------------------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. Контекст для дослідження [3](#_Toc148936394)

[Опис серверної частини та технологія реалізації (backend) 4](#_Toc148936395)

* 1. [Загальна архітектура 4](#_Toc148936396)
  2. [Авторизація 7](#_Toc148936397)
  3. [Почтова розсилка 9](#_Toc148936398)
  4. [Архітектура збереження файлів і ієрархія сховища………………..](#_Toc148936399)10

Опис клієнтського веб-додатку (frontend) [12](#_Toc148936401)

3.1 [Технічне завдання UI/UX дизайну 12](#_Toc148936402)

3.2 Практична реалізація[………………………………….………………13](#_Toc148936403)

Додаткові програми для роботи ПО. Інструкція по встановленню…....20

4.1 Загальні програми……………………………………………………...20

4.2 Алгоритм дій по конфігурації клієнтської сторони……………...….20

4.3 Алгоритм дій по конфігурації серверної сторони…………….……..20

# 

# 1. Вступ

## 1.1 Основна суть та мета розробки

Проєкт був створений як наочний 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 проєкт використовує підписну модель з різними рівнями доступу та ліцензіями на використання авторського матеріалу. Монетизаційна схема проєкта така: клієнт платить за підписку – ці гроші виплачуються авторам в залежності від кількості завантажень їх зображень (ліміт завантажень клієнта 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. Клієнт в свою чергу має змогу оцінити роботу автора, завантажити її чи підписатися на нього. Всі користувачі мають свою роль та можливості.

**1.2 Контекст для дослідження**

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 сайту, швидкість кодування та декодування зображень, об’єм займаного місця на сервері (розмір зображення), якість зображення в порівнянні з іншими методами.

## Опис серверної частини та технологія реалізації (backend).

***2.1 Загальна архітектура***

При розробці веб-застосунку зазвичай рекомендується розділити його на два рівні. API середнього рівня взаємодіє з базою даних, а веб-рівень зазвичай складається з інтерфейсу SPA (single page application) або MPA (multi page application). Таким чином, веб-додаток стає слабкіше пов'язаним, що спрощує управління та налагодження в довгостроковій перспективі.

Для написання серверної частини була використана мова php версії 8.0 та фреймворк Laravel версії 9.1 з використанням архітектурної моделі rest api. Тобто сервер обробляє запит клієтської сторони та повертає данні в json форматі, а клієнсь сторона в свою чергу отримавши відповідь, вже повністю відповідальна за рендеринг сторінки з адекватним відображенням інформації.

На рис. 1.1 можна помітити основну схему взаємодії клієнт-серверної моделі додатка

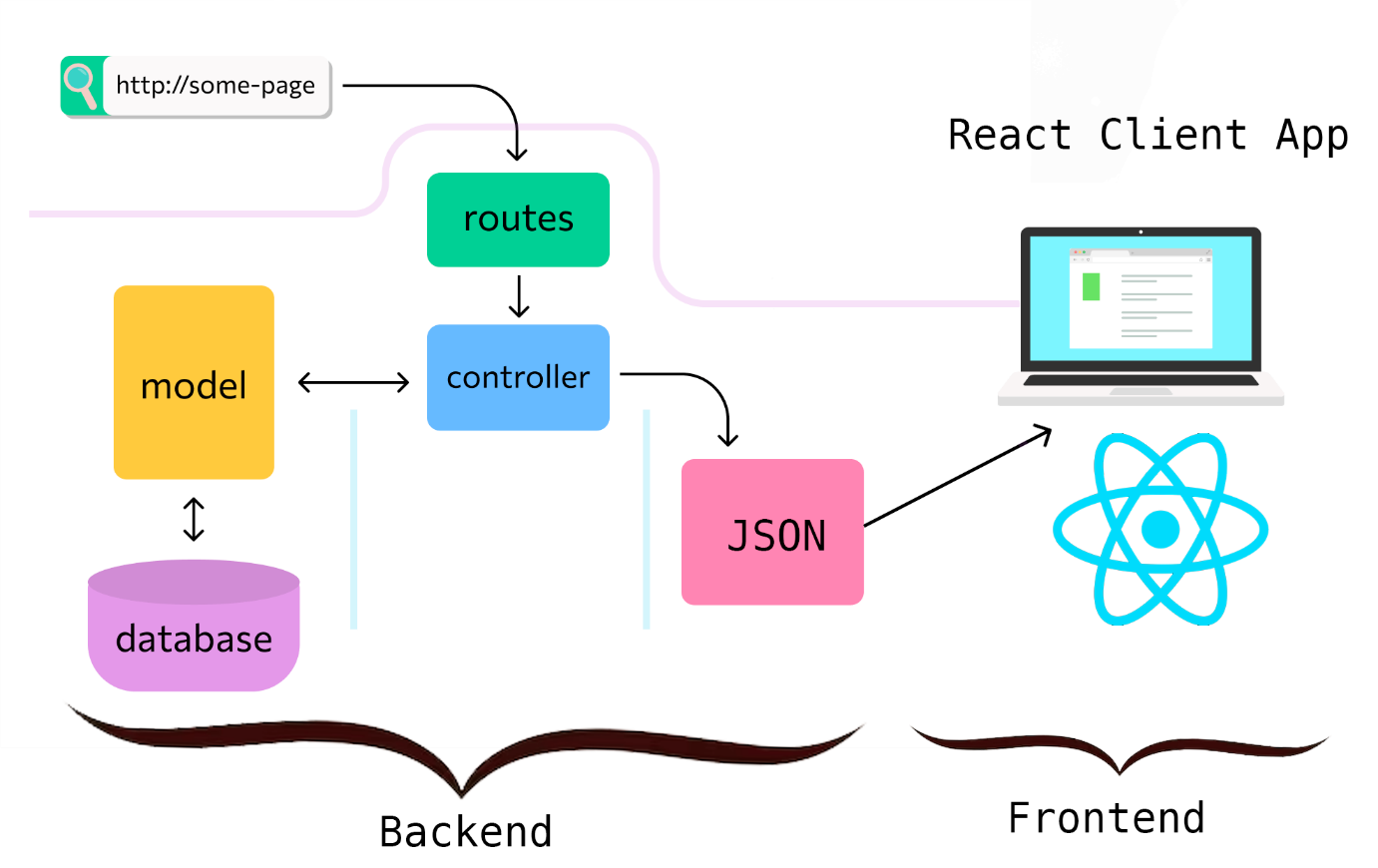


Рис. 1.1. Клієнт-серверна модель

**Business Logic Layer**

(BLL) - це компонент архітектури програмного забезпечення, який відповідає за реалізацію бізнес-логіки програми. Він знаходиться між рівнем подання та рівнем доступу до даних та відповідає за обробку даних та маніпулювання ними перед тим, як вони будуть представлені користувачеві або збережені у базі даних. BLL діє як проміжна ланка між рівнем уявлення та рівнем доступу до даних.

В моєму сервері рівню бізнес логіки відповідають такі компоненти: Controller (отримує запит від користувача, оперує рівнем доступу до даних, повертає відповідь клієнту), Middleware (посередники між маршрутизатором і контролером), Request (відповідальний за валідацію даних)

**Data Access Layer**

Оскільки Laravel має вбудований шар об'єктно-реляційного перетворювача (Object Relational Mapper, або ORM), я з легкістю зв’язав моделі скрипта з таблицями БД. Так Laravel **Eloquent** самостійно створює SQL-запити в залежності від історії взаємодії з моделлю і направляє їх до самого субд. Попередньо з’єднавшись з ним за допомогою данних, які вказані в конфігураційному файлі .env. Також Laravel має інтерфейс командного рядка **Artisan,** якийможе бути використаний для швидкого створення файлів: міграцій, моделей, “сидерів” і тд. Саме через подібні функції заточені під “**code-first**”підхід, мною була обрана така технологія.

**Data Base**

В якості системи управління базой даних я використав MySql версії 8.0 разом з phpMyAdmin для адміністрування бази в браузері. Так, як проект має безліч сутностей (**Entities**), які можуть мати відносини різних типів, моя база данних має реляційну модель і належить до 3-ї нормальної форми. В цьому можна впевнитися в рис 2.3. Сутностей звязаних з базою даних нараховується 29 класів. Загальна кількість таблиць – 41 таблиць. Сидерів (файл, який відповідає за правильне заповнення запису для таблицю) – 20 класів, факторів (файл, який генерує данні запису в таблицю) – 12 класів, міграційних файлів (файл, який створює таблицю в базі даних) – 35 файлів. Створення бд можливе за допомогою команди **php artisan migrate:refresh –seed**, що одночасно видаляє всі старі таблиці з даними, створює нові на основі актуальних даних про атрибути таблиці, та заповнює її на основі алгоритму описаному в класі **DatabaseSeeder.**

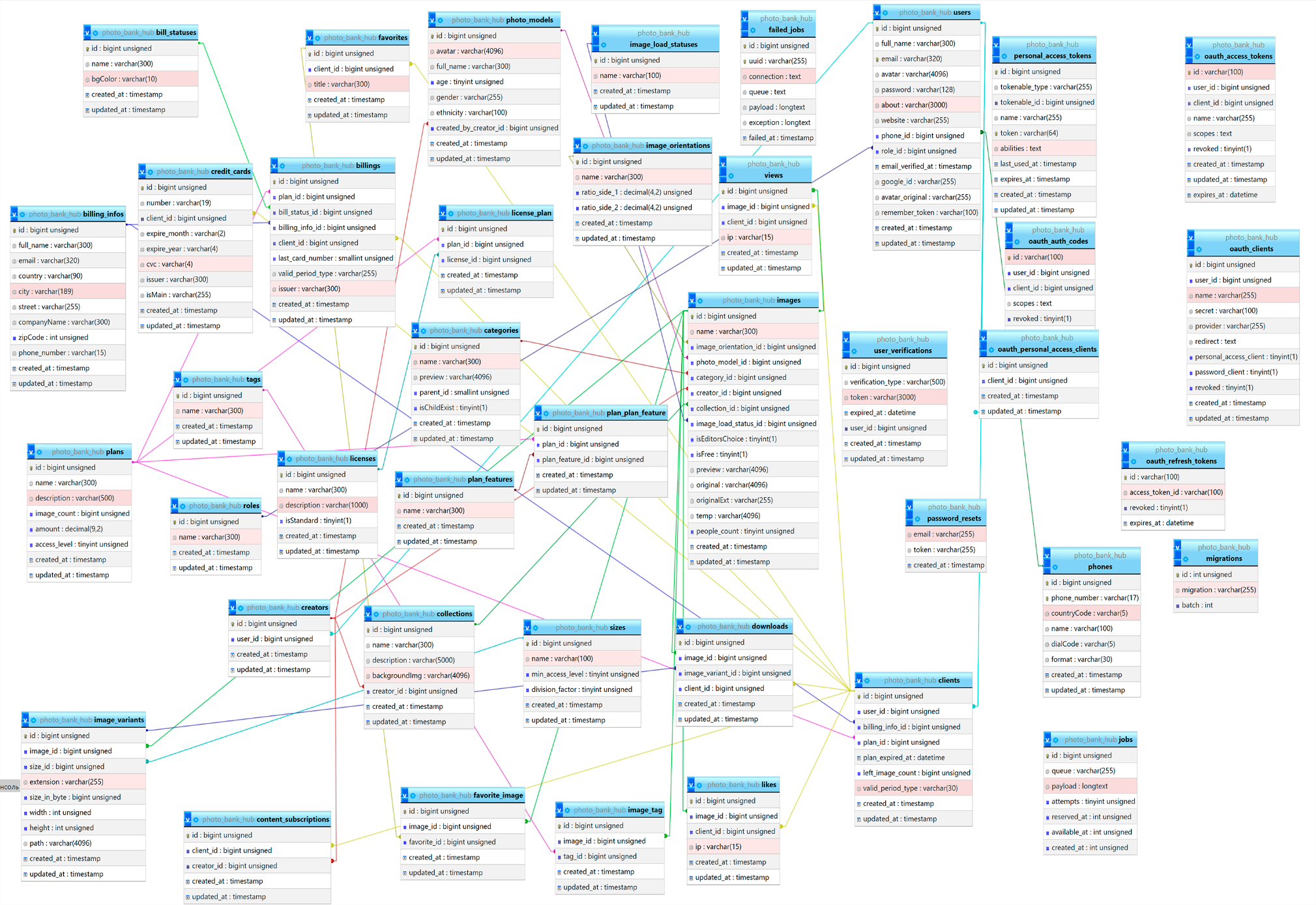


Рис 1.2 Повна діаграма бази даних

***2.2 Авторизація***

Авторизацію основана на протоколі **Oauth2**, яка вбудована в laravel пакет **Passport** версії 11.3. Оскільки API зазвичай не зберігають стан і не використовують сеанси, зазвичай використовують токени для збереження стану між запитами.

***Переваги протоколу***

**Зручність**: Замість того, щоб користувачеві створювати та реєструвати нові облікові записи в кожній програмі, яка підтримує OAuth 2.0, користувач може використовувати свої існуючі облікові дані платформи або соціальної мережі для авторизації. Це робить процес авторизації та взаємодії з додатками більш зручним та простим.

**Безпека**: OAuth 2.0 надає протокол авторизації на основі токенів, що означає, що безпосередні облікові дані користувача не передаються між різними програмами. Натомість використовуються унікальні токени (**access token** та **refresh token**), які ідентифікують користувача та надають доступ до захищених ресурсів. Це підвищує безпеку, тому що зловмисникам складніше перехоплювати та зловживати реальними обліковими даними користувача.

**Токен доступу** (access token) — це рядок, який клієнт OAuth використовує для надсилання запитів на сервер ресурсів.

**Токен оновлення** (refresh token) — це рядок, який клієнт OAuth може використовувати для отримання нового токена доступу без взаємодії з користувачем.

**Область дії** (Scope) — це механізм OAuth 2.0, що дозволяє обмежити доступ до облікового запису користувача. Програма може запросити одну або кілька областей, потім ця інформація надається користувачеві на екрані згоди, а токен доступу, виданий додатком, буде обмежений наданими областями.

**Ролі OAuth**

OAuth визначає чотири ролі: **1) Власник ресурсу** — це користувач , який дозволяє застосунку доступ до свого облікового запису. Доступ до облікового запису користувача обмежений обсягом наданих повноважень (наприклад, доступ на читання або запис). 2) **Клієнт** - це програма, яка хоче отримати доступ до облікового запису користувача. Перш ніж він зможе це зробити, він має бути авторизований користувачем, а авторизація має бути підтверджена API. 3) **Сервер ресурсів**: на сервері ресурсів розміщуються захищені облікові записи користувачів. 4) **Сервер авторизації**: Сервер авторизації перевіряє особу користувача, а потім видає токени доступу додатку.

З точки зору розробника програми API служби виконує роль ресурсу і сервера авторизації. Тому я буду називати ці ролі разом, роллю Сервісу або API.

Ось докладніше пояснення кроків на схемі:

1) Програма запитує у користувача авторизацію на доступ до ресурсів сервісу.

2) Якщо користувач авторизував запит, програма отримує дозвіл на авторизацію.

3) Програма запитує токен доступу у сервера авторизації (API), надаючи автентифікацію своєї особистості та дозвіл на авторизацію .

4) Якщо посвідчення програми автентифіковане і дозвіл на авторизацію дійсно, сервер авторизації (API) видає токен доступу до програми. Авторизація завершено.

5) Програма запитує ресурс у сервера ресурсів (API) і надає токен доступу для автентифікації.

6) Якщо токен доступу дійсний, сервер ресурсів (API) передає ресурс програмі .

Рис 2.1 Загальна схема взаємодії різних ресурсів в контексті oauth авторизації.

В рамках OAuth 2.0, представлення самого токена доступу (access token) реалізовувалась в формате JWT (JSON Web Token).

JWT містить зашифровану інформацію про користувача та дозволи, а також цифровий підпис, який дозволяє перевіряти справжність та цілісність токена. Цей токен передаватиметься між клієнтським додатком та захищеним ресурсом для перевірки доступу та авторизації.

**2.3 Почтова розсилка**

Для таких дій, як підтвердження електронної пошти, заміна пароля, повідомлення про закінчення підписки та інших, потрібен зрозумілий та простий API на базі популярної бібліотеки SwiftMailer для відправки електронної пошти.

SwiftMailer забезпечені драйверами для надсилання електронної пошти через SMTP, Mailgun, Postmark, Amazon SES та sendmail, що дозволяє швидко розпочати відправлення пошти через локальний або хмарний сервіс на ваш вибір.

***Конфігурація***

Поштові служби Laravel налаштовані через файл конфігурації config/mail.php. Кожна поштова програма, налаштована в цьому файлі, має свою власну унікальну конфігурацію і навіть свій власний унікальний транспорт, що дозволяє моєму додатку використовувати різні поштові служби для надсилання певних повідомлень електронної пошти. Наприклад, додаток може використовувати Postmark для надсилання транзакційних листів, а Amazon SES – для масових розсилок.

У конфігураційному файлі config/mail.php є масив mailers (рис 2.2). Цей масив містить зразок запису конфігурації для кожного з основних поштових драйверів / транспортів, що підтримуються Laravel, у той час як значення конфігурації default визначає, яка поштова програма буде використовуватися за замовчуванням, коли моя програма має надіслати повідомлення електронної пошти.

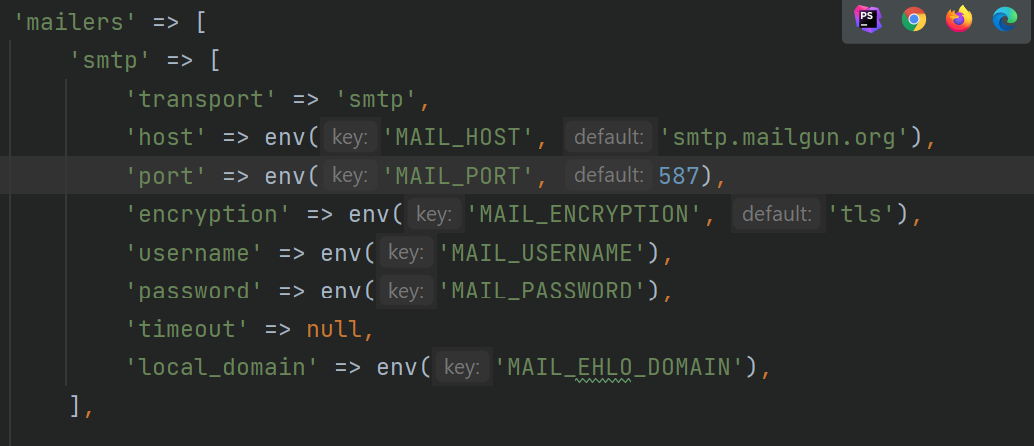


Рис 2.2 Конфігурація почтового драйверу

Для відправки повідомлень, наприклад, про закінчення підписки на мій сервіс, я використав інтерфейс Laravel Notifications для надсилання повідомлень по різних каналах: sms, email, база даних, slack.

Для того, щоб користувачі, наприклад, не чекали весь процес відправки повідомлення на почту, а в цей час могли повноцінно користуватися сайтом, в класі EmailVerificationNotification я імплементував інтерфейс ShouldQueue, який розміщує задачу (Job) в спеціальну чергу, дозволяючи виконувати подібні задачі в окремому процесі.

**2.4 Архітектура збереження файлів і ієрархія сховища**

Перед тим, як зберегти зображення отримане від користувача, система проводить ряд заходів для його стиснення та збереженню в різних розмірах. Це дозволяє досягти оптимального часу завантаження сторінок. Функціонал такого роду, було винесено в окремий класс під назвою **ImageHandler**, який належить до рівня **Service Layer**. Таке рішення відповідає принципу **DRY** (don't repeat yourself), тобто підхід, який дозволяє багаторазово використовувати один й той же відрізок коду в різних випадках з різними вхідними даними. Крім того завдяки класу Intervention\Image\Facades\Image, є можливість реалізувати функціонал додавання водяних знаків (watermark), що може запобігти використанню авторських матеріалів в неліцензійному контексті. Користувач, який завантажив зображення по підписці отримує його в повному розмірі, якості, форматі, тобто в оригінальному виді. Для запобігання витіка даних важливі дані зберігаються в приватній зоні, доступ в який відкривається за певних обставин (дійсність та відповідність рівня підписки, авторизованість і тд). В публічному секторі зберігаються: зображення категорій, стисненні зображення для загальної галереї (preview), аватари користувачів, статичні файли. Загальний вид сховища можна побачити на рис 2.3 і 2.4.

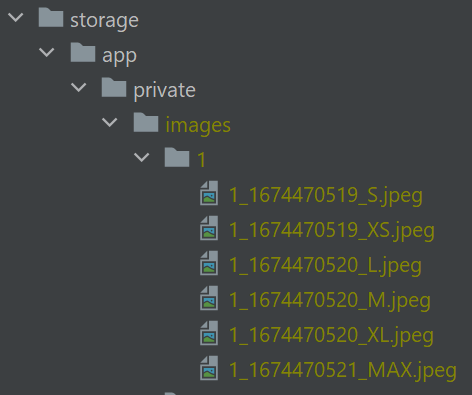


Рис 2.3 Приватна область сховища

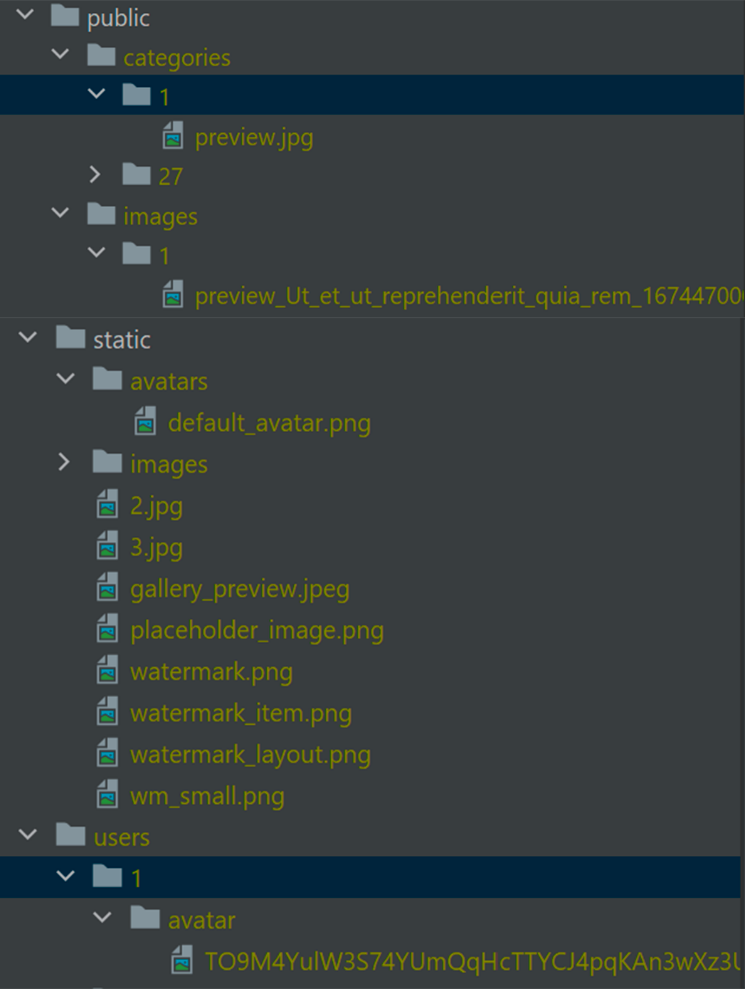
****

Рис 2.4 Публічна область сховища

1. **Опис клієнсько веб-додатку (frontend)**

**Технічне завдання UI/UX дизайну**

Під час проєктування інтерфейсу сайт, я відокремив основні вимоги до ui/ux дизайну проекту такого типу. А саме:

**Відображення фотографій**: Головна вимога для інтерфейсу сайту фотостока – це можливість красиво та ефективно відображати фотографії. Галереї та слайдшоу з плавними переходами та можливістю збільшення зображень на повний екран допоможуть користувачеві насолодитися роботами фотографа.

**Простота навігації та організація контенту**: Сайт повинен мати чітку та інтуїтивно зрозумілу навігацію, щоб відвідувачі могли легко переміщатися між різними розділами та галереями. Зручне бургер-меню або навігаційну панель допоможуть створити зрозумілу структуру сайту.

**Адаптивний дизайн**: Важливо врахувати, що сайт фотографа має бути адаптивним та добре працювати на різних пристроях (комп'ютери, планшети, мобільні телефони). Це дозволить користувачам переглядати роботи фотографа із зручністю незалежно від того, який пристрій вони використовують.

**Оптимізація завантаження фотографій**: Фотографії можуть займати багато місця, і важливо забезпечити оптимізацію завантаження, щоб користувачі не чекали тривалого відкриття кожного зображення. Використання стиснення зображень та кешування допоможе прискорити час завантаження та покращить загальний досвід користувачів.

**Контактна інформація та можливість зв'язку**: Сайт повинен надавати ясну контактну інформацію для фотографа, щоб користувачі могли зв'язатися з ним для роботи у замовлень, ставити запитання або зробити пропозиції. Форми зворотного зв'язку, а також посилання на соціальні мережі та електронну пошту можуть бути корисними для цієї мети.

Портфоліо та рекомендації: Важливо увімкнути секцію з портфоліо, яка показує кращі роботи фотографа. Це може включати різні категорії фотографій та детальні описи. Якщо є позитивні відгуки від клієнтів, їх можна додати на веб-сайт для підтвердження кваліфікації фотографа.

**Практична реалізація**

Розробка користувацького інтерфейсу була основана на технології **React** версії 18.2. Вибір пав саме на цю бібліотеку так, як в минулому я мав чимало практики використання цього інструменту для написання складних, навантажених інтерфейсів. А також завдяки ряду переваг, таких як:

**Широке застосування в індустрії**: React є однією з найпопулярніших бібліотек для розробки інтерфейсу користувача (UI) у веб-розробці. Вивчення та використання React у дипломній роботі може бути корисним для майбутньої кар'єри, оскільки багато компаній та проектів в індустрії вже використовують React.

**Компонентний підхід**: React слід компонентної архітектури, що означає поділ інтерфейсу користувача на незалежні компоненти. Компоненти React можуть бути перевикористані, що полегшує розробку, підтримку та масштабування програми. Це особливо корисно при розробці великих проектів, включаючи дипломну роботу, яка може бути складною та містити безліч різних компонентів.

**Зручність та гнучкість**: React пропонує простий та зрозумілий синтаксис на мові JavaScript. Він також підтримує безліч інструментів та бібліотек, які розширюють його функціональність та спрощують розробку. Наприклад, для керування станом програми можна використовувати бібліотеку Redux або контекстні хуки. Це дозволяє студенту більш гнучко та ефективно розробляти дипломну програму за допомогою різних інструментів та бібліотек.

**Продуктивність**: React використовує віртуальний DOM (Virtual DOM) для відстеження змін у компонентах та ефективного перемальовування лише потрібних елементів при зміні стану програми (дивитися рис 3.1). Це дозволяє досягти хорошої продуктивності та швидкого відгуку інтерфейсу.

Ви декларативно описуєте ієрархію компонентів свого веб-інтерфейсу і вводите її у віртуальну модель DOM React. Потім React піклується про синхронізацію вашого інтерфейсу користувача з фактичною DOM-моделлю браузера у відповідні моменти часу. Реалізація моделі Virtual DOM React є самодостатньою і не залежить від браузера; її навіть можна використовувати для рендерингу на стороні сервера. Virtual DOM виконує оптимізоване порівняння свого внутрішнього стану з DOM-моделями браузерів і вносить мінімальні зміни, необхідні для збереження узгодженості інтерфейсу користувача.

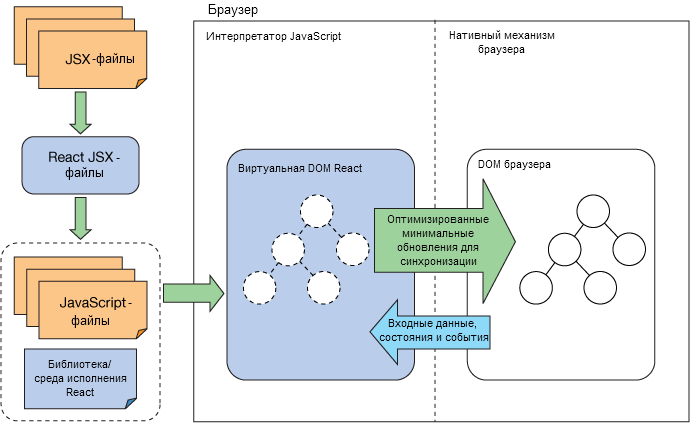


Рис 3.1 Принцип роботи бібліотеки React

**Спосіб організації структури клієнтського додатку**

На відміну від фреймворку Angular додаток побудований на бібліотеці React може мати різноманітну архітектуру залежно від маштабу розробки та інший змінних. Тобто, React надає набір інструментів і компонентів для розробки інтерфейсу користувача, але не накладає обмежень на організацію проекту. Він надає компонентний підхід, що дозволяє розбивати інтерфейс на незалежні компоненти, полегшуючи розробку, тестування та повторне використання коду.

В якості точки входу виступає файл index.js. Він буде запущений відразу після завантаження вашої програми командою **npm run start**. В ньому буде створений корінний компонент в якому й буде відбуватися подальший рендеринг. Моя ситуація, передбачає, насамперед обертання компонента **App.jsx** (основного компонента) в компонент **<BrowserRouter>**, який зберігає поточне місце розташування в адресному рядку браузера, використовуючи чисті URL-адреси та **<Provider store={store}>**, що надає спосіб передавати дані (в моєму випадку дерево станів **Redux**) через дерево компонентів без необхідності передавати реквізити (props) вручну на кожному рівні. Цей код можна побачити на рис. 3.2

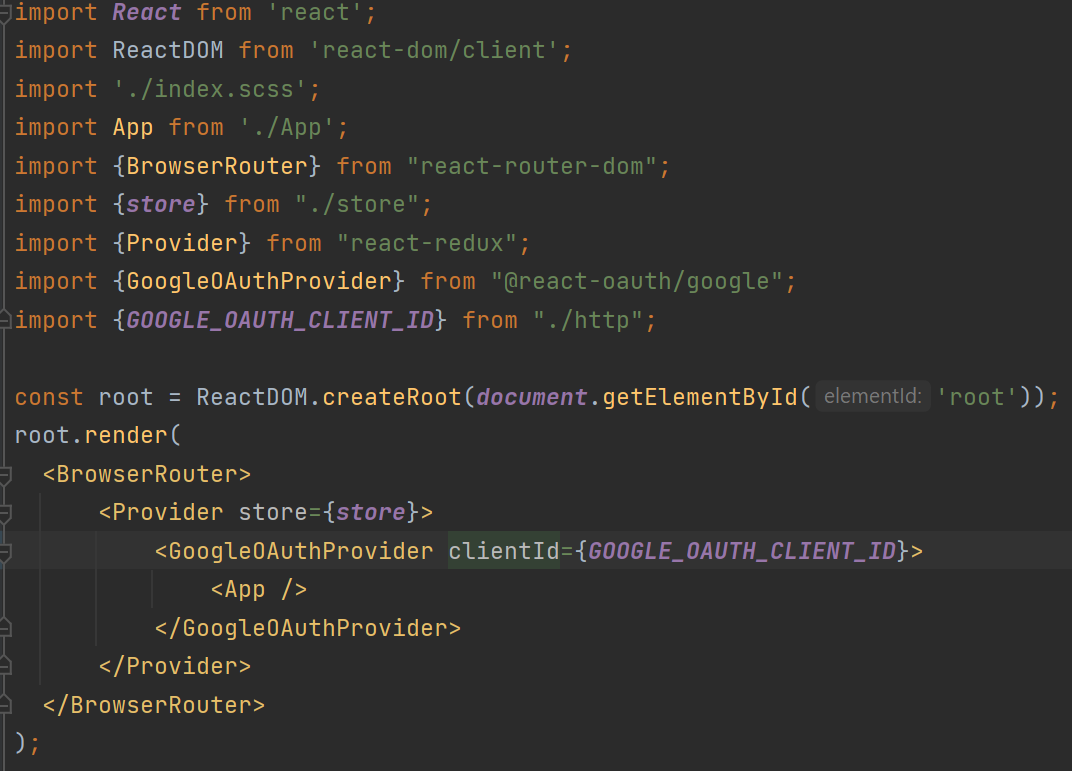


Рис 3.2 Код в точці входу додатку

**Маршрутизація**

Перейдемо до компонента App.jsx. Тут я хочу зупинитися на способі маршрутизації в додатку. Для цього я вирішив використати популярну бібліотеку **react-router-dom версії 6.5**, що дозволяє оновлювати URL-адресу при натисканні на посилання без повторного запиту іншого документа з сервера. Це забезпечує швидшу взаємодію з користувачем, оскільки браузеру не потрібно вимагати абсолютно нового документа або повторно оцінювати ресурси CSS та JavaScript для наступної сторінки.

Маршрутизація на стороні клієнта включається шляхом створення компонента **<Routes>** , який обертає компоненти <**Route>**. В свою чергу перший компонент **<Route>** в моєму випадку представляє собою, перемикач глобального шаблону (для клієнта, автора, чи адміністратора), а вже в його параметрі **children** перебираються всі основні маршрути сайту, які захищенні компонентом **<AuthGuard>** запобігаючому несанкціонованому доступі до контенту (чи авторизований користувач, яка в нього роль і тд). Все це міститься в компоненті Routing, див на рис 3.3

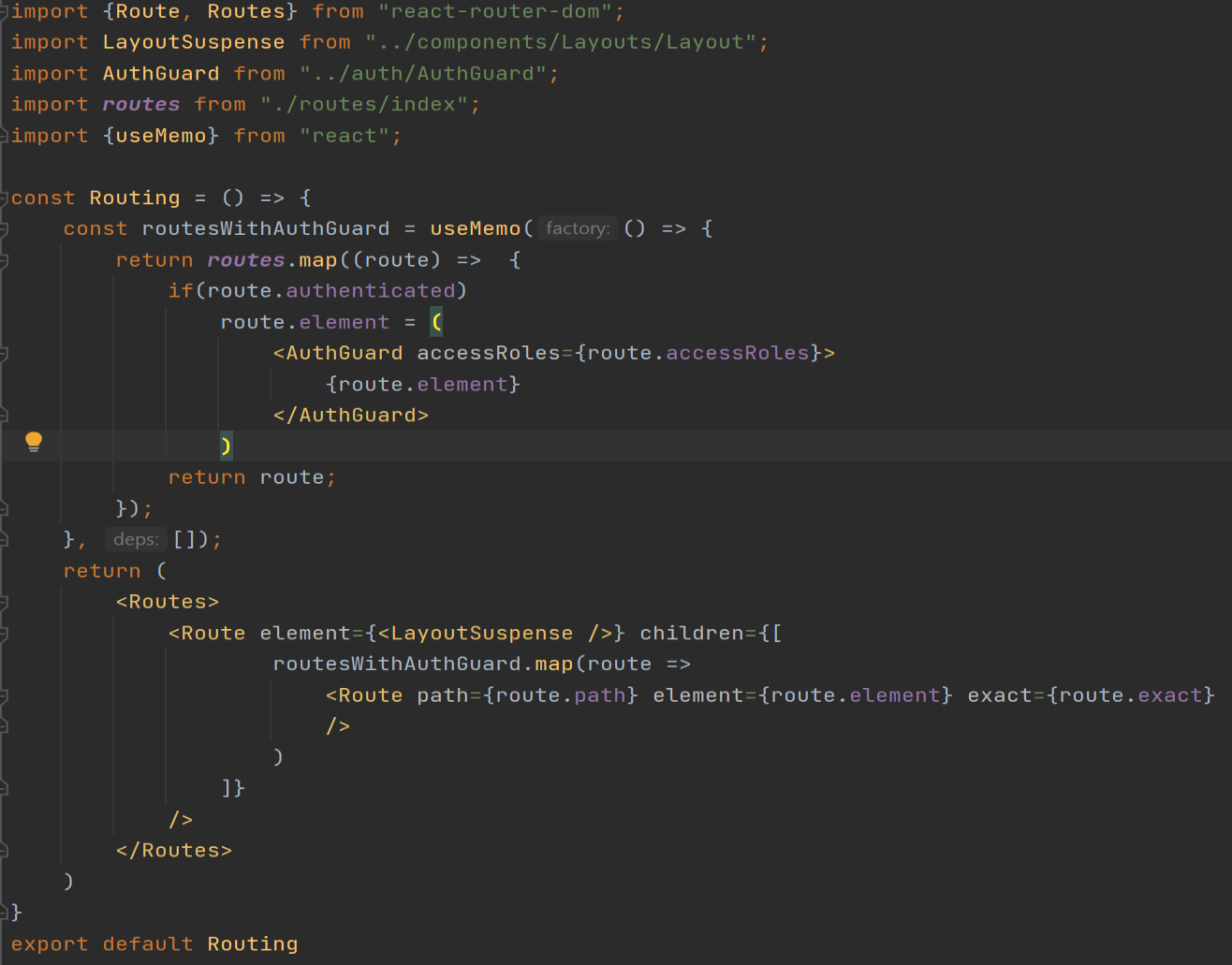


Рис. 3.3 Код маршрутизації

**Оптимізація завантаження в React**

Стосуючись теми оптимізації часу завантаження сторінок, то тут я використав стандартний підхід **Lazy Loading**, який дозволяє відкласти завантаження коду компонента, доки він не буде відображений вперше.

Саме завдяки функції **React.lazy()** вдається досягти динамічного завантаження сторінки при кожному переході в новий розділ. Що дає значне прискорення загальної роботи додатку в усіх випадках. Завантаження лише необхідних компонентів або ресурсів скорочує час завантаження і зменшує споживання пам'яті, що призводить до більш швидкої та чуйної користувальницького інтерфейсу. Тому, що тільки потрібні компоненти завантажуються в пам'ять, у той час, як компоненти, що не використовуються, можуть бути вивантажені або видалені з пам'яті, що звільняє ресурси і запобігає витоку пам'яті.

В моєму випадку всі динамічні компоненти, обернені в компонент Loadable, який додає анімацію (loader) на протязі завантаження, це дозволяє легше переносити очікування великих ресурсів. Як правило це всі окремі сторінки сайту. Приклад лінивого завантаження рис 3.4.

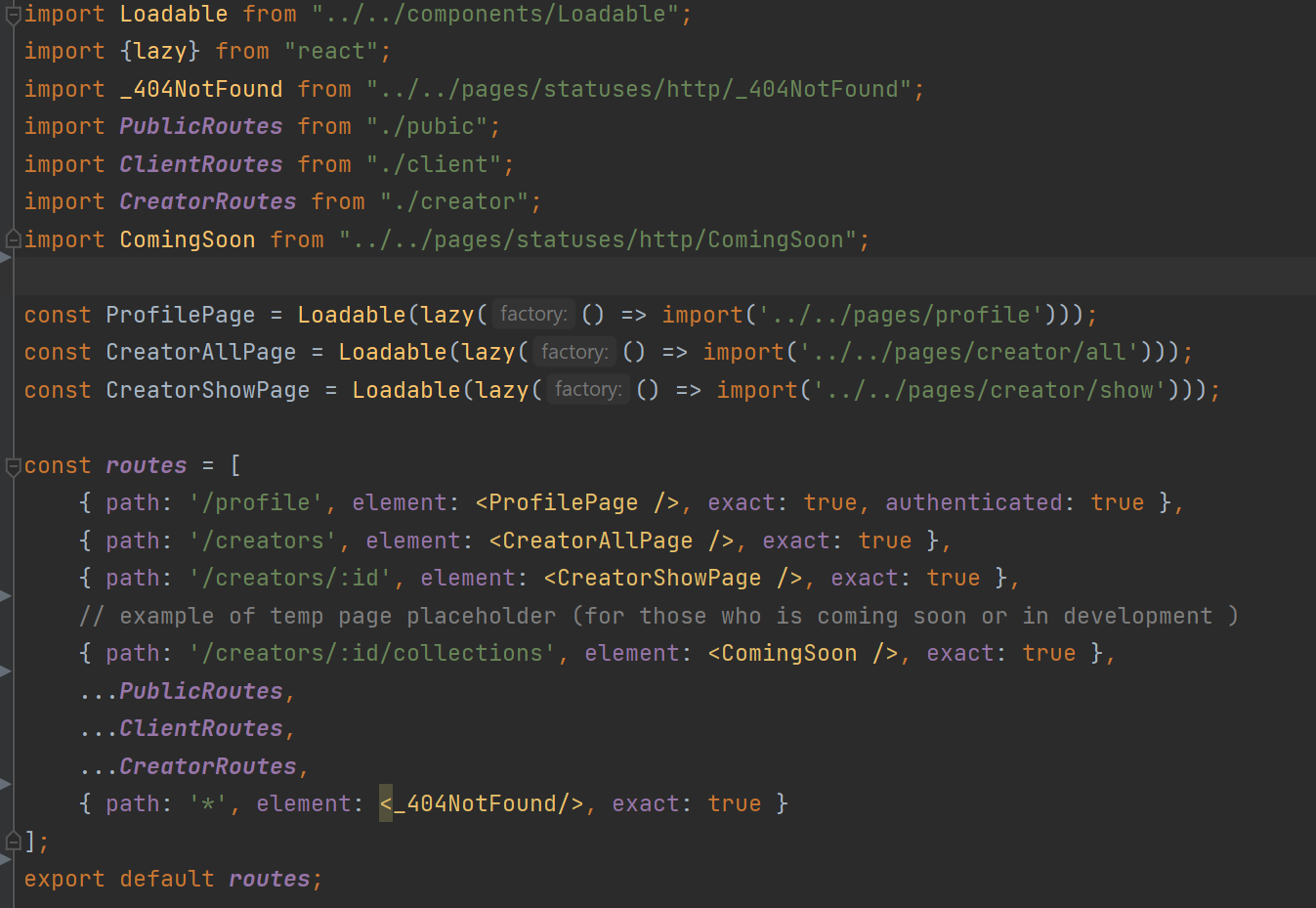


Рис 3.4 Lazy Loading в файлі-конфігурації маршрутизації клієнтського додатка

**Менеджер стану**

Традиційно для React (а також іноді і для Angular) додатків використовують бібліотеку управління станом – Redux (в моєму випадку версії 4.2), яка працює і в мене для передачі інформації про профіль поточного користувача і безпечні дані кредитної картки.

Хоча в react існують вбудовані методи реалізації подібної концепції на основі хука useContext, але вони погано масштабуються в великому проекті.

**Redux** - містить низку інструментів, що дозволяють значно спростити передачу даних сховища через контекст.

Redux слідує принципу передбачуваності - зміна стану здійснюється через чисті функції-редюсери, які перетворюють поточний стан і дії (actions) в новий стан. Це робить код більш передбачуваним та легко тестованим. Крім того, завдяки односпрямованому потоку даних та чіткому поділу логіки програми Redux забезпечує просту розширюваність.

Redux пропонує зручні інструменти для керування станом програми. Наприклад, поділ стану на частини за допомогою різних редюсерів дозволяє логічно групувати дані та відомості більш зрозумілий і організований спосіб.

Також я встановив доповнення **redux-devtools-extension**, яке полегшує відстеження змін стану та налагодження програми.

Для асинхронних функій, які звертаються до сервера, я використав проміжне програмне забезпечення - **redux-thunk.** Це дозволило писати функції з логікою всередині, які можуть взаємодіяти зі сховищем dispatch і getState методами Redux за межами компонента. Це також дозволяє перевикористати їх в інших компонентних модулях проєкту.

**Сервіси**

В моєму випадку всі сервіси (шари роботи з даними) займаються взаємодією з API та відстежують стан запита. Реалізую такий підхід використовуючи бібліотеку Axios. Конфігурацію цього http-клієнта я проводив в окремому модулі в папці http, приклад можна побачити на рис 3.5.

**Axios** - це полегшений HTTP-клієнт, заснований на $http сервісі Angular.js v1.x і схожий на власний JavaScript Fetch API. Axios заснований на обіцянках (Promise), що дає вам можливість використовувати переваги JavaScript asyncі await, отримувати більш асинхронний код, що добре читається.

Також є можливість перехоплювати, скасовувати запити, існує вбудований захист на стороні клієнта від підробки міжсайтових запитів. Після деяких важливих запитів, передбачений функціонал показу alert-повідомлення, який зникає через деякий час підтверджуючи, що запит виконаний успішно або ні. За це відповідає доповнення **react-toastify**.



Рис 3.5 Демонстрація конфігурації axios-клієнта

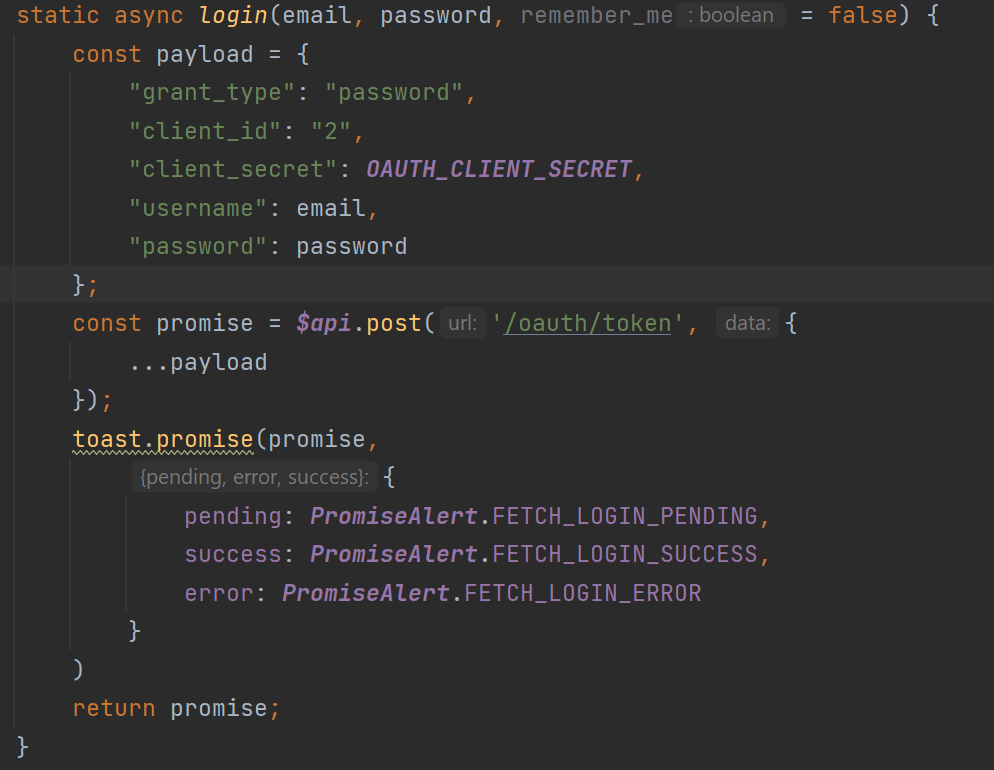


Рис 3.6 Запит на сервер використовуючи сервіс на основі axios

**Кастомні (власні) хуки.**

**Хуки** представляють функції, які дозволяють підключитися до стану та інших можливостей, які є в React.

**Кастомні React Хуки** – це можливість створення власних ділянок для перевикористання будь-якого функціоналу, винесений в окремий модуль, який в свою чергу може використовувати вбудовані хуки React (такі, як useState, useEffect, useMemo, useCallback).

Для власної розробки я прописав такі хуки:

**useDebounce** – дає можливість відправляти запити на api з вказаною затримкою (наприклад, при вводі пошукового тексту).

**useFetching** – дає можливість зручно відстежувати стан http запити в компоненті, коли створення окремого сервісу для цього не доречне.

**useObserver** – дає можливість відстежувати видимість певного елемента на екрані (використовувався в випадках динамічної підгрузки елементів)

**useQueryParams** – дає змогу перетворити масив параметрів на строку параметрів для url запита.

**4. Додаткові програми для роботи ПО. Інструкція по встановленню**

**4.1 Загальні програми**

1. Встановити редактор коду Visual Studio Code - <https://code.visualstudio.com/>
2. Встановити Git (інструкція - <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>) - <https://git-scm.com/downloadsє>

**4.2 Алгоритм дій по конфігурації клієнтської** **сторони**

Весь проєкт (клієнт+api) зберігається в гіт-репозиторії. Його можна завантажити за посиланням -<https://github.com/EvgeniyRyabchuk/PhotoBankHub>

Для запуску клієнського проєкта вам знадобляться такі програми:

1. Встановити Node JS - <https://nodejs.org/en>

Алгоритм запуску:

1. Відкрийте консоль в папці client

2. Встановіть залежності командою: npm i

3. Запустіть проект командою: npm run start.

**4.3 Алгоритм дій по конфігурації серверної сторони**

*Інструкція розрахована на windows машини*

Для запуску серверної частини вам знадобляться такі програми:

1. Встановити готова збірка програм Open Server - <https://ospanel.io/>. Потім зайти в налаштування – модулі. Далі вибрати наступні версії програм:

**Apache\_2.4-PHP\_8.0-8.1**

**PHP\_8.0**

**MySQL-8.0-Win10**

1. Увімкнути imagick в конфігурації сервера. Тобто розкоментувати значення **extension=imagick**

та замінити значення лімітів (на майбутнє)

**post\_max\_size = 100M**

**upload\_max\_filesize = 100M**

1. Встановити пакетний менеджер composer - <https://getcomposer.org/download/>

Алгоритм запуску:

1. Відкрийте консоль в папці api

2. Завантажити залежності laravel проекта:

**composer i**

3. Згенерувати ключ проекта:

**php artisan key:generate**

5. Створити приватний і публічний ключ для шифрування користувацьких паролів:

**php artisan passport:install**

6. створити oauth кліентів (Access Client, Grant Client):

**php artisan passport:client –password**

7. Створити ярлик на папку storage в папці public:

**php artisan storage:link**

8. Створити таблиці бази даних із заповненням (попередньо створивши бд в phpMySqlServer або MySql Workbrench):

**php artisan migrate –seed**

9. Запустити сервер:

**php artisan serve**

**Примітки:**

* Якщо команди php, node, composer не знаходяться командним рядком, треба впевнитися, що вони є в змінних оточення windows