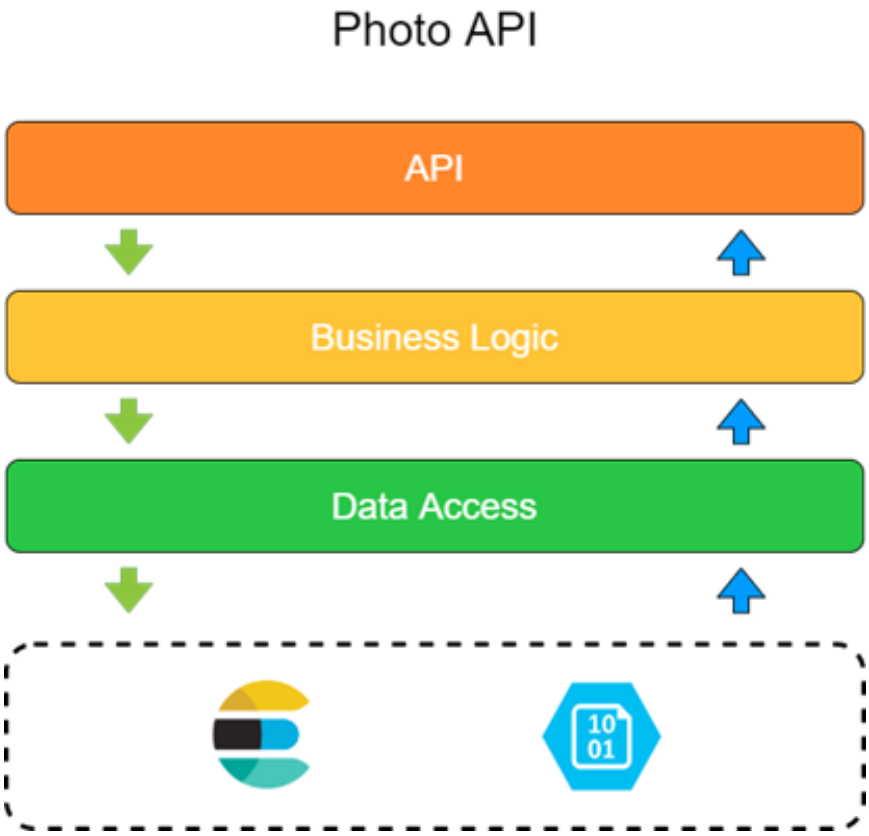

Рисунок 3.4 — Реалізація багаторівневої-архітектури.

Розглянемо рівні детальніше

а) Data Access — цей рівень забезпечує роботу із даними, файлами, базою даних, поштовими клієнтами тощо та взаємодіє з ElasticSearch та BlobStorage напряму;
б) Business Logic — містить набір компонентів, які відповідають за отримання та обробку даних від рівня представлення, використовуючи Data Access рівень;
в) API — рівень інтерфейсу та взаємодії із сервісом. Представлений у вигляді RESTful API;

Для цього сервісу логіка реалізована за допомогою сервісних класів, які представляють Anemic модель. Цей підхід обрано на протидію Rich моделі, оскільки дані являються простими, а алгоритми їх опрацювання складними.

3.3 Огляд

Отже, на цей момент архітектура додатків виглядає наступним чином:

а) Для побудови додатку клієнта використовується Angular із feature-based архітектурою;
б) Для побудови основного сервісу аплікації використовується onion архітектура із використанням CQRS підходу;
в) Для побудови сервісу із фотографіями використовується багатошарова архітектура;

Даної структури цілком достатньо аби розпочати розробку аплікації. Всі наступні компоненти будуть додані по мірі необхідності.

4 РОЗРОБКА

4.1 Авторизація

Будь-яка аплікації потребує точки входу. Окрім цього, аби користуватись системою, необхідна наявність користувачів та спосіб їх ідентифікації. Авторизація, як основний компонент системи, необхідна для того щоб зберігати налаштування клієнтів, обробляти їх, а головне відрізняти профілі один від одного.

Для входу в систему користувачеві необхідно ввести його дані, це його пошта та пароль. Цього достатньо для реєстрації й авторизації. Також є можливість здійснити вхід на основі однієї із соціальних мереж, таких як Google, Facebook та Twiter. Якщо користувач забув пароль, йому необхідно лише ввести пошту, на яку прийде новий випадково згенерований пароль. Увесь цей функціонал реалізований у вікні авторизації, ***рисунок 4.1***:

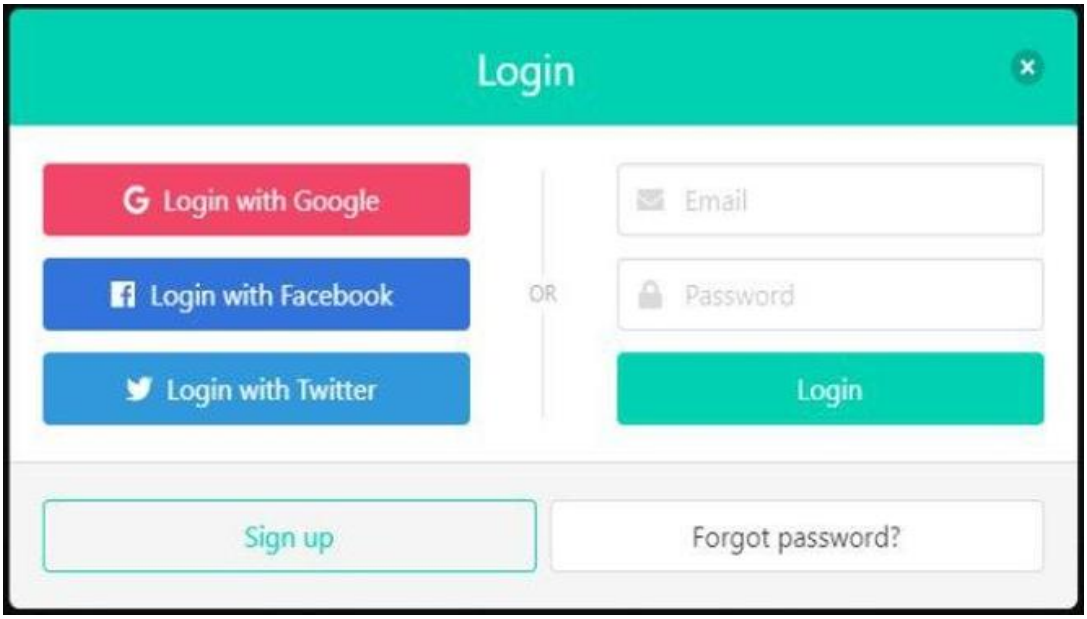

Рисунок 4.1 — Вікно авторизації.

При розробці на мікросервісній архітектурі важливе значення має спосіб реалізації автентифікації, оскільки про те що користувач увійшов у систему мають знати усі сервіси.

Таким чином автентифікації на основі сесії чи кукі не підходить, куди краще спрацьовує авторизація базована на токенах. Токен — це автономне

кодоване повідомлення, яке передається із кожним запитом, та містить інформацію про користувача. Таким чином сервіси не зберігають у себе жодної інформації про клієнта, а отримують її із запиту.

Процес автентифікації доволі складний і містить багато дрібних нюансів, про які варто подбати, наприклад, шифрування даних, оновлення токена, якщо у даного закінчився час життя тощо, тому для того щоб не реалізовувати його вручну я використав Firebase, ***рисунок 4.2***. Firebase Auth — це служба розроблена компанією Google, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта за допомогою різноманітних логін-провайдерів та містити систему управління користувачами.

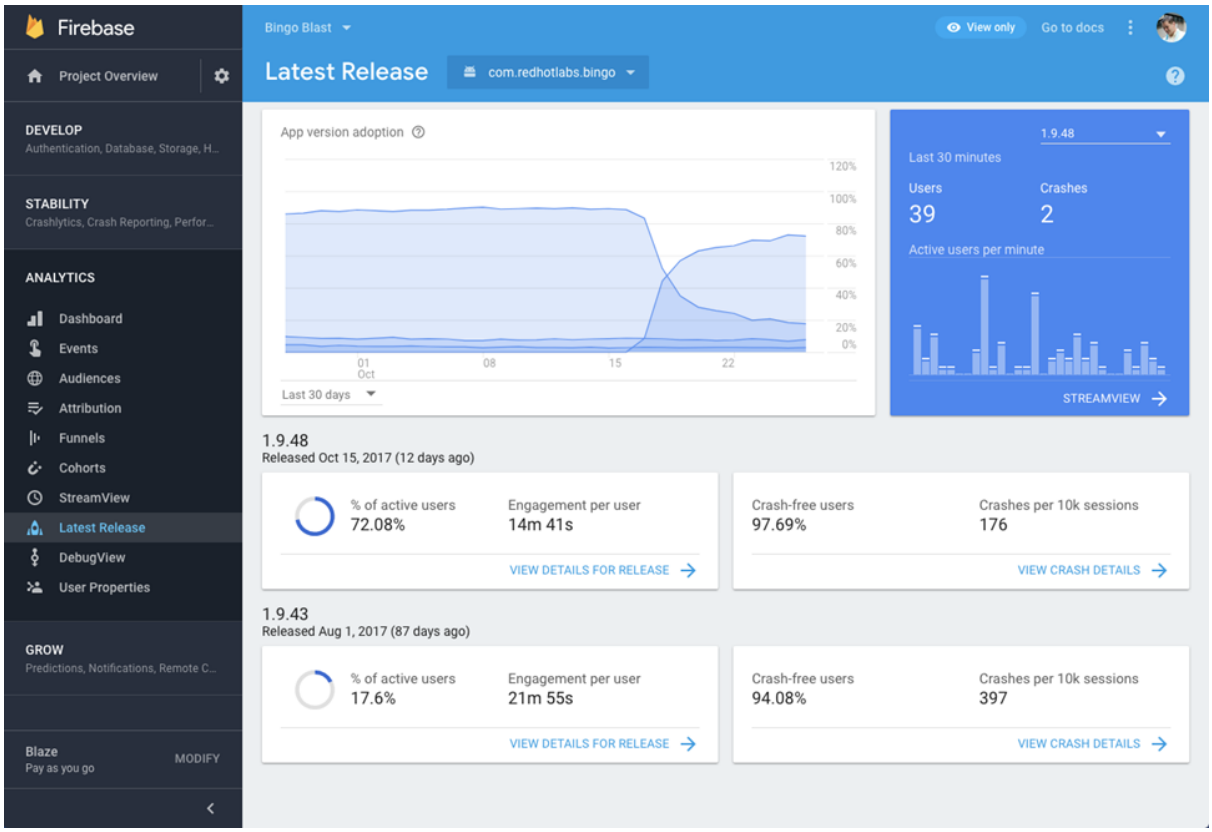

Рисунок 4.2 — Сторінка керування Firebase.

Таким чином у систему додається, ще один сторонній сервіс, який відповідатиме за авторизацію користувачів, ***рисунок 4.3***.

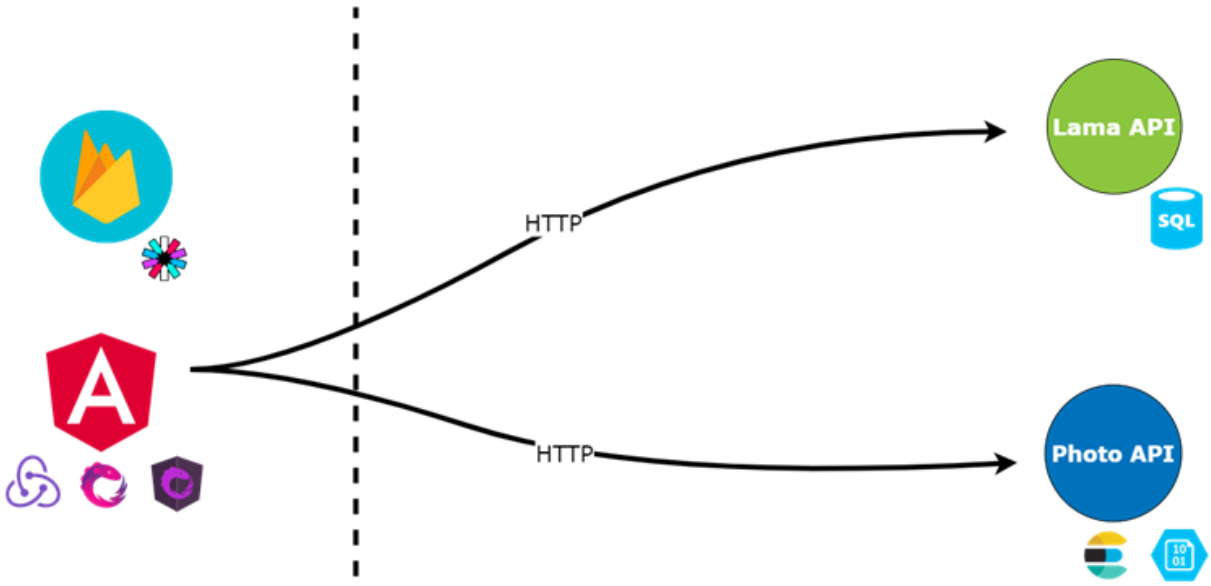

Рисунок 4.3 — Архітектура системи.

Та як бачимо зі схеми вище, можна виділити декілька основних недоліків взаємодії із сервісами:

а) клієнт явно знає про структуру системи та способи комунікації між сервісами;

б) усі клієнти мають знати про розташування кожного із сервісів, що порушує принцип доступності розподіленої системи;

в) якщо в систему потрібно додати новий сервіс чи змінити розташування вже наявних, потрібно оновити всіх клієнтів, що створює додаткові проблеми при зміні розташування сервісів;

Для такого випадку скористаємось дуже простим патерном Gateway, ***рисунок 4.4***. Це сервіс головна задача якого полягає в делегуванні запитів іншим сервісам. Таким чином ми отримаємо наступні переваги:

а) для клієнта робота із системою виглядає так ніби з однією аплікацією;

б) клієнт не знає про внутрішню архітектуру та взаємодію системи. Gateway передає дані тому сервісу, який їх потребує та по вірному каналу зв’язку;

в) для сервісів існує спільна конфігурації;

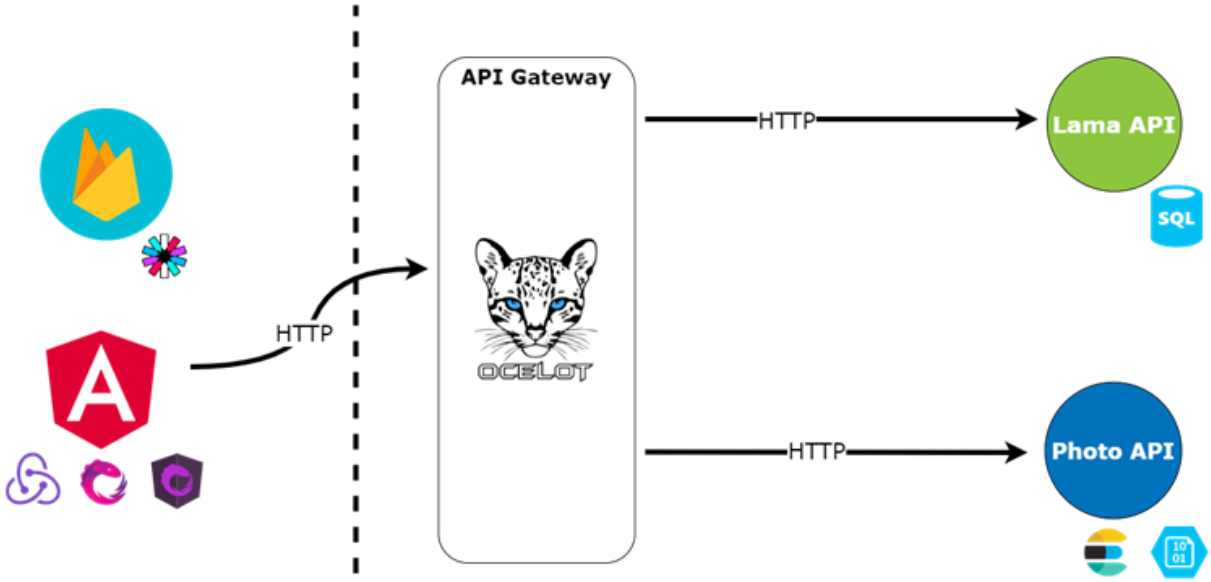

Рисунок 4.4 — Архітектура системи.

У разі вдалої реєстрації користувач отримує доступ до власного профілю, де міститься його фотографія, ім’я та пошта, ***рисунок 4.5***. Дані синхронізуються із вказаними у соціальних мережах.

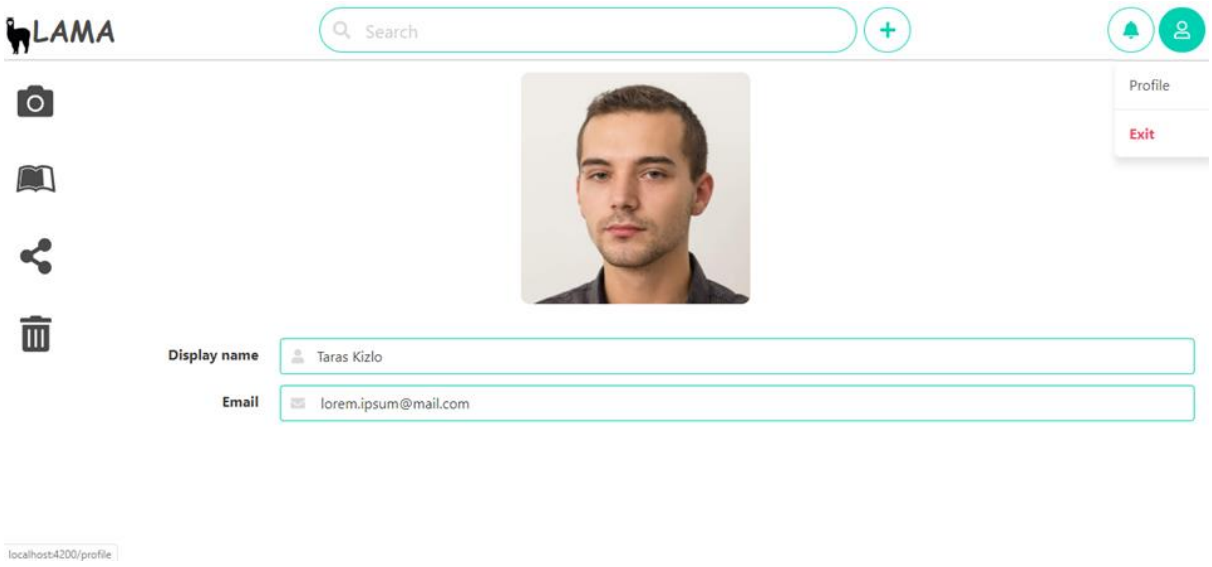

Рисунок 4.5 — Профіль користувача.

4.2 Фотографії

4.2.1 Перегляд фото

При вході у систему для користувача надається можливість перегляду та додавання нових фотографій. Якщо користуватись системою вперше вона не буди містити жодних файлів, ***рисунок 4.6***.

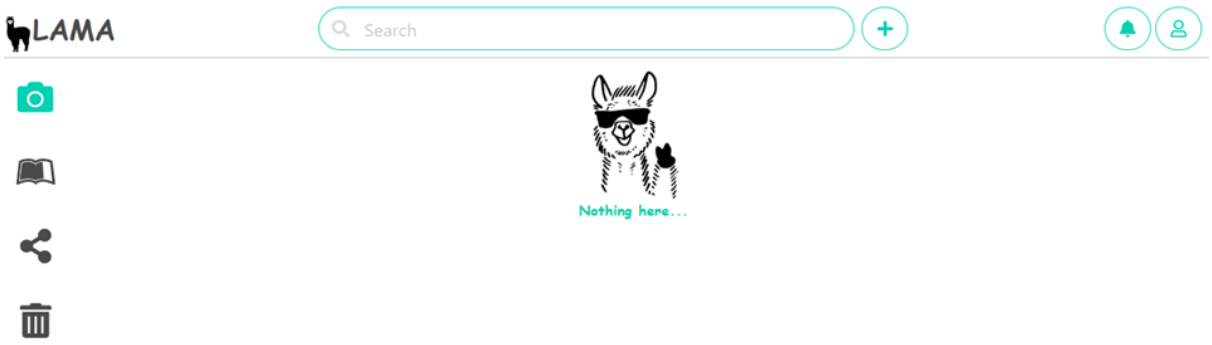

Рисунок 4.6 — Сторінка перегляду фотографій.

При тривалому використанні користувач завжди матиме доступ до фото та може змінити формат показу: панель чи список, ***рисунок 4.7***.

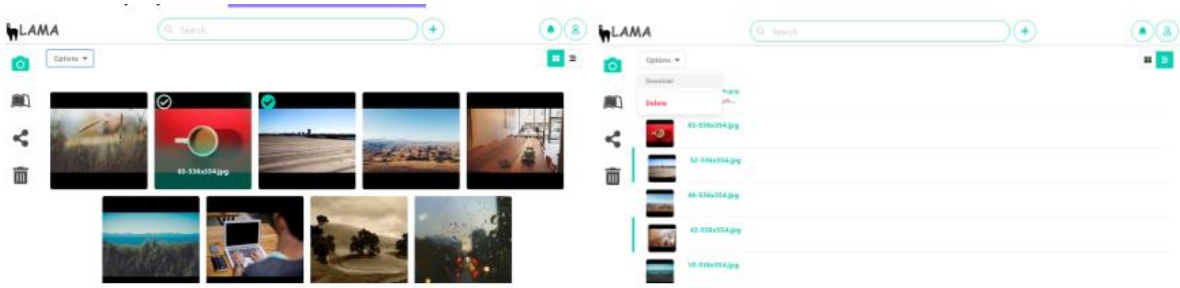

Рисунок 4.7 — Формати показу фото. Панель та список відповідно.

4.2.2 Додавання фото

Користувач завжди може додати декілька фотографій. Це робить за допомогою перетягування їх на спеціальну панель чи за допомогою відповідної кнопки, ***рисунок 4.8***.

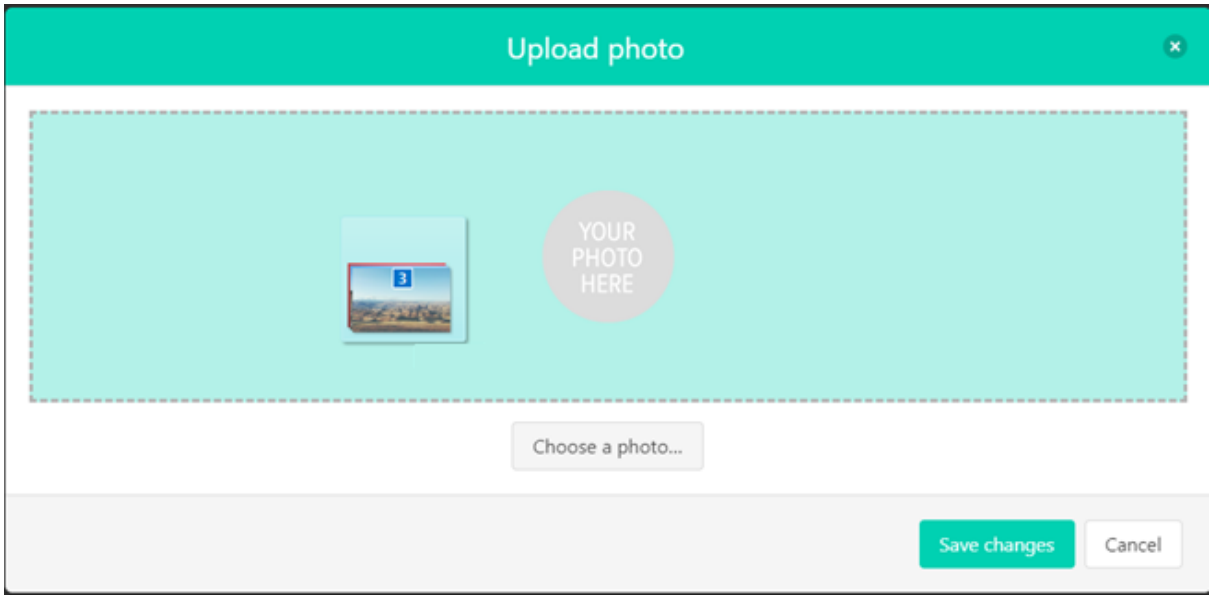

Рисунок 4.8 — Вікно для завантаження фотографій.

Таким чином можна з легкістю додати декілька фотографій, змінити їх опис, видалити вибрані фото чи додати ще декілька, тощо, ***рисунок 4.9***.

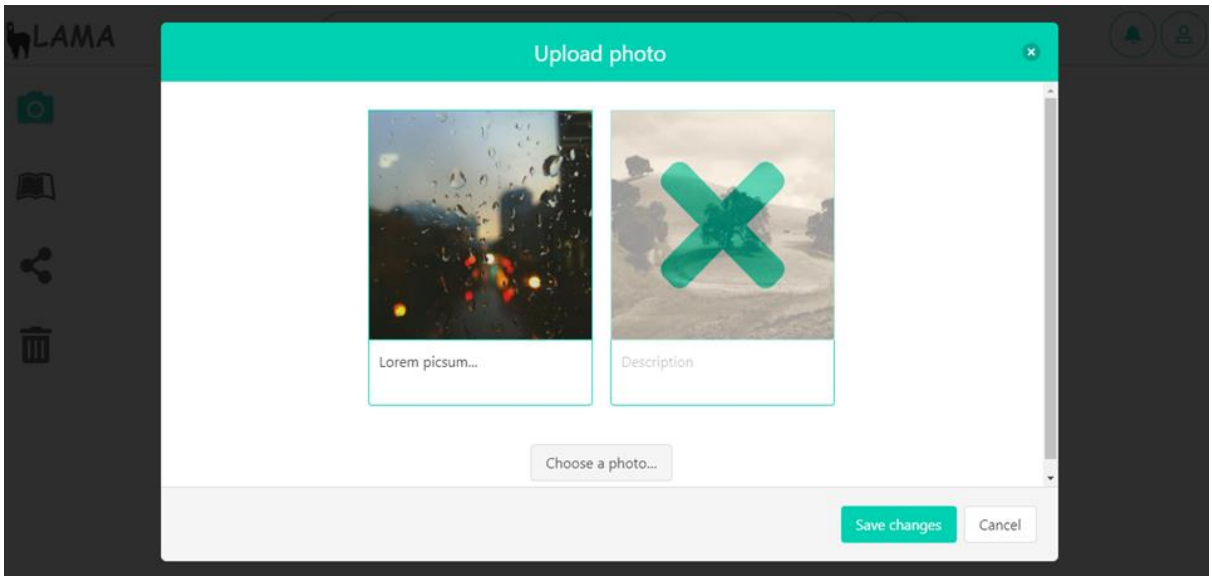

Рисунок 4.9 —Можливість редагування фотографій під час завантаження.

Завантажені фотографії зберігатимуться у спеціальному сховищі BlobStorage, а в ElasticSearch, міститиметься їхня унікальна назва. У якості унікальної назви зображення, використовується Guid.

4.2.3 Пошук

Мало хто із користувачів слідкує за тим, щоб розбивати фотографії, по альбомах, сортувати їх, надавати відповідні назви. Набагато зручніше згадати, незначну асоціацію та намагатись здійснити по ній пошук.

Саме тут у пригоді являється ElasticSearch. Він обробляє пошукові запити, здійснюючи повнотекстовий пошук. При цьому зовсім не важливо, щоб збіг був точним, оскільки дане програмне забезпечення сортує результати по мірі кращого збігу. Сам пошук здійснюється по багатьох полях: назві, опису, коментарям, але для користувачів цей процес буде непомітним завдяки заточеності цієї технології під текстовий пошук.

Також в історію пошуку записуються останні введені дані до яких можна повернутись при потребі, ***рисунок 4.10***.

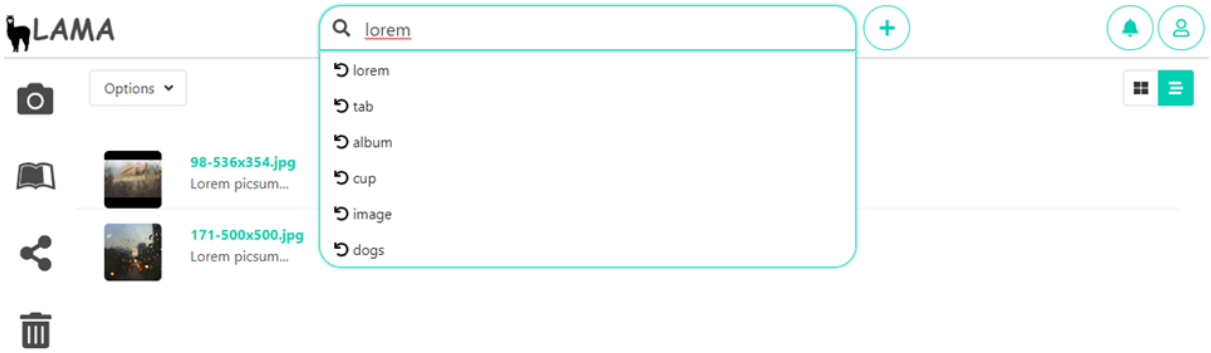

Рисунок 4.10 — Історія пошуку.

4.2.4 Видалення

Для користувача даної системи доступно багато опцій із фотографіями. Та базовими являються можливість, змінити вигляд сторінки, завантажити список обраних фотографій чи видалити не потрібні фотографії.

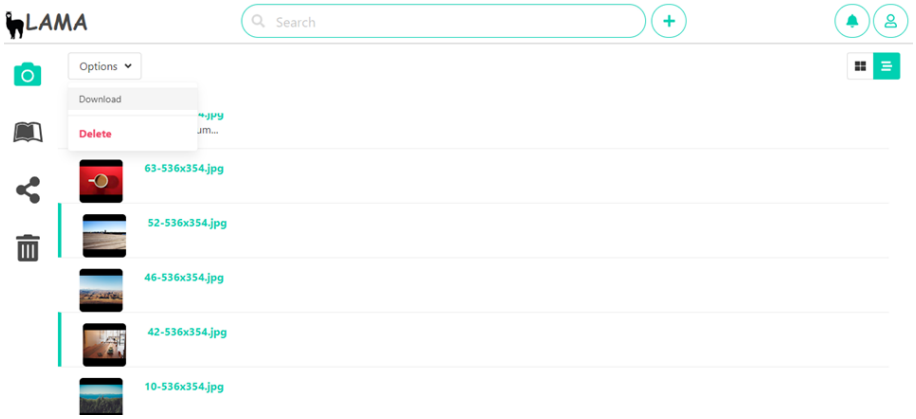

Рисунок 4.11 — Можливість видаляти обрані фото.

У разі видалення фотографії, усі дані пов’язані із нею (коментарі, список користувачів, які мають до неї доступ) теж будуть видалені із сервера, та з бази даних. При цьому тут виникають деякі труднощі, оскільки інформація про фото та коментарі знаходяться у різних сервісах. Як зазначалось вище, сервіси працюють лише зі своїм сховищем та не мають доступу до сховищ інших сервісів, а отже видалити зв’язні дані на пряму не вийде. Та й це не потрібно. Сервіс повинний відповідати лише за свої дані, а при їх зміні сповіщати інші сервіси. Усі зацікавлені сервіси мають слідкувати за змінами та реагувати на них відповідним чином.

Щодо технічної реалізації, то тут є декілька варіантів. Одним із варіантів передбачає реалізацію взаємодії по http протоколу. Таким чином клієнт надсилає запит на сервіс, а той своєю чергою надсилає запити іншим сервісам. Аби забезпечити надійність необхідно робити запити синхронними, проте це збільшить час відгуку оригінального запиту. А для того, щоб сповістити усі сервіси варто мати доступ до їх відкритого інтерфейсу взаємодії та знати, про логіку реалізації відповідних методів.

Як бачимо цей підхід містить лише недоліки хоч і являється простим у реалізації. Кращим варіантом виглядає підхід базований на подіях. Якщо у

сервісі відбувається зміна даних, він створює подію. Усі сервіси, які зацікавлені у цій події реалізовують логіку її опрацювання. При цьому всі операції відбуваються асинхронно завдяки технології гарантованого доставляння Message Bus, ***рисунок 4.11***.

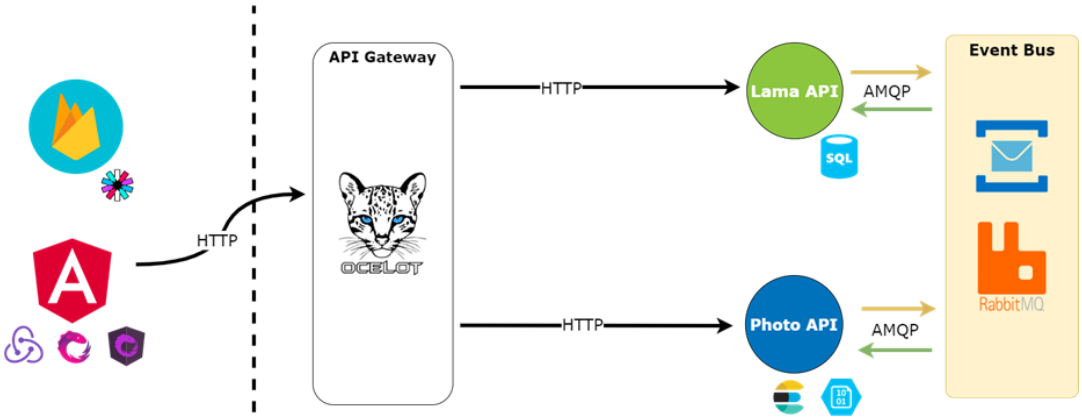

Рисунок 4.11 — Архітектура системи.

Message Bus — це шина повідомлень, сервіс, який представлений у вигляді єдиного екземпляра для усієї системи, що надає транзакційним канал зв'язку із сервісами та забезпечує доставлення усіх повідомлень у вірному порядку. При цьому забезпечується стійкість до відмов, оскільки повідомлення зберігаються у пам'яті, та будуть повторно надіслані при відновленні роботи системи, що покращує навантаження на систему, бо сервіси опрацьовують повідомлення, тоді коли не навантажені іншими командами.

Для реалізації обміну повідомленнями між сервісами можна скористатись багатьма підходами та технологіями.

Оскільки дана аплікації публікується у хмарному сховищі Azure, тоді можна використати готовий сервіс EventBus. З іншої сторони сам обмін повідомлень здійснюється за допомогою Advanced Message Queuing Protocol — протоколу, основною платформою якого являються RabbitMq.

Розглянемо різні підходи до реалізації обміну повідомлень, та оберемо вірну технологію. Один із таких варіантів це реалізації патерну Публікація-підписка. Цей шаблон проєктування реалізує механізм передачі повідомлень, в якому відправники повідомлень, не здійснюють пряме відправлення повідомлень приймачам, замість цього опубліковані повідомлення

розбиваються на категорії за класами, без знання про те, яким підписникам вони мають бути прийняті й чи взагалі будуть такі підписники. Аналогічно, підписники виявляють зацікавленість в певних класах повідомлень і приймають ті повідомлення, які їх цікавлять, без знання того, які видавці їх публікують. Цей підхід підтримується Azure сервісами та не EventBus напряму. Своєю чергою RabbitMq надає гнучкий інтерфейс, для реалізації цього підходу.

Інший спосіб реалізації обміну повідомлень — це інтеграційна шина даних. Сполучне програмне забезпечення, що забезпечує централізований та уніфікований, орієнтований на події обмін повідомленнями між різними інформаційними системами. Шина повідомлень використовується для конкретно визначених задач. Таким чином вона погано підходить для великої кількості запитів, які очікують на відповідь. Цей спосіб розриває зв’язок між тим хто публікує подію та тим хто на неї підписаний шляхом легкої зв’язності між компонентами: постачальники не знають, кому потрібна інформація, а споживачі не знають, звідки вона береться - в однієї інформації теоретично можуть бути різні постачальники та споживачі. Цей підхід чудово підходить коли одному сервісу потрібно сповістити інший, що в нього щось відбулось, щоб інші сервіси змогли на це відреагувати. Як EventBus так і RabbitMq реалізовують даний підхід.

Як бачимо обидві технології надають необхідний функціонал. Але також варто зазначити, що EventBus працює швидше у хмарному середовищі, але не доступний для локальної розробки, в той час як RabbitMq хоч і програє у швидкості, та придатний для розробки. Використаємо обидві технології та реалізуємо легкий спосіб перемикання між ними. Для цього створимо уніфікований інтерфейс інтеграційної шини даних, ***рисунок 4.12***.

```
interface IEventBus
{
    void Publish(event)
    void Subscribe(event, handler)
}
```

Рисунок 4.12 — Інтерфейс шини повідомлень.

Таким чином зміна технології відбуватиметься за допомогою незначного корегування файлу конфігурацій.

Також під час розробки, я зіткнувся із проблемою показу даних із різних сервісів. При цьому бувають ситуації різного рівня складності. Розглянемо їх детальніше.

Найпростіший варіант полягає в тому, що дані знаходяться у різних сервісах, але між собою не пов’язані, ***рисунок 4.13***. Наприклад, фотографія та список коментарів до неї. У цьому випадку необхідно здійснити два окремі запити до різних сервісів.

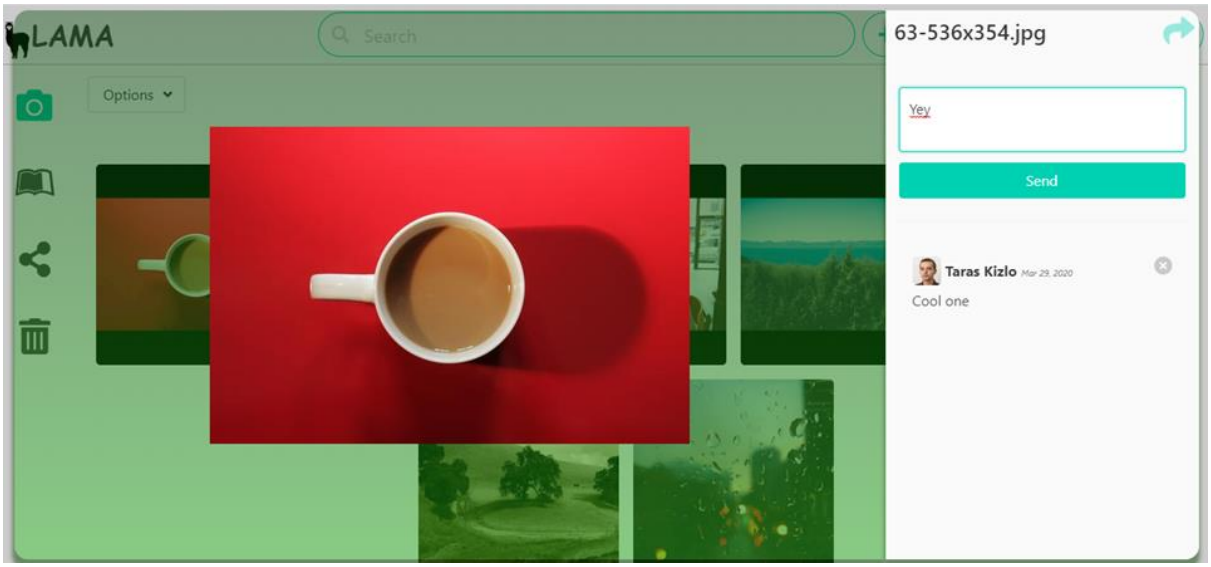

Рисунок 4.13 — Приклад взаємо не пов’язаних даних: фотографія та список коментарів.

Складніший варіант коли дані пов’язані, але у користувача до них є доступ, ***рисунок 4.14***. Так наприклад, при показі списку альбомів необхідно отримати інформацію про те яка фотографія виступає у якості обкладинки та саму фотографію. Ці дані знаходяться у різних сервісах, але на клієнті вже відомі всі фотографії і їх назви, тому варто лише отримати список альбомів та об’єднати обкладинки із доступними фотографіями.

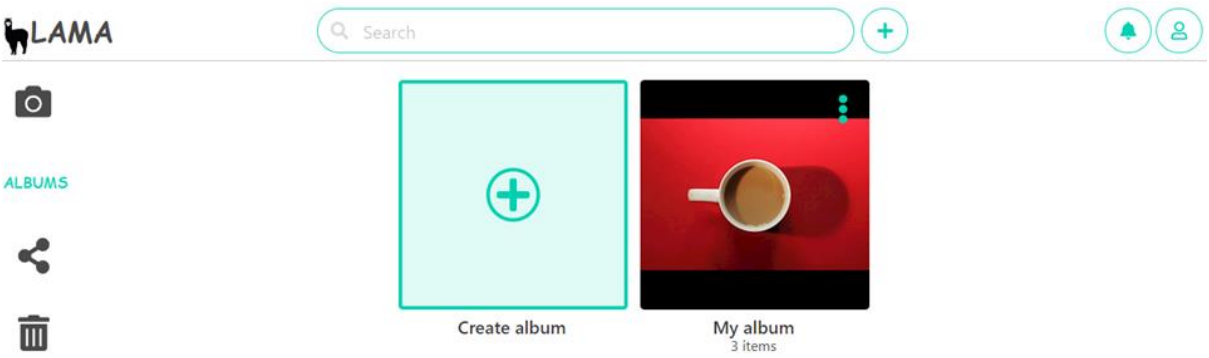

Рисунок 4.14 — Приклад даних, пов'язаних на частині клієнта: список альбомів і фото-обкладинка альбому.

Найскладніший варіант, коли необхідно показати користувачу фотографії, до яких йому надали доступ інші користувачі, ***рисунок 4.15***. Інформація про доступні фотографії та самі фото знаходяться у різних сервісах. Якщо спробувати передати всі дані клієнту та об'єднати їх отримаємо проблеми із надлишковістю даних та конфіденційністю.

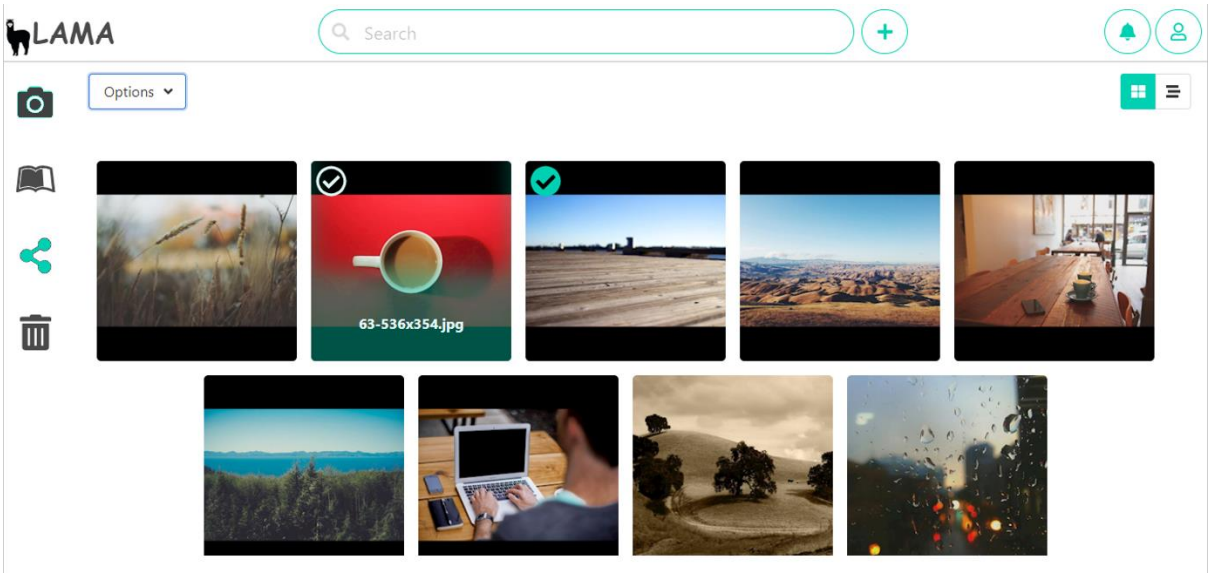

Рисунок 4.15 — Приклад даних, пов'язаних на частині сервера: список фотографій у спільному доступі.

Для цієї ситуації варто додати ще один сервіс, який буде агрегувати дані із наших сервісів, та передавати клієнту лише необхідну інформацію. Цей сервіс не містить жодної логіки, окрім нагромадження великої кількості даних

та їх об’єднання, ***рисунок 4.16***. Крім того, багато даних, таких як список усіх фото, необхідні багатьом користувачам, але виконання цього запиту навантажує систему у декілька разів через те, що спочатку запит отримує сервіс агрегатор, після нього сервіс із фотографіями, а на кінець пошук здійснюється у ElasticSearch. Цей процес можна спростити, якщо додати у сервіс агрегатор спеціальний механізм кешування. У цьому випадку кешування здійснене за допомогою розподіленого сховища пар ключ-значення, яке зберігає дані в оперативній пам’яті — Redis.

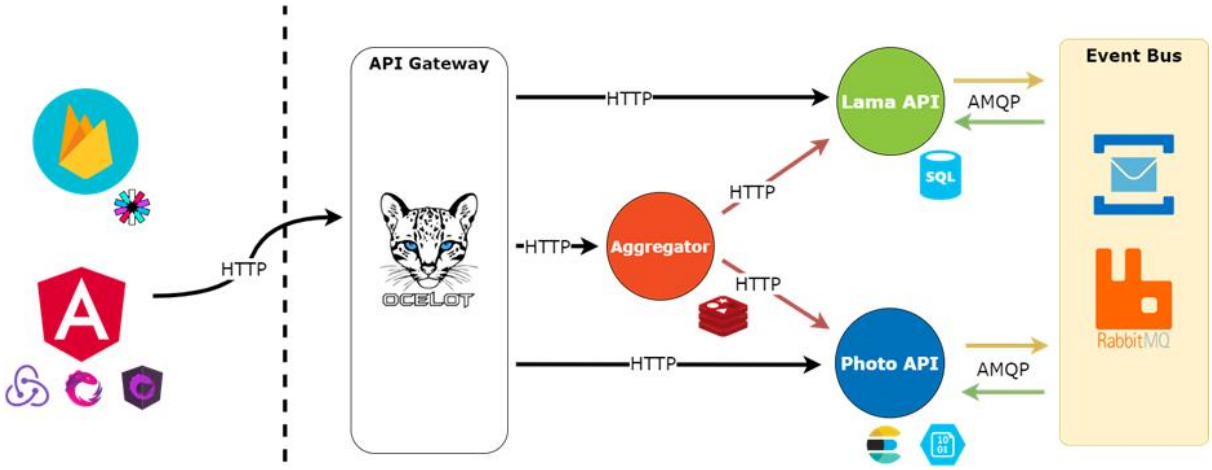

Рисунок 4.16 — Архітектура системи.

При цьому варто замітити, що після видалення фотографії остаточно не пропадають із системи. Вони поміщаються у спеціальний кошик, ***рисунок 4.17***, де будуть доступні до відновлення чи остаточного видалення протягом тридцяти днів. І лише по завершенню цього терміну фотографії будуть видалені.

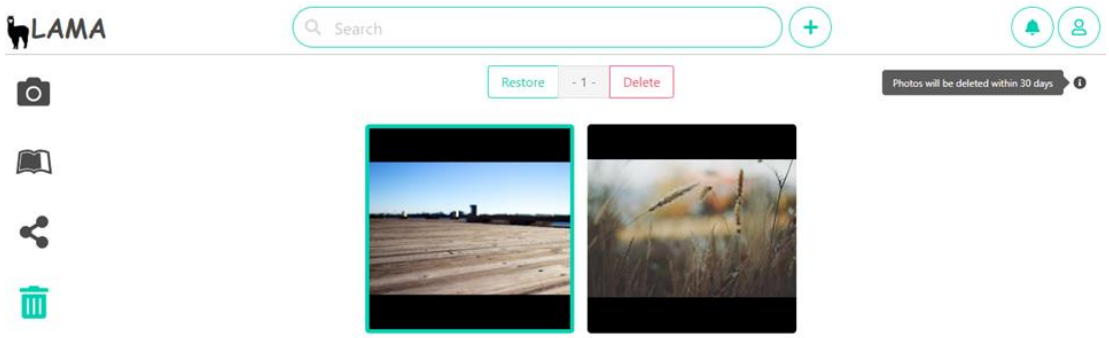

Рисунок 4.17 — Вікно видалених фотографій.

Щоб виконувати заплановані задачі до системи додається ще один компонент Hangfire, ***рисунок 4.18***.

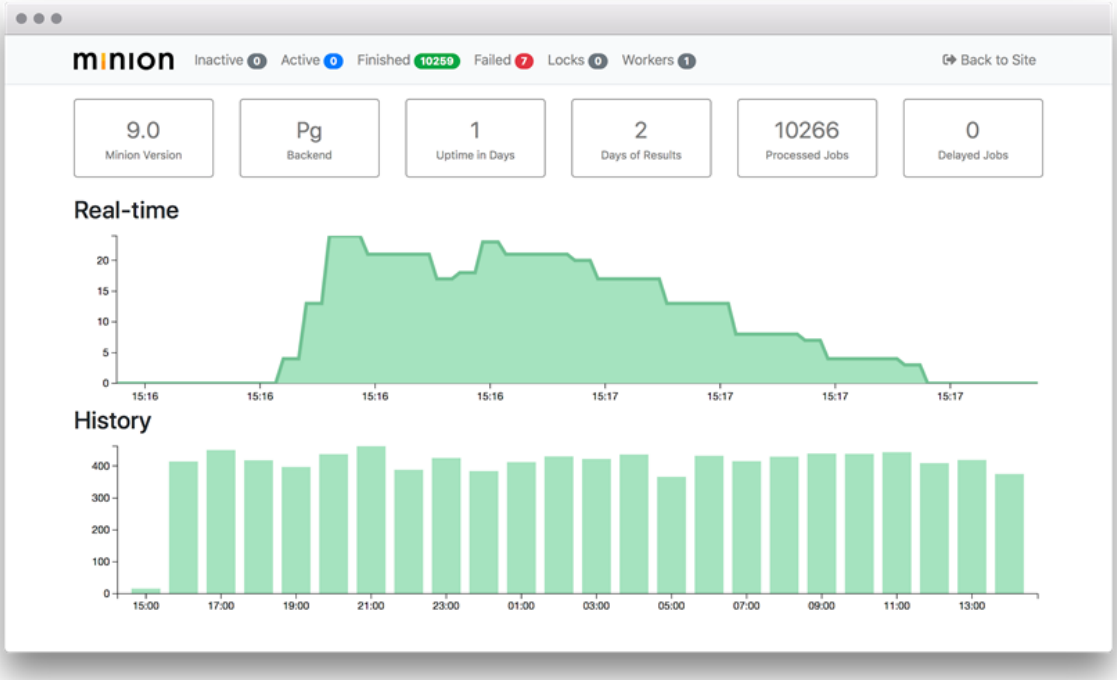

Рисунок 4.18 — Сторінка керування Hangfire.

Дана технологія дозволяє не лише виконувати заплановані задачі, але й повторювати задачі з мінімальним інтервалом, запускати задачі в ручну чи видаляти їх. Даний сервіс надає широкі можливості по масштабованості та реалізовує багатопотокову модель роботи, ***рисунок 4.19***.

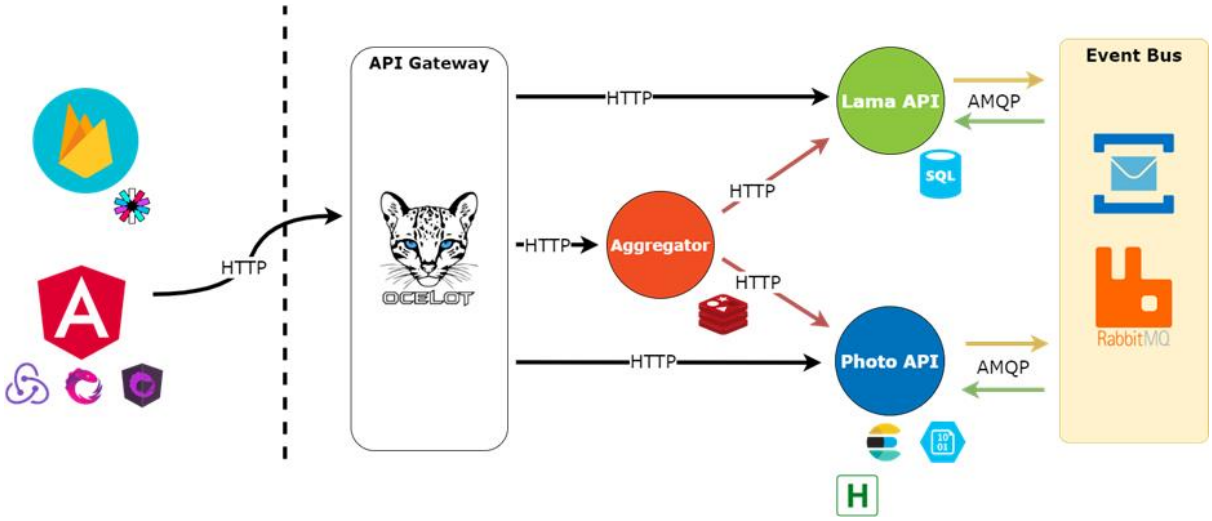

Рисунок 4.19 — Архітектура системи.

4.2.5 Створення мініатюр

При користуванні системо на даному етапі можна замітити значні проблеми оптимізації та швидкодії системи, яка прямопропорційна кількості завантажених фото. І це не дивно. Якщо користувач додав десять фотографій де кожна з них займає десять мегабайтів при наступному вході у систему необхідно завантажити сто мегабайтів даних. Хоча очевидно, що немає потреби стягувати фотографії у повному розмірі оскільки користувач все одно бачить їх зменшені версії.

Цю проблему можна вирішити, підготовляючи спеціальні мініатюри необхідного розміру, та передавати саме їх у списку всіх фотографій, а оригінальну фотографію завантажувати лише якщо користувач відкрив фото у режимі перегляду.

При цьому варто враховувати потреби користувача. Навряд чи багато клієнтів погодяться чекати поки до його фотографій створюються мініатюри, тому цей процес варто зробити асинхронним. Після того як користувач завантажить фотографії у систему, він побачить тимчасові картинки. Самі фотографії кладуться в чергу до спеціального сервісу, що робить мініатюри. Щойно до фотографій зроблені відповідні зменшені копії інтерфейс користувача оновлюється без перезавантаження сторінки, ***рисунок 4.20***.

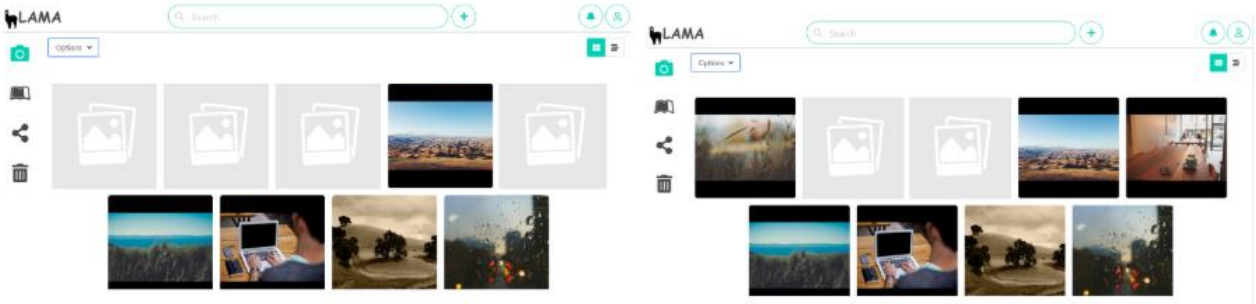

Рисунок 4.20 — Приклад роботи мініатюр.

Щоб досягнути цього результату до сервісу із фотографіями додається спеціальний процесор, який обробляє фотографії. Цей процесор являється асинхронним та не навантажує головний сервіс.

Щойно мініатюра створена, вона зберігається в базі, а її адреса відправляється клієнту за допомогою WebSocket протоколу використовуючи бібліотеку SignalR, ***рисунок 4.21***.

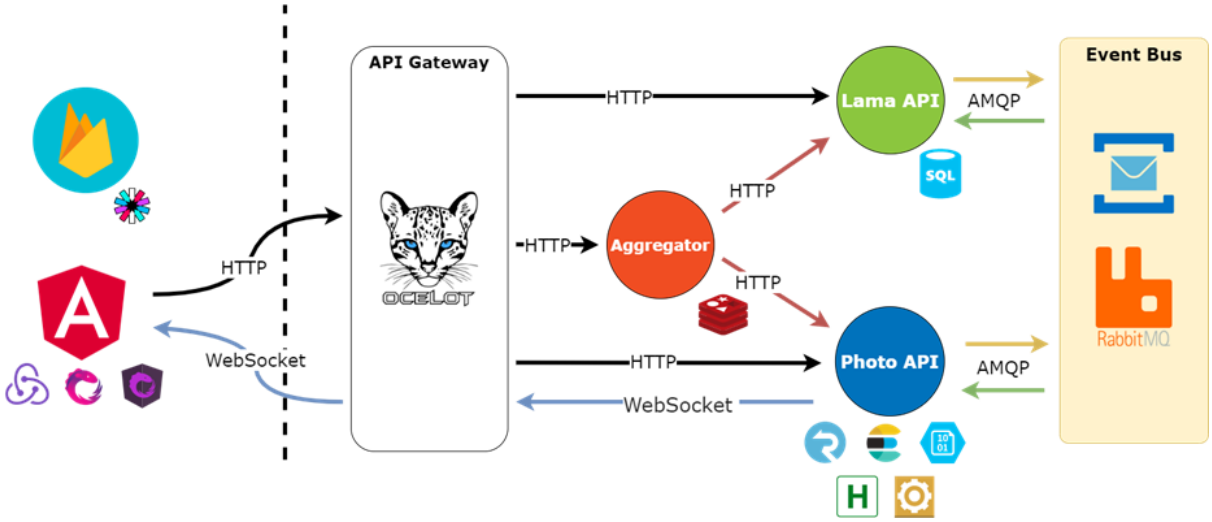

Рисунок 4.21 — Архітектура системи.

4.2.6 Завантаження фотографій

Користувач завжди може звантажити список обраних фото у вигляді архіву. При цьому фреймворк ASP не містить жодних вбудованих механізмів для роботи із файлами, тому довелось писати власний тип результату, який інкапсулює логіку збереження байтів у потоці в пам’яті.

Окрім того власний тип результату корисний, тим, що не доведеться дублювати код в інших сервісах. А дана узагальнена версія чудово підходить для завантаження файлів довільного формату.

4.2.6 Перегляд фотографій

При розширеному перегляді фото, зверху над фотографією доступна панель меню, яка надає доступ до наступних функцій:

4.2.6.1 Інформація про фотографію

При перегляді фотографії користувач завжди може змінити її опис та назву. Також він може переглянути дату, коли саме було завантажено фото, ***рисунок 4.22***.

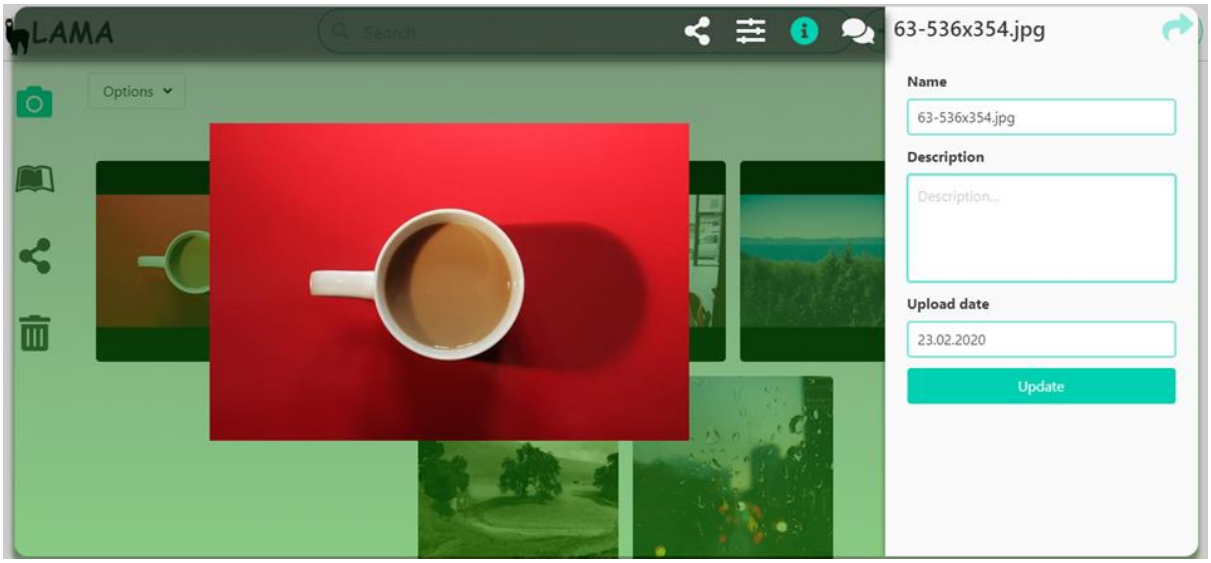

Рисунок 4.22 — Вікно детальної інформації про фотографію.

4.2.6.2 Надання спільного доступу до фотографії

Важливою особливістю даного сервісу є можливість обміну фотографіями з іншими користувачами. Щоб цього досягти варто знати адресу електронної пошти користувача, якому буде наданий доступ, ***рисунок 4.23***. Якщо даний користувач не зареєстрований у системі, то ви отримаєте сповіщення про помилку. При успішному виконанні операції, користувач отримає сповіщення та матиме доступ до фото у відповідній вкладці.

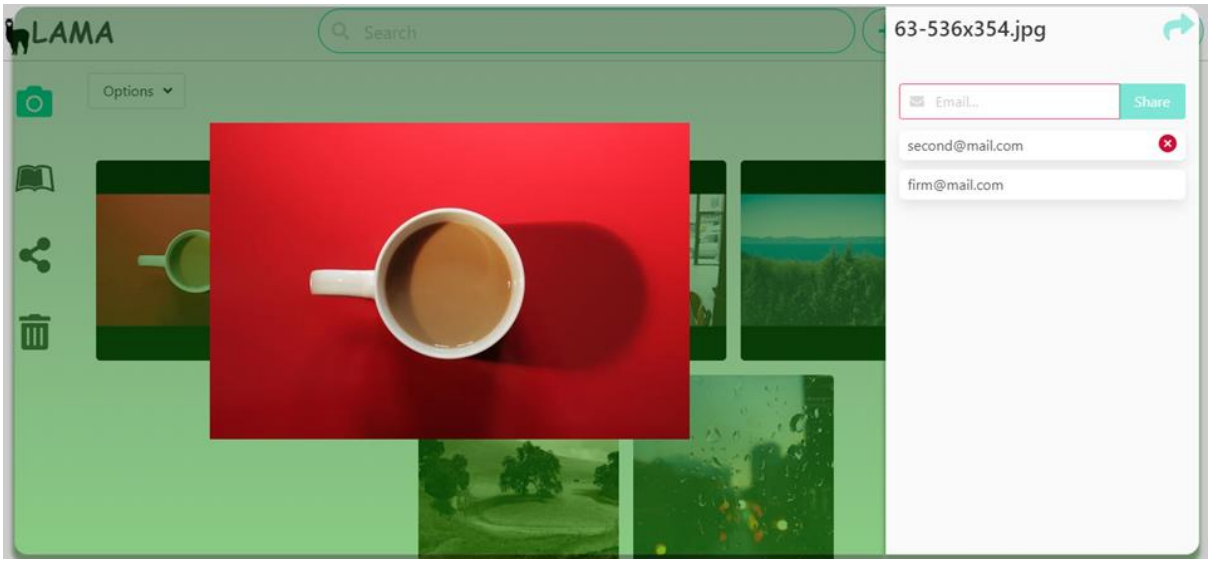

Рисунок 4.23 — Вікно надання спільного доступу до фотографії.

4.2.6.3 Коментарі

Одним із пунктом меню при перегляді фото є коментарі, ***рисунок 4.24***. Користувач може додавати коментарі лише до своїх фотографій чи тих до яких йому надали доступ інші користувачі. Також можна видалити коментар. Користувач може видалити будь-який в коментар до власного фото та ті які він створив сам.

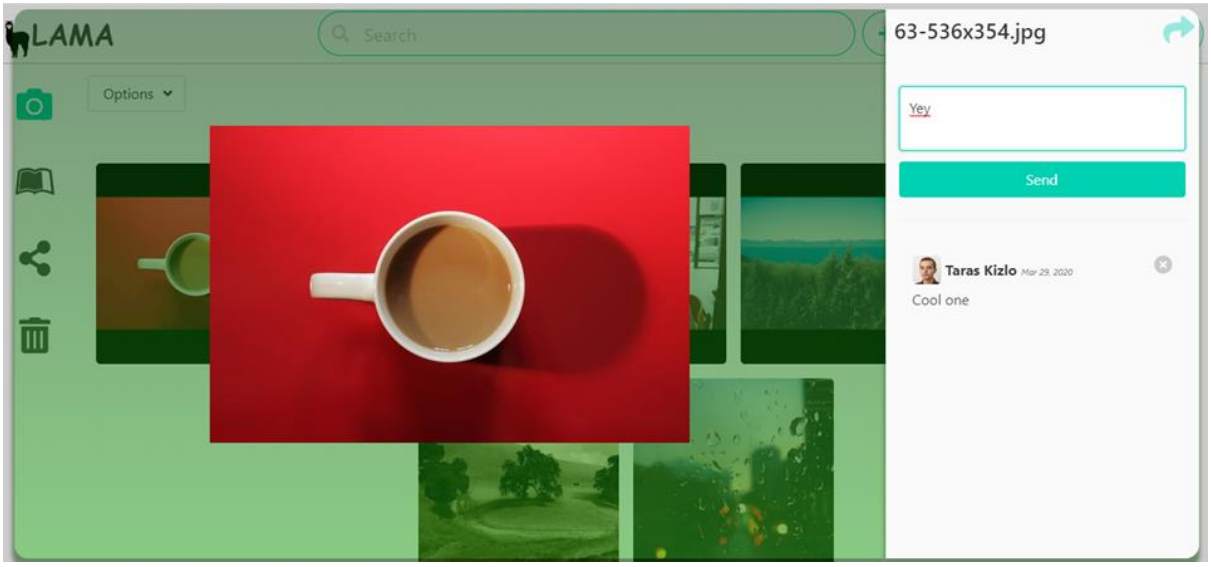

Рисунок 4.24 — Вікно списку коментарів.

4.2.6.4 Вбудований редактор

Вагомою особливістю аплікації являється вбудований редактор фотографій, ***рисунок 4.25***.

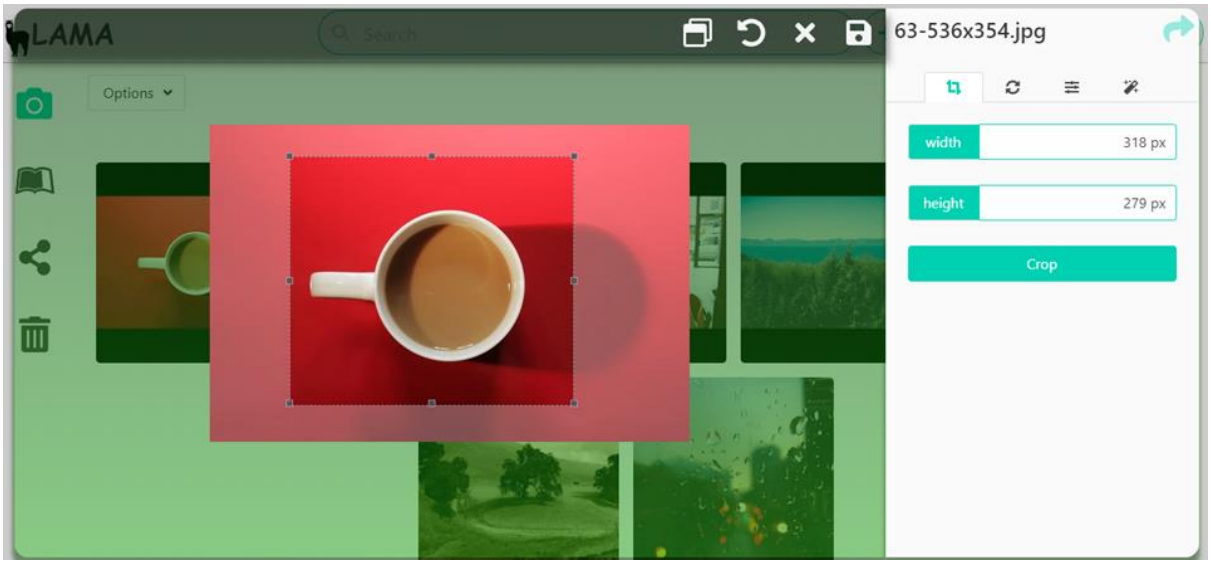

Рисунок 4.25 — Вікно вбудованого редактору.

Основними функціями редактора являються:

а) обрізання фото;

б) поворот фотографії в одну зі сторін чи віддзеркалення фото горизонтально та вертикально;

в) зміна яскравості чи контрасту;

г) застосування однієї із сімдесяти масок;

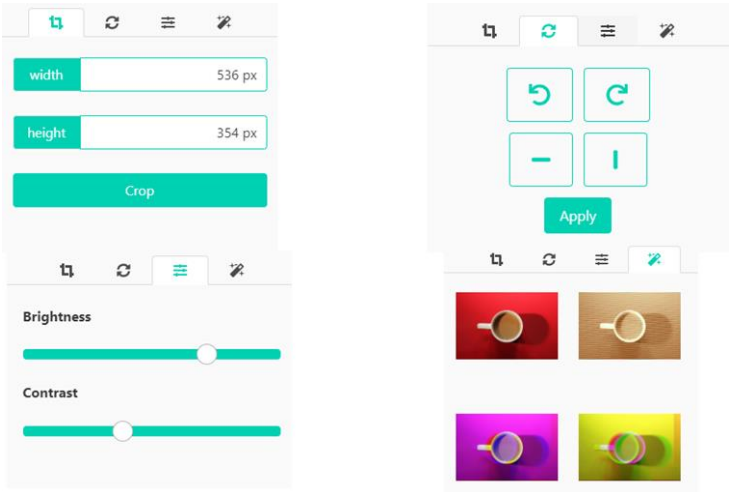

Рисунок 4.26 — Функції редактора.

Окрім того, користувач завжди може скасувати останню застосовану дію, відхилити чи зберегти зміни та при потребі відновити оригінальне зображення.

4.3 Сповіщення

4.3.1 Список сповіщень

Також механізм WebSocket'ів, який раніше використовувався для посилання адреси мініатюр клієнту, являється корисним при реалізації сповіщень. Оскільки, при потребі надіслати користувачу сповіщення, можна позбавити його необхідності оновлювати сторінку, ***рисунок 4.27***.

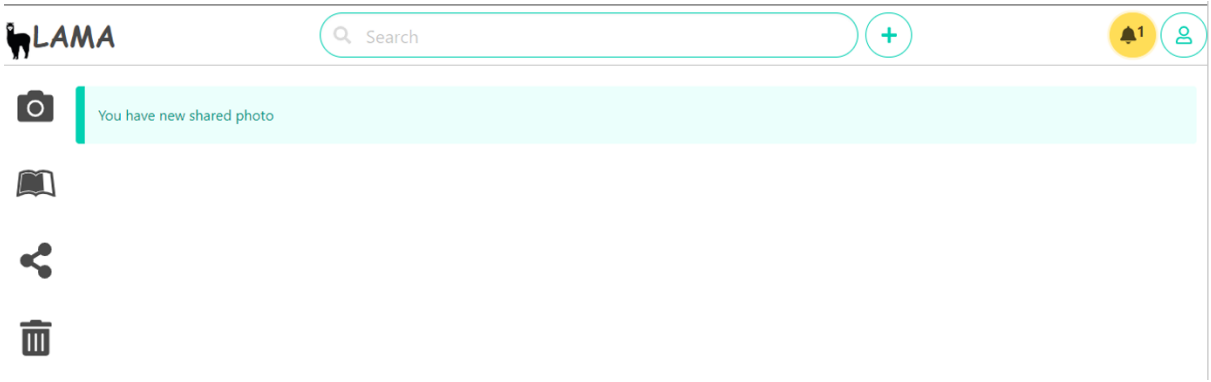

Рисунок 4.27 — Вікно перегляду списку сповіщень.

4.3.2 Статичні сповіщення

Також у системі були реалізовані два типи сповіщень: статичні та динамічні.

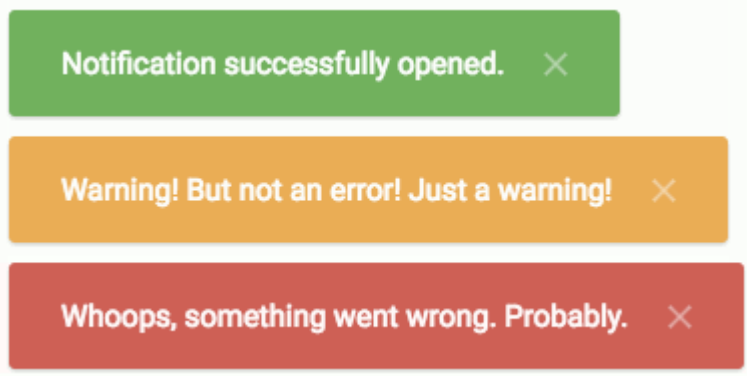

Рисунок 4.28 — Приклад сповіщень.

Статичні сповіщення — це ті сповіщення, які виникають при взаємодії між компонентами.

Як згадувалось раніше, усі компоненти поділені на спеціальні незалежні модулі, та не містять зв'язку один між одним. Тому для того, щоб реалізувати механізм обміну повідомлень на стороні клієнта довелось написати подібний

функціонал, що використовується для взаємодії між сервісами — шину повідомлень, ***рисунок 4.29***.

```
export class EventBusService {
  private events = new Subject<EventBase>();

  public emit(event: EventBase): void {
    this.events.next(event);
  }

  public on<TEvent extends EventBase>(action: (event?: TEvent) => void): Subscription {
    return this.events.pipe(filter((e: EventBase) => this.isCorrectEvent<TEvent>(e))).subscribe(action);
  }

  private isCorrectEvent<TEvent extends EventBase>(event: EventBase): event is TEvent {
    return (event as TEvent) !== undefined;
  }
}
```

Рисунок 4.29 — Програмний код реалізації шини повідомлень.

При такому підході, модуль сповіщень слідкує, за подіями та при потребі повідомляє користувача.

4.3.3 Динамічні сповіщення

Інший механізм сповіщень реалізований динамічним способом — коли користувач отримує повідомлення при відповіді на його запит.

У цьому випадку, усі запити можна розділити на два основні типи:

а) Запит без сповіщення;

б) Запит зі сповіщенням;

Запит зі сповіщенням може повертатись як явно (спеціальний тип результату), так і у випадку неопрацьованої помилки. При цьому не всі помилки необхідно показувати користувачу. Через те, що деякі винятки, являються результатом невірної роботи системи, саме для цього усі помилки, які варто показувати користувачу наслідують відповідний клас, який при обробці сервером буде обгорнутий в результат запиту, як сповіщення.

На частині клієнта додається спеціальний обробних всіх запитів, який розгортає сповіщення, для того щоб показати їх користувачу, та передає чисті дані наступним обробникам.

4.4 Альбоми

Як уже згадувалося раніше, фотографії можна групувати по альбомах, ***рисунок 4.30***. При цьому одна і та сама фотографія може міститись в декількох альбомах одночасно.

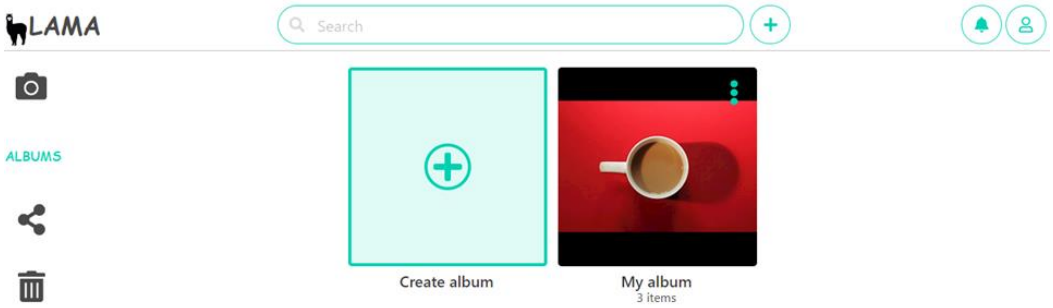

Рисунок 4.30 — Вікно перегляду списку альбомів.

Для альбомів передбачені можливість перегляду, створення, редагування та видалення.

Також є можливість легко змінювати фотографії, які входять в альбом, ***рисунок 4.31***.

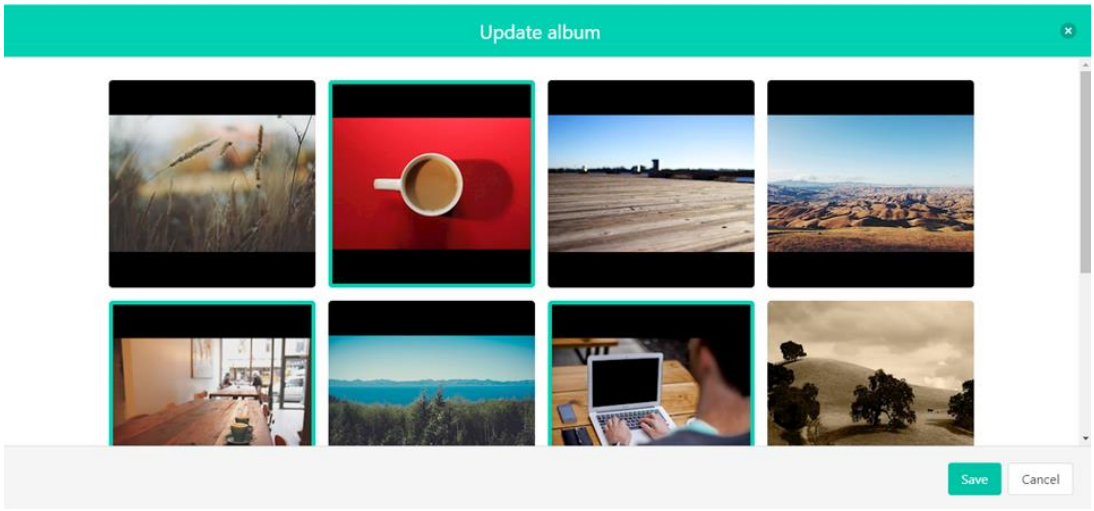

Рисунок 4.31 — Вікно зміни фотографій у альбомі.

Список фотографій в альбомі та на головній сторінці не відрізняються нічим окрім даних, що показуватися. Аби не дублювати код скористаємось тим самим компонентом, для якого розширимо поведінку в залежності від батьківської компоненти.

5 ПІДТРИМКА ТА РОЗГОРТАННЯ АПЛІКАЦІЇ

5.1 Вибір системи контролю версій

Система керування версіями — це програмний інструмент для керування версіями коду.

Це необхідний інструмент, який забезпечує користувачу можливість збереження кодової бази та при потребі повернутись до минулих версій коду.

Серед найпопулярніших виділяють Github, GitLab, Bitbucket, Azure Devops тощо. Я обрав Github оскільки він надає зручний інтерфейс для керування версіями, вбудовані можливості тестування, аналітики та розширення функціоналу із допомогою різноманітних інструментів.

5.2 Вибір системи планування

Для того щоб слідкувати за прогресом, необхідно описати завдання, та зберігати їх опис та замітки до них.

Я використовував систему TimeTracking від Github, ***рисунок 5.1***. Вона надає можливість створення карток, які описують новий функціонал чи помилку у системі, та дозволяють міняти стан картки в залежності від розміщення їх у відповідній колонці To do/In Progress Done. При цьому самі завдання можна розділяти відповідні часові проміжки, аби завжди мати готову версію продукту, та знати коли очікувати нових змін.

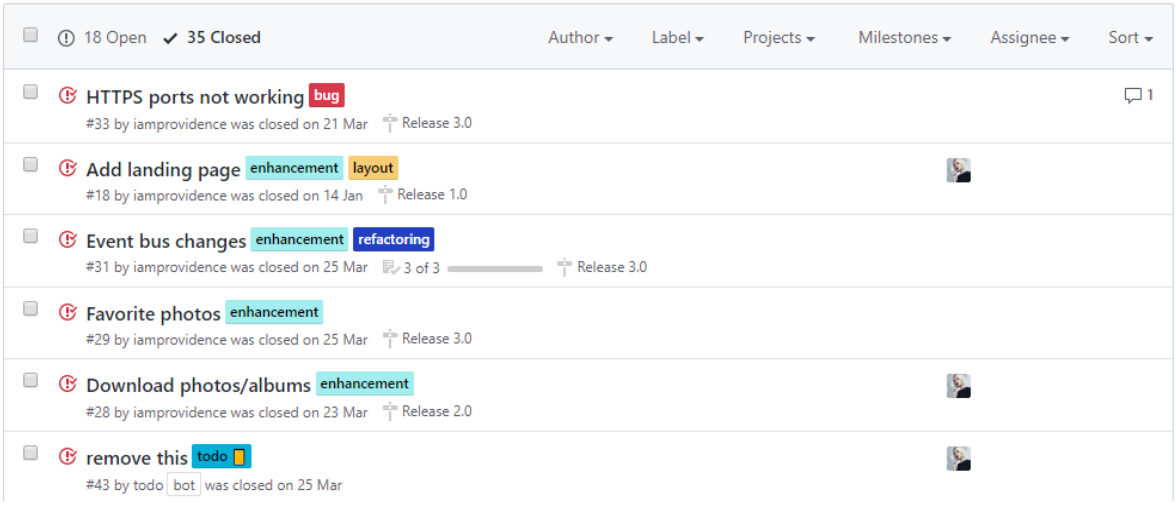

Рисунок 5.1 — Панель завдань від Github.

5.3 Неперервна інтеграція

Аби спростити процес розробки та підтримки аплікації використовувались різноманітні техніки неперервної інтеграції. Так із кожним новими змінами у систему, проєкт проходив декілька стадій.

а) валідація коду на дотримання єдиного стилю;

б) складання проєкту;

в) проходження тестів;

г) розгортання проєкт у відповідному середовищі;

Крім того, щоб уникнути повторення послідовності дій, були написані різноманітні скрипти за допомогою консольних команд, а також скриптової мови Python.

5.4 Розгортання аплікації

Щоб до даної системи отримали доступ інші користувачі її необхідно розгорнути на сервері. Одним із таких середовищ являється Azure. Але Azure надає лише місце для розміщення аплікація. Для того щоб спростити сам процес її розгортання використовується Docker. Він дозволяє записати набір команд та необхідних інструментів, для розгортання системи.

ВИСНОВОК

Отже, у рамках цієї роботи було реалізовано весь процес створення веб додаток для обміну та збереження фотографій на мікросервісній архітектурі, від планування до розгортання на хмарному сховищі. Дана система надає можливість додавати фотографії, здійснювати пошук, групувати їх по альбомах та додавати коментарі.

Аплікація являє собою вебпрограму з об'єктноорієнтованим доступом до бази даних. Для написання системи мені знадобилися різноманітні інструменти на подобі Active Server Page, Entity Framework, Elasticsearch, Docker тощо.

Також під час розробки, я переконався, що мікросервісна архітектура не підходить для всіх типів додатків. Вона покликана вирішити проблеми складності розробки великих проєктів та зовсім не піходить для незначних аплікацій.

Основними переваги використання мікросервісів являється:

а) чіткий розподіл по модулях. Завжди зрозуміло, як працює та чи інша частина коду. Це особливо важливо при розробці проєкту різними командами;

б) висока доступність. Додаток залишається доступним навіть у тому випадку, якщо працюють не всі сервіси;

в) різноманітні технології. При розробці кожного сервісу ви вільні вибирати інструменти, які найкраще підійдуть для конкретної бізнес-логіки. Наприклад, вибрати оптимальну базу даних і зручні інструменти для роботи з нею. Мікросервісна архітектура також дозволяє спробувати якусь нову технологію на окремому сервісі, не переписуючи при цьому весь додаток;

г) відносна простота розгортання. Кожен сервіс піднімається самостійно, що робить процес розгортання і налагодження більш чистим.

Серед недоліки використання мікросервісів можна зазначити:

а) складність розробки. Якщо вам потрібно швидке рішення (невеликий додаток, стислі терміни), то мікросервіси вам не підійдуть. Швидкість розробки - висока плата за доступність і модульність;

б) Складність підтримки. Кожен мікросервіс потребує окремого обслуговування, тому потрібний постійний автоматичний моніторинг.

в) узгодженість даних. Складність підтримувати цілісність даних між різними сервісами.

При цьому, я помітив можливості покращення аплікації. В перспективі розвитку проєкт є заміна HTTP протоколу на gRPC для внутрішньої взаємодії сервісів, ***рисунок 6.1***. Таким чином можна пришвидшити час відгуку, та не витрачати пам’ять на передавання непотрібних заголовків HTTP протоколу.

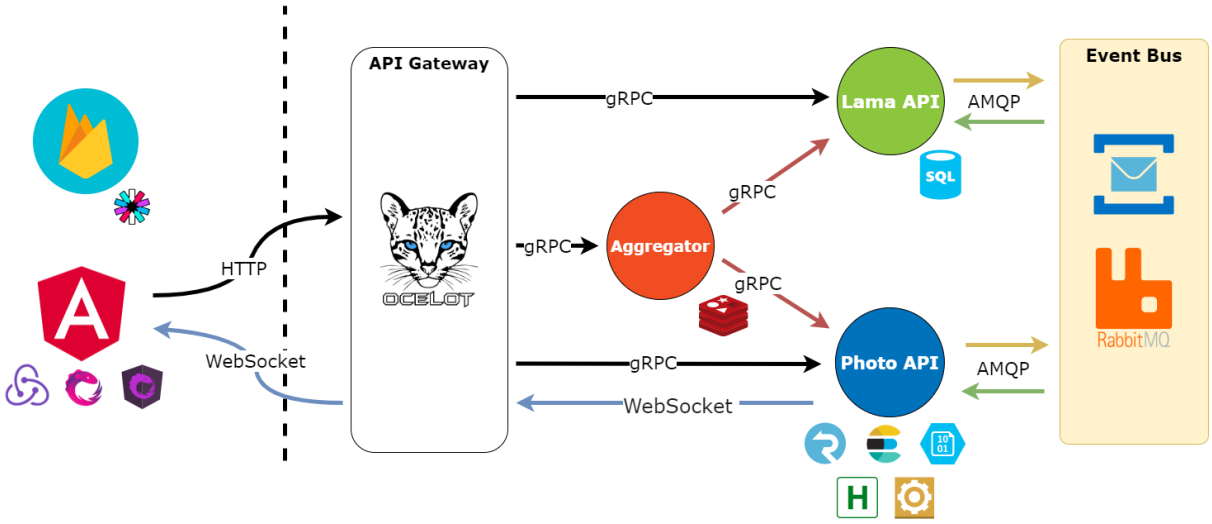

Рисунок 6.1 — Архітектура системи.

Також можна замінити спосіб взаємодії сервісів, на асинхронний. В такому випадку Gateway буде отримувати запити та зберігати їх як повідомлення. Сервісам тоді доведеться підписуватись на всі повідомлення та обробляти їх, а по завершені опрацювання повідомляти SignalR сервіс, який своєю чергою повідомить клієнта.

Оскільки всі запити у такій схемі стають асинхронними, варто винести логіку про сповіщення користувача в окремий сервіс, аби зменшити навантаження та уникнути дублювання коду, ***рисунок 6.2***.

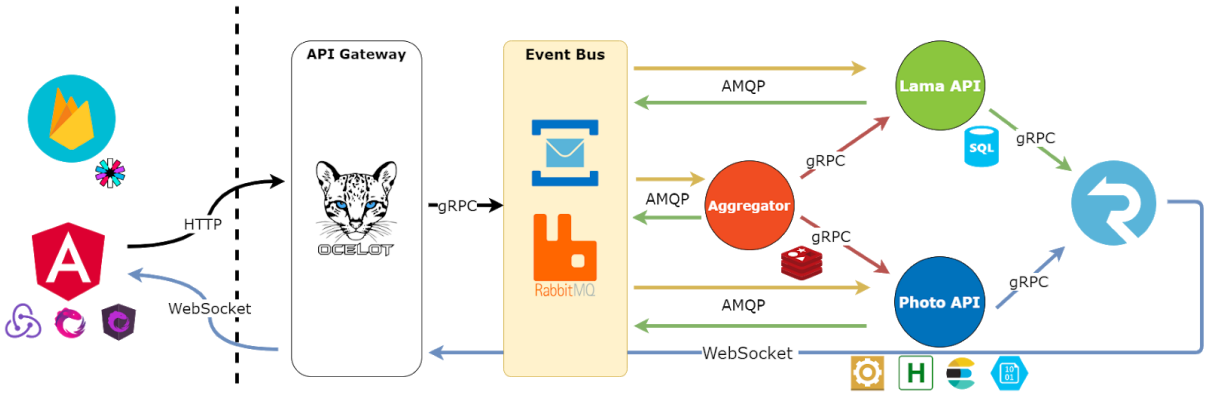

Рисунок 6.2 — Архітектура системи.

Інший варіант, без використання WebSocket протоколу передбачає реалізацію асинхронної HTTP моделі. В такому випадку на запит користувача відправлятиметься адреса за якою доступний результат та термін через який потрібно робити запит на цю адресу. У цьому разі доведеться створити сервіс, який відповідатиме за стан запитів та їх результат, ***рисунок 6.3***.

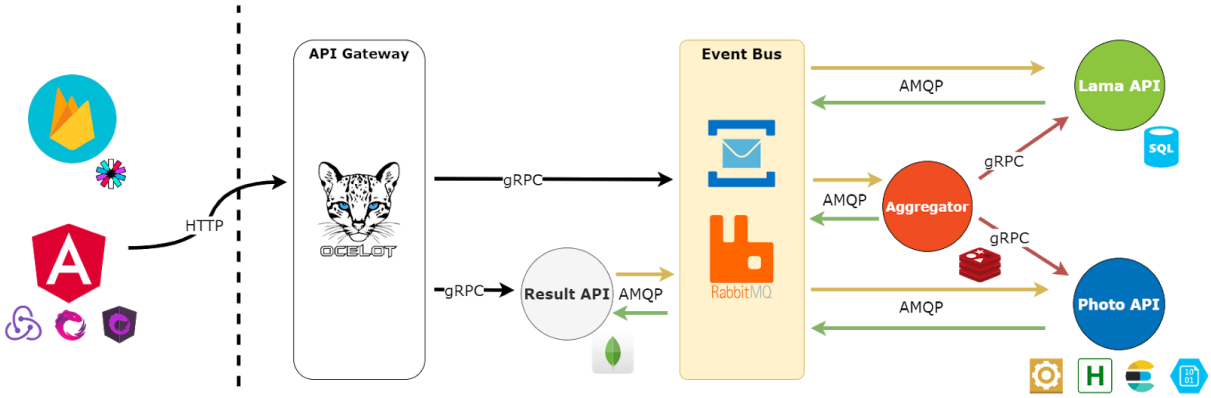

Рисунок 6.3 — Архітектура системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Берримен Дж., Тарнбулл Д. Релевантный поиск с использованием Elasticsearch и Solr, 2018. — 408 с.
2. Дэвид А. Шаппел. ESB — Сервисная Шина Предприятия — Петербург. 2008 — 294с.
3. Микросервисы. Как правильно делать и когда применять? — Режим доступу: https://habr.com/ru/company/dataart/blog/280083/
4. Andrew Troelsen. Pro C# 7: With .NET and .NET Core. — Minnesota, USA, 2017. – 1372 с.
5. Binildas A. Christudas. Service-oriented Java Business Integration — 2008. — 1500 с
6. Client Server model. — Available from: https://en.wikipedia.org/wiki/Client–server_model
7. Christian Nagel. Professional C#7 and . NET Core 2.0 7th Edition — Indianapolis, Indiana, USA, 2017. — 1948 с
8. Connection id when calling SignalR Core hub method from controller. — Available from: https://stackoverflow.com/questions/50367586/connection-id-when-calling-signalr-core-hub-method-from-controller
9. David Chappell. Enterprise Service Bus — Minnesota, USA, 2017. – 1472 с
10. How to download zip files in ASP.Net Core. — Available from: https://medium.com/@xavierpenya/how-to-download-zip-files-in-asp-net-core-f31b5c371998
11. How to get SignalR user connection id out side the hub class. — Available from: https://stackoverflow.com/questions/19447974/how-to-get-signalr-user-connection-id-out-side-the-hub-class
12. How to build a real time notification system using SignalR Core. — Available from: https://medium.com/@NanoBrasca/how-to-build-a-real-time-notification-system-using-signalr-core-1fd4160454fa

13. Implement API Gateway with ocelot. — Available from: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot
14. Implement microservices with RabbitMq. — Available from: https://insidethecpu.com/2015/05/22/microservices-with-c-and-rabbitmq/
15. Michael Bell. Service-Oriented Modeling: Service Analysis, Design, and Architecture. — 2008. — 1411 c.
16. Microservices architecture. — Available from: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/
17. Onion architecture. — Available from: https://blog.ploeh.dk/2013/12/03/layers-onions-ports-adapters-its-all-the-same/
18. SignalR documentation. — Available from: https://docs.microsoft.com/ru-ru/aspnet/core/signalr/hubs?view=aspnetcore-3.1
19. Unsupported media type error when posting to web API. — Available from: https://stackoverflow.com/questions/31526558/unsupported-media-type-error-when-posting-to-web-api
20. Using global variables in ASP.Net Core controller. — Available from: https://stackoverflow.com/questions/46482614/using-global-variable-in-asp-net-core-controller