Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

# РОЗРОБКА ОНЛАЙН-МАГАЗИНУ ДЛЯ ПРОДАЖУ АНІМЕ-ТОВАРІВ

**Текстова частина до курсової роботи**

**за спеціальністю «Комп'ютерні науки та інформаційні технології» 122**

Керівник курсової роботи
Ассистент кафедри мультимедійних технологій
Калітовський Б.В.
(прізвище та ініціали)
_____
(підпис)
«___» _____ 2022 року

Виконав студент 3 курсу
Кириченко Є.Б.
(прізвище та ініціали)
«___» _____ 2022 року

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав.кафедри

Жежерун О.П.

_____

(підпис) __

«___»_____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студента Кириченка Євгенія Борисовича факультету інформатики 3 курсу

ТЕМА РОЗРОБКА ОНЛАЙН-МАГАЗИНУ ДЛЯ ПРОДАЖУ АНІМЕ-ТОВАРІВ

Вихідні дані: Сервіс для перегляду аніме-товарів та оформлення замовлення

Індивідуальне завдання:

Вступ

1.Аналіз предметної області. Постановка завдання курсової роботи

2.Теоритичні відомості про розробку веб-застосунків

3.Реалізація програмного продукту

Висновки

Список використаної літератури

Додатки

Дата видачі «___» _____ 2022р.

Керівник _____Завдання отримав_____

(підпис) (підпис)

**Тема**: РОЗРОБКА ОНЛАЙН-МАГАЗИНУ ДЛЯ ПРОДАЖУ АНІМЕ-ТОВАРІВ

**Календарний план виконання роботи**:

| Номер | Назва етапу | Термін виоканання | Примітка |
|---|---|---|---|
| 1. | Отримання теми курсової | 12.12.2021 | |
| 2. | Огляд літератури за темою роботи | 12.01.2022 | |
| 3. | Аналіз актуальності питання | 19.01.2022 | |
| 4. | Огляд аналогів | 27.01.2022 | |

| | | | |
|---|---|---|---|
| 5. | Постановка завдання | 01.02.2022 | |
| 6. | Аналіз технологій розробки | 01.03.2022 | |
| 7. | Аналіз технічого завдання | 11.03.2022 | |
| 8. | Вибір технологій розробки | 18.03.2022 | |
| 9. | Створення ЄР-моделі | 24.03.2022 | |
| 10. | Створення схем БД | 31.03.2022 | |
| 11. | Розробка серверної частини | 31.04.2022 | |
| 12. | Розробка клієнтської частини | 12.05.2022 | |
| 13. | Поєдання клієнта та сервера | 22.05.2022 | |
| 14. | Перегляд змісту роботи керівником | 30.06.2022 | |
| 15. | Внесення змін та правок відповідно до зауважень керівника | 04.06.2022 | |

Студент Кириченко Є.Б.

Керівник Калітовський Б.В.

«___»_____

**ЗМІСТ**

# АНОТАЦІЯ

В даній роботі досліджується ніша онлайн магазинів аніме-товарів. Пояснюється актуальність питання. Розглянуто аналоги, знайдено їх плючи та мінуси. Поставлено за задачу розробити власний додаток, онлайн магазин з продажу товарів, пов'язаних з аніме. В теоретичній частина досліджені основні технлогії, принципи  та підходи розробки веб-застосувань. В практичній описується реалізація додатку – від з'ясування ТЗ і до готового додатку.

# ВСТУП

В природі людини закладений один дуже цікавий принцип – досягти бажаного як можна меншими витратами енергії. Зараз, щоб знайти майже будь-яку інформацію достатньо здійснити пошук в інтернеті. Товари не виняток. Можна не просто швидко знайти характеристики будь-якого товару, але й придбати його, не виходячи з дому. Варто лише натиснути декілька кнопок - і готово. Це стало можливим завдяки появі онлайн магазинів. Кожного року з'являється величезна кількість таких веб-додатків, метою яких є задовільнити потреби клієнта. А так як кожен з нас клієнт з великою кількістю (потреб) створення нових онлайн-магазинів має сенс.

Метою є створення сервісу для придбання аніме-товарів. У людей з'явиться більший вибір подібних сервісів, щоб швидко знаходити та замовляти потрібні їм товари

Завданням є розробка зручного веб-додатку, що має приємний та зрозумілий інтерфейс для роботи з користувачем

Об'єктом дослідження є магазини аніме-товарів, які вже існують на ринку Програмне забезпечення.

Бекенд буде розроблений на Node.js. Для фронтенду буде використаний React.js . Для рівня даних буде використаний MongoDB. Написання коду здійснюватиметься у середовищі для розробки WebStorm.

**Структура курсової роботи**

Робота складається з трьох розділів. У першому відбувається аналіз предметної області, а саме:

-Визначається актуальність теми

-Проводиться аналіз аналогів, що вже є на ринку

-Ставиться задача.

У другому розділі надані теоретичні відомості щодо технологій розробки веб-застосунків.

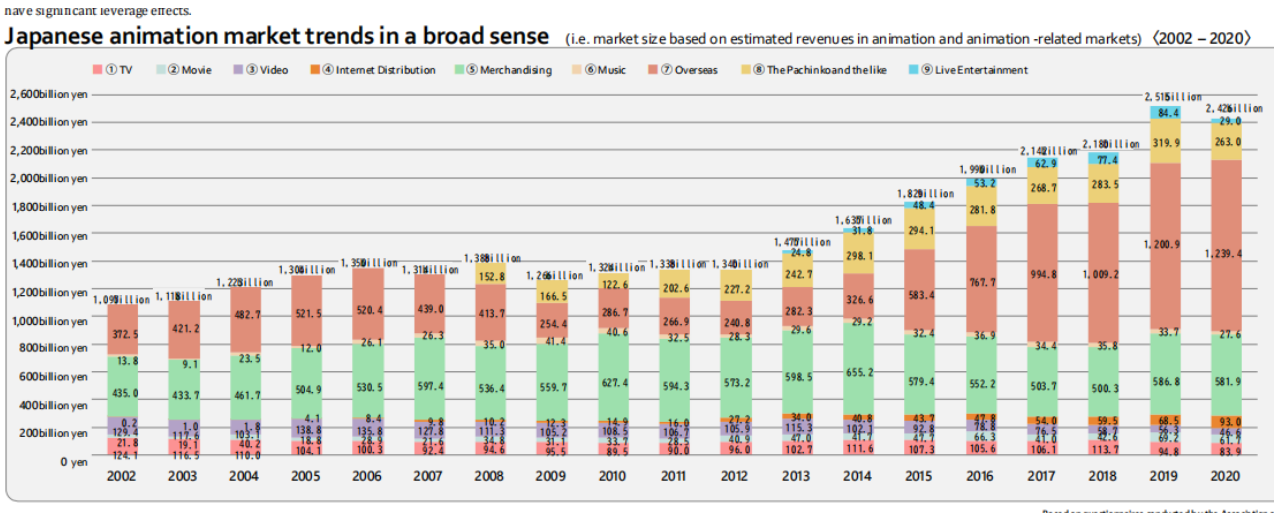
У третьому розділі розповідається з-за допомогою яких технологій та яким чином реалізовна програма. В цьому розділі чітко формулюється технічне завдання та описані кроки його виконання, включаючи структуру класів, що є у програмі, структуру бд. В кінці розділу відбувається тестування готової програми.

# РОЗДІЛ 1: ВИВЧЕННЯ НІШИ

## 1.1 Аналіз актуальності питання

В останні десятиліття аніме набуло шаленої популярності, яка продовжує рости по всьому світу, і за прогнозами не збирається загасати. Фанати та любителі даного жанру все частіше замовляють та задумуються над тим аби замовити певну річ пов'язану з улюбленим аніме.

Популярність цієї ніші має такі темпи росту, що подібний магазин має всі шанси на те, щоб користуватись великим попитом. Саме тому я обрав дану тематику для свого онлайн-магазину.



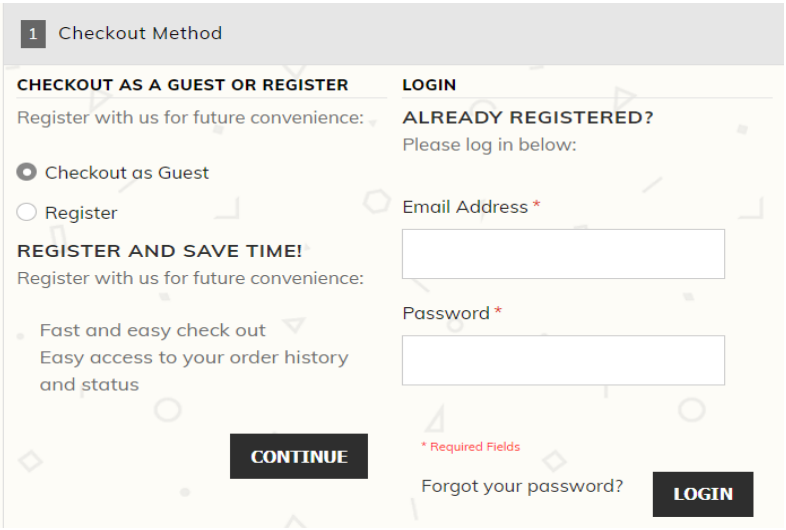

**The Association of Japanese Animations** chart [1]

## 1.2 Огляд існуючих онлайн-магазинів

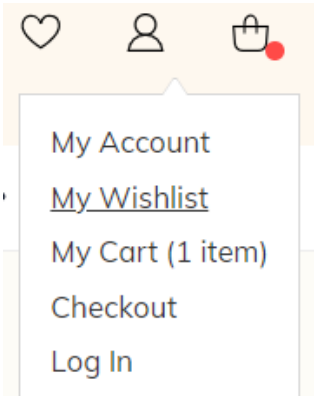
Розглянемо сайти, які пропонують нам асортимент товарів, пов'язаних з аніме.Почнемо зі світових. [2]

1) JLIST [3]

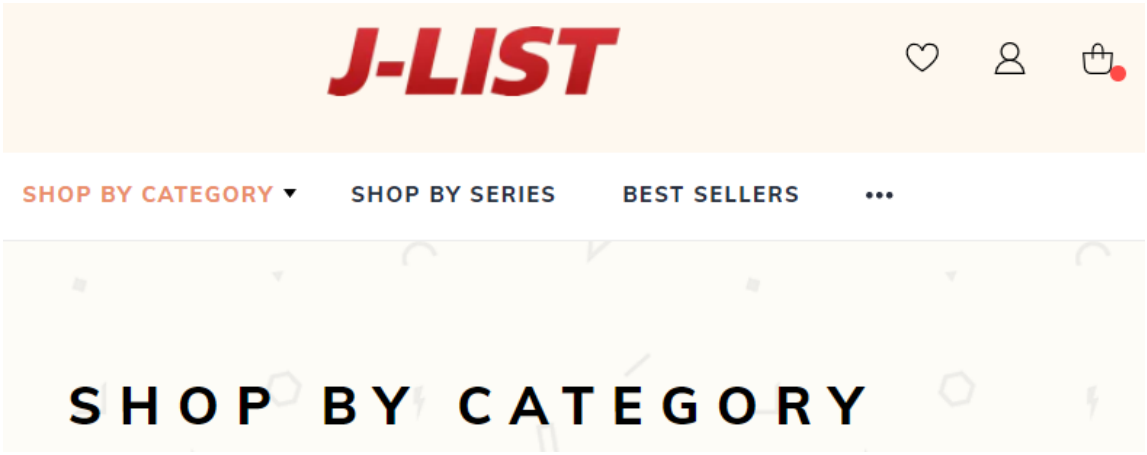
Перше, що варто відзначити - у магазина приємний та зрозумілий інтерфейс. Є можливість зареєструватись, проте покупку можна здійснити й без реєстрації, що є величезним плюсом.



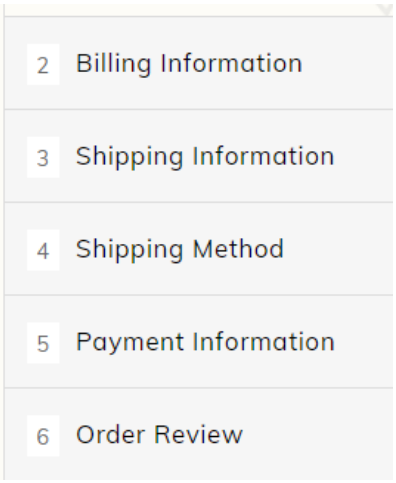

Товар можна додати в корзину або до списку бажань



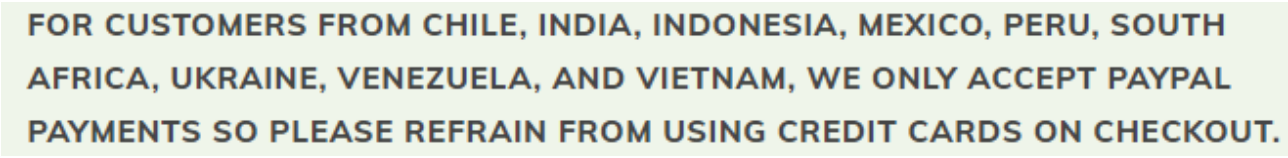

Є можливість сортування товарів за категоріями, ціною, та певним аніме.




Також, важливо, що можна дивитись деталі замовлення та його стан




Варто відзначити, що є певні обмеження для покупок

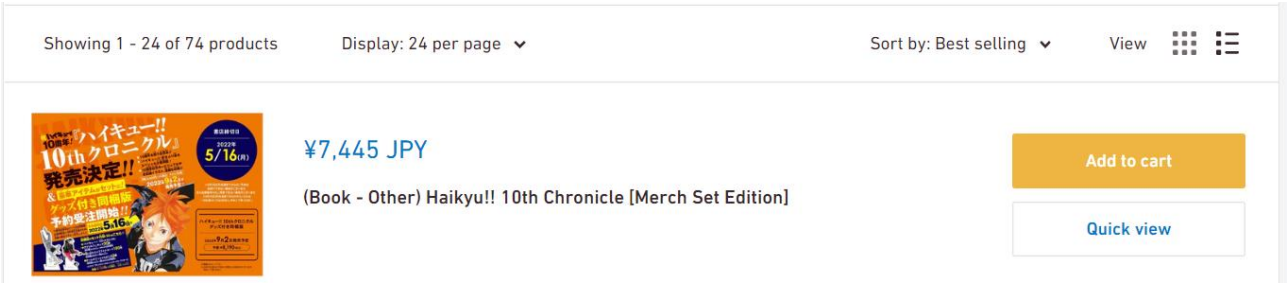



Так користувачі з України мають оплатити покупку обов'язково з-за допомогою PayPal.

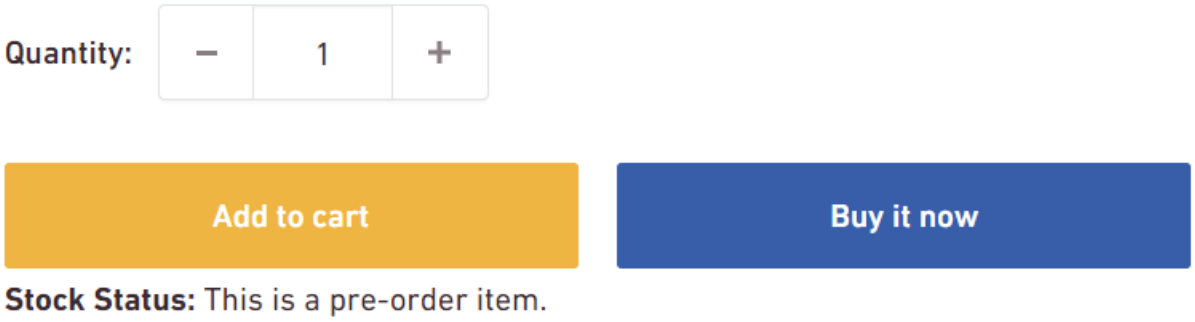
До недоліків варто віднести, що сайт повільно вантажиться. Також, на жаль, щоб купити товар треба спершу додати його в кошик. Цього не можна зробити одразу зі сторінки товару, а це могло б зекономити час. Також, на сайті не можна змінити мову, немає коментарів.

2) Animate[4]

Що одразу кидається в очі, так це валюта – японська єна




Є один величезний плюс: можна купити товар з його сторінки.

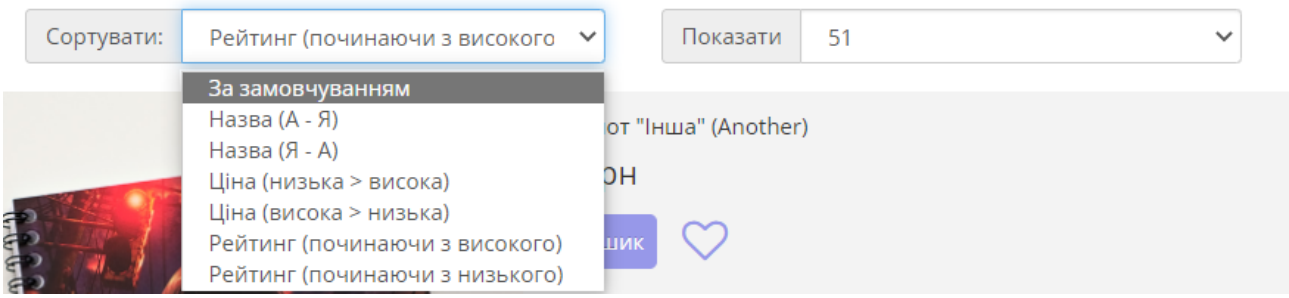



Але тут же виникає мінус: покупки можуть здійснювати лише авторизовані користувачі. Крім того, на сайті я не бачу коментарів.

Ці сайти я знайшов в одному рейтингу. Тепер поглянемо, які сайти видає нам гугл за запитом українською мовою.

1) MURASAKI[5]

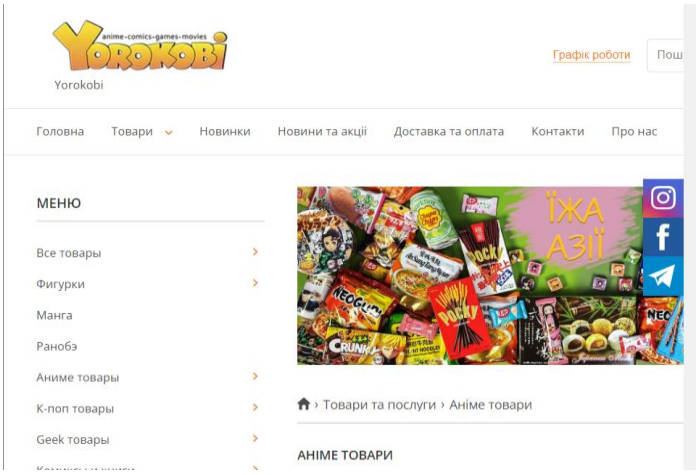
Є навігація по сайту. Є сортування.



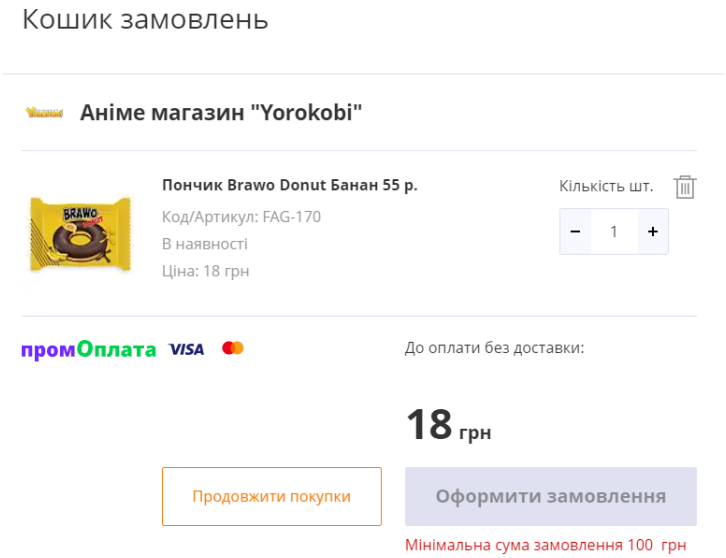
Проте не можна поставити максимальну припустиму ціну. Придбати товар зі сторінки не можна. Тут можна читати коментарі, а за умови наявності облікового запису і створювати.В якості служби підтримки є один номер телефону. Є можливість написати текстове звернення зв'завись через телеграм чи іншу соціальну мережу, проте я не думаю, що всім клієнтам сподобається така альтернатива.

2) YOROKOBI [6]

Одразу бачимо, що у сайту проблеми з перекладом, адже на українську перекладаються лише назви розділів зверху.




Обмеження на мінімальне замолення у розмірі 100 гривень.




І це один з сайтів, який випаадає майже одразу при пошуку аніме-товарів українською мовою.

З цього випливає, що на нашому ринку з даною продукцією можна і треба конкурувати. Є недоліки, які можна врахувати при створенні власного веб-сервісу. Треба подумати над локалізацією, можливістю покупки зі сторінки товару, способами оплати, певними корисними фільтрами та можливістю придбання для неавторизованих користувачів.

## 1.3 Постановка завдання

1.Вибір зручних та оптимальних технологій розробки
2.Аналіз логіки взаємодії клієнта та сервера, моделювання ЄР-моделі.
3.Вимоги до програми
- реєстрація та автентифікація користувача
- пошук товарів (за категоріями)
- перегляд сторінки товару
- користувачі матимуть можливість переглядати інформацію про товар
- додавання товару до корзини
- перегляд інформації про стан замовлення
- додавання/видалення/редагування товарів адміністраторами
- у програмі буде фільтр категорій для легшого знаходження потрібного товару

# ВИСНОВКИ РОЗДІЛ 1

В даному розділі було з’ясовано чому тема є актуальною, був виявлений закономірний ріст доходів сфери аніме. Також, передбачено, що популярність сфери ростиме з часом.

Також, було досліджено нішу магазинів з аніме-товарами, були виявлені плюси та недоліки магазинів. До плюсів можна віднести можливість покупки зі сторінки товару, покупки без реєстрації акаунту, наявність служби підтримки, фільтри сортування. До мінусів – відсутність названих плюсів. Виявлено

прогалину на ринку українських аніме-шопів. Постановлено, що розробка додатку є перспективною та корисною.

Було поставлене завдання.

# РОЗДІЛ 2: Аналіз теорії для розробки застосунків

## 2.1 Види веб-сервісів

Створюючи веб-додаток, ми маємо розуміти, який саме вид веб-сайту ми хочемо реалізувати.

Умовно їх можна розбити на декілька типів: [17]

1)eCommerce. В дану категорію можна віднести будь-який сайт, де ми можемо здійснювати покупки.

2)Business. Дані сайти є презентацією певного бізнесу. На сайті є контактна інформація та інформація про компанію та послуги, які вона представляє.

3) Entertainment. Дані сайти заробляють не шляхом продажу, а розміщенням реклами.

4)Portfolio. Власне сайт, де можна подивитись роботи людини, описані її вміння. Ціллю є пошук нових клієнтів.

5)Media.

6) Non-profit. Сайт для донатів.

7)Educational. Надають навчальні курси. Підписка може бути платною, можуть продавати курси, також доходи можуть приходити з реклами.

8)Personal. Людина створює сайт, щоб писати там свої думки. Може бути не пов'язаним з доходами.

9)Web-portal. Інформація з різних сайтів, порталів, джерел зібрана в одному місці.

10) Wiki / forum. Сайт спільного користування, де будь-хто може додавати інформацію або вносити поправки.

Очевидно, що застосунок «ОНЛАЙН-МАГАЗИНУ ДЛЯ ПРОДАЖУ АНІМЕ-ТОВАРІВ» відноситься до категорії eCommerce та має риси Business,

адже зазвичай подібні додатки місять логотип, контактну інформацію, та інформацію про розробників.

## 2.2 Основні технології розробки веб-сервісів

### 2.2.1 Паттерни

Є різні паттерни для розробки веб-сервісів:

Model View Controller

Page Controller

Front Controller

Intercepting Filter

Page Cache

Observer

Для розробки веб-сервісу я використовую шаблон MVC. (Я вважаю, що для поставленого мною завдання немає сенсу використовувати інші, складніші шаблони *)

Не можна не відзначити, що є дуже схожі з MVC паттерни [15] – MVP та MVMM.

MVP – ModelViewPresenter.

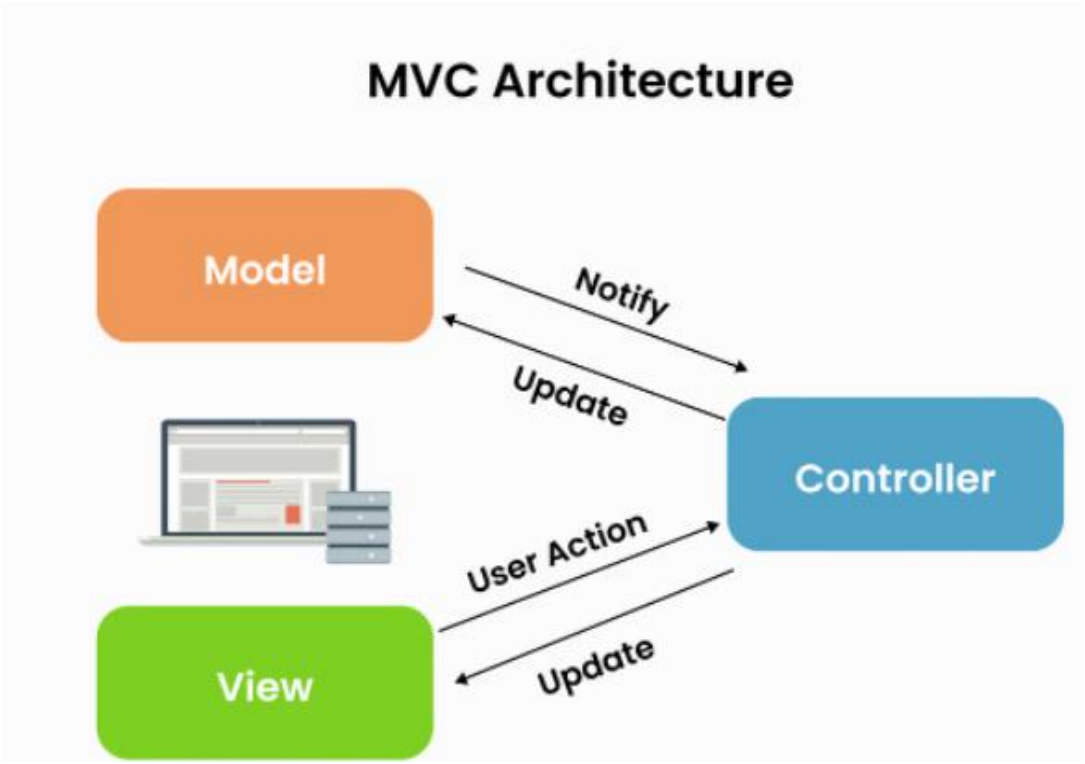
MVVM – Model View ViewModel.

MVC середній з них по перфомансу, проте легкий в тестуванні окремих компонент, легкий у розширенні, тобто додавати нові функції простіше, ніж при використанні інших шаблонів, так як View не має жодних посилань на контролер, тоді як в MVVM View посилається на View-Model, а в MVP View зсилається на Presenter. Також над програмою з MVC легше працювати великій кількості людей, що може знадобитись у майбутньому.

Даний шаблон дозволяє нам[7] відділити представлення даних від бізнес логіки. Таким чином ми можемо змінювати інтерфейс користувача, не зачіпаючи логіку програми.

Модель відповідає за зв'язок з базою даних. View за візуальне представлення даних – те, що бачить користувач. Controller – зв’язує модель та в’ю і представляє логіку нашої програми, може викликати методи моделі та маніпулювати з БД.

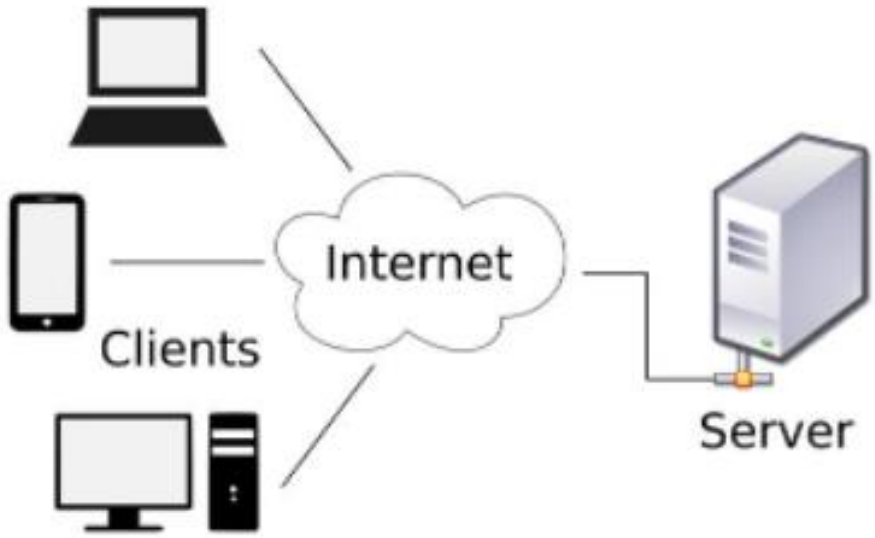
В контролері ми, наприклад, забираємо дані з бази даних і передаємо у view, щоб їх побачив користувач.




## 2.2.2 Peer-to-peer та Client server

Існують різні архітектури для написання веб-додатків. Peer-to-peer та **Client server.**Вони мають відмінності. [30] Клієнт-серверна архітектура має визначених клієнтів та серверів, коли в Peer-to-peer кожен клієнт може бути сервером. В Клієнт-серверній архітектурі головна мета – поширювати інформацію, а в Peer-to-peer підтримувати звя’язок між клієнтами-серверами. В клієнт-серверній архітектурі клієнт може лише виконувати запити до даних, тоді як в Peer-to-peer клієнти-сервери можуть надавати і відповіді на запити.

З вищесказаного випливає, що в моєму випадку буде використана клієнт-серверна архітектура.Так, у додатку буде модель:




При написанні веб-застосунка я використаю клієнт-серверну архітектуру.

## 2.2.3 Взаємодія клієнта з сервером

Клієнт спілкується з сервером з-за допомогою http реквестів. Існують наступні HTTP запити:POST PUT GET DELETE PATCH. Сервер має обробляти можливі запити. Контролер буде викликати методи CRUD відповідні до запитів. CRUD – create read update delete.Так операції crud пов'язані з відповідними операціями з бд:

[31]

| NAME | DESCRIPTION | SQL EQUIVALENT |
|---|---|---|
| Create | Adds one or more new entries | Insert |
| Read | Retrieves entries that match certain criteria (if there are any) | Select |
| Update | Changes specific fields in existing entries | Update |
| Delete | Entirely removes one or more existing entries | Delete |

Ось так з http операціями:

| CRUD | HTTP |
|---|---|
| CREATE | POST/PUT |
| READ | GET |
| UPDATE | PUT/POST/PATCH |
| DELETE | DELETE |

Забезпечити роботу з http запитами на сервері можна за допомогою REST API або SOAP API. [10][11]

REST - REpresentational State Transfer -

Архітектурний стиль

SOAP - Simple Object Access Protocol – формат протоколу поверх http для розробки SOAP API.

REST орієнтований саме на використання HTTP в якості транспортного протокола, в той час

SOAP може використовуватись з будь-яким типом транспортного протокола.

SOAP використовує SOAP XML, чіткий формат, що є XML файлом там включає в себе елементи

Envelope, Header, Body та Fault, для запитів та відповідей.

REST же не має фіксованого формату для запитв та відповідей. Ми можемо використовувати як xml так і json, чи будь-який інший формат.

Зазвичай використовують REST з Json. Такі сервіси простіше та швидше розробляти. Також, за рахунок використання Json замість зменшує витрати на передачу інформації також Json легше читати та парсити. Отже, варіант з REST та Json ми використаємо.

## 2.2.4 Компоненти веб-додатків

Веб-додатки можуть мати різну кількість різних компонентів:

1) Одна БД і один сервер. Це напростіший варіант, проте є проблеми з надійністю, адже якщо сервер вийде з ладу або буде збій з бд, веб-додаток одразу вийде з ладу

2) Декілька серверів, одна база даних. В цьому випадку при виході з ладу одного сервера, додаток продовжить працювати. Проте, якщо буде збій в бд, додаток все ще зламається

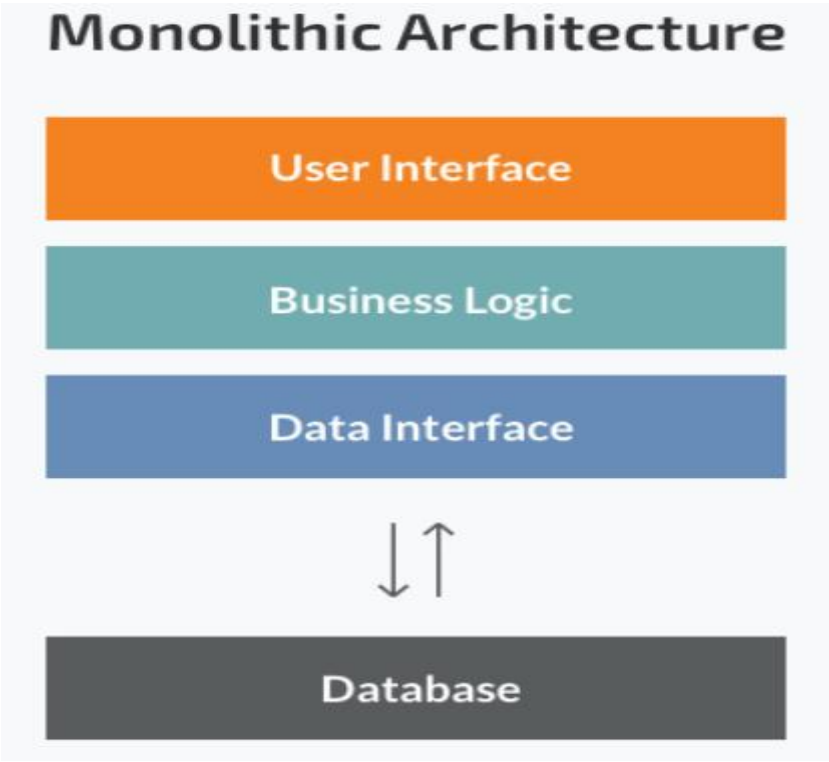
3)Декілька серверів і декілька БД – найнадійніший варіант. У всіх БД можуть зберігатись одні й ті ж дані або дані можуть розподілятися між декількома бд

Перший варіант підходить для навчальних цілей або для реалізації тестового проекту. Його ми й оберемо, адже він швидший у реалізації. У майбутньому цю архітектуру можна змінити .

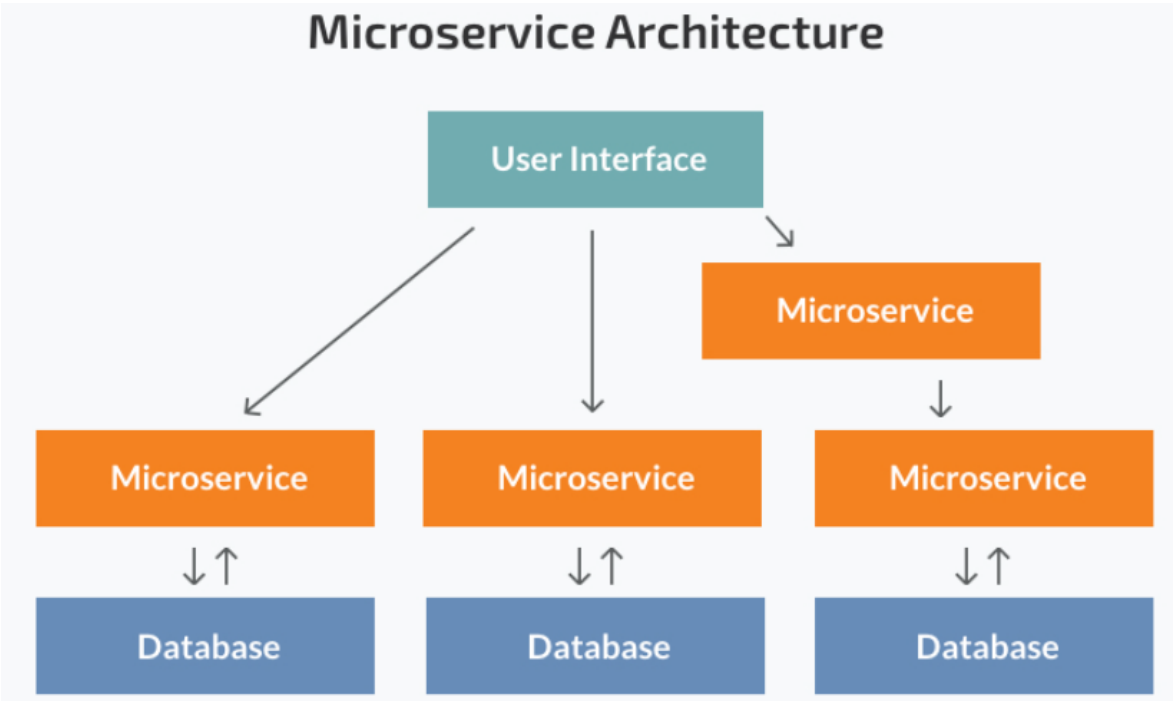
## 2.2.5 Моноліт та мікросервіси

При розробці сервісу важливо знати, що є 2 основні архітектурні системи.[13]

1) Моноліт – підхід, який визначає програму як одну неподільну одиницю. Це означає, що всі компоненти програмного забезпечення зібрані в одному місц, тож такі застосунки мають недоліки, що стосуються модульності.




2) Мікросервіси – декілька незалежних одиниць, кожна з яких відповідає за певний сервіс, має свою відділену логіку, свою базу даних.



Тож, коли який підхід обрати? Моноліт варто обрати коли:

1) команда розробників досить невелика.
2) застосунок досить простий і просто немає сенсу розбивати його на декілька мікросервісів.
3) коли треба швидко розробити і запустити додаток – відповідно будуть менші витрати на розробку.

Не варто обирати мікросервіси завжди лише через те, що це популярний підхід.На початку створення додатку у нас може бути небагато функцій, розбивати які на мікросервіси буде просто нелогічним рішенням. Варто зазначити, що деякі великі компанії, як Нетфлікс чи Амазон починали з монолітної архітектури, а вже згодом перейшли на мікросервіси.

Отже, з приведених вище аргументів випливає, що мій застосунок буде монолітним.

# ВИСНОВКИ: РОЗДІЛ 2

В другому розділі розібрана теорія, що може стати у нагоді при розробці власного застосунку. Було обрано паттерн MVC, названо його відмінності від інших схожих, виявлено його сильні сторони.

Порівняно дві архітекутри розробки – Client server та peer-to-peer, пояснена їх принципова різниця. З'ясовано, коли яка архітекутра застосовується та вирішено розробляти додаток згідно з клієнт-серверною архітектурою.

Описана взаємодія клієнта з сервером, порівняно два API для обробки http запитів на сервері: REST і SOAP, виявлені сильні сторони REST API , аргументовано його застосування.

Розібрано основні варіанти компонування елементів веб-додатків: баз даних та серверів.

Порівняно архітектурні підходи розробки застосунків моноліт та мікросервіси. Виявлені сильні сторони моноліта, обрання цього підходу як більш доречного.

# РОЗДІЛ 3 :РЕАЛІЗАЦІЯ

## 3.1 Аналіз ТЗ

Технічне завдання полягає в створенні онлайн магазину аніме-товарів. Для цього нам потрібно реалізувати декілька груп користувачів та їх поведінку у системі. Серед користувачів у нас покупці та адміністратори. (Покупці можуть бути авторизовані або ні, від цього можуть залежати їх  можливість у сервісі).

Користувачу будуть надані наступні можливості:

1) Авторизація/реєстрація. Авторизований користувач може зберігати товари в кошик.

2) Перегляд каталогу товарів, додавання їх до кошика

3) Придбання товару. Можна зібрати замовлення, додаючи в кошик, або придбати певний продукт одразу зі сторінки замовлення. ????

4) Редагування особистої інформації.

Адміністратору будуть надані наступні можливості:

1) Додавати/Видаляти/Оновлювати категорії товарів.
2) Додавати/Видаляти/Оновлювати товари..
3) Видаляти/Оновлювати замовлення, його статус.????

Вимоги до даних

- Користувач: може зареєструвати акаунт, має пошту, id, пароль, та може бути або не бути адміністратором. Клієнт може форомити замовлення.
- Категорія: категоріями керує адімінстратор. Категорія має назву та унцкальний ідентифікатор
- Продукт: адміністратор може керувати ними, користувач – додавати. продукт має назву, ціну, рейтинг, назву файлу з зображенням, та належить до однієї категорії
- Замовлення: скаладається з масиву елементів, які мають id товару, так кількість екземплярів для придбання. Замовлення оформлює клієнт.

- Коментарі: складаються з тексту та оцінки . Коментарі залишають користувачі для певного товару.

## 3.2 Вибір засобів розробки

Для розробки бекенду веб-застосунка я обрав Node.js . Node.js [18][19] – це середовище виконання Javascript, засноване на Javascript рушій Chrome V8. V8 – рушій, написаний на C++, що здатен прискорити виконання Javascript коду Node.js має багато плюсів, а саме [27]:

-open source – кожен може додавати чи змінювати код

-кросплатформеність – працює на Windows, Linux, Unix, Mac OS X ітд

-асинхронне програмування лежить в основі Node.js . Наприклад в ASP чи PHP кщо треба відкрити файл на сервері, та повернути клієнту, до того як файл повернеться на клієнт, інші запити не обробляються. В той же час в Node.js відкриття та читання файлу будуть в іншому потоці. За рахунок асинхронності сервер на ноді не має чекати завершення однієї операції повернення даних, щоб почати робити іншу.

-швидкість роботи коду (за рахунок Chrome V8 рушія).

-легко масштабується за рахунок того, що в Node js використовується спеціальна однопоточна модель, яка не блокує потоки вводу-виводу.

Є й інші популярні засоби розробки[21], серед яких Python Django, проте Node js пропонує більше бібліотек, а розробку додатку робить більш гнучкою.

Для написання серверного коду на Node js я використаю фреймворк Express.js. Він спрощує розробку, надає інстурменти для реалізації роутингу, мідлвейрів, авторизації. З обраними засобами розробки сервер досить просто розробляти, і додаток працюватиме швидко.

Для розробки клієнтської частини використаний React.js – Javascript бібліотека , що дозволяє сторювати інтерактивний UI. React здатен оновлювати лише ті

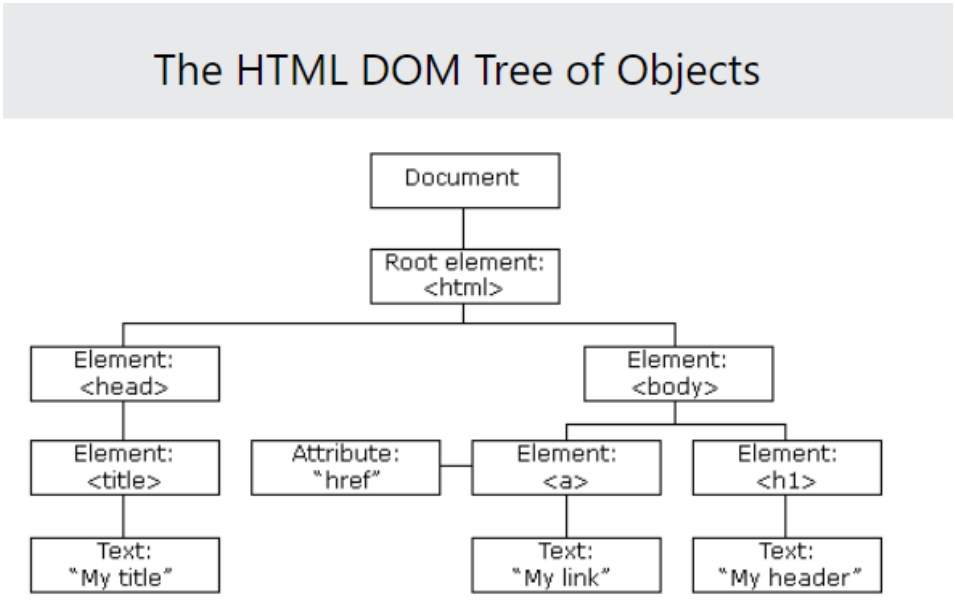
компоненти, які цього потребують. Є певні пункти, які пояснюють чому React є хорошим рішенням для побудови UI:

- JSX – Javascript XML. За допомогою JSX можна писати html код в React. JSX працює швидіе за javascript + html


[22]

Як JSX спрощує написання коду

- Virtual DOM. DOM – [23] Document Object Model – дерево html елементів, яке створюється при завантаженні браузером веб-сторінки.




Робота з DOM відбувається досить повільно, тому в React є Virtual DOM – абстракція HTML DOM. React оптимізований через те, що перерендерюються тільки ті елементи нашого дерева, які зазнали змін. Щоб дізнатись, які елементи

зазнали змін,[24][25] при зміні стану компоненту створюється ще один віртуальний DOM.Далі вони порівнюються – порівнювати 2 VDOM значно швидше, адже операції на VDOM виконуються значно швидше, ніж на DOM. Ми знаходимо різницю між двома VDOM, і тільки тоді оновлюємо наш DOM. VDOM працює швидше, адже містить в собі ReactElement, а не ReactComponent. Це дозволяє швидше порівнювати елементи, так я не потрібно рендерити сторінку для порівняння дерев

- Використання компонентів – блоки в Реакті, що дозволяють робити композиції елементів.

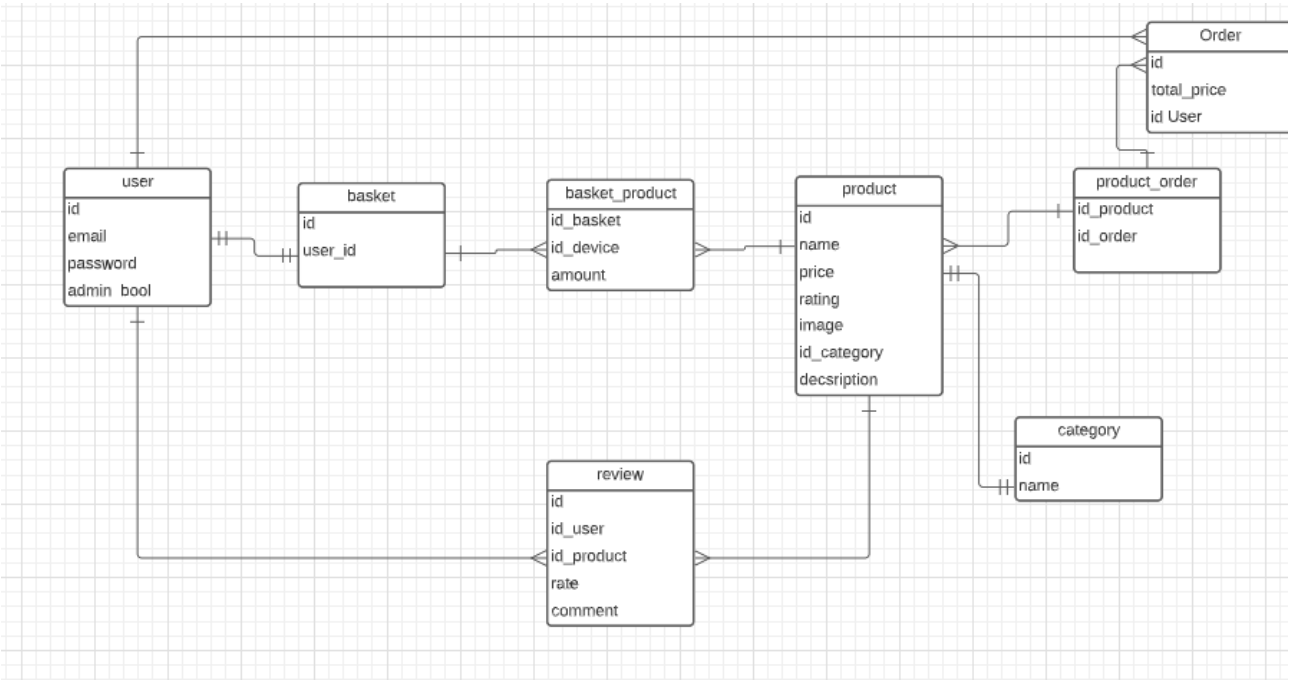
Для рівня даних я обрав MongoDB. Mongo DB - документоорієнтована база даних, це означає, що запис – не рядок таблиці, а документ у форматі BSON (бінарний JSON).

Mongo DB має ряд переваг: [26]

1) Можна працювати з документами у форматі JSON, це зрозумілий формат, який легко читати. В JSON можна вкладати інші JSON. Можна додавати поля, або упускати взагалі. Також у багатьох сучасних мовах програмування є ObjectMapper’и, що дозволяють читати та записувати JSON об'єкти.
2) Масштабованість.

# 3.3 Опис реалізації

Для початку побудуємо ER модель нашого додатку




Отже, у нас є наступні моделі:

User – наш користувач. Він має поля id email password та Boolean значення, яке визначатиме чи є користувач адміном. По дефолту воно буде false.

Basket – кошик, куди корисьтувач зможе додавати товари.

У одного користувача може бути тільки один кошик. Кощик містить айді користувача та власний id.

Product – товар. Містить айді, назву, ціу, рейтинг, категорію та опис.

В кошику може бути багато продуктів, а один продукт може бути в різних кошиках. Тож маємо зв'язуючу сутність basket-product.

Order – замовлення. Замовлення має айді суму та айді користувача. Так як продукт може відноситись до багатьох замовлень, і замовлення може містити багато продуктів, маємо допоміжну реляцію product-order, яка зберігає id замовлення та id продукту.

Category – має назву категорії та id. Співвідноситься до продукту 1 до 1

Review – містить id, id користувача, який залишив review, id продукту, до якого створений review, оцінку та текстовий коментар. Користувач може створювати багато review, а review обов'язково належить одному користувачу. У одно продукту може бути багато review, залишених різними користувачами.

## 3.3.1 Server

Застосунок побудований за принципами клієнт-серверної взаємодії. Сервер має обробляти можливі запити які надходять мережею від клієнта. Також сервер має підтримувати зв'язок з базою даних. Сервер може діставати потрібні дані, після чого відправляти їх на клієнт. За допомогою ER-моделі застосунку побудовані відповідні моделі в базі даних. Використовується база даних MongoDb. Роботу з нею можна максимально спростити, використовуючи Mongoose – це ODM(Object database Modelling), що дозволяє створити схему реляції прямо з node js та потім звертатись до цієї схеми для виконання запиті до БД. Також є стандартний функціонал для здійснення деяких операцій CRUD.

Так створюється схема

```
const Product = new mongoose.Schema ( definition: {
    title: {type:String , required: true, unique: true },
    price: {type:Number , required: true},
    review:[{type:mongoose.Schema.Types.ObjectId, ref: 'Review'}],
    img: {type:String , required: true},
    id_category: {type: mongoose.Schema.Types.ObjectId, ref:'Category', required: true },
    description: {type:String , required: true},
    size: {type:String, default: 'def_size'},
    color: {type:String , default: 'def_color'},

}, options: {timestamps: true})
module.exports = mongoose.model( name: 'Product', Product)
```

Так можна робити запити. Тут метод пошуку за id та оновлення – один зі стандартних методів  mongoose.

```
const updatedProduct = await Product.findByIdAndUpdate(product._id,  product, options: {new:true} );
```

А ось так описуються зв'язки між схемами. Зображена схема зв'язків Order, яка має посилання на реляцію продукти та користувача.

```
userId: { type: mongoose.Schema.Types.ObjectId, ref:'User', required: true },
products: [{type:mongoose.Schema.Types.ObjectId, ref: 'Product'}],
```

Додаток виконується згідно з MVC-паттерном. Відповідно є контроллери, які можуть керувати даними, які містяться в БД. Контролери реалізовують потрібні CRUD операції – CREATE, READ, UPDATE, DELETE.

При взаємодії клієнта та сервера, перший відправляє запит, який відповідає певному маршруту. Запитам, які ідуть з клієнта, роутер ставить у відповідність методи контролера. Роутер керує тим, які дії з рівнем даних виконає Котнтролер залежно від маршруту. Сервер використовує роутери. Вказується маршрут, за яким працюватиме роутер, та сам роутер.

```
server.use('/api/product',productRouter)
```

```
router.post( path: '/',productController.create)
router.get( path: '/', productController.getAll)
router.get( path: '/:id', productController.getByID )
router.put( path: '/', productController.update)
router.delete( path: '/:id', productController.delete)
```

Тобто при POST запиті на api/product викличеться метод CREATE productController`a, який створює новий екземпляр продукту, звертаючись до бд.

```
const prod = await Product.create({title, price, img:fileName, id_category, description })
```

## 3.3.1.1 Сервер. Авторизація.

Для авторизації є відповідний роутер та контроллер

```
router.post( path: '/signin', authController.signin)
router.post( path: '/signup', authController.signup)
```

Варто відзначити, що авторизація у веб застосунках може відбуватись за різними сценаріями[28]. Можуть використовуватись token, session або cookie. Також, token можна зберігати в cookie.

1)cookie based автентифікація

Клієнт авторизується. Сервер перевріяє дані, та якщо вони валідні – сервер створює сесію, яку записує в базу даних і відправить set-cookie хедер, який містить sessionId на клієнт. Далі всі запити від клієнта включають в собі кукі,

sessionId якої порівнюється з тим, яке знаходиться в базі на сервері. Це відбувається щоб перевірити валідність sessionId клієнта.

2)Авторизація з токенами

Клієнт авторизується, якщо все нормально – на клієнт відправляється токен. Токен зберігається в локальному сховищі, сховищі кукі, або сховищі сесії. Далі токен додається в authorization header запитів клієнта. При надсиланні запиту з клієнта токен дістається та перевіряється на валідність.

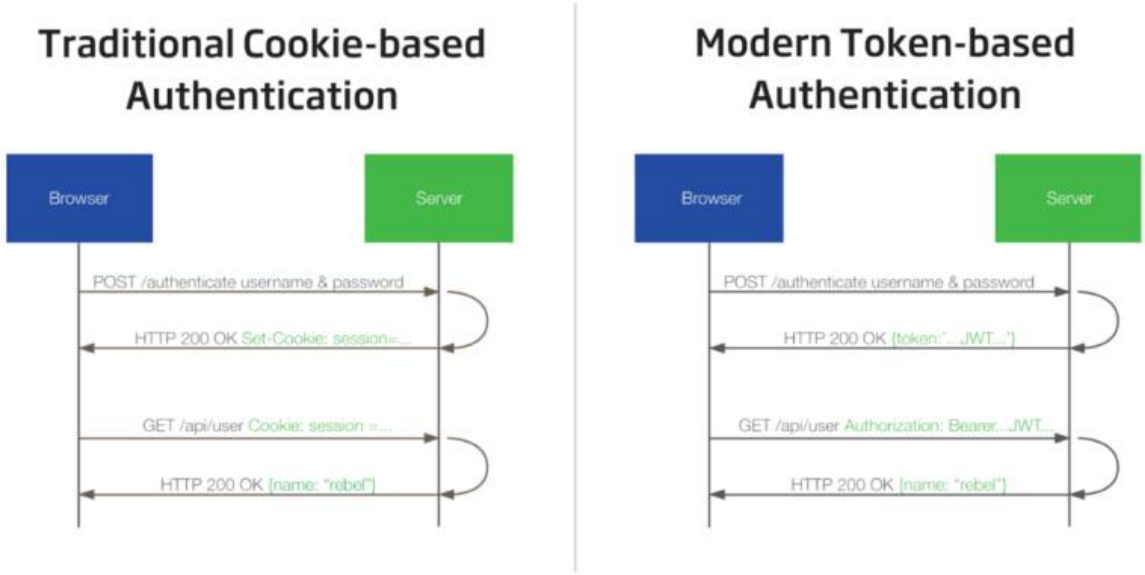
Авторизація за допомогою токена має певні плюси:

1)Сервер не має зберігати токени, як це відбувається з сесіями

2)Різні служби та домени можуть доступатися до API сервісів з таким типом авторизації. Одне API може обслуговувати і Web-платформи і Android, IOS-додатки.

3)Швидкість. Не потрібен пошук по базі, в якій зберігаються сесії. Не потрібен пошук користувача в базі юзерів, щоб зрозуміти чи користувач адмін, адже роль можна зберігати в токені.

Згідно з аргументами приведеними вище, для авторизації використовуються токени.




Тут токен генерується та повертається на клієнт.

```
const token = genJWT(user._id, user.email, user.isAdmin)
return res.json(token)
```

Токен включає в собі дані про id role email. Також можна додавати інші корисні дані, але важливо не додавати забагато інформації, адже токен буде довше генеруватись та декодуватись.

```
const genJWT = (id, email, role) => {
    return jwt.sign(
        payload: {id, email, role},
        process.env.SECRET_KEY,
        options: {expiresIn: '12h'}
    )
}
```

На сервері реалізовані middleware для перевірки валідності токена користувача та для первірки ролі користувача. Для багатьох операцій потрібна авторизація – для її перевірки є метод verifyToken. Для перевірки ролі адміна - verifyTokenAndAdmin.

```
router.post( path: '/',verifyTokenAndAdmin, productController.create)
router.get( path: '/', productController.getAll)
router.get( path: '/:id',verifyToken,  productController.getByID )
router.put( path: '/',verifyTokenAndAdmin, productController.update)
router.delete( path: '/:id',verifyTokenAndAdmin, productController.delete)
```

Якщо певний middleware не проходить, користувач не може виконати дії за маршрутом.

Приклад middleware

```
const verifyTokenAndAuthorization = (req, res, next) => {
    verifyToken(req, res, next: () => {
        if (req.user.id === req.params.id || req.user.isAdmin) {
            next();
        } else {
            res.status(403).json("You are not allowed to do that!");
        }
    });
};
```

Функція некст використовується для переходу далі по ланцюжку middleware’ів або до функції контроллера.

Наш сервер здатен обслуговувати запити користувача за маршрутами авторизації, кошика, замовлення, рев'ю, користувача та продукта за допомогою контроллерів.

# Client

Клієнт – елемент View паттерну MVC. Клієнтський інтерфейс реалізований за допомогою React js. Клієнт має надавати користувачу інтерфейс, та реагувати надії користувача, відправляючи запити на сервер.

Почнемо з інтерфейсу. Реакт дозволяє нам створювати компоненти, суміщаючи js код з реакт компонентами.

```
<ListGroup>
    {product.categories.map(cat=>
    <ListGroup.Item key = {cat.id}
                    style={{cursor: 'pointer'}}
    onClick={()=>product.setActiveCategory(cat)}
    active = {cat.id === product.activeCategory.id}> {cat.name} </ListGroup.Item>
    )}
</ListGroup>
```

Так – тут ми виводимо список категорій за допомгою компоненту ListGroup, в кожен елемент якої передаємо значення з коду. Такий код писати швидко та неважко.

В головному класі застосунку є компоненти AppRouter та Navbar, створені у процесі розробки.

```
function App() {
  return (
      <BrowserRouter>
        <MyNavbar/>
        <AppRouter/>
      </BrowserRouter>
  );
```

AppRouter реалізовує роутинг по застосунку. Спершу первіряємо авторизовані маршрути, потім інші, і якщо користувач намагається перейти за неіснуючим маршрутом, повертаємо його на головну сторінку за допомогою вбудованого компонента Navigate.

```
<Routes>
    {isAuth && authorizateRoutes.map(({path : string, Component}) =>
        <Route key={path} path={path}  element={<Component/>} exact/>
    )}
    {publicRoutes.map(({path : string, Component : function(): any | ...}) =>
        <Route key={path} path={path}  element={<Component/>} exact/>
    )}
    <Route path="*" element={<Navigate to={SHOP_ROUTE} replace />} />

</Routes>
```

```
<Route key={path} path={path}  element={<Component/>} exact/>
```

Тут установлюється відповідність між маршрутом сторінкою, яку потрібно рендерити.

Другий елмент в App це MyNavbar. Навігаційна панель буде доступною з усіх сторінок. В залежності від того чи авторизований користувач, вона малюється по-різному.

.

```
<h2 className="m-auto">{isRegistration ?
    "Реєстрація":'Авторизація'}</h2>
```

```
<Row className="d-flex justify-content-between mt-3 pl-3 pr-3">
    {isRegistration ?

        <div>
            <Link to={LOGIN_ROUTE}>Увійти</Link>
        </div>
        :
        <div>
         <Link to={REGISTRATION_ROUTE}>Зареєструватися</Link>
        </div>
    }
    <Button
        variant={"outline-success"}
        onClick={isRegistration?signUp:signIn}
    >
        {isRegistration ?  "Реєстрація":'Авторизація' }
    </Button>
</Row>
```

Для з’ясування поточної локації використовується хук useLocation. Так визначається на які саме сторінці ми знаходимось – реєстрації чи авторизації.

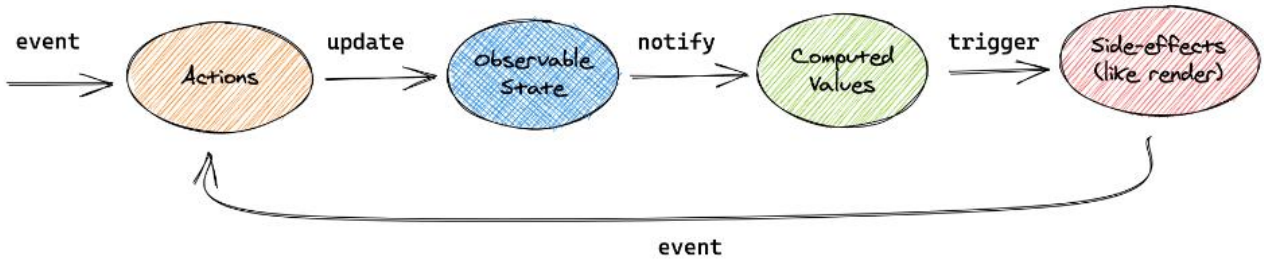
```
const location = useLocation()
const isRegistration = location.pathname === REGISTRATION_ROUTE
```

Для навігації по сторінкам використовується хук useNavigate.

```
const navigate = useNavigate()
```

```
onClick={()=>navigate(PRODUCT_ROUTE + '/' + product.id)
```

Деякі елементи обгорнуті в observer. В застосунку використовується state manager mobx[29]. State manager дозволяє нам слідкувати за змінами елементів. В результаті ми ререндеримо сторінку лише тоді, коли є зміни, потрібні для перемальовки.




Якщо є дія, яка змінює стан,  чи змніюється якийсь компонент, відбувається ререндеринг.

Наші компоненти часто потребують однієї і тієї ж інформації. Таку інформацію можена винести у глобальне сховище, та потім діставати з контексту.

Створення контексту в глобальному середовищі

```
export const Context = createContext( defaultValue: null)
```

Передача інформації в контекст

```
<Context.Provider value = {{
    user: new UserStore(),
    product: new ProductStore()
}}>
```

В нашому випадку інформацію про товари та користувача можна дістати в будь-якому компоненті.

Також в програмі використовуються hook useState, що дозволяє нам вказувати на зміну стану елемента для подальшої перемальовки.

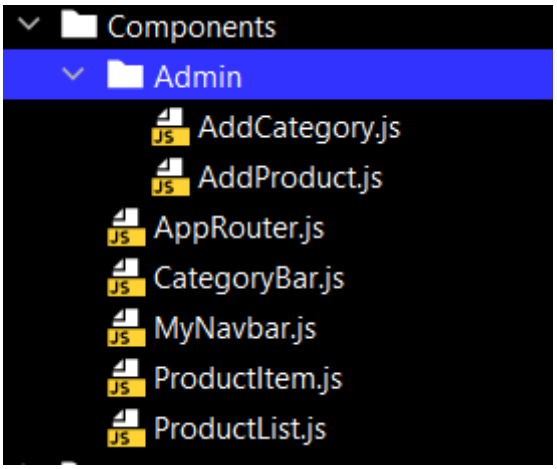
```
const [password, setPassword] = useState( initialState: '')
```

```
onChange={e => setPassword(e.target.value)}
```

Ми використовуємо setPassword – відбувається ререндеринг компонента.

У нашому додатку є такі основні сторінки:


Та такі основні компоненти:




Список компонентів можна розширити.

Shop.js – головна сторінка нашого веб-застосунку.

Складається вона з CategoryBar – панелі категорій та ProductList'а – списку продуктів. В свою чергу ProductList відображає елементи ProductItem`а. ProductItem відповідає за вишляд одного Product`а на загальній сторінці Shop.

Сторінка адміна відповідає за інтер фейс адміністратора. Містить в собі модальні вікна AddCategory та AddProduct для додавання категорії та товару відповідно.

Auth.js має вікна аторизації та реєстрації.

За допомогою useLocation визначаємо поточну локацію та яку сторінку відображати

```
const location = useLocation()
const isRegistration = location.pathname === REGISTRATION_ROUTE
```

```
<div>
    <Link to={LOGIN_ROUTE}>Увійти</Link>
</div>
:
<div>
 <Link to={REGISTRATION_ROUTE}>Зареєструватися</Link>
</div>
```

Вікна реєстрації та авторизації практично ідентичні тому ми не розділяли їх на два компоненти.

Верстка всіх компонентів, сторінок відбувалася за допомогою react bootstrap,який має багато готових класів для елементів.

# Клієнт-серверна взаємодія

Тож, є клієнт, є сервер. Тепер реалізуємо їх взаємодію. Для цього створимо API на клієнті. Тут реалізовані методи взаємодії з сервером. А саме створення продукту, пошук одного, пошук всіх, створення типу, та пошук всіх типів.
Тут ми просто відправляємо на сервер запит, якщо треба то з інофрмацією отриманою на клієнті.

```
export const fetchCategory = async () => {
    const {data} = await $host.get( url: 'api/category')
    return data
}
```

1)API користувача

Реалізовані методи входу та реєстрації

```
export const signup = async(email,password) => {
    console.log(email)
const {data} = await $host.post( url: 'api/auth/signup', data: {email, password} )
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}

export const signin = async(email,password) => {
    const {data} = await $host.post( url: 'api/auth/signin', data: {email, password} )
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}
```

Звертаємось до сервера, отримуємо токен і кладемо токен в localStorage.

2)API продуктів

Реалізовані методи роботи з категоріями та продуктами.

При завантаженні сторінки за допомого хуку useEffect ми дістаємо списки категорій та продуктів за допомогою нашого API.

```
useEffect( effect: ()=>{
  fetchCategory().then(cat=>product.setCategories(cat))
  fetchProduct().then(prod => product.setProducts(prod))
})
```

В елементі Shop, головної сторінки. Тут продукт ми дістаємо з контексту та ініціалізуємо.

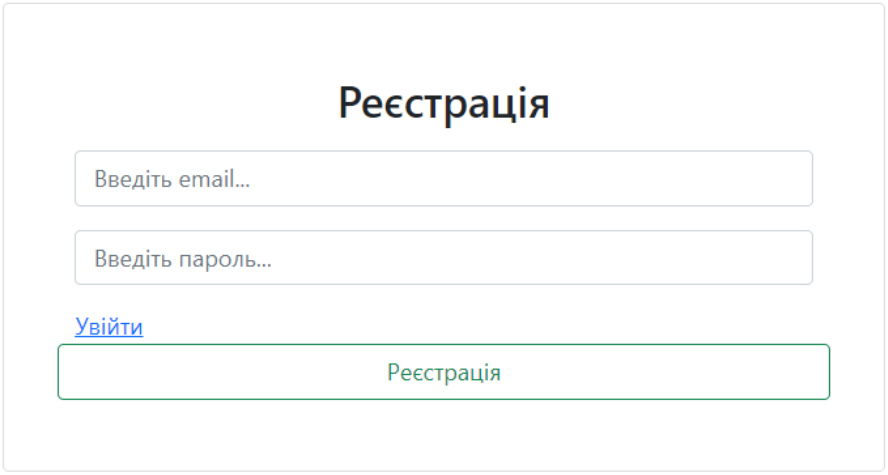
Звернення в АПІ до сервера

```
export const fetchCategory = async () => {
    const {data} = await $host.get( url: 'api/category')
    return data
}
```

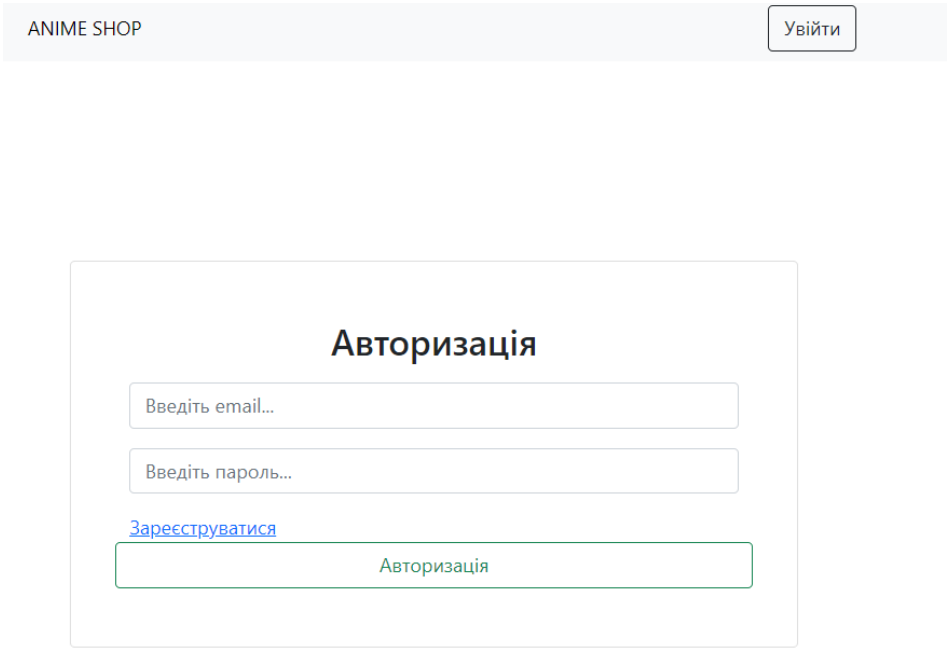
```
export const fetchProduct = async (categoryId) => {
    const {data} = await $host.get( url: 'api/product', config: {params: {
            categoryId
        }})
    return data
```

# Інтерфейс

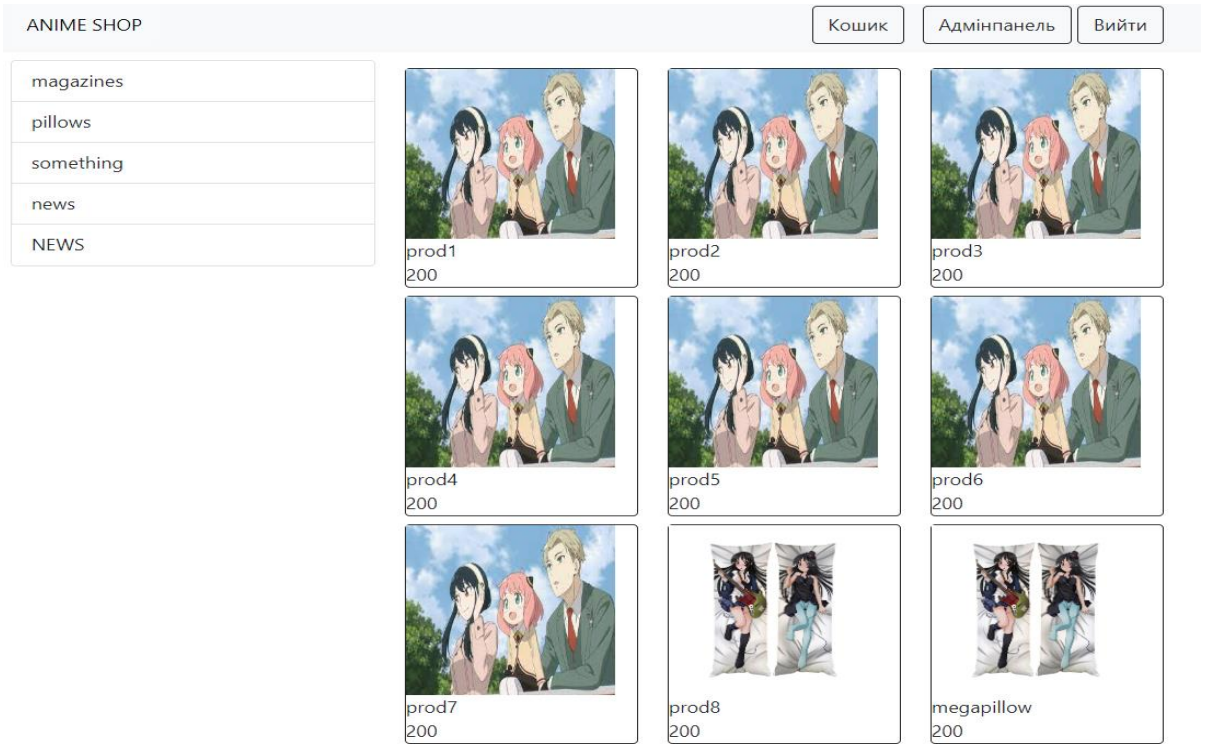
На Сторінці реєстрації користувач може створити власний обліковий запис




Сторінка авторизації

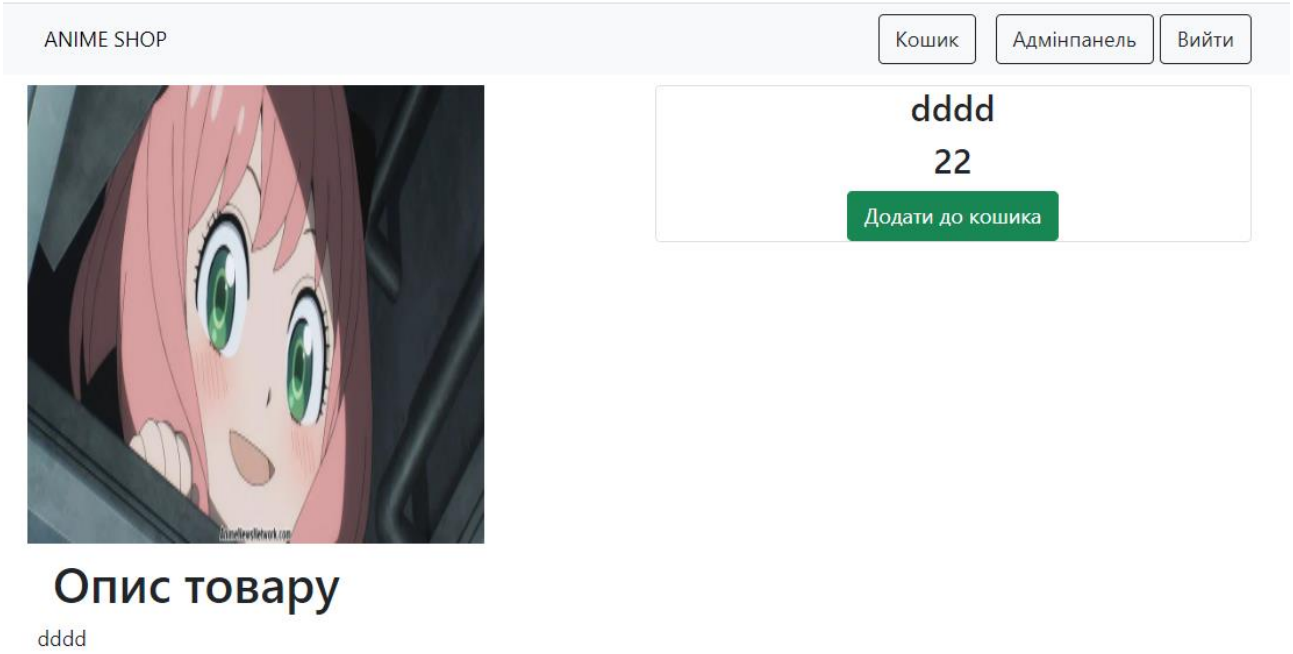



На сторінці shop розміщена продукція магазину, можна перейти на адмінпанель, вийти з акаунту, переглянути товари за категоріями

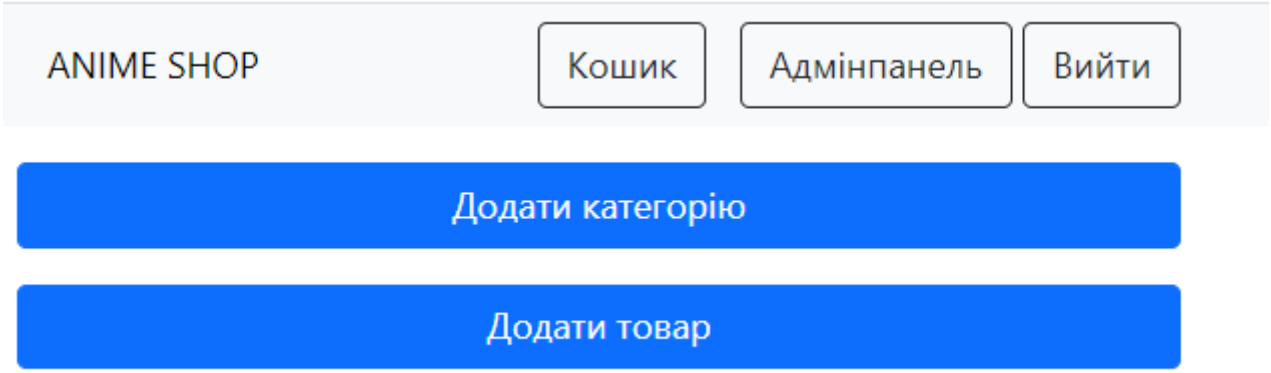


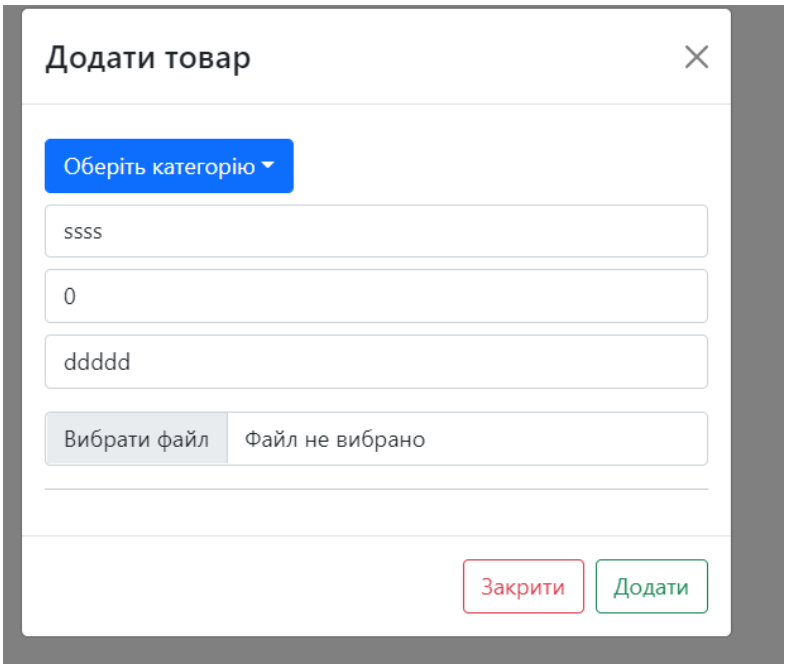

(Категорії та товари вантажаться з беку)

Сторінка товару містить назву, ціну, кнопку додавання до кошика та опис товару




Інтерфейс адміністратора включає можливості додавання нового товару та категорії




Додати товар
Оберіть категорію
ssss
0
ddddd
Вибрати файл
Файл не вибрано
Закрити
Додати

# ВИСНОВКИ: РОЗДІЛ3

В цьому розділі проаналізовано технічне завдання, визначені види користувачів та їх поведінка в системі. Також визначено вимоги до даних, об'єкти програми та взаємодія між ними.

Детально проаналізовані переваги та сильні сторони виокристання Node.js для розробки серверної частини та React.js для клієнтської. Для бази даних обрано MongoDB.

Описані серверна та клієнтська сторони застосунку. Поясненні основні принципи їх роботи з прикладами. Описана взаємодія клієнта з сервером, та сервера з базою.

Продемонстровані та описані інтерфейси адміністратора та користувача.

# ВИСНОВОК

Поставлені задачи були частоко виконані: розроблений веб-застосунок, онлайн-магазин аніме товарів. Є можливість зареєструвати акаунт, зайти та вийти. Можливості перегляду товарів, за необхідністю за категоріями, та перегляд сторінки окремого товару. Реалізовний базовий інтерфейс для адміністраторів Досліджено плюси та мінуси додатків-аналогів.

Розглянуті різіні архітектури розробки веб застосунків, паттерни та різні підходи до розробки додатків.

Було обрано технологію для розробки, що доволяє виконати поставлені задачі. Різноманітні бібліотеки полегшили процес розробки.

В результаті розроблено застосунок для здійснення покупок. Вдосконалити його можна розширивши інтерфейс адміна та інтерфейс користувача, закінчивши реалізацію кошика та офромлення замовлень.

# Список джерел

[1] – Anime chart

https://aja.gr.jp/english/japan-anime-data

[2] – Рейтинг сайтів з аніме-товарами

11 Best Places To Buy Anime Merch - Mofluid.com

[3] – Інтернет магазин JLIST

Kimetsu no Yaeba Demon Slayer Parody Onahole (jlist.com)

[4] – Інтернет магазин Murasaki

Murasaki - інтернет-магазин аніме в Україні

[5] – Інтернет магазин animate

【animate】(Book) Haikyu!! 10th Chronicle [Merch Set Edition]【official】| Anime Merch Shop

[6] – Інтернет магазин Yorokobi

Аніме товари - купить с доставкой по всей Украине, от Одесской компании "Аниме магазин "Yorokobi"" - Сторінка 232

[7] – MVC

Model-View-Controller | Enterprise Solution Patterns Using Microsoft. NET 2003 (flylib.com)

[8] – MVC Node js app

How to Build and Structure a Node.js MVC Application - SitePoint

[9] – MVC

DotNetSolutions: MVC (Model View Controller) architecture workflow for beginners. (msdotnetsol.blogspot.com)

[10] – SOAP vs REST

Comparing SOAP vs REST APIs (restfulapi.net)

[11] – SOAP vs REST

Различия REST и SOAP / Хабр (habr.com)

[12] – Компоненти веб-застосувань

What is Web Application Architecture? Components, Models, and Types (hackr.io)

[13] – Моноліт та мікросервіси

https://medium.com/transparent-data-eng/monoliths-vs-microservices-benefits-and-drawbacks-a-comparision-9e7a462b8e3a

[14] – Моноліт та мікросервіси

Microservices vs Monolith: which architecture is the best choice? (n-ix.com

[15] – Паттерни проектування та їх співставлення

MVC vs MVVM- A complete guide with comparison (intuz.com)

[16] – Паттерни проектування та їх співставлення

MVC vs MVP vs MVVM : 10 Differences You Should Know (xperti.io)

[17] – Типи веб-сайтів

12 Popular Types of Websites You Can Create | HostGator

[18] – Node js

Node.js (nodejs.org)

[19] – чим є Node js

Чем на самом деле является Node.js? / Хабр (habr.com)

[20] – знаймоство з Node js

Node.js Introduction (w3schools.com)

[21] – Порівняння Django vs Node js

Django vs. Node JS | Difference between Django and Node JS - Javatpoint

[22] – Переваги React jsx

React JSX (w3schools.com)

[23] – Що таке DOM

JavaScript HTML DOM (w3schools.com)

[24] – Що таке VDOM, різниця між DOM та VDOM

The difference between Virtual DOM and DOM - React Kung Fu

[25] – як працює VDOM

How does the virtual DOM work in React! - DEV Community

[26] – переваги MongoDb

Why Use MongoDB And When To Use It? | MongoDB

[27] – переваги React js

ReactJS Tutorials - GeeksforGeeks

[28] – робота різних видів аутентифікації

https://www.section.io/engineering-education/cookie-vs-token-authentication/

[29] – використання mobX

https://mobx.js.org/react-integration.html

[30] – peer-to-peer vs client server

Comparison of "peer-to-peer" vs "client-server" Network Models (networkstraining.com)

[31] – Http і crud

https://nordicapis.com/crud-vs-rest-whats-the-difference/