== Титульна сторінка ==

Зміст

[Вступ 4](#_Toc130907121)

[1 Опис предметної області 5](#_Toc130907122)

[1.1 Основні поняття 5](#_Toc130907123)

[1.2 Основний алгоритм. 5](#_Toc130907124)

[2 Постанова завдання 6](#_Toc130907125)

[2.1 Мета створення програми 6](#_Toc130907126)

[2.2 Функції програми . 6](#_Toc130907127)

[2.3 Вимоги до проектованої системи 6](#_Toc130907128)

[2.4 Вимоги до надійності 6](#_Toc130907129)

[2.5 Умови роботи програми 6](#_Toc130907130)

[2.6 Умови розповсюдження програми 6](#_Toc130907131)

[3 Програмування 7](#_Toc130907132)

[3.1 Обгрунтування вибору середовища розробки системи 7](#_Toc130907133)

[3.2 Обгрунтування вибору середовища функціонування системи 7](#_Toc130907134)

[3.3 Основні рішення щодо реалізації компонентів системи 7](#_Toc130907135)

[3.3.1 Використовувані моделі даних 7](#_Toc130907136)

[3.3.2 Структурна схема програми 7](#_Toc130907137)

[3.3.3 Розробка модулів системи . 7](#_Toc130907138)

[4 Методика роботи користувача з системою 8](#_Toc130907139)

[4.1 Керівництво програміста 8](#_Toc130907140)

[4.1.1 Призначення і умови використання програми 8](#_Toc130907141)

[4.1.2 Характеристики програми 8](#_Toc130907142)

[4.1.3 Звертання до програми (файл проекту) 8](#_Toc130907143)

[4.1.4 Вхідні і вихідні дані 8](#_Toc130907144)

[4.2 Керівництво оператора 8](#_Toc130907145)

[4.2.1 Призначення і умови використання програми 8](#_Toc130907146)

[4.2.2 Виконання програми 8](#_Toc130907147)

[4.2.3 Повідомлення оператору 8](#_Toc130907148)

[5 Організаційно – економічний розділ 9](#_Toc130907149)

[5.1 Планування розробки програмного продукту 9](#_Toc130907150)

[5.2 Розрахунок витрат на розробку програмного продукту 9](#_Toc130907151)

[5.2.1 Складання кошторису витрат на розробку 9](#_Toc130907152)

[5.2.2 Розрахунок собівартості програмного продукту 9](#_Toc130907153)

[5.3 Оцінка ефективності проекту 9](#_Toc130907154)

[6 Охорона праці користувачів комп’ютерів 10](#_Toc130907155)

[6.1 Правове збезпечення заходів щодо охорони праці користувачів комп’ютерів 10](#_Toc130907156)

[6.2 Електробезпека та пожежобезпека у приміщеннях з персональними комп’ютерами 10](#_Toc130907157)

[6.3 Причини виникнення, загальна характеристика та класифікація надзвичайних ситуацій 10](#_Toc130907158)

[Висновки 11](#_Toc130907159)

[Список літератури 12](#_Toc130907160)

[Додатки 13](#_Toc130907161)

# Вступ

Завдяки прогресу та розвитку інтернету обсяг доступної інформації для людей значно збільшився. Раніше отримання певної інформації було нелегким завданням, але зараз з легкістю можна знайти будь-яку інформацію через пошукові мережі та доступ до всесвітньої мережі даних. Інформація стала відкритою для усіх бажаючих, що призвело до значного збільшення її обсягу. Для зберігання великої кількості даних потрібен зручний сервіс, який зможе це робити

Не так давно вчені підрахували, що сучасна людина за тиждень отримує стільки інформації, скільки людина середньовіччя отримувала за все життя. Людська психіка має певні обмеження. Експериментально доведено, що мозок звичайної людини здатен сприймати і безпомилково обробляти інформацію зі швидкістю не більше 25 біт на секунду (в одному слові середньої довжини міститься якраз 25 біт). При такій швидкості поглинання інформації людина за життя може прочитати не більше трьох тисяч книг. І то – за умови, що буде щодня освоювати по 50 сторінок. Мало того, що ми не встигаємо вивчити велику частину інформації, яка накопичується, вона ще й швидко старіє і вимагає заміни. Вперше над цим фактом задумалися вчені у 70-х роках минулого століття. Тоді і почав використовуватись термін «інформаційний вибух». [1]

Мета цієї дипломної роботи полягає в створенні багатошарової клієнт-серверної архітектури, яка спрощує та удосконалює обробку великих обсягів інформації, зокрема, за рахунок зручної маніпуляції статичними даними.

# 1 Опис предметної області

## Основні поняття

Предмет науки (предметна область) – це ті сторони, зв’язки, відношення об’єкта, які вивчаються даною наукою [2]

Мета цієї дипломної роботи полягає в створенні багатошарової клієнт-серверної архітектури, яка спрощує та удосконалює обробку великих обсягів інформації, зокрема, за рахунок зручної маніпуляції статичними даними.

Розробка програмного забезпечення (ПЗ) – це вид діяльності та процес, спрямований на створення та підтримку працездатності, якості та надійності ПЗ, використовуючи технології, методологію та практики з інформатики, керування проектами, математики, інженерії та інших областей знання. Як і інші традиційні інженерні дисципліни, розробка ПЗ має справу з проблемами якості, вартості та надійності. Деякі програми містять мільйони рядків вихідного коду, які, як очікується, повинні правильно виконуватися в умовах, що змінюються. Складність ПЗ порівнянна зі складністю найбільш складних сучасних машин (таких, наприклад, як літаки). [3]

Елементами сервісу введення статистики є: категорії, статичні дані, графік, авторизація, активні сесії користувача,

Категорії в сервісі служать для об’єднання даних, які користувач може ввести в систему для зручного збереження та роботою з даними різного виду, таких як, наприклад, різні види спортивних вправ або ж різні сфери

Статичні дані в сервісі - дані, які користувач може вводити в систему, наприклад, кількість відвідувачів сайту за день або продажі за тиждень, обираючи для них якусь категорію.

Графік в сервісі - візуальне представлення статистичних даних у вигляді графіка з різними режимами відображення. Графіки допомагають користувачам аналізувати дані і зробити висновки про тенденції і зміни у певному періоді часу.

Автентифікація - це процес перевірки особистості користувача. Технологія перевірки автентичності забезпечує контроль доступу для систем, перевіряючи, чи облікові дані користувача збігаються з обліковими даними в базі даних авторизованих користувачів або на сервері автентифікації даних [4]

Активні сесії користувача - це перелік сеансів, що знаходяться у відкритому стані на різних пристроях, що користується користувач. Кожна сесія відображається відповідно до типу пристрою, часу та інших параметрів. Це дозволяє користувачам управляти своїм доступом до сервісу та контролювати власну безпеку в Інтернеті.

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. [5]

Після аналізу досліджуваної предметної області було визначено, що найбільш ефективним рішенням для реалізації програмного застосування є багатошарова архітектура. Це дає змогу створити зручний та масштабований сервіс, який зможе оптимально опрацьовувати великі обсяги інформації.

## Основний алгоритм

Основна ідея автоматизованої системи полягає в заміні ручного збору та обробки статистичної інформації на її автоматизований збір та обробку в комп'ютерній системі. Для досягнення цієї мети, створюється відповідна інформаційна система, яка міститиме базу даних з інформацією про категорії статичних даних, користувачів та статистичні дані.

Проектування бази даних починається з концептуального проектування, де визначаються всі об’єкти, що використовуються в базі даних, їх характеристики та зв'язки між ними. База даних розташовується в хмарі та автоматично створюється у разі її відсутності.

Після з'єднання з базою даних, програма отримує доступ до виконання різноманітних запитів в базу даних, таких як додавання, редагування, видалення та читання інформації. Також, система передбачає підтримку авторизації користувачів та обробку даних з форм.

У результаті роботи автоматизованої системи статистична інформація буде збиратися та зберігатися в базі даних, яка забезпечить швидкий та легкий доступ до неї, а також можливість проведення різноманітного аналізу та створення звітів.

# 2 Постанова завдання

## 2.1 Мета створення програми

Метою дипломної роботи є створення онлайн-сервісу, на основі багатошарової клієнт-серверної архітектури, який дозволить користувачам зберігати та аналізувати статистичні дані

## Функції програми

Після проведення дослідження відповідної предметної області, необхідно розробити програму з різноманітними функціями, щоб забезпечити користувачам можливість ефективно використовувати її потенціал. Нижче наведено фукнції, які повина включати програма:

* Авторизація за допомогою облікового запису Google для зручного входу в систему без необхідності створювати новий обліковий запис та запам'ятовування логіну та паролю.
* Можливість створювати, редагувати, видаляти та змінювати порядок категорій.
* Користувачі повині мати змогу переглядати інформацію про категорії в зручному табличному форматі.
* Можливість створювати, редагувати, видаляти та змінювати порядок груп категорій. Одна категорія може належати до декількох груп.
* Можливість створювати, редагувати, видаляти та змінювати порядок підкатегорій. Підкатегорії повинні значно полегшити статичний облік даних користувачам
* Можливість створювати, редагувати та видаляти записи статики в системі на основі категорій. Ця функція дозволяє зберігати важливі дані та інформацію.
* Можливість перегляду активних сеансів користувача та їх завершення в разі необхідності.
* Візуальне відображення статичних даних у вигляді графіка з різноманітними параметрами групування та фільтрації.
* Можливість переглядати записи статистики у вигляді таблиці з фільтраціями та сортуваннями за різними критеріями.
* Можливість безпечного виходу з облікового запису без втрати даних.
* Забезпечення безпеки даних. Забезпечення цілісності та безпеки даних є найважливішою функцією, яка гарантує, що інформація користувачів буде захищена від несанкціонованого доступу.

## Вимоги до проектованої системи

Основні вимоги до проектованої системи включають:

* Забезпечення чіткого та зрозумілого інтерфейсу взаємодії з користувачем, що містить всі необхідні елементи та легкий у використанні, а також адаптивний до різних розмірів екранів.
* Забезпечення зрозумілості використання програми, включаючи наявність текстових описань дій та підказок.
* Збереження даних, які завантажує користувач на сервері, та забезпечення їх доступності у будь-який момент часу.
* Забезпечення правильної роботи всіх функцій програми та можливості скасування змін.
* Забезпечення надійності та конфіденційності персональних даних користувача, які використовує програма.

Додаткові вимоги до проектованої системи включають:

* Забезпечення швидкої та ефективної роботи системи.
* Сумісність з різними операційними системами та браузерами.
* Забезпечення можливості редагування та видалення даних користувача.
* Забезпечення можливості інтеграції з іншими програмами або сервісами.
* Забезпечення можливості масштабування та розвитку системи в майбутньому.

## Вимоги до надійності

До основних вимог до надійності та безпеки програми відносяться наступні:

* Забезпечення конфіденційності та цілісності персональних даних користувача. Це означає, що будь-яка інформація, яку користувач надає в рамках програми, повинна бути захищена від несанкціонованого доступу і використання третіми особами. Також необхідно забезпечити захист від можливих атак хакерів та зловмисників.
* Повідомлення користувача про будь-які помилки або проблеми, що виникають при завантаженні або роботі програми. Користувач повинен бути повідомлений про будь-які виникненні помилки та отримувати достатньо інформації для їх виправлення.
* Забезпечення індивідуального доступу до даних. Кожен користувач повинен мати можливість зайти в програму лише за своїм акаунтом і мати доступ тільки до своїх власних даних. Таким чином, забезпечується захист персональних даних від доступу третіх осіб.
* Програма повинна чітко інтерпретувати та зберігати інформацію. Всі дані, введені користувачем, повинні бути збережені на сервері програми та бути доступними для перегляду та редагування користувачем в будь-який момент часу. Програма повинна чітко інтерпретувати цю інформацію та відображати її у зрозумілому форматі для користувача. Також необхідно забезпечити надійне зберігання даних на сервері та їх резервне копіювання, щоб у разі виникнення проблем з сервером, дані користувача були збережені в безпечному місці.

## Умови роботи програми

Для коректної роботи клієнтської частини програми необхідно мати браузер, який має мінімальну версію не нижче 2020 року. Найбільш підходящими для використання є такі браузери, як Google Chrome, Safari, Mozilla Firefox, Opera та Microsoft Edge.

Рекомендується мати останню версію браузера, оскільки це забезпечує оптимальну швидкість роботи та захист від можливих вразливостей. Також рекомендується використовувати встановлені оновлення та плагіни для браузера для запобігання можливих проблем з безпекою.

Як можна бачити, згідно статистики популярності різних браузерів “Desktop Browser Market Share Worldwide Mar 2022 - Mar 2023”, від statcounter, яка зображена на рисунку 1.5 клієнтська частина програми має коректно працювати у всіх сучасних та найбільш використовуваних браузерах. [6]



Рисунок 1.5 – Статистика популярності різних браузерів в світі

Серверна частина програмного забезпечення працює на NodeJS - це засіб виконання JavaScript, який дозволяє розробникам створювати серверні додатки. Різні версії NodeJS мають різний рівень підтримки нових функцій та можуть мати відмінності в швидкості та стабільності роботи. Для того, щоб забезпечити правильну роботу сервісу, рекомендується використовувати одну з наступних LTS версій NodeJS: 18.15.0, 16.20.0 або 14.21.3.

Варто відзначити, що NodeJS на сьогодні є найпопулярнішою платформою для розробки клієнт-серверних додатків, оскільки вона дозволяє розробникам писати серверний код на JavaScript, що дозволяє швидко та ефективно створювати високоякісні додатки.

На рисунку 1.6 зображена статистика найбільш використовуваних веб-фреймворків та бібліотек за 2022 рік. Як можна бачити, NodeJS займає перше місце [7]



1.6 – Статистика найбільш використовуваних веб-фреймворків та бібліотек за 2022 рік

## 2.6 Умови розповсюдження програми

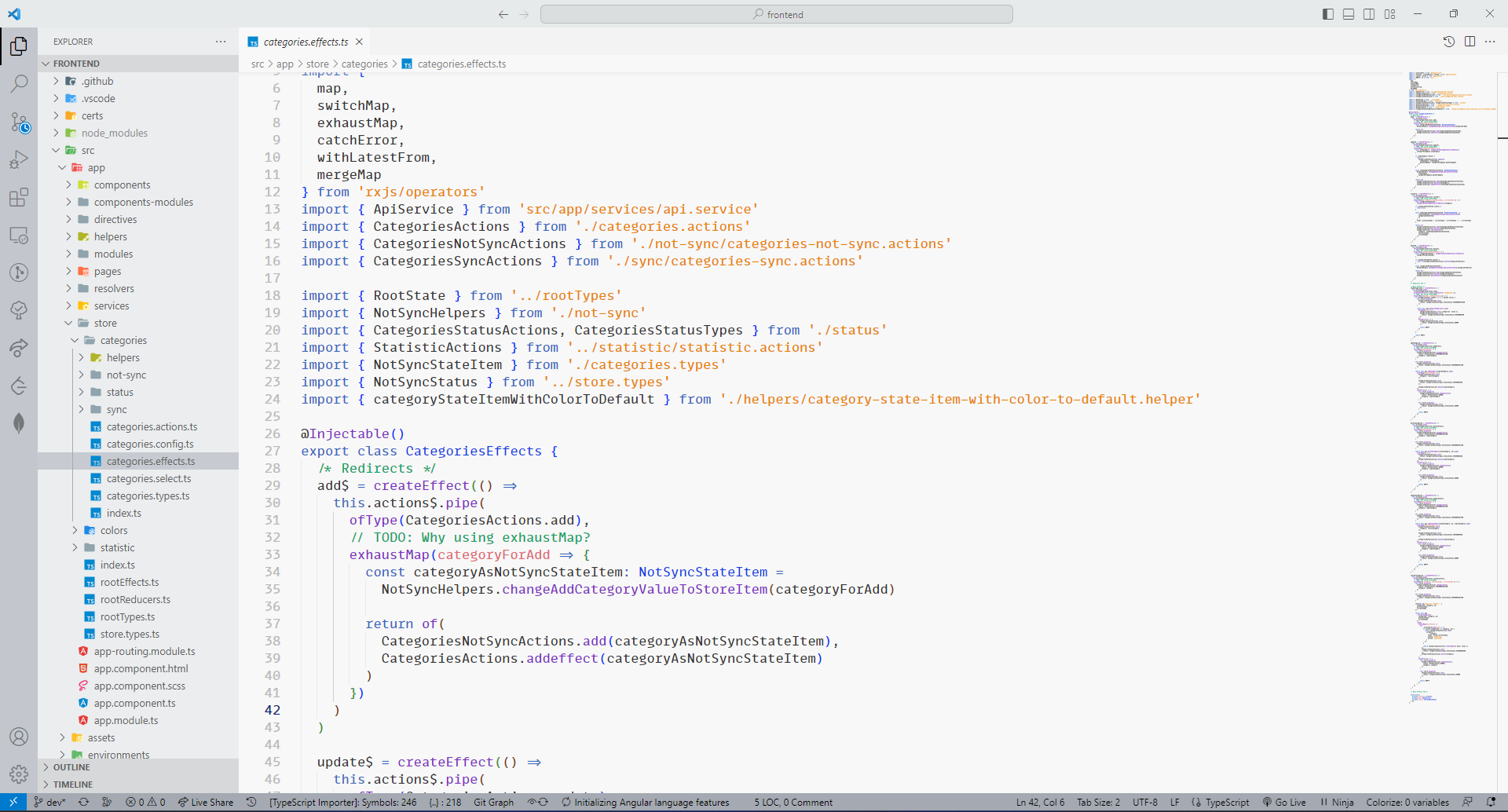
Даний програмний продукт є безкоштовним і доступним за допомогою мережі Інтернет. Щоб мати можливість користуватися програмою, користувачеві необхідно увести в адресний рядок свого браузера наступну адресу: "https://counter-ltlaitoff.vercel.app". Ця адреса є посиланням на веб-сторінку, на якій розміщений даний програмний продукт. Веб-сторінка містить інформацію про програму та надає можливість користувачам скористатися нею безкоштовно. Варто відзначити, що таке розповсюдження дозволяє забезпечити швидкий та зручний доступ до програмного продукту для широкого кола користувачів з усього світу.

# 3 Програмування

## 3.1 Обгрунтування вибору середовища розробки системи

В якості середовища розробки сервісу був обраний Visual Studio Code — це легкий, але потужний редактор вихідного коду, який може працювати навіть в браузері та доступний для Windows, macOS і Linux. Поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов і середовищ виконання (таких як C++, C#, Java, Python, PHP, Go, .NET). Вигляд основного вікна Visual Studio Code зображений на рисунку 3.1 [3.1]

[3.1] - https://code.visualstudio.com/docs

Рисунок 3.1 – Вигляд основного вікна

Він підтримує ряд мов програмування, зокрема мову програмування TypeScript, яка була використана для написання сервісу, підсвічування синтаксису, IntelliSense, рефакторинг та налагодження програми, навігацію по коду, підтримку Git та інші можливості для розробки різноманітних додатків

Visual Studio Code є оптимальним рішенням для розробки цільного додатку. Він підтримує велику кількість мов, що стане в нагоді при розробці проектів, в яких використані різні рішення щодо реалізації їх компонентів.

Також в цьому середовищі розробки є вбудована функція відкриття і перегляду не тільки одного файлу, а й цілої папки проекту: її вміст буде відображатись у лівій частині інтерфейсу. Це робить зручним швидкий доступ до різних файлів проекту і налагодження їх зв’язку між собою, а разом з підтримкою багатьох рішень розробки дозволяє швидко перемикати увагу з однієї частини проекту на іншу, не витрачаючи часу на відкриття інших програм.

Великою перевагою VS Code є присутній у ньому вбудований відладчик, меню якого показано на рисунку 3.2, програм майже для будь-якої обраної мови програмування. Є можливість запустити файл на виконання не переходячи в інші програми, що прискорює знаходження і виправлення помилок.



Рисунок 3.2 – Демонстрація панелі для роботи з вбудованим

відладчиком Visual Studio Code

Також реалізовані функції навігації по коду, доповнення типових конструкцій і контекстної підказки(IntelliSense), які суттєво прискорюють роботу над проектом. Контекстні підказки показані на рисунку 3.3

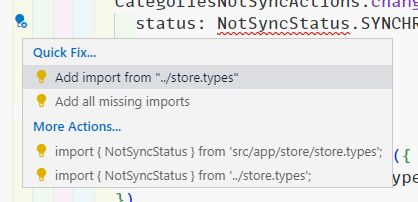
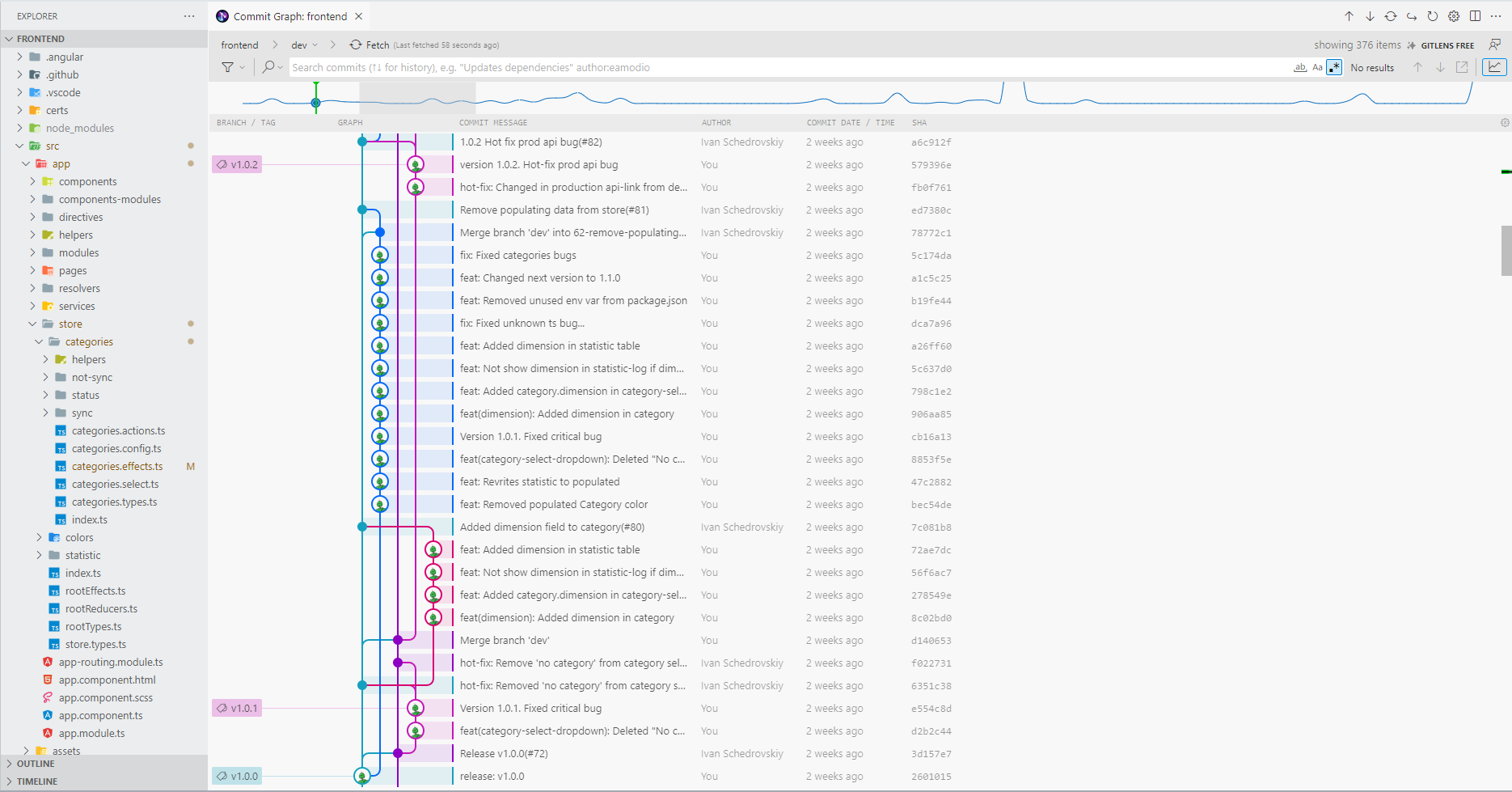


Рисунок 3.3 – Демонстрація контекстних підказок

Основною перевагою Visual Studio Code є реалізована зручна вбудована система управління надлаштуваннями і розширеннями: їх можна завантажувати і встановлювати прямо у редакторі на вкладинці Extensions. До розширень належать, наприклад, пакети підтримки мов, бібліотеки або відладчики до них, додатки для підсвітки і форматування коду та інші надлаштування, що розширюють його функціонал або підвищують комфортність роботи у редакторі. На рисунку 3.4 продемонстрована робота розширення «GitLens»

Рисунок 3.4 – Демонстрація роботи розширення «GitLens»

Важливою особливістю VS Code є простий і зрозумілий інтерфейс. Всі елементи розбиті на групи, їх небагато, що робить роботу в редакторі зручною і легко зрозумілою навіть для новачка. Також інтерфейс можна повністю налаштувати: від зовнішнього вигляду і теми до положення елементів управління. Користувач може змінити вікно програми для максимізації власного комфорту, або спеціально під розробку конкретного додатку

В якості систему контролю версіями був обраний Git.

Система контролю версій - це система, що записує зміни у файл або набір файлів протягом деякого часу, так щоб розробник зміг повернутися до певної версії пізніше. [3.2]

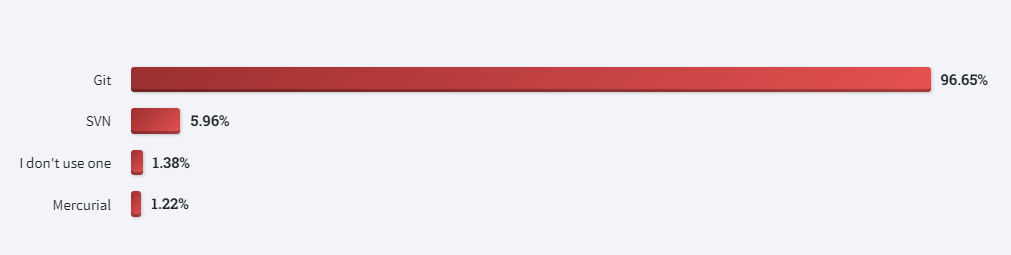
[3.2] - <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

У 2005 році відносини між спільнотою розробників ядра Linux і комерційною компанією, що розробила BitKeeper почали псуватись, і безкоштовне використання продуктом було скасовано. Це підштовхнуло розробників Linux (і зокрема Лінуса Торвальдса, автора Linux) розробити власну систему, ґрунтуючись на деяких з уроків, які вони дізналися під час використання BitKeeper. [3.3]

[3.3] - https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git

Як видно з графіку зображеного на рисунку 3.5 популярності різних систем контролю версій 2022 року серед професійних розробників від StackOverFlow на данй момент Git є найбільш популярною системою контролю версій. [3.4]

[3.4] - https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems

Рисунок 3.5 – Графік популярності різних систем контролю версій 2022 року серед професійних розробників від StackOverflow

Для зручної побудови та тестування API, зокрема зі стандартом REST, було обрано Postman.

API (Application Programming Interface) забезпечує взаємодію між двома системами. API дозволяє надсилати інформацію безпосередньо з однієї програми до іншої, обминаючи інтерфейс взаємодії з користувачем. API приймає запит, передає інформацію системі, обробляє її та повертає відповідь.

REST (Representational State Transfer) – стандарт архітектури взаємодії додатків і сайтів, що використовує протокол HTTP. Особливість REST в тому, що сервер не запам'ятовує стан користувача між запитами. Іншими словами, ідентифікація користувача і всі параметри виконання операції передаються в кожному запиті.

Для передачі запитів і відповідей під час роботи з API використовується протокол HTTP

HTTP (Hypertext Transfer Protocol) – це протокол передачі гіпертексту, який дозволяє клієнту та серверу спілкуватися по мережі за допомогою запиту/відповіді. Це протокол рівня додатка, який покладається на TCP/IP для своїх послуг.

HTTPS – це розширення протоколу передачі гіпертексту(HTTP). Він використовується для безпечного спілкування через комп’ютерну мережу. [3.6]

[3.6] - https://training.qatestlab.com/blog/technical-articles/use-postman-in-testing/

Postman - це платформа для розробки та використання API, яка дозволяє спростити кожен етап процесу створення та вдосконалення API, включаючи тестування та документування. Завдяки Postman розробники можуть створювати API швидше і ефективніше.

Крім того, платформа Postman забезпечує візуалізацію запитів і відповідей, що полегшує налагодження API та дозволяє зменшити час на коригування помилок. [3.7]

[3.7] - https://www.postman.com/

Вигляд головного вікна Postman зображено на рисунку 3.6

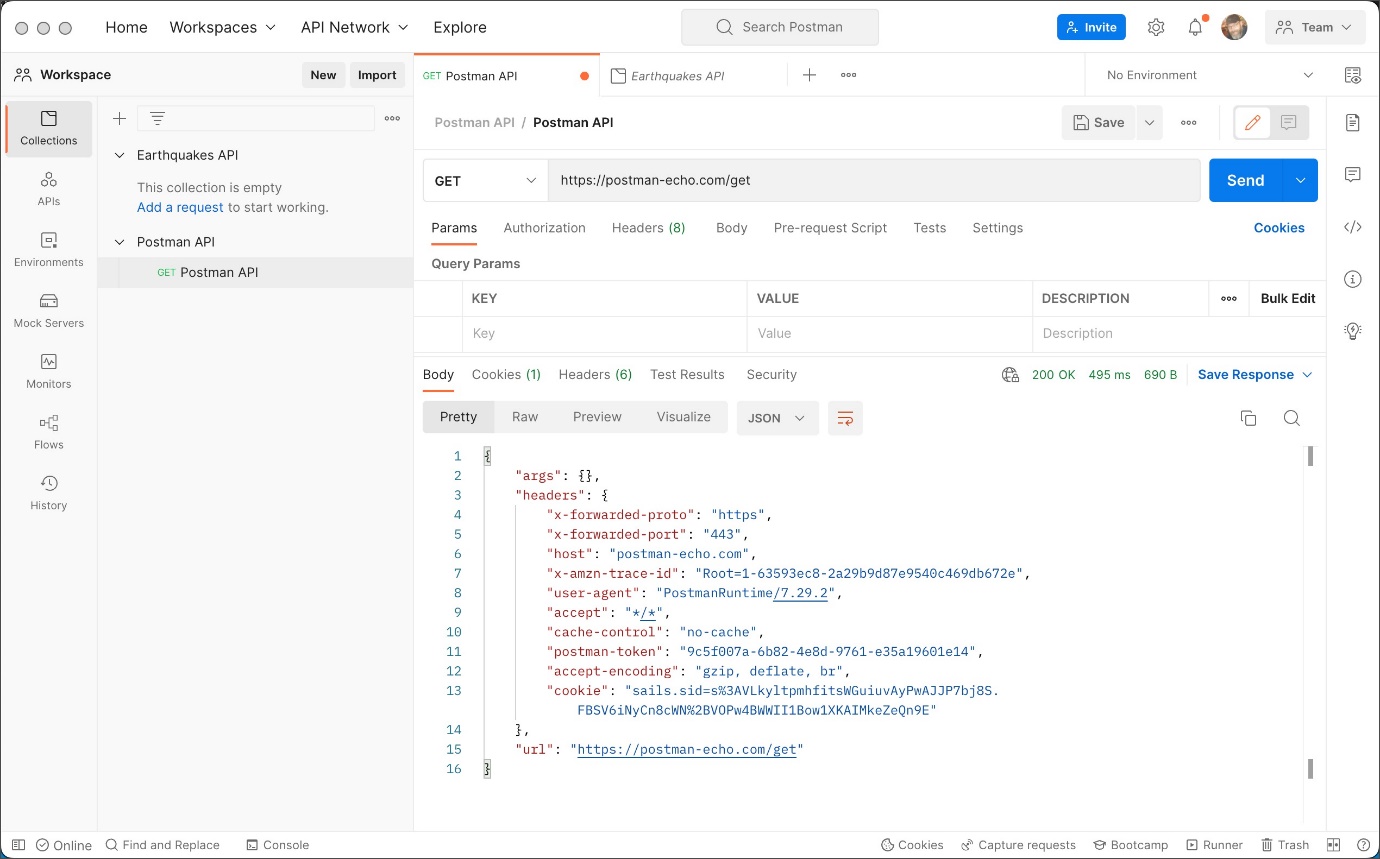


Рисунок 3.6 – Вигляд головного вікна Postman

Swagger UI дозволяє візуалізувати та взаємодіяти з ресурсами API, не маючи жодної реалізації логіки. Вона автоматично генерується з OpenAPI (раніше відомого як Swagger) специфікації, а візуальна документація спрощує реалізацію на боці сервера та використання на боці клієнта.

Swagger UI в дипломній роботі використовується для швидкої розробки документації API для більш зручного використання серверної частини при розробці клієнтської та може використовуватись іншими розробниками при роботі з API в майбутньому [3.8]

[3.8] - https://swagger.io/tools/swagger-ui/

## 3.2 Обгрунтування вибору середовища функціонування системи

Клієнтська частина сервісу повинна функціонувати або ж в браузері користувача, або в вигляді PWA застосунку

Браузер – програмне забезпечення яке переносить користувача до будь-якого куточку Інтернету і дає змогу переглядати текст, зображення та відео де б користувач не знаходився.

Браузер переносить вас до будь-якого закуточка Інтернету. Він бере інформацію з інших частин Інтернету та показує її на вашому комп'ютері чи мобільному пристрої. Інформація передається за допомогою протоколу передачі гіпертексту (Hypertext Transfer Protocol), який визначає, як передаються текст, зображення та відео в Інтернеті. Ця інформація повинна надсилатися та показуватися у сумісному форматі, щоб користувачі будь-якого браузера, у будь-якому куточку світу, могли побачити цю інформацію. [3.9]

[3.9] - <https://www.mozilla.org/uk/firefox/browsers/what-is-a-browser/>

Progressive Web Applications (PWA), яка була анонсована Google у 2015 році. PWA — це сучасні вебсайти, які наділені характеристиками нативних додатків: можливістю запуску з робочого столу девайсу із доступом до його вбудованих функцій, спроможністю роботи offline, наявністю push-сповіщень тощо.

Виявлено, що особливі переваги від використання PWA можуть отримати компанії у сфері e-commerce й інформаційного бізнесу. Зручність використання таких додатків, високий рівень UI/UX, їх швидкодія та невибагливість до ресурсів, можливість роботи без підключення до інтернету сприяють розширенню мобільної присутності компаній у вебпросторі, зростанню конверсії та збільшенню доходів, які надходять з інтернет-каналів. Важливою перевагою є невисока вартість розробки та супроводу PWA у порівнянні з їх нативними аналогами, що пояснюється кросплатформеністю готового продукту

Створення прогресивного вебдодатку у порівнянні з нативним обходиться компаніям у середньому в 3-4 рази дешевше, у деяких випадках у 10-15 разів. Це пояснюється необхідністю розробки окремих додатків не тільки під різні платформи, а й версії операційних систем. Так вартість нативного додатку для App або Play Store починається від 10 тис. дол. США.

[3.10]

[3.10] - <https://kneu.edu.ua/userfiles/zb_mise/99/8.pdf>

PWA дозволяє значно зменшити розмір застосунку на мобільних додатках. На рисунку 3.7 приведено графічне порівняння розмірів застосунку “Pinterest” використовуючи нативні застосунки та PWA

[3.11]

[3.11] - https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154

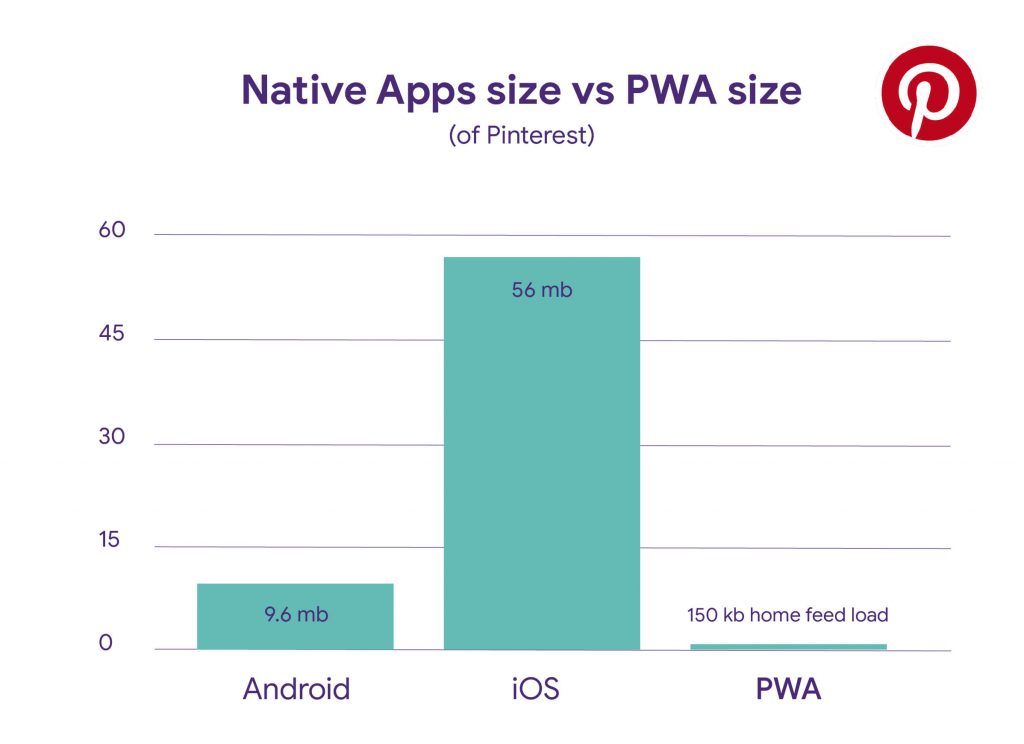


Рисунок 3.7 – Графічне порівняння розмірів застосунку “Pinterest” використовуючи нативні застосунки та PWA

Для коректної роботи клієнтської частини програми необхідно мати браузер, який має мінімальну версію не нижче 2020 року. Найбільш підходящими для використання є такі браузери, як Google Chrome, Safari, Mozilla Firefox, Opera та Microsoft Edge.

Рекомендується мати останню версію браузера, оскільки це забезпечує оптимальну швидкість роботи та захист від можливих вразливостей. Також рекомендується використовувати встановлені оновлення та плагіни для браузера для запобігання можливих проблем з безпекою.

Як можна бачити, згідно статистики популярності різних браузерів “Desktop Browser Market Share Worldwide Mar 2022 - Mar 2023”, від statcounter, яка зображена на рисунку 3.8 клієнтська частина програми має коректно працювати у всіх сучасних та найбільш використовуваних браузерах. [6]



Рисунок 3.8 – Статистика популярності різних браузерів в світі

Серверна частина програмного забезпечення працює на NodeJS - це засіб виконання JavaScript, який дозволяє розробникам створювати серверні додатки. Різні версії NodeJS мають різний рівень підтримки нових функцій та можуть мати відмінності в швидкості та стабільності роботи. Для того, щоб забезпечити правильну роботу сервісу, рекомендується використовувати одну з наступних LTS версій NodeJS: 18.15.0, 16.20.0 або 14.21.3.

Варто відзначити, що NodeJS на сьогодні є найпопулярнішою платформою для розробки клієнт-серверних додатків, оскільки вона дозволяє розробникам писати серверний код на JavaScript, що дозволяє швидко та ефективно створювати високоякісні додатки.

На рисунку 3.9 зображена статистика найбільш використовуваних веб-фреймворків та бібліотек за 2022 рік. Як можна бачити, NodeJS займає перше місце [7]



3.9 – Статистика найбільш використовуваних веб-фреймворків та бібліотек за 2022 рік

## 3.3 Основні рішення щодо реалізації компонентів системи

### 3.3.1 Використовувані моделі даних

Для розробки даного сервісу використовується документо-орієнтована модель даних. На рисунку 3.x представлений приклад документо-орієнтованої моделі

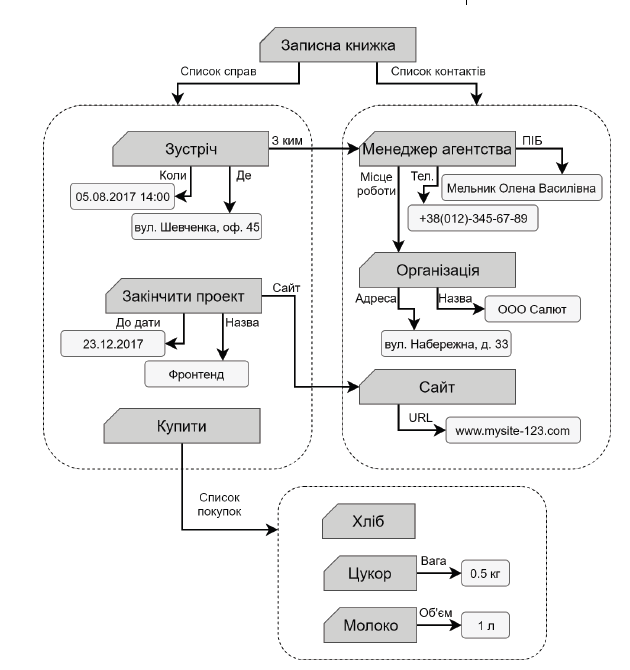


Рисунок 3.x - Приклад документо-орієнтованої моделі даних

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 [x]

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

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. [x]

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

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: “User”, “Statistic”, “Category”, “Color”, “CategoryGroup”

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

Спроектована документо-орієнтована модель зображена на рисунку 3.x

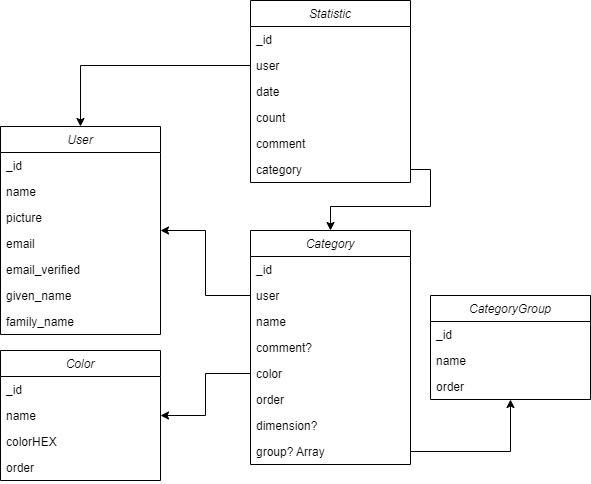


Рисунок 3.x - Спроектована модель даних

Документ «User» (\_id, name, picture, email, email\_verified, given\_name, family\_name) зберігає інформацію про користувачів. Опис документа «User» представлений в таблиці 3.x1, схема документа представлена на рисунку 3.xx1

Таблиця 3.x1 – Опис документа «User»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| name | string | Ім’я та призвіще користувача |
| picture | string | Аватар |
| email | string | Електрона пошта |
| email\_verified | boolean | Чи підтверджена пошта |
| given\_name | string | Ім’я користувача |
| family\_name | string | Призвіще користувача |

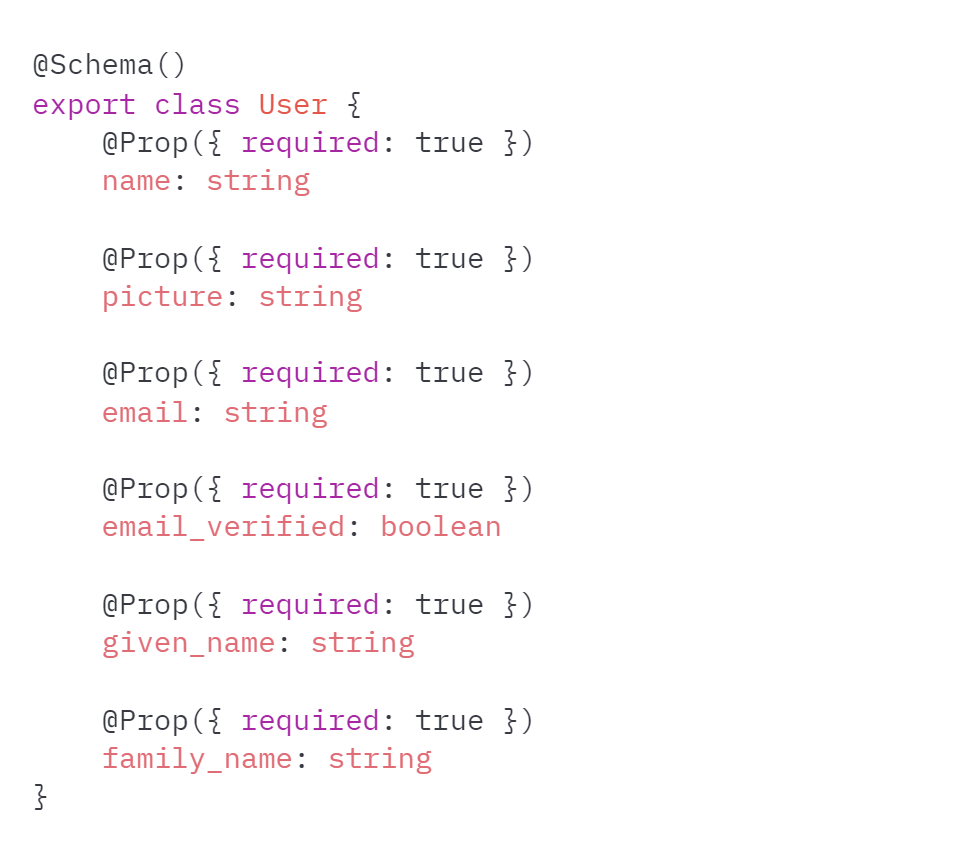


Рисунок 3.xx1 – Схема документа «User»

Документ «Statistic» (\_id, user, date, count, comment, category) зберігає інформацію про користувачів. Опис документа «Statistic» представлений в таблиці 3.x2, схема документа представлена на рисунку 3.xx2

Таблиця 3.x2 – Опис документа «Statistic»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| user | ObjectID | Користувач . Посилання на User |
| date | Date | Дата запису |
| count | number | Кількість |
| comment | string | Коментар |
| category | ObjectID | Категорія. Посилання на Category |

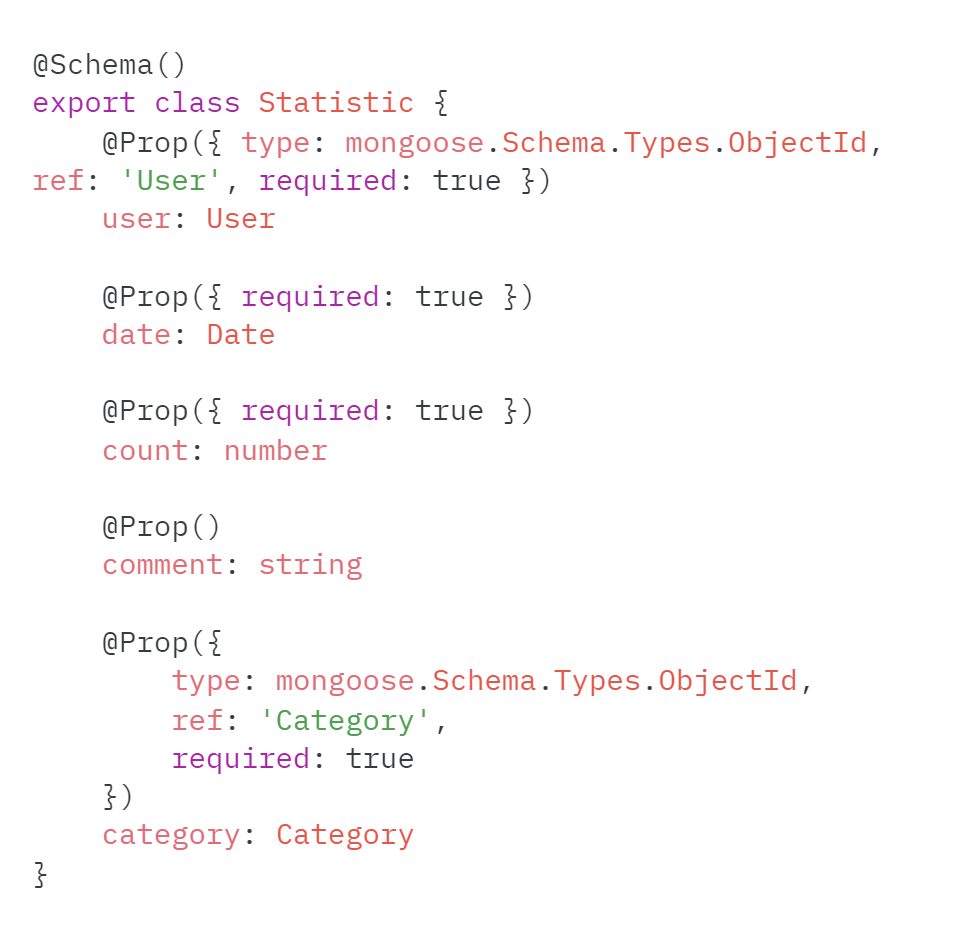


Рисунок 3.xx2 – Схема документа «Statistic»

Документ «Category» (\_id, user, name, comment, color, order, dimension, group) зберігає інформацію про користувачів. Опис документа «Category» представлений в таблиці 3.x3, схема документа представлена на рисунку 3.xx3

Таблиця 3.x3 – Опис документа «Category»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| user | ObjectID | Користувач. Посилання на User |
| name | string | Назва |
| comment | string | Кількість |
| color | ObjectID | Колір. Посилання на Color |
| order | number | Порядок категорії |
| dimension | string | Одиниці виміру |
| group | ObjectID | Група категорії. Посилання на CategoryGroup |

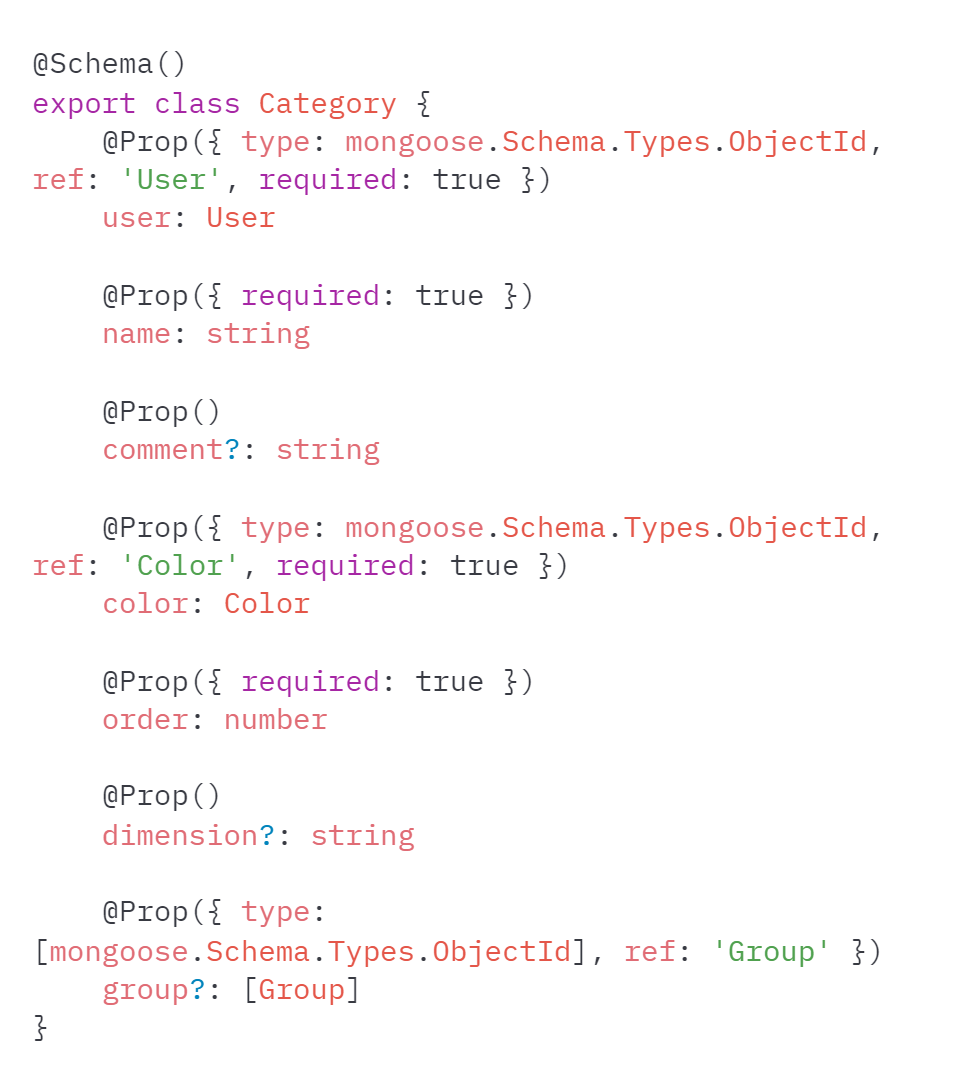


Рисунок 3.xx3 – Схема документа «Category»

Документ «Color» (\_id, name, colorHEX, order) зберігає інформацію про користувачів. Опис документа «Color» представлений в таблиці 3.x4, схема документа представлена на рисунку 3.xx4

Таблиця 3.x4 – Опис документа «Color»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| name | string | Назва коліру |
| colorHEX | string | Колір в HEX форматі |
| order | number | Порядок коліру |



Рисунок 3.xx4 – Схема документа «Color»

Документ «Color» (\_id, name, colorHEX, order) зберігає інформацію про користувачів. Опис документа «Color» представлений в таблиці 3.x4, схема документа представлена на рисунку 3.xx4

Таблиця 3.x4 – Опис документа «Color»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| name | string | Назва коліру |
| colorHEX | string | Колір в HEX форматі |
| order | number | Порядок коліру |



Рисунок 3.xx4 – Схема документа «Color»

Документ «CategoryGroup» (\_id, name, order) зберігає інформацію про користувачів. Опис документа «CategoryGroup» представлений в таблиці 3.x5, схема документа представлена на рисунку 3.xx5

Таблиця 3.x5 – Опис документа «CategoryGroup»

|  |  |  |
| --- | --- | --- |
| Назва ключа | Тип значення | Опис |
| \_id | ObjectID | Унікальне поле |
| name | string | Назва групи категорій |
| order | number | Порядок групи категорій |

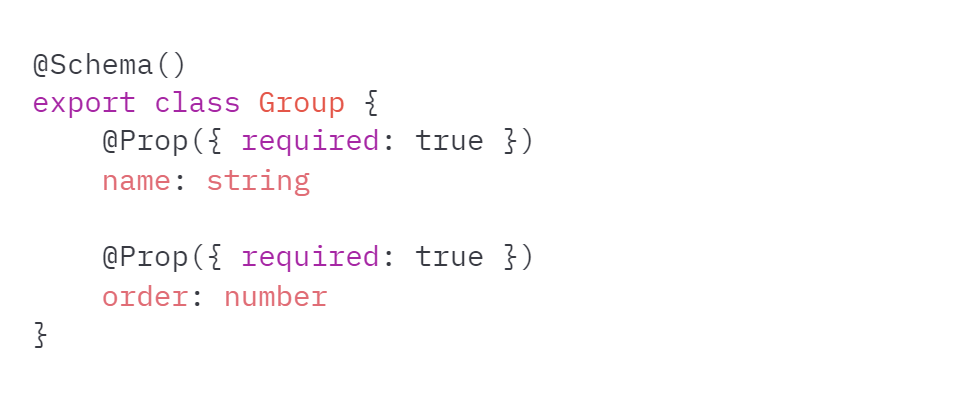


Рисунок 3.xx5 – Схема документа «CategoryGroup»

### 3.3.2 Структурна схема програми

### 3.3.3 Розробка модулів системи

# 4 Методика роботи користувача з системою

## 4.1 Керівництво програміста

### 4.1.1 Призначення і умови використання програми

### 4.1.2 Характеристики програми

### 4.1.3 Звертання до програми (файл проекту)

### 4.1.4 Вхідні і вихідні дані

## 4.2 Керівництво оператора

### 4.2.1 Призначення і умови використання програми

### 4.2.2 Виконання програми

### 4.2.3 Повідомлення оператору

# 5 Організаційно – економічний розділ

## 5.1 Планування розробки програмного продукту

## 5.2 Розрахунок витрат на розробку програмного продукту

### 5.2.1 Складання кошторису витрат на розробку

### 5.2.2 Розрахунок собівартості програмного продукту

## 5.3 Оцінка ефективності проекту

# 6 Охорона праці користувачів комп’ютерів

## 6.1 Правове збезпечення заходів щодо охорони праці користувачів комп’ютерів

## 6.2 Електробезпека та пожежобезпека у приміщеннях з персональними комп’ютерами

## 6.3 Причини виникнення, загальна характеристика та класифікація надзвичайних ситуацій

# Висновки

# Список літератури

[1] - <https://enigma.ua/articles/informatsiyniy_vibukh_i_globalizatsiya_svitovoi_politiki>

[2] - <https://eprints.kname.edu.ua/10895/1/%D0%A1%D0%B8%D1%81%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_1_8%D0%BD.pdf>

[3] - <http://www.rusnauka.com/40_OINBG_2014/Informatica/3_182487.doc.htm>

[4] - https://er.nau.edu.ua/bitstream/NAU/46983/1/%D0%A4%D0%9A%D0%9A%D0%9F%D0%86\_2020\_125\_%D0%9A%D0%BE%D0%BB%D1%8F%D0%BA%D0%B0%D0%90%D0%92.pdf

[5] - <http://deep.kiev.ua/attachments/093_2016d_Kravchuk.pdf>

[6] - https://gs.statcounter.com/browser-market-share/desktop/worldwide

[7] - <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>

# Додатки