МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

**Звіт**

**з курсу «Об'єктно-орієнтовний аналіз і проектування ПЗ»**

Виконала команда:

LastDayEfforts:

Боровий Орест (ПМі-52)

Монець Данило (ПМі-52)

Львів 2020

Метою даної роботи є створення веб-застосунку та відкритого API, побудованого на основі мікросервісної архітектури та RESTful API, який буде обслуговувати підсистему для перенесення стилю зображення, а також розробка зручного API для сторонніх розробників, які бажатимуть включити в свою систему можливості системи з перенесення стилю зображення.

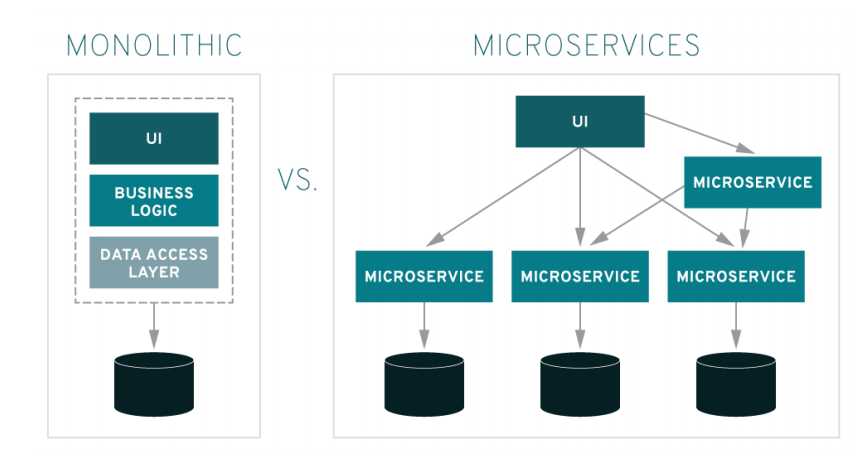
У результаті виконання роботи розроблено два основні сервіси: сервіс, який обслуговує підсистему з перетворення зображень та сервіс для користувацького інтерфейсу, запроектовано дві бази даних PostgreSQL та сховище даних Redis, підключений брокер повідомлень RabbitMQ та створено дві асинхронні черги задач Celery.

У ході роботи розроблено веб-застосунок, який розширює можливості користувачів у сфері редагування та перетворення зображень, оскільки дає змогу накладати на зображення певний вибраний стиль. Система побудована на RESTful API та мікросервісній архітектурі, реалізовано зручний графічний користувацький інтерфейс. Застосунок може бути популярним серед користувачів, які люблять редагувати або змінювати свої фотографії і графічні зображення, зокрема користувачів соціальних мереж.

Використані технології:

1. Система побудована на основі мікросервісної архітектури та RESTful API

Мікросервіси – це архітектурний підхід до побудови додатків. Архітектурно мікросервіси розподілені та слабко з'єднані між собою, тому зміни однієї команди на одному сервісі не зламають весь додаток. Основна перевага від побудови мікросервісів полягає в тому, що команди розробників здатні швидко створювати нові компоненти додатків для задоволення мінливих потреб бізнесу



REST - скорочення від Representational State Transfer, що можна перекласти як «передача репрезентативного стану». Це стиль проектування розподілених систем за допомогою обмежень. Центральною абстракцією в REST є ресурс. А головні обмеження виглядають так:

• клієнт-серверна модель;

• взаємодія без збереження стану;

• логічний інтерфейс.

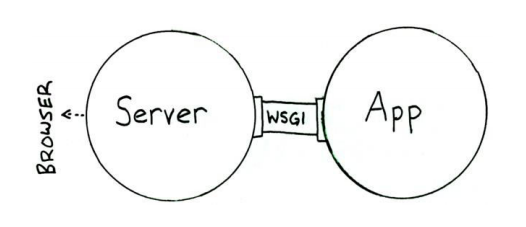
1. Система побудована з використанням фреймворку Django та інтерфейсу WSGI

WSGI (Web-Server Gateway Interface) є нащадком CGI(Common Gateway Interface). Коли веб почав розвиватися, CGI розростався через підтримку величезної кількості мов і через відсутність інших рішень. Однак, таке рішення було повільним і обмеженим. WSGI був розроблений як інтерфейс для маршрутизації запитів від веб-серверів (Apache, Nginx і т.д.) на веб-додатки. У найпростішому випадку WSGI складається з двох основних сутностей:

• веб-сервер (Nginx, Apache і т. д.);

• веб-додаток, написаний на мові Python.

Принцип роботи: Веб-сервер виконує код і відправляє пов'язану з http-запитом інформацію і callbackфункцію в веб-додаток. Потім запит на стороні програми обробляється і надсилається відповідь на веб-сервер.



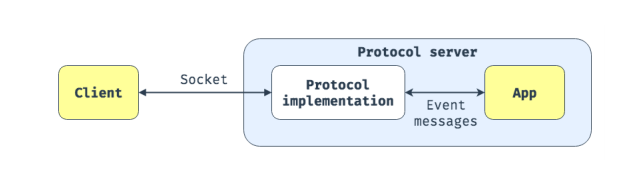
1. Також було використано можливості фреймворку FastAPI та інтерфейсу ASGI

FastAPI - відносно новий веб-фреймворк, написаний на мові програмування Python для створення REST (а якщо сильно постаратися то і GraphQL) API, заснований на нових можливостях Python 3.6+, таких як: підказки типів (type-hints), нативна асинхронність (asyncio). Крім усього іншого, FastAPI щільно інтегрується з OpenAPI-schema і автоматично генерує документацію для вашого API за допомогою Swagger і ReDoc.

Високорівнево ASGI можна розглядати як сполучна ланка, яка дозволяє асинхронним Python серверів і додатків взаємодіяти один з одним. Він повторює безліч архітектурних ідей з WSGI, і часто подається як його наступник з вбудованою асинхронністю. ASGI складається з двох різних компонентів:

1. Сервера протоколу (protocol server) - слухає сокети і перетворює їх в з'єднання і повідомлення про події всередині кожного з'єднання.

2. Додатка (application), яке живе всередині сервера протоколу, його екземпляр створюється один раз для кожного з'єднання і обробляє повідомлення про події в міру їх виникнення.

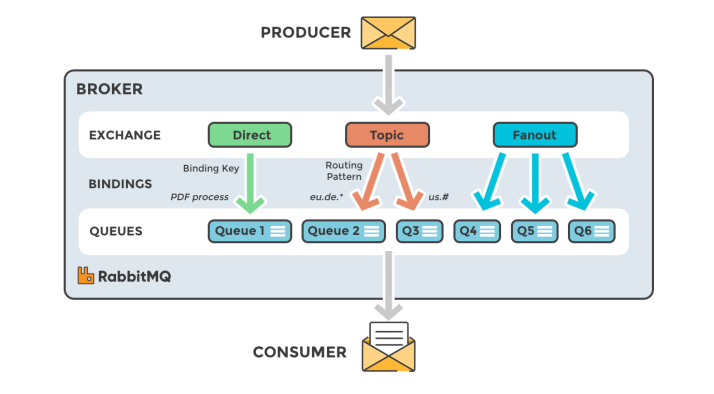


1. Для реалізації паттерну publish/subscribe було використано брокер повідомлень RabbitMQ

RabbitMQ – це брокер повідомлень з відкритим вихідним кодом. Він маршрутизує повідомлення за всіма базовими принципами протоколу AMQP описаним в специфікації. Відправник передає повідомлення брокеру а той доставляє його одержувачу. RabbitMQ реалізує і доповнює протокол AMQP.

Основна ідея моделі обміну повідомленнями в RabbitMQ полягає в тому, що producer (видавець) не надсилає повідомлення безпосередньо в чергу. Насправді і досить часто видавець навіть не знає, чи буде повідомлення взагалі доставлено в будь-яку чергу.

Замість цього видавець може надсилати повідомлення лише на обмін. З одного боку, обмін отримує повідомлення від видавців, а з іншого — відправляє їх у черги. Обмін повинен точно знати, що робити з отриманим повідомленням. Чи повинно воно бути додано в певну чергу? Чи повинно воно бути додано в кілька черг? Або повідомлення потрібно ігнорувати.



1. Було використано асинхронну чергу задач Celery

Celery – черга завдань, побудована на системі асинхронної передачі повідомлень. У програмуванні, Celery можна використовувати в якості сховища для відкладених завдань. Програма, яка передала завдання, може безперешкодно продовжувати працювати, а згодом вона звертається до celery для підтвердження закінчення процесу обчислення, щоб отримати необхідні дані. Незважаючи на те, що celery написана на Python, працювати з нею можна при використанні будь-якої мови програмування за допомогою webhooks.

1. Для розподіленого сховища даних було використано Redis

Redis (розшифровується як Remote Dictionary Server) - це швидке сховище даних типу "ключ-значення" в пам'яті з відкритим вихідним кодом для використання в якості бази даних, кешу, брокера повідомлень або черги. Проект виник, коли Сальваторе Санфіліппо, початковий Розробник Redis, намагався поліпшити масштабованість стартапу в Італії. Redis забезпечує час відгуку на рівні часток мілісекунди і дозволяє додаткам, що працюють в режимі реального часу, виконувати мільйони запитів в секунду.

1. Також було використано веб-сокети та технологію Sockets.io

Socket.IO - бібліотека JavaScript, заснована (написана поверх) на веб-сокетах та інших технологіях. Вона використовує веб-сокети, коли вони доступні, або такі технології, як Flash Socket, AJAX Long Polling, AJAX Multipart Stream, коли вебсокети недоступні.

Головними перевагами Socket.IO є наступне:

• на відміну від Веб-сокетів, Socket.IO дозволяє відправляти повідомлення всім підключеним клієнтам. Наприклад, ви пишете чат і хочете повідомляти всіх користувачів про підключення нового користувача. Ви легко можете це реалізувати за допомогою однієї операції. При використанні веб-сокетів, для 29 реалізації подібного завдання вам буде потрібно список підключених клієнтів і відправка повідомлень по одному;

• у веб-сокетах складно використовувати проксування і балансувальники навантаження. Socket.IO підтримує ці технології з коробки;

• як зазначалося раніше, Socket.IO підтримує поступову (витончену) деградацію;

• Socket.IO підтримує Автоматичне перепідключення при розриві з'єднання; • з Socket.IO легше працювати

1. Для спрощеного розгортання веб-застосунку було використано Docker

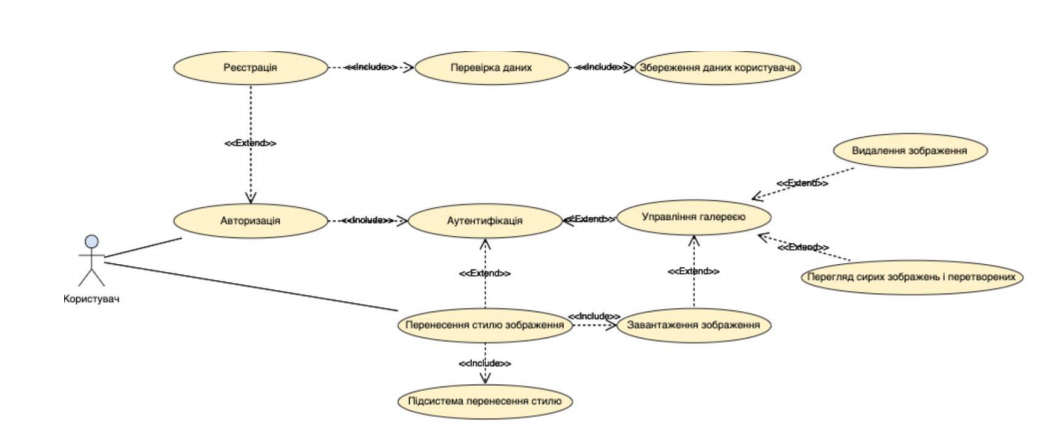
Docker - це інструмент, призначений для спрощення створення, розгортання та запуску програм за допомогою контейнерів. Контейнери дозволяють розробнику пакувати додаток з усіма необхідними йому частинами, такими як бібліотеки та інші залежності, і розгортати його як один пакет. Тим самим, завдяки контейнеру, розробник може бути впевнений, що програма працюватиме на будь-якій іншій машині Linux незалежно від будь-яких налаштованих параметрів, які можуть мати машини, які можуть відрізнятися від машини, що використовується для написання та тестування коду.

**Діаграма прецедентів** – схема, яка відображає функціонал проектованої системи. Складається з таких ключових елементів:

• суб’єкт – проектована система;

• актор – користувач, який взаємодіє із системою;

• прецедент – набір дій, які виконуються системою для отримання якогось результату.



Діаграма, зображена вище, складається з таких прецедентів:

• реєстрація;

• перевірка даних;

• збереження даних користувача;

• авторизація;

• автентифікація;

• управління галереєю;

• видалення зображення;

• завантаження зображення;

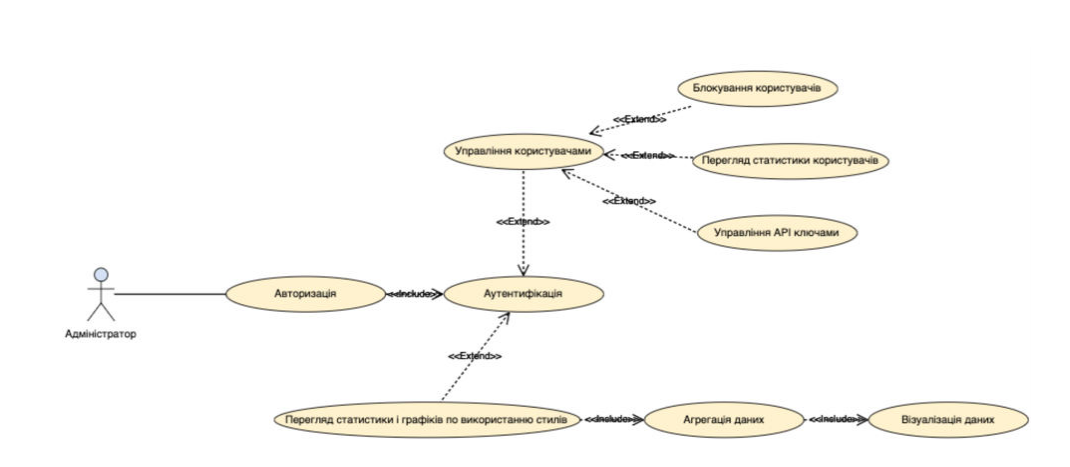
• перегляд сирих зображень і перетворених;

• перенесення стилю зображення;

• підсистема перенесення стилю

Як бачимо з діаграми, користувач має змогу авторизуватись або зареєструватись, якщо його профіль ще відсутній. Після цього користувач може керувати галереєю, переглядати її та змінювати стилі своїх зображень.

Діаграма прецедентів для ролі адміністратора:



Діаграма, зображена вище, складається з таких прецедентів:

• авторизація;

• автентифікація;

• управління користувачами;

• блокування користувачів;

• перегляд статистики користувачів;

• управління АРІ користувачами;

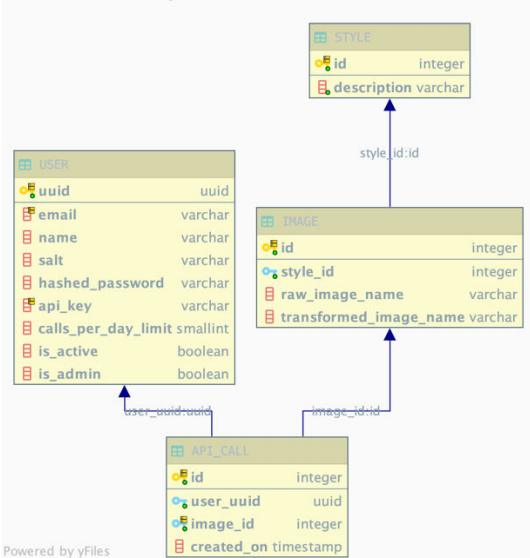
• перегляд статистики і графіків по використанню стилів;

• агрегація даних;

• візуалізація даних.

Як можна бачити , адміністратор має змогу після авторизації управляти користувачами та переглядати статистику щодо користувачів і стилів.

У реалізованій системі повинно бути дві бази даних (БД), оскільки для реалізації системи в мікросервісній архітектурі, кожен сервіс повинен мати свою БД. Тому сервіс для перенесення стилю зображення і сервіс для користувацького інтерфейсу мають різні власні БД.



Вище зображено схему БД сервісу для перенесення стилю зображення, яка скрадається з 4 таблиць:

• користувачі;

• стилі;

• зображення;

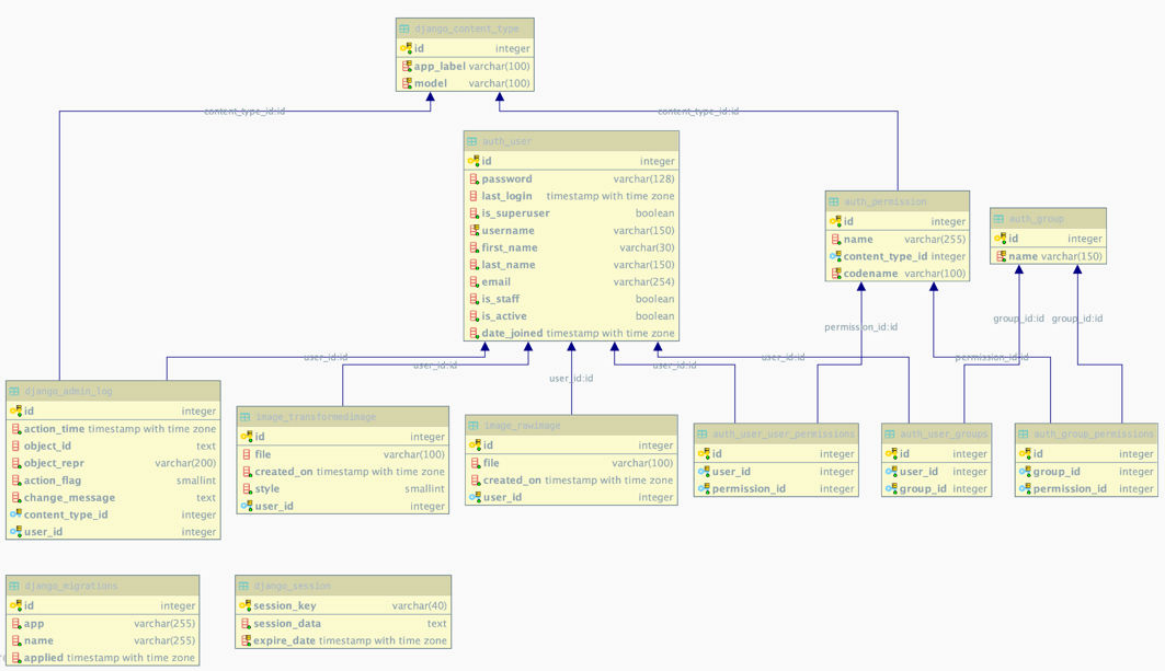
• запити на перетворення зображення.

Встановлено такі відношення:

• «Зображення» мають зв’язок із «Стилі» як «багато до одного», тобто багато зображень можуть мати один стиль;

• «Користувачі» мають зв’язок із «Запити в систему» як «один до багатьох», тобто один користувач може мати багато запитів;

• «Зображення» мають зв’язок із «Запити в систему» як «один до до одного», тобто під час одного запиту можна перетворити одне зображення.



Вище зображено схему БД сервісу для користувацького інтерфейсу, яка скрадається з 12 таблиць, з яких 9 службових або створених фреймворком і 3 власних:

• користувачі;

• перетворені зображення;

• початкові зображення.

Також встановлені такі відношення:

• «Перетворені зображення» та «Сирі зображення» мають зв’язок із «Користувачі» як «багато до одного», тобто один користувач може мати багато зображень.