МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ

«ЗАПОРІЗЬКИЙ ЕЛЕКТРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Циклова комісія спеціальності

121 Інженерія програмного забезпечення

спеціалізація «Розробка програмного забезпечення»

реалізація багатошарової

клієнт-серверної архітектури на прикладі створення системи онлайн продаж

Пояснювальна записка до курсової роботи

121.44.29.01 ПЗ

Викладач Ольга СЛАДКОВА

Члени комісії Ян БАСОК

Алла СМІРНОВА

Студент гр. РПЗ 19 2/9 Іван ЩЕДРОВСЬКИЙ

2022

РЕФЕРАТ

Пояснювальна записка до курсової роботи містить 76 сторінки,   
38 рисунків, 9 таблиць, 1 додаток, 24 джерел.

Метою курсової роботи є розробка програмного сервісу онлайн продаж.

Програмне забезпечення розроблене у середовищі Microsoft Visual   
Studio Code за допомогою мови програмування TypeScript.

Програма працює на ПК під керуванням операційних систем: Microsoft Windows 10, Arch Linux та Ubuntu.

В загальному розділі розглянуті такі питання: опис предметної області, постанова завдання, обґрунтування вибору середовища розробки системи та середовища функціонування системи.

В спеціальному розділі розглянуті основні рішення щодо реалізації компонентів системи, методика роботи користувача з системою, яка включає інструкцію програмісту та інструкцію оператору.

СЕРВІС, КЛІЄНТ, СЕРВЕР, КОРИСТУВАЧ, ПРОДУКТ, ЗАМОВЛЕННЯ, РІВНІ АДМІНІСТРАЦІЇ, КАТЕГОРІЇ, ТИПИ, JavaScript, TypeScript, NodeJS, Express, СУБД MongoDB, ДОКУМЕНТО-ОРІЄНТОВАНА МОДЕЛЬ

ЗМІСТ

Вступ 4

1. Загальний розділ 6
   1. Опис предметної області 6
   2. Постанова завдання 16
   3. Обґрунтування вибору середовища розробки системи 17
   4. Обґрунтування вибору середовища функціонування системи 18
2. Спеціальний розділ 20
   1. Основні рішення щодо реалізації компонентів системи 20
   2. Використані моделі даних 39
   3. Методика роботи користувача з системою 52
      1. Керівництво програміста 52
      2. Керівництво оператора 53

Висновки 57

Перелік джерел посилання 58

Додаток А (обов’язковий) - Текст програми 59

Вступ

З розвитком прогресу і винайденням інтернету, об’єм доступної людині інформації зростає з кожним днем. Якщо раніше отримання певної інформації для більшості людей було не тільки важким, а іноді і неможливим завданням, сьогодні дістати з кишені смартфон та набрати в ньому запит в популярній пошуковій мережі не є проблемою: доступ до всесвітньої мережі даних, мільйонів документів є майже у кожної людини.

Інформаційні технології мають сьогодні пріоритетне значення в багатьох сферах діяльності й визначають майбутній розвиток суспільства.

Одним з таких напрямків є інтернет-бізнес, який починає привертати все більше значення в сучасному світі, і в нашій країні відповідно, у зв'язку з тенденцією до загальної глобалізації економіки. Інтернет і електронна торгівля грають в цьому процесі одну з найважливіших ролей. Поява абсолютного нового виду зв'язку зумовило перегляд підприємців процесу організації бізнесу.

Інтернет відкриває широкі можливості не тільки для великих підприємств. Багато початківців бізнесменів відразу відкривають свою справу в Інтернеті, так як для цього потрібна менша сума первинних інвестицій, ніж у сфері традиційної торгівлі. А компанії, які працюють за принципом прямих поставки, вкладають кошти виключно в веб-сайт і рекламу.

Західні покупці, а вслід за ними і українські, все частіше купують товари, не виходячи з дому, посилаючись на нестачу часу, невеликий асортимент товарів у традиційних магазинах і високі ціни. В Україні в початку 2000-х тільки інтузіасти зробили електронні покупки, сьогодні, за даними статистики, хоча б раз це зробили 4 з 5 чоловік. Розвиток бізнесу в інтернеті на сьогоднішній день крайне перспективне напрямок для підприємців, що і обумовлює актуальність даної роботи.

Розробка веб-сайтів електронної комерції – це нелегке завдання. У нього є свої проблеми. Оскільки велика кількість веб-користувачів відвідують сайт електронної комерції щодня, тому потрібно його підтримувати. Для цього може знадобитися залучити експертів веб-розробників електронної комерції від надійної та відомий компанії з розробки електронної комерції веб-сайтів, що забезпечує інтуїтивний процес веб-розробки електронної комерції з легкістю.

Темою моєї курсової роботи було обрано "Розробка багатошарової клієнт-серверної архітектури на прикладі створення сервісу онлайн продаж". Метою є створення зручного, легкого в використанні сервісу онлайн продаж.

Середою розробки сервісу було обрано сучасну інтегровану середу Visual Studio Code. Ця середа дозволяє швидко та ефективно працювати при написанні коду, налагодженні, компілюванні та складанні проекту.

1 Загальний розділ

1.1 Опис предметної області

Предметна область — це частина реального світу, що розглядається в межах певного контексту.

Тематикою моєї роботи є створення сервісу онлайн продажів.

Сервіс онлайн продажів - сервіс в інтернеті, де відбувається прямий продаж товарів споживачеві (юридичній або фізичній особі). При цьому розміщення споживацької інформації, замовлення товару і угода відбуваються там само, всередині мережі (на сайті інтернет-магазину).

Розробка програмного забезпечення — це процес, спрямований на створення та підтримку працездатності, якості та надійності програмного забезпечення, використовуючи технології, методологію та практики з інформатики, керування проєктами, математики, інженерії та інших областей знання.

Електронна комерція — це сфера цифрової економіки, що включає всі фінансові та торгові транзакції, які проводяться за допомогою комп'ютерних мереж, та бізнес-процеси, пов'язані з проведенням цих транзакцій.

Елементами сервісу онлайн продаж є: товар, користувач, каталог, кошик, особистий кабінет

Каталог — список, перелік товарів, складений у певному порядку, щоб полегшити їхній пошук.

Товар — продукт природи і людської праці або тільки людської праці у матеріальній і нематеріальній субстанції та у формі послуг, який завдяки своїм властивостям здатний задовольняти наявні чи передбачувані суспільні потреби.

Кошик - список, перелік товарів які користувач планує купити. Створений щоб полегшити перегляд, видалення або редагування кількості товарів

Особистий кабінет - це особливий розділ, який дозволяє користувачу отримати доступ до даних про стан та деталей замовлення.

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними.

Сервіс — це обслуговування населення, забезпечення його побутових потреб.

Існує два загальних абстрактних понять Архітектури - перший пов’язаний з розбиттям системи на найбільш значимі складові частини; в другому випадку маються на увазі деякі конструктивні рішення, котрі після їх прийняття важко піддаються внесенню змін. Також, є розуміння того, що існує більше одного способу описання архітектури і ступінь важливості кожного з них змінюється з плином життєвого циклу системи

В більшості корпоративних додатків відслідковується та чи інша форма архітектурного «розшарування»

Концепція шарів(або рівнів) – одна з загально використовуваних моделей, використовуваних розробниками програмного забезпечення для розділення складних систем на більш прості частини

На рисунку 1.1 представлений приклад розділення додатку на шари



Рисунок 1.1 – Приклад розділення додатку на шари

Описуючи систему в термінах архітектурних шарів, зручно сприймати підсистеми, з яких вона складається у вигляді «багатошарового пирога».

Шар більш високого рівня користується послугами, що надає нижній шар, але той не знає про існування сусіднього верхнього рівня. Більше того, зазвичай кожен проміжний шар приховує нижній шар від верхнього.

Розділення системи на шари надає цілий ряд переваг:

- окремий шар можна сприймати як єдине самодостатнє ціле, не піклуючись про наявність інших шарів;

- можна обрати альтернативну реалізацію базових шарів;

- залежність між шарами зводиться до мінімуму;

- кожен шар є кандидатом на стандартизацію;

- якісно створений шар може слугувати основою для декількох різних шарів більш високого рівня.

Схемі розшарування властиві певні недоліки: шари здатні вдало інкапсулювати багато, але не все; модифікація одного шару одночасно пов’язана з потребою внесення каскадних змін в інші шари.

Другим недоліком є те, що наявність додаткових шарів знижує продуктивність системи. При переході від шару до шару сутності зазвичай піддаються трансформації з одного представлення в інше.

Не зважаючи на це, інкапсуляція нижче розташованих шарів дозволяє досягнути істотних переваг. Наприклад, оптимізація шару транзакцій зазвичай приводить до підвищення продуктивності всіх шарів, що розташовані вище.

Поняття шару набуло очевидної значущості в середині 1990-х років з появою архітектури клієнт-сервер. Це були системи з двома шарами, клієнт відповідав за роботу інтерфейсу користувача і виконання коду додатка, а роль сервера виконувала СКБД

Двошарова архітектура програмного забезпечення зазвичай відповідає моделі “товстого” клієнта. В такій моделі серверні компоненти системи відповідають, головним чином, за організацію зберігання і доступу до даних, а всі або більшість функцій прикладної обробки даних виконуються на стороні клієнтської частини.

Шари прикладних рішень і засоби підтримки виконання програм прикладного шару, що входять в системний шар функціонують на робочій станції, а засоби організації зберігання і доступу до даних - здебільшого на сервері.

На рисунку 1.2 представлено приклад двошарової архітектури



Рисунок 1.2 - Приклад двошарової архітектури

Головними перевагами такої архітектури є:

- Простота системи, у порівняні з тришаровою і багатошаровою архітектурами;

- Гарантія цілісності даних;

- Повна підтримка одночасної роботи багатьох користувачів.

Але така архітектура має досить значні недоліки, а саме:

- Необхідність більш потужного комп’ютера в якості сервера та потужних клієнтських машин, здатних забезпечити і бізнес логіку і графічний інтерфейс;

- Відсутність масштабування. Слабкий захист від взлому;

- Бізнес-логіка повністю на стороні клієнта. При її зміні треба повністю оновлювати клієнтське ПЗ.

Через недоліки двошарової архітектури на зміну їй прийшла трьохшарова

На рисунку 1.3 представлено приклад трьохшарової архітектури



Рисунок 1.3 - Приклад трьохшарової архітектури

Основними шарами даної архітектури є шар представлення; домен, котрий ще називають шаром бізнес-логіки та шар даних, який узагальнює джерела даних. Кожен з них має визначені функції та несе відповідальність за частину роботи, виконувану додатком, шари можуть розміщуватися не тільки локально на одному пристрої, а і бути розділеними, наприклад представлення на клієнтській частині, а бізнес-логіка і джерело даних – на серверній частині додатку

Шар представлення виконує надання послуг, відображення даних, обробку подій користувацького інтерфейсу, обслуговування HTTP-запитів, підтримку функцій командної строки та API пакетного використання.

Шар домену - бізнес-логіку додатку, специфічні алгоритми

Джерело даних - запити до бази даних, обмін повідомленнями, управління транзакціями, тощо

Шару представлення стосується усе, що пов’язане зі взаємодією користувача з системою. Він може бути простим, як командна строка чи текстове меню, але зараз користувачу, ймовірніше за все, доведеться мати справу з графічним інтерфейсом, оформленим у стилі «товстого» клієнта.

Головна задача шару представлення – транслювати команди користувача у формат, зрозумілий шару бізнес-логіки.

Логіка домену – описує основні функції додатку, призначені для досягнення поставленої перед ним цілі. До цих функцій належать обчислення на основі введених і збережених процедур, перевірка усіх елементів даних і обробка команд, що надходять від шару представлення, а також передача інформації шару джерела даних.

Іноді шари організовують таким чином, щоб бізнес-логіка повністю приховувала джерело даних від представлення. Однак частіше код представлення може звертатися до джерела даних безпосередньо. Хоча такий варіант менш бездоганний з теоретичної точки зору, в практичному використанні він нерідко більш зручний та доцільний; код представлення може інтерпретувати команду користувача, активізувати функції джерела даних для отримання відповідних порцій інформації з бази даних, звернутися до засобів бізнес-логіки для аналізу цієї інформації і виконання необхідних розрахунків і тільки після цього відобразити відповідну картину на екрані.

Джерело даних – це підмножина функцій, що забезпечують взаємодію зі сторонніми системами, котрі виконують завдання в інтересах додатку. Код цієї категорії несе відповідальність за моніторинг транзакцій, управління іншими додатками, обмін повідомленнями тощо. Для більшості корпоративних додатків основна частина логіки джерела даних концентрується в коді СКБД

Індустрія не стояла на місці та розширила поняття трирівневої архітектури до багаторівневої. Логічно модель має таку ж саму структуру, але всеохоплююче використання Інтернету внесло свої корективи, ставши важливою частиною багатьох програмних додатків.

Веб-сервіси (а пізніше REST дані) стали більш інтегровані в додатки. Як наслідок, шар даних, як правило, стали розщеплювати на рівень зберігання даних (сервер баз даних) і рівень доступу до даних. У комплексних системах для уніфікації доступу до баз даних і веб-сервісів розробляють додатковий рівень класів-обгорток.

Веб-браузери були менш потужним, ніж традиційні додатки клієнтського рівня і логіка користувацького інтерфейсу розділилися між браузером з JavaScript і сервером з додатком веб-сервера, що містить у собі логіку користувацького інтерфейсу.

Шари все далі і далі набували більш розмитого характеру із додаванням збережуваних процедур усіма основними постачальниками баз даних і баз даних з відкритим вихідним кодом. Це призвело до поширення практики переносу деяких частин бізнес-логіки від бізнес-рівня на рівень бази даних, тобто з’явилася концепція створення рівнів в межах рівнів.

Так як під впливом Інтернету, технологічних інновацій і сервісів архітектура додатку стала більш розмитою, трьохшарова модель додатку розвинулася у багаторівневу архітектуру

На рисунку 1.4 представлено приклад багатошарової архітектури



Рисунок 1.4 – Приклад багатошарової архітектури

Багатошарова архітектура з’явилася завдяки створенню декількох рівнів в інших рівнях, а саме в рівні представлення:

- компоненти графічного інтерфейсу, котрі відповідають за відображення графічних елементів;

- компоненти процесів графічного інтерфейсу, котрі реагують на події, що відбуваються у графічному інтерфейсі.

Великі корпоративні додатки часто структуровані навколо бізнеспроцесів та бізнес-компонентів. Ці поняття розглядаються в рамках цілого ряду компонентів, сутностей, агентів та інтерфейсів бізнес-рівня:

- бізнес-компоненти – програмні реалізації концепцій чи процесів. Вони складаються з усіх артефактів необхідних для представлення, реалізації, розгортування конкретної концепції як автономного елемента більшої системи, котрий можна використовувати повторно;

- бізнес-сутності – це структури, що виступають контейнерами даних. Вони інкапсулюють та приховують деталі специфічного формату представлення даних. Наприклад, бізнес сутність може інкапсулювати набір записів, отриманих з бази даних. Пізніше, ця ж бізнес-сутність може бути змінена для огортання в XML-документ з мінімальним впливом на інші частини додатку;

- сервісні інтерфейси – додаток може надавати частину його функціоналу як сервіс, котрий можуть використовувати інші додатки. В ідеалі він приховує деталі реалізації і надає тільки тонкий шар інтерфейсу;

- бізнес-процеси – відображають діяльність бізнесу на високих рівні абстракції системи, як-то обробка замовлення, підтримка користувача, закупка товару.

Шар даних теж зазнав певних внутрішніх метаморфоз, внаслідок чого з’явилися наступні шари:

- компоненти доступу до даних – ізолюють бізнес-шар від деталей реалізації, специфічних для сховища даних. Дозволяє мінімізувати вплив зміни постачальника бази даних, зміни представлення даних, наприклад, схеми бази даних, інкапсулює весь код, що маніпулює конкретною одиницею даних в одному місці, що надзвичайно спрощує підтримку та тестування;

- сервісні шлюзи – бізнес-компоненти часто повинні отримувати доступ до внутрішніх та зовнішніх сервісів чи додатків. Сервісний шлюз – це компонент, що інкапсулює інтерфейс, протокол та код, потрібний для використання сервісів. Наприклад, бізнес-рішення часто потребує інформацію з деякого сервісу для завершення бізнес-процесу. Воно делегуватиме всю взаємодію з цим сервісом шлюзу. Сервісний шлюз надає можливість з меншими зусиллями змінити зовнішній сервіс на інший. Також даний підхід надає змогу емулювати зовнішній сервіс, наприклад, для тестування доменного рівня.

Надодачу до описаних шарів багатошарова архітектура визначає набір фундаментальних сервісів, котрі потенційно можуть використовувати усі інші шари. Ці сервіси діляться на три базові категорії:

- шар безпеки – сервіси цього шару підтримують безпеку додатку;

- шар операційного управління – ці сервіси оперують компонентами і зв’язаними з ними ресурсами і також торкаються таких вимог як масштабованість та відмовостійкість;

- шар сервісів комунікації – сервіси, котрі надають можливість спілкуватися різним шарам між собою.

Переваги даної архітектури – гарна точка відправлення для побудови власних додатків. Розробнику, що використовує даний підхід, дістаються найбільші позитивні риси розшарованого додатку. Але є й певні аспекти архітектури, які додають відповідальності, а саме, для важких, комплексних рішень необхідно правильно розділяти доменний рівень, особливо, якщо можливість повторного використання компонентів є в пріоритеті або якщо розробник проектує сімейство рішень, що базується на наборі компонентів. У такому випадку типовим є заміна одного бізнес-шару класичного тришарового додатку трьома. [1]

1.2 Постанова завдання

Метою курсової роботи є створення системи онлайн продаж. Дана система здійснює продаж аксесуарів для домашніх тварин.

Система повинна дозволити покупцям здійснювати швидкий та детальний перегляд продукції.

Сервіс повинен включати в себе функцію реєстрації користувача, залишення відгуків та здійснення замовлення, додавання товару.

Після вивчення предметної області і створення відповідної бази даних необхідно створити сервіс, який має наступні функції:

- Вибір необхідних товарів с фільтрацією по будь-якій характеристиці;

- Створення аккаунтів, зміни інформації про них

- Додавання та видалення відгуків про товар;

- Створення замовлень, перегляд інформації про них;

- Зручне та швидке маніпулювання адміністратором даними про товари(додавання, видалення, змінення).

Повинна бути реалізована 2-х рівнева система адміністрації: модератори та адміністратори

Вимоги до надійності та безпеки:

- Тільки адміністратор має право додавання, змінення чи видалення товару;

- Модератор має право видаляти відгуки про товари та змінювати статуси замовлення;

- Адміністратор має право змінювати інформацію про постачальників, категорії та типи товарів.

Етапами розробки програмного проєкту є:

- Формування вимог;

- Проєктування системи;

- Реалізація;

- Тестування та виправлення помилок, якщо такі є;

- Впровадження;

- Експлуатація та супровід.

1.3 Обґрунтування вибору середовища розробки системи

Visual Studio Code - інтегрована середа розробки програмних продуктів, розроблена Microsoft та доступна на всіх операційних системах.

Visual Studio Code надає можливість швидко розробляти продукти завдяки вбудованому функціоналу та багатьом розширенням.

Також ця середа розробки надає можливість запускати та відлагувати програму завдяки точкам зупинки, стеку викликів та інтерактивній консолі

Visual Studio Code має вбудований графічний інтерфейс для роботи с Git, який надає можливість переглядати відмінності, проміжні файли, робити знімки змін, відправляти знімки на сервер та отримувати їх.

Редактор має багато розширень, які дозволяють налаштувати його як візуальну складову, так і функціональну [2]

Git — це безкоштовна розподілена система контролю версій із відкритим вихідним кодом, призначена для швидкої та ефективної обробки будь-яких проектів, від малих до дуже великих.

Git простий у вивченні та має невеликий розмір з блискавичною продуктивністю. Він перевершує такі інструменти SCM, як Subversion, CVS, Perforce і ClearCase, завдяки таким функціям, як дешеве локальне розгалуження, зручні робочі області та численні робочі процеси. [3]

Postman — це платформа API для створення та використання API. Postman спрощує та оптимізує кожен крок життєвого циклу API щоб створювати кращі API — швидше.

Postman може зберігати та керувати специфікаціями API, документацією, рецептами робочих процесів, тестами та результатами, показниками та всім іншим, що стосується API.

Платформа Postman містить повний набір інструментів, які допомагають прискорити життєвий цикл API — від проектування, тестування, документування та моделювання до спільного використання та видимості ваших API.

Підхід Postman до управління, що базується на повному життєвому циклі, дає змогу відмовитись від розробки свойого сервісу, створюючи API кращої якості та сприяючи співпраці між командами розробників і командами дизайнерів API. [4]

Swagger - сервіс, який дозволяє спростити розробку API за допомогою професійного набору інструментів з відкритим кодом.

Swagger дозволяє спроектувати та змоделювати API відповідно до стандартів на основі специфікацій.

Покращує та спрощує роботу завдяки інтерактивної документації API.

Також Swagger виконує прості функціональні тести API без додаткових витрат [5]

1.4 Обґрунтування вибору середовища функціонування системи

Середовищем функціонування сервісу є NodeJS.

Різні версії NodeJS мають різну підтримку нових функцій. Сервіс буде правильно працювати в версіях NodeJS:

- NodeJS 14.19.1;

- NodeJS 16.18.1;

- NodeJS 18.12.1.

NodeJS на даний момент є найбільш популярною платформою для розробки клієнт-серверних застосунків

На рисунку 1.5 представлено графік порівняння використання технологій за опитом розробників за 2020 рік. Я видно з графіку більше ніж 50% розробників в своїй роботі використовують NodeJS

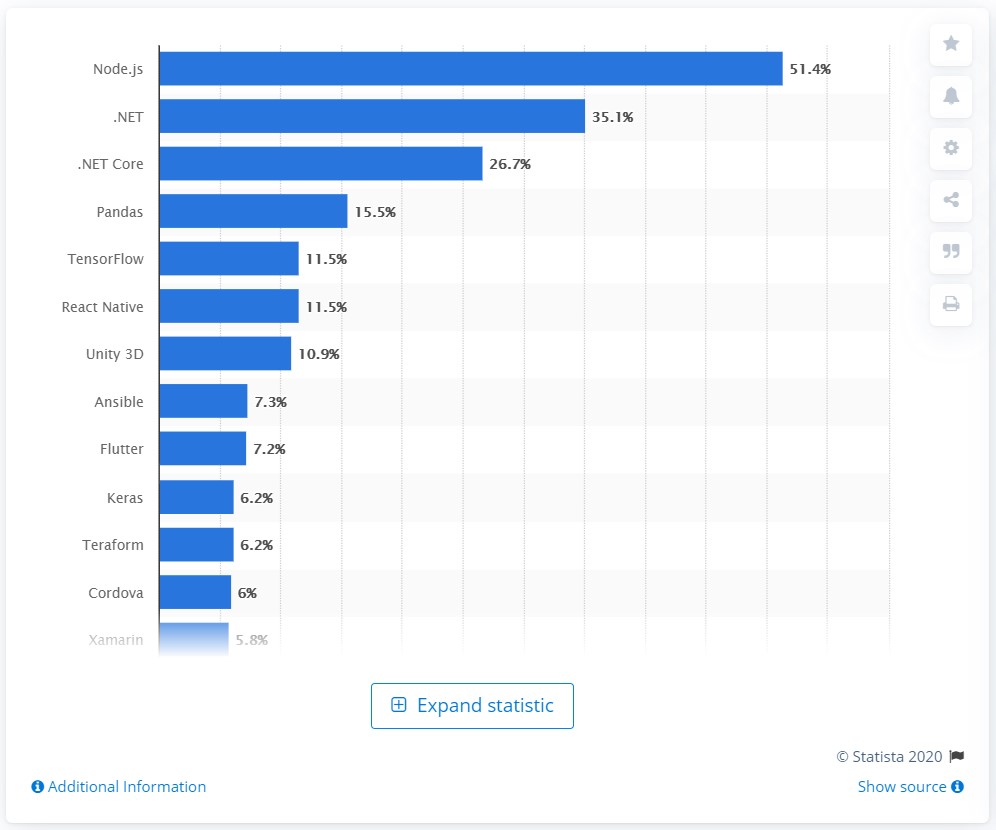


Рисунок 1.5 – Графік популярності різних технологій

2 Спеціальний розділ

2.1 Основні рішення щодо реалізації компонентів системи

JavaScript – це легка, інтерпретована, об'єктно-орієнтована мова з функціями першого класу, а також найвідоміша скриптова мова для веб-сторінок, але також використовується в багатьох не браузерних оточеннях.

Інтерпретована означає, що сирцевий код програми не перетворюється повністю у машинний код для виконання, як у компільованих, а виконується рядок за рядком з допомогою спеціальної програми інтерпретатора

Доказом того, що JavaScript - інтерпретована мова програмування є дещо різна поведінка одного й того самого коду в різних оточеннях

JavaScript є об'єктно-орієнтованою мовою через те, що в основі мови лежить поняття об’єкта – певної сутності, яка об’єднує в собі поля(дані) і методи(виконувані об’єктом дії)

JavaScript має функції першого класу, бо розглядає функції як об’єкти першого класу, тобто сутність, яка може бути побудована під час виконання програми, передаватись як параметр, повертатись з підпрограми, або присвоюватись змінній. Це означає, що JavaScript підтримує передачу функції як аргументів інших функцій, повернення їх як результат інших функцій, присвоювання їх змінним або збереження в структурах даних

На рисунку 2.1 представлено логотип JavaScript



Рисунок 2.1 – Логотип JavaScript

JavaScript може виконуватись як за допомогою браузерного оточення або, наприклад, NodeJS. NodeJS та більшість браузерів на даний момент використовують в якості рушія V8 який компілює JavaScript код безпосередньо у власний машинний код, минаючи стадію проміжного байт-коду, а також ефективно керує пам’яттю.

На рисунку 2.2 представлено логотип рушія JavaScript V8



Рисунок 2.2 - Логотип рушія JavaScript V8

JavaScript - прототипно-орієнтована мультипарадигменна мова сценаріїв.

Прототипно-орієнтованою, бо в цій мові відсутнє поняття класу, а повторне використання (успадкування) проводиться шляхом клонування наявного примірника об’єкта – прототипу.

Мультипарадигменна – бо підтримує декілька підходів або стилів программування: динамічний, об'єктно-орієнтований, імперативний та функціональний стилі.

JavaScript може функціонувати і як процедурна, і як об'єктно-орієнтована мова. Об'єкти можна створювати програмно під час виконання, шляхом приєднання методів і властивостей або порожніх об'єктів під час виконання, на відміну від синтаксичних визначень класів у мовах, що компілюються, таких як С++ або Java. Після того, як об'єкт був створений, він може бути використаний як план (або прототип) для створення схожих об'єктів.

Динамічні можливості JavaScript включають створення об'єктів під час виконання, змінну кількість параметрів, динамічне створення скриптів (за допомогою eval), перебір об'єктів (за допомогою for ... in), відновлення вихідного коду (програми на JavaScript можуть декомпілювати тіла функцій назад у вихідний код).

JavaScript є динамічно типізованою мовою програмування, тобто основна частина перевірок типів виконується під час виконання програми, а не під час компіляції. У динамічній типізації значення мають типи, а змінні – ні, тому змінна може містити значення будь-якого типу.

Динамічність дозволяє простіше писати програму, але може заплутати розробника, а в деяких ситуація помилка в коді через типізацію може випадково попасти до клієнта та викликати масу помилок.

Тому над JavaScript існує безліч надбудов, які додають ті, чи інші функції до мови, а потім компілюються в JavaScript який далі виконується. Найбільш популярною надбудовою є TypeScript. [6]

TypeScript — це строго типізована мова програмування, яка ґрунтується на JavaScript і надає кращі інструменти будь-якого масштабу.

TypeScript додає до JavaScript додатковий синтаксис для підтримки більш тісної інтеграції з редактором. Дозволяє ловити помилки на ранніх стадіях розробки.

На рисунку 2.3 представлено логотип TypeScript

Изображение выглядит как текст, коллекция картинок

Автоматически созданное описание

Рисунок 2.3 - Логотип TypeScript

Код TypeScript перетворюється на JavaScript, який виконується будь-де, де працює JavaScript: у браузері, на NodeJS або Deno.

TypeScript розуміє JavaScript і використовує визначення типу, щоб надати чудові інструменти без додаткового коду. [7]

В силу повної зворотної сумісності адаптація наявних застосунків з JavaScript на TypeScript може відбуватися поетапно, шляхом поступового визначення типів. Підтримка [динамічної типізації](https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F) зберігається — компілятор TypeScript успішно обробить і не модифікований код на JavaScript.

Основний принцип — будь-який код на JavaScript сумісний з TypeScript, тобто в програмах на TypeScript можна використовувати стандартні JavaScript-бібліотеки і раніше створені напрацювання. Більш того, можна залишити наявні JavaScript-проекти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли, які не заважатимуть розробці і прямому використанню проекту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

JavaScript на 2022 рік є найбільш популярною мовою програмування в світі, а TypeScript займає 4 місце [octoverse.github.com]

На рисунку 2.4 представлено графік популярності мов програмування

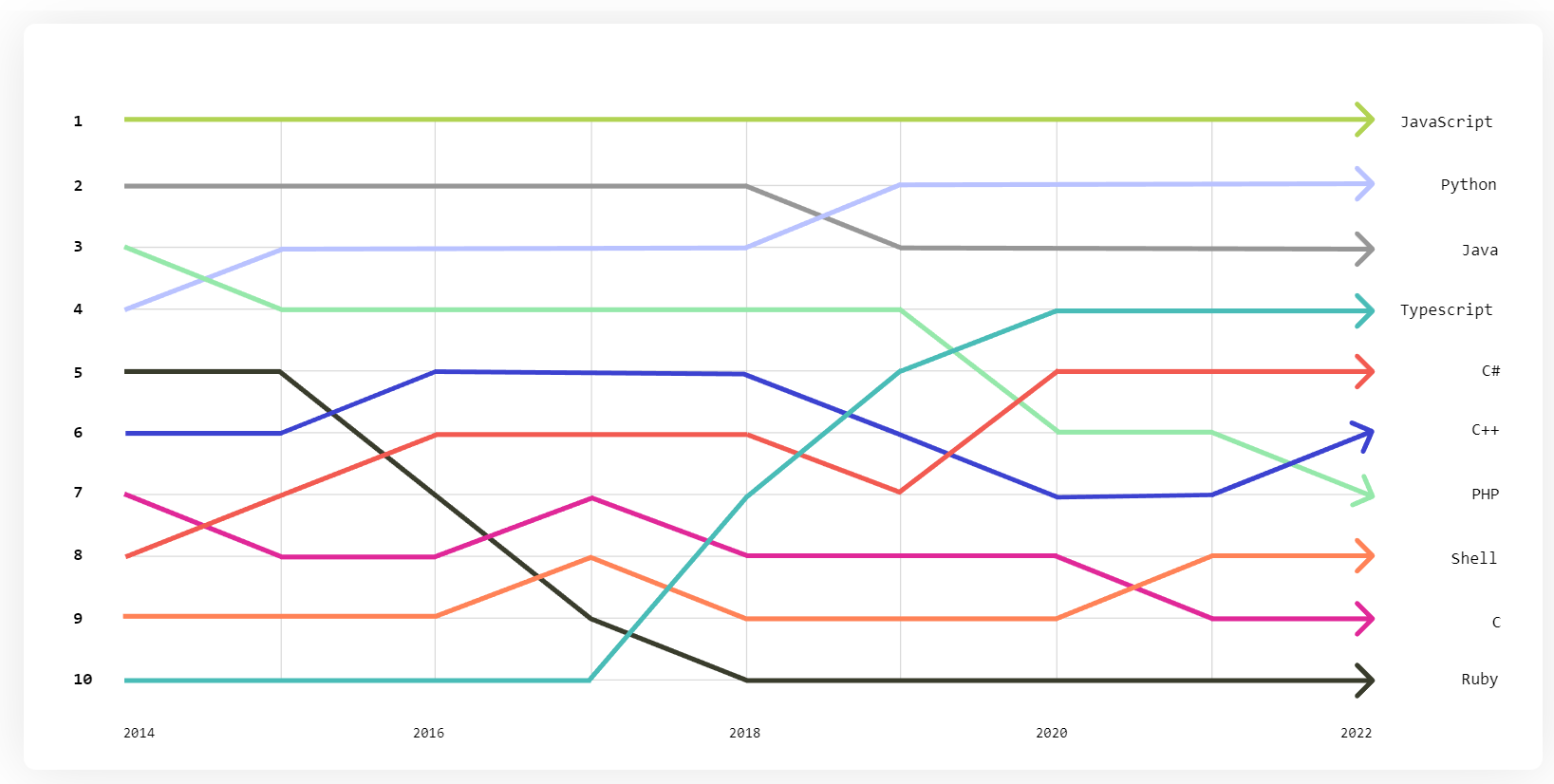


Рисунок 2.4 - Графік трендів мов програмування

NodeJS — платформа, побудована на JavaScript–рушієві V8, з [відкритим](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D1%82%D0%B8%D0%BA%D0%B0_%D0%B2%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D1%83) [кодом](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%80%D1%86%D0%B5%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) для виконання високопродуктивних мережевих [застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA), написаних мовою [JavaScript](https://uk.wikipedia.org/wiki/JavaScript).

З відкритим кодом означає, що кожна людина може запропонувати свої зміни, які надалі будуть перевірені та можуть попасти в наступну версію.

Раніше, до створення NodeJS, JavaScript застосовувався для обробки даних в браузері користувача. NodeJS надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання. Таким чином NodeJS перетворила JavaScript з вузькоспеціалізованої мови для браузерів в мову загального призначення.[8]

На рисунку 2.5 представлено логотип NodeJS

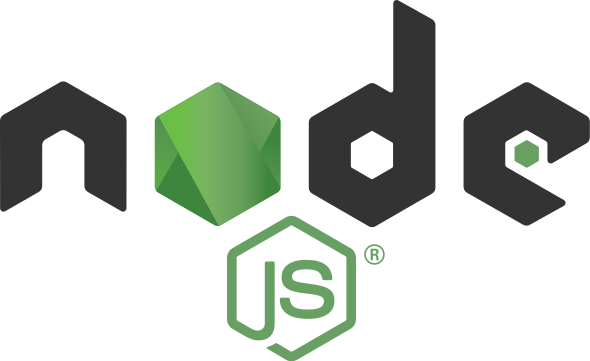


Рисунок 2.5 - Логотип NodeJS

Node Version Manager (далі - nvm) – контролер версій NodeJS, створений для швидкої розробки та тестування застосунку без необхідності встановлювати нову чи стару версії NodeJS для кожного запуску. [9]

На рисунку 2.6 представлено логотип nvm

Изображение выглядит как текст, знак, темный

Автоматически созданное описание

Рисунок 2.6 - Логотип Node Version Manager

Node Package Manager - менеджер пакетів, що входить до складу NodeJS, найбільший у світі реєстр програмного забезпечення. Розробники пакетів з відкритим кодом з усіх континентів використовують npm для спільного використання та запозичення пакетів, а багато організацій також використовують npm для керування приватною розробкою.

Node Package Manager складається з клієнта командного рядка, який взаємодіє з віддаленим реєстром. Це дозволяє користувачам користуватися модулями JavaScript та розповсюджувати їх. В головному реєстрі npm доступно понад 477 000 пакунків. Реєстр не має процедури перевірки, а це означає, що знайдені там пакунки можуть бути низькоякісними або небезпечними. Натомість Node Package Manager спирається на звіти користувачів, щоб видаляти пакунки, якщо вони порушують політику безпеки (є незахищеними, зловмисними або низькоякісними). npm показує статистику, включаючи кількість завантажень та кількість пакунків, щоб допомогти розробникам оцінювати якість пакетів.

Node Package Manager може управляти пакунками, які є локальними залежностями певного проекту, а також глобально інстальованими інструментами JavaScript. При використанні npm як менеджера залежності для локального проекту, можна встановити одною командою всі залежності проекту через файл package.json. У файлі package.json кожна залежність може визначати діапазон дійсних версій, використовуючи схему семантичної версії, що дозволяє розробникам автоматично оновлювати свої пакети, одночасно уникаючи небажаних змін.

На рисунку 2.7 представлено логотип npm



Рисунок 2.7 - Логотип Node Package Manager

На рисунку 2.8 представлено package.json файл проєкту



Рисунок 2.8 – package.json файл проєкту

pnpm(далі pnpm) - це новий менеджер пакунків для JavaScript, створений на основі npm для спрощення процесу інсталяції пакетів у програмах вузла. PNPM є альтернативою NPM. Він дотримується тих же принципів, що й NPM, але має деякі додаткові функції, які роблять його потужнішим, ніж його попередник.

На рисунку 2.9 представлено візуалізацію роботи PNPM

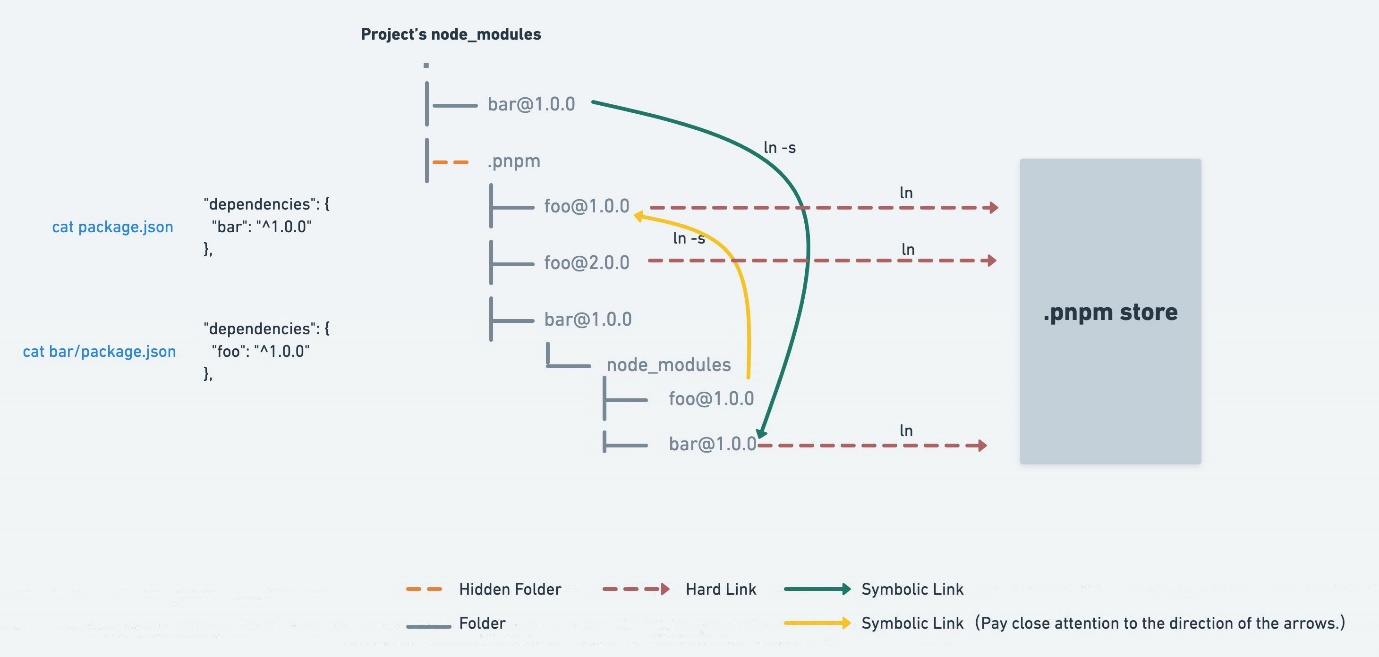


Рисунок 2.9 - Візуалізація роботи pnpm

Основною мотивацією створення pnpm була економія дискового простору та підвищення швидкості встановлення. Під час використання npm, якщо у розробника є 100 проектів, які використовують якусь залежність, буде 100 копій цієї залежності, збережених на диску. За допомогою pnpm залежність зберігатиметься в сховищі з адресою вмісту, тому:

- Якщо в проекті використовуються різні версії залежності, до сховища додаються лише ті файли, які відрізняються версія від версії. Наприклад, якщо він містить 100 файлів, а нова версія містить зміни лише в одному з цих файлів, pnpm update до сховища буде додано лише 1 новий файл, замість того, щоб клонувати всю залежність лише для єдиної зміни;

- Усі файли зберігаються в одному місці на диску. Коли пакети інстальовано, їхні файли жорстко пов’язуються з цього єдиного місця, не займаючи додаткового місця на диску. Це дає змогу поділитися залежностями однієї версії між проектами.

Як наслідок, економія великої кількості місця на диску, пропорційна кількості проектів і залежностей, а також набагато швидше встановлення.[11]

На рисунку 2.10 представлено логотип pnpm

Изображение выглядит как текст, седзи, здание, окно

Автоматически созданное описание

Рисунок 2.10 - Логотип pnpm

MongoDB — [документо-орієнтована система керування базами даних](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85" \o "Документо-орієнтована система керування базами даних) з [відкритим](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) [вихідним кодом](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%87%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4), яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і [реляційними системами](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D0%A1%D0%9A%D0%91%D0%94) котролю базами даних, функціональними і зручними у формуванні запитів. MongoDB написана на мові [C++](https://uk.wikipedia.org/wiki/C%2B%2B) і поширюється в рамках ліцензії [AGPLv](https://uk.wikipedia.org/wiki/AGPL)3.

MongoDB підтримує зберігання документів в [JSON](https://uk.wikipedia.org/wiki/JSON)-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання [великих бінарних об'єктів](https://uk.wikipedia.org/wiki/BLOB), підтримує [журналювання](https://uk.wikipedia.org/wiki/%D0%96%D1%83%D1%80%D0%BD%D0%B0%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Журналювання) операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми [Map/Reduce](https://uk.wikipedia.org/wiki/Map/Reduce" \o "Map/Reduce), підтримує [реплікацію](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BB%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F_(%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)) і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який з реплікацією даних можна побудувати горизонтально масштабований [кластер](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного [сервера](https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80) на кластер проводиться без зупинки роботи БД простим додаванням нових машин.

СКБД управляє наборами [JSON](https://uk.wikipedia.org/wiki/JSON)-подібних документів, що зберігаються в бінарному форматі [BSON](https://uk.wikipedia.org/wiki/BSON). Зберігання і пошук файлів в MongoDB відбувається завдяки викликам протоколу [GridFS](https://uk.wikipedia.org/w/index.php?title=GridFS&action=edit&redlink=1" \o "GridFS (ще не написана)). [12]

На рисунку 2.11 представлено логотип MongoDB

Изображение выглядит как текст, знак

Автоматически созданное описание

Рисунок 2.11 - Логотип mongoDB

Mongoose — це бібліотека моделювання об’єктних даних (ODM) для MongoDB і Node.js. Він керує зв’язками між даними, забезпечує перевірку схеми та використовується для перекладу між об’єктами в коді та представленням цих об’єктів у MongoDB.

На рисунку 2.12 представлено візуалізацію роботи NodeJS та MongoDB через Mongoose

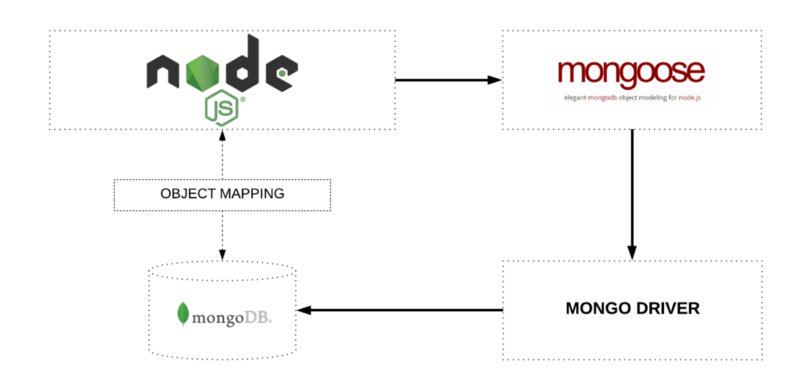


Рисунок 2.12 - Візуалізація роботи Node та Mongo через Mongoose

«Схема» Mongoose — це структура даних документа (або форма документа), яка виконується через прикладний рівень.

«Моделі» — це конструктори вищого порядку, які беруть схему та створюють екземпляр документа, еквівалентний записам у реляційній базі даних.

Модель Mongoose є оболонкою схеми Mongoose. Схема Mongoose визначає структуру документа, значення за замовчуванням, валідатори тощо, тоді як модель Mongoose забезпечує інтерфейс до бази даних для створення, запитів, оновлення, видалення записів тощо. [13]

Для передачі даних по World Wide Web (всесвітній паутині) в курсовій роботі використовуються та є найбільш популярними протоколи HTTP та HTTPS

Hypertext Transfer Protocol (далі - HTTP) - протокол передачі гіпертексту - це протокол передачі гіпертекстових документів, як-от HTML. За своєю суттю HTTP є прикладним протоколом і надає стандартні правила взаємодії трьох категорій - веб-браузерів (клієнтів), серверів і проксі-серверів.

Він створений для зв'язку між веб-браузерами та веб-серверами, хоча в принципі HTTP може використовуватись і для інших цілей. Протокол слідує класичної клієнт-серверної моделі, коли клієнт відкриває з'єднання для створення запиту, а потім чекає на відповідь.

HTTP - це протокол без збереження стану, тобто сервер не зберігає жодних даних (стан) між двома парами "запит-відповідь".

Сам HTTP побудований поверх набору протоколів TCP. HTTP використовує структурований текст гіпертексту, який встановлює логічний зв'язок між вузлами, які містять текст. Він також відомий як протокол без збереження стану, оскільки кожна команда виконується окремо, без використання посилання на попередню команду запуску. [14]

HTTPS розшифровується як Hyper Text Transfer Protocol Secure. Це дуже просунута та безпечна версія HTTP. Використовується порт №. 443 передачі даних. Це дозволяє створити безліч безпечних транзакцій шляхом шифрування зв'язку за допомогою SSL. За своєю суттю HTTPS – це комбінація протоколів SSL/TLS та HTTP. Він забезпечує зашифровану та безпечну ідентифікацію мережевого сервера.

HTTPS також дозволяє створювати безпечне зашифроване з'єднання між сервером та браузером. Він пропонує двонаправлену безпеку даних. Це допоможе захистити потенційно конфіденційну інформацію від крадіжки.[15]

Representational state transfer (далі - REST) – це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який зазвичай використовується для побудови веб-служб. Термін REST був запроваджений у 2000 році Роєм Філдінгом, одним із авторів HTTP-протоколу. Системи, що підтримують REST, називаються RESTful-системами.

REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, наприклад URL. Кожна URL, у свою чергу, має строго заданий формат.

Відсутність додаткових внутрішніх прошарків означає передачу даних у тому вигляді, як і самі дані. Тобто. ми не загортаємо дані в XML, як це робить SOAP та XML-RPC, не використовуємо AMF, як це робить Flash і т.д. Просто надаємо самі дані.

Кожна одиниця інформації однозначно визначається URL – це означає, що URL є первинним ключем для одиниці даних.

Як відбувається управління інформацією сервісу – це повністю й грунтується на протоколі передачі. Найбільш поширений протокол звичайно ж є HTTP. Так ось, для HTTP дія над даними визначається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма чотирма методами, так і лише за допомогою GET та POST. [16]

ExpressJS - backend фреймворк для створення web-застосунків, а також для створення RESTful APIs з Node.js, є безкоштовним та знаходиться в open-source. Його називають де-факто стандартною серверною структурою для Node.js

Express - мінімалістичний та гнучкий веб-фреймворк для програм Node.js, надає широкий набір функцій для мобільних та веб-додатків. Маючи у своєму розпорядженні безліч службових методів HTTP та проміжних обробників, створити надійний API можна швидко та легко. Express надає тонкий шар фундаментальних функцій веб-застосунків, які не заважають вам працювати з давно знайомими та улюбленими вами функціями Node.js. [17]

На рисунку 2.13 представлено логотип ExpressJS



Рисунок 2.13 - Логотип ExpressJS

Специфікація OpenAPI (OAS) визначає стандартний мовний інтерфейс для RESTful API, який дозволяє як людям, так і комп’ютерам відкривати та розуміти можливості сервісу без доступу до вихідного коду, документації або перевірки мережевого трафіку. При правильному визначенні споживач може розуміти віддалену службу та взаємодіяти з нею з мінімальною кількістю логіки реалізації.

Визначення OpenAPI потім може використовуватися інструментами створення документації для відображення API, інструментами створення коду для створення серверів і клієнтів різними мовами програмування, інструментами тестування та багатьма іншими випадками використання.

OpenAPI дозволяє описати весь API, зокрема:

- Доступні кінцеві точки (/користувачі) і операції на кожній кінцевій точці (GET /користувачі, POST /користувачі);

- Параметри операції Вхід і вихід для кожної операції;

- Методи автентифікації;

- Контактна інформація, ліцензія, умови використання та інша інформація.

Специфікації API можуть бути написані в YAML або JSON. Формат простий у вивченні та читається як людьми, так і машинами.

На рисунку 2.14 представлено логотип OpenAPI



Рисунок 2.14 – Логотип OpenAPI

Swagger — це набір інструментів із відкритим вихідним кодом, створених на основі специфікації OpenAPI, які можуть допомогти вам проектувати, створювати, документувати та використовувати REST API.

Використовувані інструменти Swagger:

- Swagger UI – рендерить визначення OpenAPI як інтерактивну документацію;

- Swagger Codegen – генерує серверні заглушки та клієнтські бібліотеки з визначення OpenAPI;

- Swagger Parser – автономна бібліотека для аналізу визначень OpenAPI;

- Swagger APIDom – забезпечує єдину уніфіковану структуру для опису API для різних мов опису та форматів серіалізації.

Здатність API описувати власну структуру є корінням усієї приголомшливості OpenAPI. Після написання специфікація OpenAPI та інструменти Swagger можуть сприяти подальшій розробці API різними способами

Swagger UI є одним із найпопулярніших інструментів для створення інтерактивної документації з вашого документа OpenAPI. Swagger UI створює інтерактивну консоль API, щоб користувачі могли швидко дізнатися про ваш API та експериментувати із запитами. Крім того, Swagger UI (який є активно керованим проектом з ліцензією Apache 2.0) підтримує останню версію специфікації OpenAPI (3.x) та інтегрується з іншими інструментами Swagger.[18]

На рисунку 2.15 представлено логотип Swagger

Изображение выглядит как текст, коллекция картинок

Автоматически созданное описание

Рисунок 2.15 - Логотип Swagger

На рисунку 2.16 представлено вміст yaml-файлу, який описує взаємодію користувача з запитами на взяття типів



Рисунок 2.16 - Опис взаємодії користувача з запитами на взяття типів в проєкті

Prettier — це самовпевнений форматувальник коду який бере код і передруковує його з нуля, враховуючи задані стилі.

Prettier забезпечує узгоджений **стиль** коду (тобто форматування коду, яке не впливає на абстрактне синтаксичне дерево) у всій кодовій базі, оскільки ігнорує оригінальний стиль, аналізуючи його та повторно друкуючи проаналізоване абстрактне синтаксичне дерево із власними правилами.

Безумовно, головна причина використання Prettier – не витрачати час на форматування, тим самим пришвидшити написання проєкту та зберегти якість.

Prettier - єдиний «гід зі стилю», який працює повністю автоматично.[19]

На рисунку 2.17 представлено логотип Prettier



Рисунок 2.17 – Логотип Prettier

ESLint — це проект із відкритим вихідним кодом, який допомагає знаходити та виправляти проблеми з кодом JavaScript. ESLint статично аналізує код, щоб швидко знаходити проблеми.

Багато проблем, які ESLint знаходить, виправляються автоматично. Виправлення ESLint враховують синтаксис, тому у не виникає помилок, які виникають через традиційні алгоритми пошуку та заміни.

ESLint дозволяє попередньо обробляти код, використовувати спеціальні аналізатори та створювати власні правила, які працюють разом із вбудованими правилами ESLint. Налаштовувати ESLint так, щоб він працював саме так, як це потрібно для проекту. [20]

На рисунку 2.18 представлено логотип ESLint



Рисунок 2.18 – Логотип ESLint

Nodemon — це інструмент, який допомагає розробляти програми на основі Node.js шляхом автоматичного перезапуску програми node, коли виявляються зміни файлів у каталозі. Nodemon не потребує жодних додаткових змін у коді чи методі розробки. Nodemon є заміною обгортки для NodeJS. [21]

Postman — це платформа API для створення та використання API. Postman спрощує кожен кроки життєвого циклу API та оптимізує розробку, щоб створювати кращі API швидше.

Postman починався просто як побічний проект для вирішення конкретної проблеми: Абхінав Астана, генеральний директор і співзасновник Postman, мав намір створити інструмент, який би спростив процес тестування API. Оскільки використання інструменту швидко поширилося, Абхінав найняв двох своїх колишніх колег, Анкіта Собті та Абхіджита Кейна, щоб вони допомогли йому створити Postman, Inc. Зараз Postman є провідною у світі платформою API, і троє засновників очолюють компанію донині. [22]

2.2 Використовуванні моделі даних

Для розробки даного сервісу використовується документо-орієнтована модель даних. На рисунку 2.19 представлений приклад документо-орієнтованої моделі

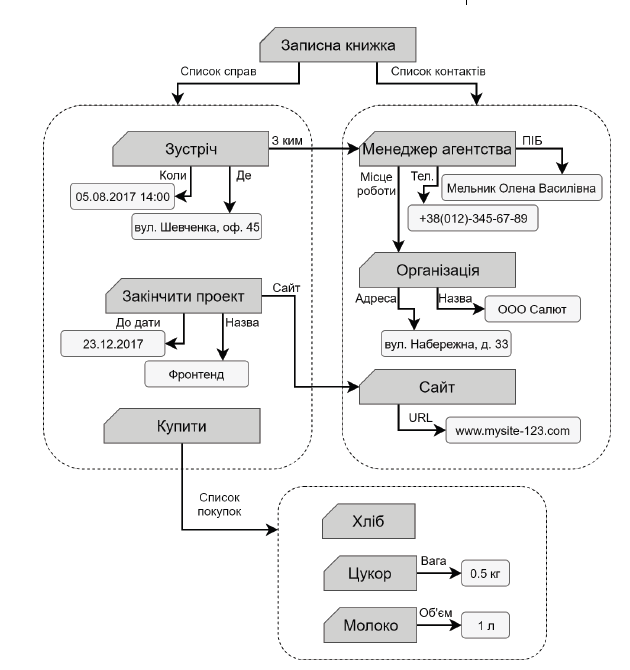


Рисунок 2.19 - Приклад документо-орієнтованої моделі даних

Перевагами документо-орієнтованої моделі бази даних є:

- Семантичке наповнення. Дані у документо-орієнтованій базі даних не можуть бути описані за допомогою єдиної схеми. Замість цього, дані описують самі себе, тобто кожен елемент даних має особисту семантичну схему;

- Структурна незалежність та незалежність даних. Характерною особливістю слабоструктурованих даних є те, що описова інформація, яка зазвичай виділяється в окрему схему, присутня в самих даних. Тому дані в документо-орієентованій моделі не залежать від будь-яких зовнішніх структур та схем;

- Відповідність стандартам опису даних. Наявність загальних стандартів для представлення даних, таких як XML, JSON, YAML;

- Простота проектування, реалізації, керування та використання;

- Висока масштабованість;

- Ключові переваги NoSQL баз в розподілених системах полягають в процедурах шаринга і реплікації.

Недоліками документо-орієнтованої моделі даних:

- Відсутність транзакцій. Відсутність єдиної схема та специфіка області застовування документо-орієнтованих баз даних поки що не дають змоги реалізувати повноцінний механізм транзакій;

- Відсутність стандартизованої мови маніпулювання даними;

- Недостатня цілісність даних. Задача котролю цілісності даних ніяк не регулюється системою контролю баз даних та повина бути реалізована у коді додатків, що працюють з цією базою даних.[23]

В роботі використовується MongoDB - крос-платформна, потужна, гнучка, документно-орієнтована база даних, що легко масштабується.

MongoDB складається з баз даних, які зберігають в собі колекції.

Колекція – іменована безліч об'єктів, при цьому один об'єкт належить лише одній колекції.

Об'єкт – сукупність властивостей, включаючи унікальний ідентифікатор \_id.

Властивість – сукупність назви і відповідного йому типу і значення.

Типи властивостей – string, number, float, array, object, buffer, date, boolean, null, objectId.

Підтримуються операції вибірки (count, group, MapReduce), вставки, зміни та видалення.

Зв'язків між об'єктами немає, об'єкти можуть лише зберігати інші об'єкти у властивостях.

Підтримуються як унікальні, так і композитні індекси. Індекси можна накладати на властивості вкладених об'єктів. Швидкість MongoDB гарно себе показує при виконані операції вставки даних, роблячи їх дуже швидко.[24]

Формат зберігання і формат передачі об'єктів по мережі один і той же, так що для вибірки якогось об'єкта треба всього лише знайти його позицію за індексом і повернути клієнту шматок файлу певної довжини.

В якості унікального ідентифікатора використовується 12-байтне унікальне число, що генерується на клієнті.

По-перше немає проблеми з синхронізацією реплік, тобто можна незалежно робити вставки на дві різні машини, і конфлікту не виникне. По-друге, не буде нісенітниці з переповненням цілого числа, ну і після перевтілення бази даних, пошукачі не будуть адресувати на нові статті за старими посиланнях.

В процесі проектування даного програмного продукту для побудови об'єктно-орієнтованої моделі було виділено наступні об’єктні множини: «Types», «Categories», «Producers», «Users», «AccountLevels», «Orders», «Reviews», «ProductsAllInfo», «Products»,

Аналіз визначених об’єктів і атрибутів дозволяє виділити сутності бази даних і побудувати її логічну схему.

Спроектована документо-орієнтована модель зображена на рисунку 2.20

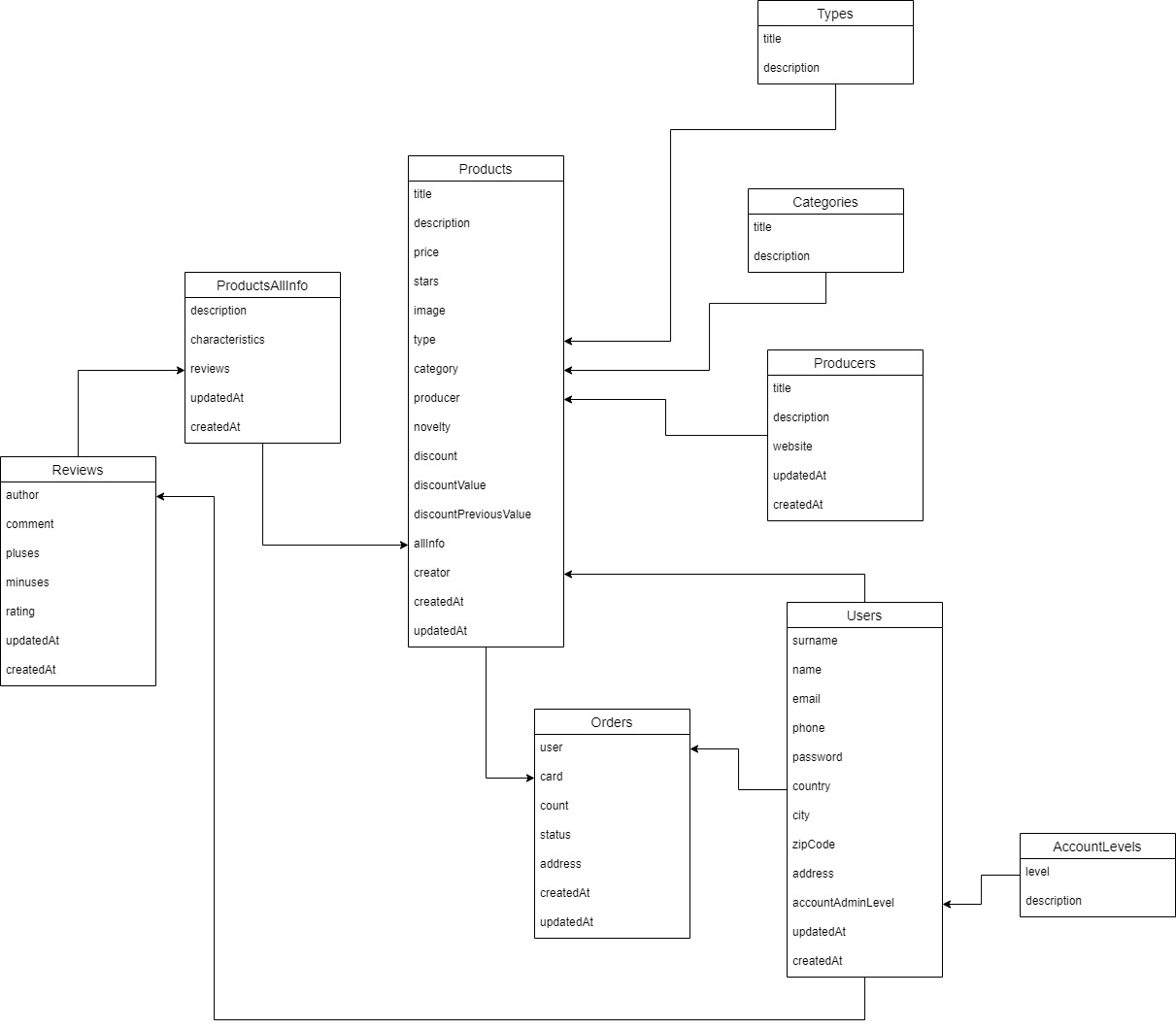


Рисунок 2.20 - Спроектована модель даних

Документ «Types» (\_id, title, description) зберігає інформацію про типи тварин. Опис документа «Types» представлений в таблиці 2.1, схема документа представлена на рисунку 2.21

Таблиця 2.1 – Опис документа «Types»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |

|  |  |  |
| --- | --- | --- |
| title | string | Заголовок |
| description | String | Опис |

Продовження таблиці 2.1 – Опис документа «Types»

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.21 – Схема документа «Types»

Документ «Categories» (\_id, title, description) зберігає інформацію про типи тварин. Опис документа «Categories» представлений в таблиці 2.2, схема документа представлена на рисунку 2.22

Таблиця 2.2 – Опис документа «Categories»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| title | string | Заголовок |
| description | String | Опис |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.22 – Схема документа «Categories»

Документ «Producers» (\_id, title, description, website) зберігає інформацію про типи тварин. Опис документа «Producers» представлений в таблиці 2.3, схема документа представлена на рисунку 2.23

Таблиця 2.3 – Опис документа «Producers»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| title | string | Заголовок |
| description | String | Опис |
| website | String | Посилання на вебсайт |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.23 – Схема документа «Producers»

Документ «AccountLevels» (\_id, level, description) зберігає інформацію про типи тварин. Опис документа «AccountLevels» представлений в таблиці 2.4, схема документа представлена на рисунку 2.24

Таблиця 2.4 – Опис документа «AccountLevels»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| level | number | Рівень користувача |
| description | String | Опис рівня |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.24 – Схема документа «AccountLevels»

Документ «Users» (\_id, surname, name, email, phone, password, country, city, zipCode, address, accountAdminLevel) зберігає інформацію про типи тварин. Опис документа «Users» представлений в таблиці 2.5, схема документа представлена на рисунку 2.25

Таблиця 2.5 – Опис документа «Users»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| surname | String | Призвіще |
| name | String | Ім’я |
| email | String | Почта |
| phone | String | Телефон |
| password | String | Пароль |
| country | String | Країна |
| zipCode | Number | Код міста |

Продовження таблиці 2.5 – Опис документа «Users»

|  |  |  |
| --- | --- | --- |
| address | String | Адреса |
| AccountAdminLevel | ObjectID | Посилання на документ «AccountsLevel» |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.25 – Схема документа «Users»

Документ «Reviews» (\_id, author, comment, pluses, minuses, rating) зберігає інформацію про типи тварин. Опис документа «Reviews» представлений в таблиці 2.6, схема документа представлена на рисунку 2.26

Таблиця 2.6 – Опис документа «Reviews»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальний ідентифікатор |
| author | ObjectID | Посилання на документ «Users» |
| comment | String | Коментарій |
| pluses | String | Плюси |

Продовження таблиці 2.6 – Опис документа «Reviews»

|  |  |  |
| --- | --- | --- |
| minuses | String | Мінуси |
| rating | Number | Рейтинг |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.26 – Схема документа «Reviews»

Документ «ProductsAllInfo» (\_id, description, characteristics, reviews) зберігає інформацію про типи тварин. Опис документа «ProductsAllInfo» представлений в таблиці 2.7, схема документа представлена на рисунку 2.27

Таблиця 2.7 – Опис документа «ProductsAllInfo»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальний ідентифікатор |
| description | String | Опис продукту |
| characteristics | String | Характеристики продукту |
| reviews | Array<ObjectID> | Масив посилань на документ «Reviews» |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.27 – Схема документа «ProductsAllInfo»

Документ «Product» (\_id, title, description, price, stars, image, type, category, producer, novelty, discount, discountValue, discountPreviousPrice, allInfo, creator) зберігає інформацію про типи тварин. Опис документа «Product» представлений в таблиці 2.8, схема документа представлена на рисунку 2.28

Таблиця 2.8 – Опис документа «Product»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальний ідентифікатор |
| title | Object | Назва продукту в якості об’єкту |
| title.en | String | Ангійська назва продукту |
| title.ua | String | Українська назва продукту |
| description | String | Короткий опис продукту в якості об’єкту |
| description.en | String | Ангійський короткий опис продукту |
| description.ua | String | Український короткий опис продукту |

Продовження таблиці 2.8 – Опис документа «Product»

|  |  |  |
| --- | --- | --- |
| price | Number | Ціна |
| stars | Number | Рейтинг |
| image | Buffer | Картинка в форматі base64 |
| type | ObjectID | Посилання на документ «Types» |
| category | ObjectID | Посилання на документ «Categories» |
| producer | ObjectID | Посилання на документ «Producers» |
| novelty | Boolean | Позначка нового товару |
| discount | Boolean | Позначка скидки на товар |
| discountValue | Number | Розмір скидки |
| discountPreviousPrice | Number | Ціна товару до скидки |
| allInfo | ObjectID | Посилання на документ «ProductsAllInfo» |
| creator | ObjectID | Посилання на документ «Users» |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.28– Схема документа «Product»

Документ «Orders» (\_id, title, description, price, stars, image, type, category, producer, novelty, discount, discountValue, discountPreviousPrice, allInfo, creator) зберігає інформацію про типи тварин. Опис документа «Orders» представлений в таблиці 2.9, схема документа представлена на рисунку 2.29

Таблиця 2.9 – Опис документа «Orders»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальний ідентифікатор |
| user | ObjectID | Посилання на документ «Users» |

Продовження таблиці 2.9 - Опис документа «Orders»

|  |  |  |
| --- | --- | --- |
| card | ObjectID | Посилання на документ «Product» |
| count | Number | Кількість товару |
| productPrice | Number | Ціна за одиницю |
| status | String | Статус замовлення |
| address | String | Адреса |

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.29 – Схема документа «Orders»

2.3 Методика роботи користувача с системою

2.3.1 Керівництво програміста

Основним призначення програми є надання клієнтській частині потрібну їй інформацію за допомогою запитів.

Оскільки вся програма розбита на модулі внесення подальших змін буде максимально швидким та мати мінімальну кількість перешкод

Основними модулями програми є

- Моделі - схеми та моделі mongoose, які описують дані MongoDB;

- Роути – уловлювачі запитів клієнтської частини, які потім викликають потрібні контролери;

- Контролери – Головні обробники програми, які за допомогою mongoose витягують дані з MongoDB, обробляють дані враховуючи параметри запиту та повертають результат;

- Хелпери – Допоміжні функції програми;

- Сваггер – Опис роутів для створення візульного інтерфейсу.

Мінімальні вимоги до середовища функціонування системи:

- CPU: 1 vCPU Type-A;

- RAM: 1 Гибибайт;

- SSD: 2 Гибибайт;

- Інтернет трафік на місяць: 3 Тибибайт;

- Операційна система: Ubuntu, Debian, Arch;

- Встановлена база даних: MongoDB 4.4;

- Програмне оточення: NodeJS.

2.3.2 Керівництво оператора

Перед запуском сервісу треба створити змінні оточення:

- Змінна PORT – Відповідає за порт на якому буде працювати сервіс;

- Змінна MONGO\_USERNAME – Відповідає за ім’я користувача, яке буде використане при авторизації в MongoDB ;

- Змінна MONGO\_PASSWORD – Відповідає за пароль користувача, який буде використаний при авторизації в MongoDB;

- Змінна MONGO\_DATABASE – Відповідає за ім’я бази даних.

Для запуску сервісу треба виконати команду: node index.js

Після запуску, в браузері потрібно відкрити сторінку сервісу за посиланням <http://localhost:8000/api-docs>

Вигляд сторінки сервісу продемонстрований на рисунку 2.30

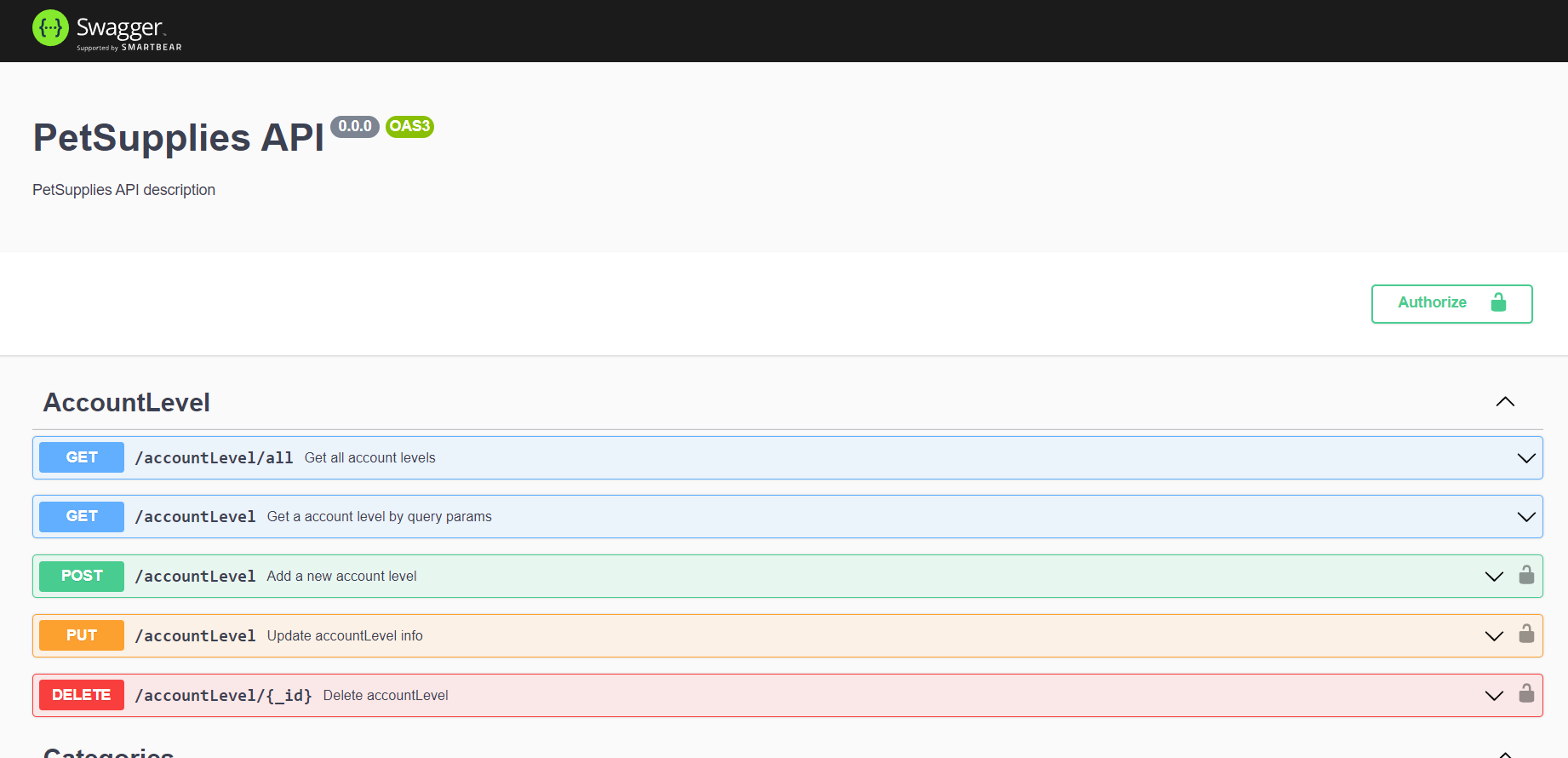


Рисунок 2.30 – Вигляд сторінки сервісу API

Всі можливості сервісу поділені на категорії, які можна згортати при необхідності  
На прикладі категорії AccountLevel розглянемо основні можливості сервісу. Як видно з рисунку 2.30 сервіс надає можливість робити операції з даними:

- взяття(GET);

- створення або додавання(POST);

- зміна(PUT);

- видалення(DELETE)

На операціях додавання, змінювання та видалення на рисунку 2.30 є замок, який говорить про те, що для виконання цієї операції користувач повинен бути авторизований в системі та мати необхідний рівень доступу

На рисунку 2.30 всі операції знаходяться в згорнутому вигляді

Для виконання операції потрібно розгорнути вкладку з нею, після чого на екрані користувача з’явиться схема запросу, яка включає в себе параметри або ж тіло запиту та відповіді(Responses) серверу.

Приклад тіла запиту представлений на рисунку 2.31

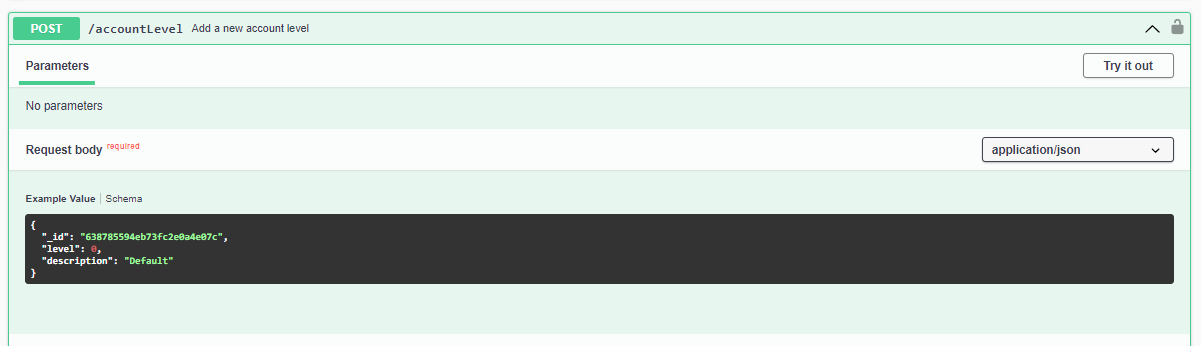


Рисунок 2.31 – Приклад тіла запиту

В відповідях запиту можливо переглянути що саме поверне сервер користувачу після виконання запиту.

При успішному запиті сервер поверне відповідь з кодом 200 та відповідний до прикладу результат.

При помилкі на сервері, або в параметрах запиту повернеться 500 помилка з відповідним до прикладу результатом

При помилці авторизації сервер поверне 401 помилку та повідомлення

Приклад відповідей серверу представлені на рисунку 2.32

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.32 – Приклад відповідей запиту

Для виконання запиту потрібно натиснути на кнопку «Try it out», ввести необхідні дані та натиснути кнопку «Execute»

Після виконання запиту в секції відповідей(Responses) з’явиться curl(команда, яку виконала клієнтська частина), Request url (посилання, на яке був виконаний запит), а нижче код, тіло та заголовки відповіді серверу

Приклад виконаного запиту представлений на рисунку 2.33

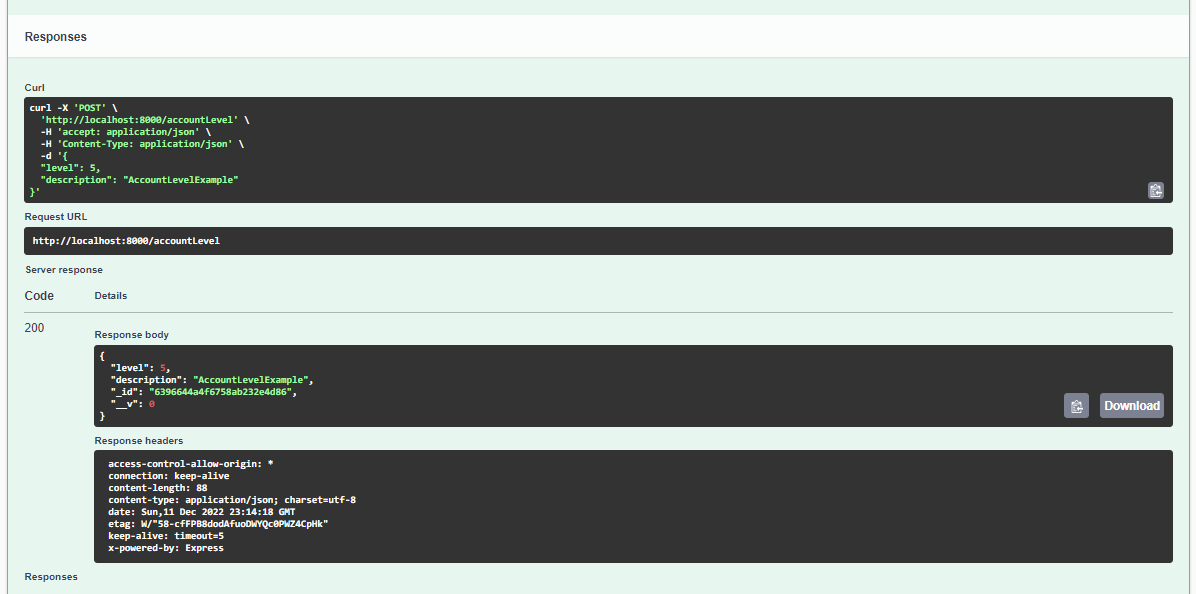


Рисунок 2.33 – Приклад виконаного запиту

ВИСНОВКИ

В результаті виконання курсової роботи була створена система онлайн продаж. Дана система здійснює продаж аксесуарів для домашніх тварин.

Система дозволяє адміністраторам та користувачам здійснювати швидкий та детальний перегляд продукції.

Сервіс включає в себе функцію реєстрації користувача, залишення відгуків та здійснення замовлення, додавання товару.

Створений сервіс має наступні функції:

- Вибір необхідних товарів с фільтрацією по будь-якій характеристиці;

- Створення аккаунтів, зміни інформації про них;

- Додавання та видалення відгуків про товар;

- Створення замовлень, перегляд інформації про них;

- Зручне та швидке маніпулювання адміністратором даними про товари(додавання, видалення, змінення).

Реалізована 2-х рівнева система адміністрації: модератори та адміністратори

Враховані та виконані вимоги до надійності та безпеки:

- Тільки адміністратор має право додавання, змінення чи видалення товару;

- Модератор має право видаляти відгуки про товари та змінювати статуси замовлення;

- Адміністратор має право змінювати інформацію про постачальників, категорії та типи товарів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 http://deep.kiev.ua/attachments/093\_2016d\_Kravchuk.pdf

2 <https://code.visualstudio.com/#meet-intellisense>

3 <https://git-scm.com/>

4 <https://www.postman.com/>

5 <https://swagger.io/>

6 https://developer.mozilla.org/ru/docs/conflicting/Web/JavaScript

7 <https://www.typescriptlang.org/>

8 https://nodejs.org/en/about/

9 https://github.com/nvm-sh/nvm

10 https://www.npmjs.com/about

11 https://pnpm.io/motivation

12 https://aws.amazon.com/ru/documentdb/what-is-mongodb

13 <https://www.freecodecamp.org/news/introduction-to-mongoose-> for-mongodb-d2a7aa593c57

14 https://developer.mozilla.org/ru/docs/Web/HTTP

15 <https://www.ukraine.com.ua/blog/seo-optimization/chto-takoe->https-i-zachem-on-nuzhen-kazhdomu-sajtu.html

16 https://habr.com/ru/post/38730/

17 https://expressjs.com/

18 https://swagger.io/docs/specification/about/

19 https://prettier.io/docs/en/index.html

20 https://dev.to/shivambmgupta/eslint-what-why-when-how-5f1d

21 <https://www.npmjs.com/package/nodemon>

22 https://www.postman.com/

23 І. О. Бардус, М. І. Лазарєв, А. О. Ніценко БАЗИ ДАНИХ У СХЕМАХ (НА ОСНОВІ ФУНДАМЕНТАЛІЗОВАНОГО ПІДХОДУ)]

24 <https://www.geeksforgeeks.org/what-is-mongodb-working-and->features/

ДОДАТОК А

**(обов’язковий)**

**Текст програми**

src\index.ts

import dotenv from'dotenv';import express from'express';import{connect,mongo}from'mongoose';import cors from'cors';import swaggerUI from'swagger-ui-express';import swaggerJsDoc from'swagger-jsdoc';import{HomeRouter,AccountLevelRouter,CategoriesRouter,ProducerRouter,TypesRouter,OrdersRouter,}from'./routes';import mongoose from'mongoose';import bodyParser from'body-parser';console.log(new mongoose.Types.ObjectId('638387eb6f33fac689be5e82'));dotenv.config();connect(`mongodb+srv:).then(()=>{console.log('[Server]: DB connected')}).catch(err=>{console.error('[Server]: DB error connect',err)});const app=express();const port=process.env.PORT;app.use(cors());const options={failOnErrors:true,definition:{openapi:'3.0.0',info:{title:'PetSupplies API',version:'0.0.0',description:'PetSupplies API description',servers:[`http:},},apis:['src/routes/\*.ts','src/swagger/\*.yaml'],};const openapiSpecification=swaggerJsDoc(options);app.use('/api-docs',swaggerUI.serve,swaggerUI.setup(openapiSpecification));app.use(bodyParser.json());app.use(bodyParser.urlencoded({extended:true}));app.use(HomeRouter);app.use(CategoriesRouter);app.use(TypesRouter);app.use(ProducerRouter);app.use(AccountLevelRouter);app.use(OrdersRouter);app.listen(port,()=>{console.log(`[Server]:Server is running at http:})

src\controllers\AccountLevel\createNewAccountLevel.ts

import mongoose,{Types}from'mongoose';import{Request,Response}from'express';import{AccountLevel}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckID}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;level:number;description:string}};const getQueryParams=(body:Request['body']):ErrorMessageAnswer|getQueryParamsOk=>{const{\_id,level,description}=body;let resultId:undefined|Types.ObjectId=undefined;if(\_id!==undefined){if(typeof \_id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(\_id)){return createErrorMessage('Id string must be ObjectId isValid')};resultId=new Types.ObjectId(\_id)};if(typeof description!=='string'){return createErrorMessage('Description must be string')};let resultLevel:number|undefined=undefined;resultLevel=Number(level);if(resultLevel===undefined||resultLevel===NaN){return createErrorMessage('Level must be number | undefined')};return{status:Status.OK,value:{id:resultId,level:resultLevel,description},}};export const createNewAccountLevel=async(req:Request,res:Response)=>{const params=getQueryParams(req.body);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const newAccountLevelParams=getValidParamsWithCheckID(params.value);new AccountLevel(newAccountLevelParams).save((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\AccountLevel\deleteAccountLevel.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{AccountLevel}from'../../models';import{Codes}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';export const deleteAccountLevel=(req:Request,res:Response)=>{const id=req.params.\_id;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const idAsObjectId=new Types.ObjectId(id);AccountLevel.deleteOne({\_id:idAsObjectId},error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(createSuccessMessage(`AccountLevel ${id}successfully deleted`))})}

src\controllers\AccountLevel\getAccountLevel.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{AccountLevel}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;level:number|undefined;description:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,level,description}=req.query;let resultId:undefined|Types.ObjectId=undefined;if(id!==undefined){if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id)};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};let resultLevel:number|undefined=undefined;if(level!==undefined){if(Number(level)===NaN){return createErrorMessage('Level must be number | undefined')};resultLevel=Number(level)};return{status:Status.OK,value:{id:resultId,level:resultLevel,description},}};export const getAccountLevel=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const findParams=getValidParamsWithCheckAll(params.value);AccountLevel.findOne(findParams,{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\AccountLevel\getAllAccountLevels.ts

import{Request,Response}from'express';import{AccountLevel}from'../../models';import{Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';export const getAllAccountLevels=(req:Request,res:Response)=>{AccountLevel.find({},{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\AccountLevel\index.ts

export{createNewAccountLevel}from'./createNewAccountLevel';export{deleteAccountLevel}from'./deleteAccountLevel';export{getAccountLevel}from'./getAccountLevel';export{updateAccountLevel}from'./updateAccountLevel';export{getAllAccountLevels}from''

src\controllers\AccountLevel\updateAccountLevel.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{AccountLevel}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId;level:number|undefined;description:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,level,description}=req.query;if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const resultId=new Types.ObjectId(id);if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};let resultLevel:number|undefined=undefined;if(level!==undefined){if(Number(level)===NaN){return createErrorMessage('Level must be number | undefined')};resultLevel=Number(level)};return{status:Status.OK,value:{id:resultId,level:resultLevel,description},}};export const updateAccountLevel=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,level,description}=params.value;const updatedParams=getValidParamsWithCheckAll({id,level,description,});AccountLevel.updateOne({\_id:id},updatedParams,null,error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};AccountLevel.findOne({\_id:id}).exec((findError,findResult)=>{if(findError){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+findError.name))};return res.status(Codes.OK).json(findResult)})})}

src\controllers\Category\createNewCategory.ts

import mongoose,{Types}from'mongoose';import{Request,Response}from'express';import{Category,User}from'../../models';import{Status,SuccessMessageAnswer,ErrorMessageAnswer,Codes,}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';import{getValidParamsWithCheckID}from'../../helpers/getValidParams';const Buffer=mongoose.Types.Buffer;type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;title:string;description:string}};const getQueryParams=(body:Request['body']):ErrorMessageAnswer|getQueryParamsOk=>{const{\_id,title,description}=body;let resultId:undefined|Types.ObjectId=undefined;if(\_id!==undefined){if(typeof \_id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(\_id)){return createErrorMessage('Id string must be ObjectId isValid')};resultId=new Types.ObjectId(\_id)};if(typeof title!=='string'){return createErrorMessage('Title must be string')};if(typeof description!=='string'){return createErrorMessage('Description must be string')};return{status:Status.OK,value:{id:resultId,title,description},}};export const createNewCategory=async(req:Request,res:Response)=>{const params=getQueryParams(req.body);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description}=params.value;const newCategoryParams=getValidParamsWithCheckID({id,title,description,});new Category(newCategoryParams).save((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Category\deleteCategory.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Category,User}from'../../models';import{Codes}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';export const deleteCategory=(req:Request,res:Response)=>{const id=req.params.\_id;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const idAsObjectId=new Types.ObjectId(id);Category.deleteOne({\_id:idAsObjectId},error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(createSuccessMessage(`Category ${id}successfully deleted`))})}

src\controllers\Category\getAllCategories.ts

import{Request,Response}from'express';import{Category}from'../../models';import{Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';export const getAllCategories=(req:Request,res:Response)=>{Category.find({},{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Category\getCategories.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Category,User}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;title:string|undefined;description:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,title,description}=req.query;let resultId:undefined|Types.ObjectId=undefined;if(id!==undefined){if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id)};if(title!==undefined){if(typeof title!=='string'){return createErrorMessage('Title must be string')}};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};return{status:Status.OK,value:{id:resultId,title,description},}};export const getCategories=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description}=params.value;const findParams=getValidParamsWithCheckAll({id,title,description});Category.find(findParams,{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Category\index.ts

export{createNewCategory}from'./createNewCategory';export{getCategories}from'./getCategories';export{deleteCategory}from'./deleteCategory';export{updateCategory}from'./updateCategory';export{getAllCategories}from''

src\controllers\Category\updateCategory.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Category,User}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId;title:string|undefined;description:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,title,description}=req.query;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const resultId=new Types.ObjectId(id);if(title!==undefined){if(typeof title!=='string'){return createErrorMessage('Title must be string')}};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};return{status:Status.OK,value:{id:resultId,title,description},}};export const updateCategory=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description}=params.value;const updatedParams=getValidParamsWithCheckAll({id,title,description});Category.updateOne({\_id:id},updatedParams,null,(error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};Category.findOne({\_id:id}).exec((findError,findResult)=>{if(findError){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+findError.name))};return res.status(Codes.OK).json(findResult)})})}

src\controllers\Order\createNewOrder.ts

import mongoose,{Types}from'mongoose';import{Request,Response}from'express';import{Order,Product,User}from'../../models';import{Status,SuccessMessageAnswer,ErrorMessageAnswer,Codes,}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';import{getValidParamsWithCheckID}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;user:Types.ObjectId;product:Types.ObjectId;count:number;productPrice:number|undefined;address:string|undefined;status:string|undefined}};const getQueryParams=(body:Request['body']):ErrorMessageAnswer|getQueryParamsOk=>{const{id,user,product,count,productPrice,address,status}=body;let resultId:undefined|Types.ObjectId=undefined;let resultUser:undefined|Types.ObjectId=undefined;let resultProduct:undefined|Types.ObjectId=undefined;if(id!==undefined){if(typeof id!=='string'){return createErrorMessage('\_id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('\_id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id)};if(typeof user!=='string'){return createErrorMessage('user must be undefined | string')};if(!Types.ObjectId.isValid(user)){return createErrorMessage('user string must be ObjectId.isValid')};resultUser=new Types.ObjectId(user);if(typeof product!=='string'){return createErrorMessage('product must be undefined | string')};if(!Types.ObjectId.isValid(product)){return createErrorMessage('product string must be ObjectId.isValid')};resultProduct=new Types.ObjectId(product);let resultCount:undefined|number=undefined;let resultProductPrice:undefined|number=undefined;if(Number(count)===NaN){return createErrorMessage('address must be string')};resultCount=Number(count);if(productPrice!==undefined){if(Number(productPrice)===NaN){return createErrorMessage('address must be string')};resultProductPrice=Number(productPrice)};if(address!==undefined){if(typeof address!=='string'){return createErrorMessage('address must be string')}};if(status!==undefined){if(typeof status!=='string'){return createErrorMessage('status must be string')}};return{status:Status.OK,value:{id:resultId,user:resultUser,product:resultProduct,count:resultCount,productPrice:resultProductPrice,address,status,},}};const checkProductIsExists=async(id:Types.ObjectId):Promise<boolean>=>{const productResult=await Product.findOne({\_id:id});return productResult!==null};const checkUserIsExists=async(id:Types.ObjectId):Promise<boolean>=>{const result=await User.findOne({\_id:id});return result!==null};export const createNewOrder=async(req:Request,res:Response)=>{const params=getQueryParams(req.body);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const newOrderParams=getValidParamsWithCheckID<getQueryParamsOk['value']>(params.value);if((await checkProductIsExists(newOrderParams.product))===false){return res.status(Codes.ERROR).json(createErrorMessage('Product id is not valid'))};if((await checkUserIsExists(newOrderParams.user))===false){return res.status(Codes.ERROR).json(createErrorMessage('User id is not valid'))};new Order(newOrderParams).save((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Order\deleteOrder.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Order}from'../../models';import{Codes}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';export const deleteOrder=(req:Request,res:Response)=>{const id=req.params.\_id;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const idAsObjectId=new Types.ObjectId(id);Order.deleteOne({\_id:idAsObjectId},error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(createSuccessMessage(`Order ${id}successfully deleted`))})}

src\controllers\Order\getAllOrders.ts

import{Request,Response}from'express';import{Order}from'../../models';import{Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';export const getAllOrders=(req:Request,res:Response)=>{Order.find({},{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Order\getOrder.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Order}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;user:Types.ObjectId|undefined;product:Types.ObjectId|undefined;count:number|undefined;productPrice:number|undefined;address:string|undefined;status:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,user,product,count,productPrice,address,status}=req.query;let resultId:undefined|Types.ObjectId=undefined;let resultUser:undefined|Types.ObjectId=undefined;let resultProduct:undefined|Types.ObjectId=undefined;if(id!==undefined){if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id)};if(user!==undefined){if(typeof user!=='string'){return createErrorMessage('user must be undefined | string')};if(!Types.ObjectId.isValid(user)){return createErrorMessage('user string must be ObjectId.isValid')};resultUser=new Types.ObjectId(user)};if(product!==undefined){if(typeof product!=='string'){return createErrorMessage('product must be undefined | string')};if(!Types.ObjectId.isValid(product)){return createErrorMessage('product string must be ObjectId.isValid')};resultProduct=new Types.ObjectId(product)};let resultCount:undefined|number=undefined;let resultProductPrice:undefined|number=undefined;if(count!==undefined){if(Number(count)===NaN){return createErrorMessage('address must be string')};resultCount=Number(count)};if(productPrice!==undefined){if(Number(productPrice)===NaN){return createErrorMessage('address must be string')};resultProductPrice=Number(productPrice)};if(address!==undefined){if(typeof address!=='string'){return createErrorMessage('address must be string')}};if(status!==undefined){if(typeof status!=='string'){return createErrorMessage('status must be string')}};return{status:Status.OK,value:{id:resultId,user:resultUser,product:resultProduct,count:resultCount,productPrice:resultProductPrice,address,status,},}};export const getOrder=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const findParams=getValidParamsWithCheckAll(params.value);Order.find(findParams,{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Order\index.ts

export{createNewOrder}from'./createNewOrder';export{getOrder}from'./getOrder';export{deleteOrder}from'./deleteOrder';export{updateOrder}from'./updateOrder';export{getAllOrders}from''

src\controllers\Order\updateOrder.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Order}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId;user:Types.ObjectId|undefined;product:Types.ObjectId|undefined;count:number|undefined;productPrice:number|undefined;address:string|undefined;status:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,user,product,count,productPrice,address,status}=req.query;let resultId:undefined|Types.ObjectId=undefined;let resultUser:undefined|Types.ObjectId=undefined;let resultProduct:undefined|Types.ObjectId=undefined;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id);if(user!==undefined){if(typeof user!=='string'){return createErrorMessage('user must be undefined | string')};if(!Types.ObjectId.isValid(user)){return createErrorMessage('user string must be ObjectId.isValid')};resultUser=new Types.ObjectId(user)};if(product!==undefined){if(typeof product!=='string'){return createErrorMessage('product must be undefined | string')};if(!Types.ObjectId.isValid(product)){return createErrorMessage('product string must be ObjectId.isValid')};resultProduct=new Types.ObjectId(product)};let resultCount:undefined|number=undefined;let resultProductPrice:undefined|number=undefined;if(count!==undefined){if(Number(count)===NaN){return createErrorMessage('address must be string')};resultCount=Number(count)};if(productPrice!==undefined){if(Number(productPrice)===NaN){return createErrorMessage('address must be string')};resultProductPrice=Number(productPrice)};if(address!==undefined){if(typeof address!=='string'){return createErrorMessage('address must be string')}};if(status!==undefined){if(typeof status!=='string'){return createErrorMessage('status must be string')}};return{status:Status.OK,value:{id:resultId,user:resultUser,product:resultProduct,count:resultCount,productPrice:resultProductPrice,address,status,},}};export const updateOrder=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id}=params.value;const updatedParams=getValidParamsWithCheckAll(params.value);Order.updateOne({\_id:id},updatedParams,null,error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};Order.findOne({\_id:id}).exec((findError,findResult)=>{if(findError){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+findError.name))};return res.status(Codes.OK).json(findResult)})})}

src\controllers\Producer\createNewProducer.ts

import mongoose,{Types}from'mongoose';import{Request,Response}from'express';import{Producer}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckID}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;title:string;description:string;website:string}};const getQueryParams=(body:Request['body']):ErrorMessageAnswer|getQueryParamsOk=>{const{\_id,title,description,website}=body;let resultId:undefined|Types.ObjectId=undefined;if(\_id!==undefined){if(typeof \_id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(\_id)){return createErrorMessage('Id string must be ObjectId isValid')};resultId=new Types.ObjectId(\_id)};if(typeof title!=='string'){return createErrorMessage('Title must be string')};if(typeof description!=='string'){return createErrorMessage('Description must be string')};if(typeof website!=='string'){return createErrorMessage('Website must be string')};return{status:Status.OK,value:{id:resultId,title,description,website},}};export const createNewProducer=async(req:Request,res:Response)=>{const params=getQueryParams(req.body);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description,website}=params.value;const newProducerParams=getValidParamsWithCheckID({id,title,description,website,});new Producer(newProducerParams).save((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Producer\deleteProducer.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Producer}from'../../models';import{Codes}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';export const deleteProducer=(req:Request,res:Response)=>{const id=req.params.\_id;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const idAsObjectId=new Types.ObjectId(id);Producer.deleteOne({\_id:idAsObjectId},error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(createSuccessMessage(`Producer ${id}successfully deleted`))})}

src\controllers\Producer\getAllProducers.ts

import{Request,Response}from'express';import{Producer}from'../../models';import{Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';export const getAllProducers=(req:Request,res:Response)=>{Producer.find({},{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Producer\getProducer.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Producer}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;title:string|undefined;description:string|undefined;website:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,title,description,website}=req.query;let resultId:undefined|Types.ObjectId=undefined;if(id!==undefined){if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id)};if(title!==undefined){if(typeof title!=='string'){return createErrorMessage('Title must be string')}};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};if(website!==undefined){if(typeof website!=='string'){return createErrorMessage('Website must be string')}};return{status:Status.OK,value:{id:resultId,title,description,website},}};export const getProducer=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description,website}=params.value;const findParams=getValidParamsWithCheckAll({id,title,description,website,});Producer.findOne(findParams,{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Producer\index.ts

export{createNewProducer}from'./createNewProducer';export{deleteProducer}from'./deleteProducer';export{getProducer}from'./getProducer';export{updateProducer}from'./updateProducer';export{getAllProducers}from''

src\controllers\Producer\updateProducer.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Producer}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId;title:string|undefined;description:string|undefined;website:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,title,description,website}=req.query;if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const resultId=new Types.ObjectId(id);if(title!==undefined){if(typeof title!=='string'){return createErrorMessage('Title must be string')}};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};if(website!==undefined){if(typeof website!=='string'){return createErrorMessage('Website must be string')}};return{status:Status.OK,value:{id:resultId,title,description,website},}};export const updateProducer=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description,website}=params.value;const updatedParams=getValidParamsWithCheckAll({id,title,description,website,});Producer.updateOne({\_id:id},updatedParams,null,error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};Producer.findOne({\_id:id}).exec((findError,findResult)=>{if(findError){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+findError.name))};return res.status(Codes.OK).json(findResult)})})}

src\controllers\Product\ProductPost.ts

import mongoose from'mongoose';import{Category,Producer,ProductsAllInfo,Type,User}from'../../models';export const productPostController=async(data:any)=>{console.log(data);const notValidateFields={title:{ua:data.titleUA,en:data.titleEN,},description:{ua:data.descriptionUA,en:data.descriptionEN,},price:data.price,stars:data.stars,novelty:data.novelty,discount:data.discount,discountValue:data.discountValue,discountPreviousValue:data.discountPreviousValue,};const queryType=data.type;const categoryType=data.category;const producerType=data.producer;const creatorQuery=data.creator;const typeValue=await typeValidate(queryType);const categoryValue=await categoryValidate(categoryType);const producerValue=await producerValidate(producerType);const allInfoValue=await allInfoCreate();const creatorValue=await creatorValidate(creatorQuery);if(typeValue.status==='error'){return typeValue};if(categoryValue.status==='error'){return categoryValue};if(producerValue.status==='error'){return producerValue};if(creatorValue.status==='error'){return creatorValue};return{...notValidateFields,type:typeValue.value,category:categoryValue.value,producer:producerValue.value,creator:creatorValue.value,allInfo:allInfoValue.value,}};const typeValidate=async(type:any)=>{if(typeof type!=='string'){return{status:'error',message:'Invalid type',}};if(type.includes(',')){const typeArray=type.split(',');const equals=typeArray.map(item=>{return{title:item}});const result=await Type.find({$or:equals});if(result.length!==typeArray.length){return{status:'error',message:'Invalid type',}};return{status:'ok',value:result.map(item=>item.\_id),}};const result=await Type.findOne({title:type});if(result===null){return{status:'error',message:'Invalid type',}};return{status:'ok',value:result.\_id,}};const categoryValidate=async(category:any)=>{if(typeof category!=='string'){return{status:'error',message:'Invalid category',}};if(category.includes(',')){const categoryArray=category.split(',');const equals=categoryArray.map(item=>{return{title:item}});const result=await Category.find({$or:equals});if(result.length!==categoryArray.length){return{status:'error',message:'Invalid category',}};return{status:'ok',value:result.map(item=>item.\_id),}};const result=await Category.findOne({title:category});if(result===null){return{status:'error',message:'Invalid category',}};return{status:'ok',value:result.\_id,}};const producerValidate=async(value:any)=>{if(typeof value!=='string'){return{status:'error',message:'Invalid producer',}};const result=await Producer.findOne({title:value});if(result===null){return{status:'error',message:'Invalid producer',}};return{status:'ok',value:result.\_id,}};const allInfoCreate=async()=>{const allInfo=await new ProductsAllInfo({description:'description',characteristics:'characteristics',reviews:[],}).save();return{status:'ok',value:allInfo.\_id,}};const creatorValidate=async(value:any)=>{if(typeof value!=='string'){return{status:'error',message:'Invalid creator',}};const result=await User.findOne({\_id:new mongoose.Types.ObjectId(value)});if(result===null){return{status:'error',message:'Invalid creator',}};return{status:'ok',value:result.\_id,}}

src\controllers\Type\createNewType.ts

import mongoose,{Types}from'mongoose';import{Request,Response}from'express';import{Type}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckID}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;title:string;description:string}};const getQueryParams=(body:Request['body']):ErrorMessageAnswer|getQueryParamsOk=>{const{\_id,title,description}=body;let resultId:undefined|Types.ObjectId=undefined;if(\_id!==undefined){if(typeof \_id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(\_id)){return createErrorMessage('Id string must be ObjectId isValid')};resultId=new Types.ObjectId(\_id)};if(typeof title!=='string'){return createErrorMessage('Title must be string')};if(typeof description!=='string'){return createErrorMessage('Description must be string')};return{status:Status.OK,value:{id:resultId,title,description},}};export const createNewType=async(req:Request,res:Response)=>{const params=getQueryParams(req.body);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description}=params.value;const newCategoryParams=getValidParamsWithCheckID({id,title,description,});new Type(newCategoryParams).save((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Type\deleteType.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Type}from'../../models';import{Codes}from'../../types';import{createErrorMessage,createSuccessMessage,}from'../../helpers/messages';export const deleteType=(req:Request,res:Response)=>{const id=req.params.\_id;if(typeof id!=='string'){return createErrorMessage('Id must be string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const idAsObjectId=new Types.ObjectId(id);Type.deleteOne({\_id:idAsObjectId},error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(createSuccessMessage(`Type ${id}successfully deleted`))})}

src\controllers\Type\getAllTypes.ts

import{Request,Response}from'express';import{Type}from'../../models';import{Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';export const getAllTypes=(req:Request,res:Response)=>{Type.find({},{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Type\getType.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Type}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId|undefined;title:string|undefined;description:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,title,description}=req.query;let resultId:undefined|Types.ObjectId=undefined;if(id!==undefined){if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};resultId=new Types.ObjectId(id)};if(title!==undefined){if(typeof title!=='string'){return createErrorMessage('Title must be string')}};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};return{status:Status.OK,value:{id:resultId,title,description},}};export const getType=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description}=params.value;const findParams=getValidParamsWithCheckAll({id,title,description});Type.findOne(findParams,{\_\_v:0}).exec((error,result)=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};return res.status(Codes.OK).json(result)})}

src\controllers\Type\index.ts

export{createNewType}from'./createNewType';export{getType}from'./getType';export{deleteType}from'./deleteType';export{updateType}from'./updateType';export{getAllTypes}from''

src\controllers\Type\updateType.ts

import{Types}from'mongoose';import{Request,Response}from'express';import{Type}from'../../models';import{Status,ErrorMessageAnswer,Codes}from'../../types';import{createErrorMessage}from'../../helpers/messages';import{getValidParamsWithCheckAll}from'../../helpers/getValidParams';type getQueryParamsOk={status:Status.OK;value:{id:Types.ObjectId;title:string|undefined;description:string|undefined}};const getQueryParams=(req:Request):ErrorMessageAnswer|getQueryParamsOk=>{const{id,title,description}=req.query;if(typeof id!=='string'){return createErrorMessage('Id must be undefined | string')};if(!Types.ObjectId.isValid(id)){return createErrorMessage('Id string must be ObjectId.isValid')};const resultId=new Types.ObjectId(id);if(title!==undefined){if(typeof title!=='string'){return createErrorMessage('Title must be string')}};if(description!==undefined){if(typeof description!=='string'){return createErrorMessage('Description must be string')}};return{status:Status.OK,value:{id:resultId,title,description},}};export const updateType=async(req:Request,res:Response)=>{const params=getQueryParams(req);if(params.status===Status.ERROR){return res.status(Codes.ERROR).json(params)};const{id,title,description}=params.value;const updatedParams=getValidParamsWithCheckAll({id,title,description});Type.updateOne({\_id:id},updatedParams,null,error=>{if(error){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+error.name))};Type.findOne({\_id:id}).exec((findError,findResult)=>{if(findError){return res.status(Codes.ERROR).json(createErrorMessage('Something went wrong.'+findError.name))};return res.status(Codes.OK).json(findResult)})})}

src\helpers\getValidParams.ts

import{Types}from'mongoose';export function getValidParamsWithCheckID<T extends{id:Types.ObjectId|undefined}>(value:T):Omit<T,'id'>&{\_id?:Types.ObjectId}{const result=Object.keys(value).reduce((acc,current)=>{if(current!=='id'){return{...acc,[current]:value[current]}};if(value[current]!==undefined){return{...acc,\_id:value[current]}};return acc},{})as Omit<T,'id'>&{\_id?:Types.ObjectId};return result};export function getValidParamsWithCheckAll<T extends{id:Types.ObjectId|undefined}>(value:T){const checkedId=getValidParamsWithCheckID(value);const res=Object.fromEntries(Object.entries(checkedId).filter(([key,value])=>value!==undefined));return res}

src\helpers\messages.ts

import{Status,ErrorMessageAnswer,SuccessMessageAnswer}from'../types';export const createErrorMessage=(message:string):ErrorMessageAnswer=>({status:Status.ERROR,message:message,});export const createSuccessMessage=(message:string):SuccessMessageAnswer=>({status:Status.OK,message:message,})

src\models\AccountLevel.ts

import{Schema,model}from'mongoose';const accountLevelSchema=new Schema({level:{type:Number,required:true},description:{type:String,required:true},});const AccountLevel=model('AccountLevel',accountLevelSchema);export{AccountLevel}

src\models\Category.ts

import{Schema,model}from'mongoose';const categorySchema=new Schema({title:{type:String,required:true},description:{type:String,required:true},});const Category=model('Category',categorySchema);export{Category}

src\models\index.ts

export{Type}from'./Type';export{Review}from'./Review';export{ProductsAllInfo}from'./ProductsAllInfo';export{Category}from'./Category';export{Producer}from'./Producer';export{AccountLevel}from'./AccountLevel';export{User}from'./User';export{Order}from'./Order';export{Product}from''

src\models\Order.ts

import{Schema,model}from'mongoose';import{User}from'.';import{Product}from'./Product';const orderSchema=new Schema({user:{type:Schema.Types.ObjectId,ref:User,required:true},product:{type:Schema.Types.ObjectId,ref:Product,required:true},count:{type:Number,required:true},productPrice:{type:Number,required:true},address:{type:String,required:true},status:{type:String,required:true},},{timestamps:true,});const Order=model('Order',orderSchema);export{Order}

src\models\Producer.ts

import{Schema,model}from'mongoose';const producerSchema=new Schema({title:{type:String,required:true},description:{type:String,required:true},website:{type:String,required:true},},{timestamps:true,});const Producer=model('Producer',producerSchema);export{Producer}

src\models\Product.ts

import{Schema,model}from'mongoose';import{Category,Producer,ProductsAllInfo,Type,User}from'.';const productSchema=new Schema({title:{en:{type:String,required:true},ua:{type:String,required:true},},description:{en:{type:String,required:true},ua:{type:String,required:true},},price:{type:Number,required:true},stars:{type:Number,required:true},image:{type:Buffer,contentType:String,required:true},type:{type:[Schema.Types.ObjectId],ref:Type,required:true},category:{type:[Schema.Types.ObjectId],ref:Category,required:true},producer:{type:Schema.Types.ObjectId,ref:Producer,required:true},novelty:{type:Boolean,default:false,required:true},discount:{type:Boolean,default:false,required:true},discountValue:{type:Number,default:-1,required:true},discountPreviousPrice:{type:Number,default:-1,required:true},allInfo:{type:Schema.Types.ObjectId,ref:ProductsAllInfo,required:true,},creator:{type:Schema.Types.ObjectId,ref:User,required:true},},{timestamps:true,});const Product=model('Product',productSchema);export{Product}

src\models\ProductsAllInfo.ts

import{Schema,model}from'mongoose';import{Review}from'.';const productsAllInfoSchema=new Schema({description:{type:String,required:true},characteristics:{type:String,required:true},reviews:{type:[Schema.Types.ObjectId],ref:Review},},{timestamps:true,});const ProductsAllInfo=model('ProductsAllInfo',productsAllInfoSchema);export{ProductsAllInfo}

src\models\Review.ts

import{Schema,model}from'mongoose';import{User}from'./User';const rewiewsSchema=new Schema({author:{type:Schema.Types.ObjectId,ref:User},comment:{type:String,required:true},pluses:{type:String,required:true},minuses:{type:String,required:true},rating:{type:Number,required:true},},{timestamps:true,});const Review=model('Review',rewiewsSchema);export{Review}

src\models\Type.ts

import{Schema,model}from'mongoose';const typeSchema=new Schema({title:{type:String,required:true},description:{type:String,required:true},});const Type=model('Type',typeSchema);export{Type}

src\models\User.ts

import{Schema,model}from'mongoose';import{AccountLevel}from'./AccountLevel';const userSchema=new Schema({surname:{type:String,required:true},name:{type:String,required:true},email:{type:String,required:true},phone:{type:String,required:true},password:{type:String,required:true},country:{type:String,required:true},city:{type:String,required:true},zipCode:{type:Number,required:true},address:{type:String,required:true},accountAdminLevel:{type:Schema.Types.ObjectId,ref:AccountLevel,required:true,},},{timestamps:true,});const User=model('User',userSchema);export{User}

src\routes\accountLevel.ts

import Router,{Request,Response}from'express';import{createNewAccountLevel,deleteAccountLevel,getAccountLevel,updateAccountLevel,getAllAccountLevels,}from'../controllers/AccountLevel';const AccountLevelRouter=Router();AccountLevelRouter.get('/accountLevel/all',getAllAccountLevels);AccountLevelRouter.get('/accountLevel',getAccountLevel);AccountLevelRouter.post('/accountLevel',createNewAccountLevel);AccountLevelRouter.delete('/accountLevel/:\_id',deleteAccountLevel);AccountLevelRouter.put('/accountLevel',updateAccountLevel);export{AccountLevelRouter}

src\routes\categories.ts

import Router,{Request,Response}from'express';import{createNewCategory,deleteCategory,getAllCategories,getCategories,updateCategory,}from'../controllers/Category';const CategoriesRouter=Router();CategoriesRouter.use((req:Request,res:Response,next)=>{console.log('[Server]: Time = ',Date.now());next()});CategoriesRouter.get('/categories/all',getAllCategories);CategoriesRouter.get('/categories',getCategories);CategoriesRouter.post('/categories',createNewCategory);CategoriesRouter.put('/categories',updateCategory);CategoriesRouter.delete('/categories/:\_id',deleteCategory);export{CategoriesRouter}

src\routes\home.ts

import Router,{Request,Response}from'express';const router=Router();router.use((req:Request,res:Response,next)=>{console.log('[Server]: Time = ',Date.now());next()});router.get('/',(req:Request,res:Response)=>{res.status(200).json({page:'home',type:'get'})});router.post('/',(req:Request,res:Response)=>{res.status(200).json({page:'home',type:'post'})});export{router as HomeRouter}

src\routes\index.ts

import Router,{Request,Response}from'express';import{User}from'../models';export{HomeRouter}from'./home';export{AccountLevelRouter}from'./accountLevel';export{CategoriesRouter}from'./categories';export{ProducerRouter}from'./producers';export{TypesRouter}from'./types';export{OrdersRouter}from''

src\routes\orders.ts

import Router,{Request,Response}from'express';import{getAllOrders,getOrder,createNewOrder,updateOrder,deleteOrder,}from'../controllers/Order';const OrdersRouter=Router();OrdersRouter.get('/order/all',getAllOrders);OrdersRouter.get('/order',getOrder);OrdersRouter.post('/order',createNewOrder);OrdersRouter.put('/order',updateOrder);OrdersRouter.delete('/order/:\_id',deleteOrder);export{OrdersRouter}

src\routes\producers.ts

import Router,{Request,Response}from'express';import{createNewProducer,deleteProducer,getProducer,updateProducer,getAllProducers,}from'../controllers/Producer';const ProducerRouter=Router();ProducerRouter.get('/producer/all',getAllProducers);ProducerRouter.get('/producer',getProducer);ProducerRouter.post('/producer',createNewProducer);ProducerRouter.delete('/producer/:\_id',deleteProducer);ProducerRouter.put('/producer',updateProducer);export{ProducerRouter}

src\routes\types.ts

import Router,{Request,Response}from'express';import{createNewType,deleteType,getType,updateType,getAllTypes,}from'../controllers/Type';const TypesRouter=Router();TypesRouter.get('/type/all',getAllTypes);TypesRouter.get('/type',getType);TypesRouter.post('/type',createNewType);TypesRouter.delete('/type/:\_id',deleteType);TypesRouter.put('/type',updateType);export{TypesRouter}

src\types\index.ts

export\*from''

src\types\main.ts

export enum Status{OK='ok',ERROR='error',};export enum Codes{OK=200,ERROR=500,};type MessageAnswer<T extends Status>={status:T;message:string};export type SuccessMessageAnswer=MessageAnswer<Status.OK>export type ErrorMessageAnswer=MessageAnswer<Status.ERROR>

src\utils\initializeDatabase\AccountLevel.ts

import mongoose,{Schema,ObjectId}from'mongoose';import{AccountLevel}from'../../models';const initializeDatabaseAccountLevel=async()=>{await AccountLevel.deleteMany({});const createAccountLevelInitialize=(\_id:mongoose.Types.ObjectId,level:number,description:string)=>{return new AccountLevel({\_id:\_id,level:level,description:description,}).save(error=>{if(error){console.error(`AdminLevel ${level}error`,error);return};console.log(`AdminLevel ${level}has been added`)})};createAccountLevelInitialize(new mongoose.Types.ObjectId('63838c77675cff96068e1823'),0,'Default');createAccountLevelInitialize(new mongoose.Types.ObjectId('63838c77675cff96068e1824'),1,'Moderator');createAccountLevelInitialize(new mongoose.Types.ObjectId('63838c77675cff96068e1825'),2,'Admin')};export{initializeDatabaseAccountLevel}

src\utils\initializeDatabase\Category.ts

import mongoose,{Schema,ObjectId}from'mongoose';import{Category}from'../../models';const initializeDatabaseCategory=async()=>{await Category.deleteMany({});const createCategoryInitialize=(\_id:mongoose.Types.ObjectId,title:string,description:string)=>{return new Category({\_id:\_id,title:title,description:description,}).save(error=>{if(error){console.error(`Category ${title}error`,error);return};console.log(`Category ${title}has been added`)})};createCategoryInitialize(new mongoose.Types.ObjectId('638387eb6f33fac689be5e81'),'Care','Care');createCategoryInitialize(new mongoose.Types.ObjectId('638387eb6f33fac689be5e82'),'Walk','Walk');createCategoryInitialize(new mongoose.Types.ObjectId('638387eb6f33fac689be5e83'),'Health','Health');createCategoryInitialize(new mongoose.Types.ObjectId('638387eb6f33fac689be5e84'),'Toys','Toys');createCategoryInitialize(new mongoose.Types.ObjectId('638387eb6f33fac689be5e85'),'SleepingPlaces','SleepingPlaces');createCategoryInitialize(new mongoose.Types.ObjectId('638387eb6f33fac689be5e80'),'Bowls','Bowls')};export{initializeDatabaseCategory}

src\utils\initializeDatabase\index.ts

import{initializeDatabaseCategory}from'./Category';import{initializeDatabaseType}from'./Type';import{initializeDatabaseAccountLevel}from'./AccountLevel';const initializeDataBase=()=>{initializeDatabaseType();initializeDatabaseCategory();initializeDatabaseAccountLevel()};export{initializeDataBase}

src\utils\initializeDatabase\Type.ts

import mongoose,{Schema,ObjectId}from'mongoose';import{Type}from'../../models';const initializeDatabaseType=()=>{Type.deleteMany({},()=>{});const createTypeInitialize=(\_id:mongoose.Types.ObjectId,title:string,description:string)=>{return new Type({\_id:\_id,title:title,description:description,}).save(error=>{if(error){console.error(`Type ${title}error`,error);return};console.log(`Type ${title}has been added`)})};createTypeInitialize(new mongoose.Types.ObjectId('6383529877588dc76e74919c'),'Cats','Cats');createTypeInitialize(new mongoose.Types.ObjectId('6383529877588dc76e74919d'),'Dogs','Dogs')};export{initializeDatabaseType}