== 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.

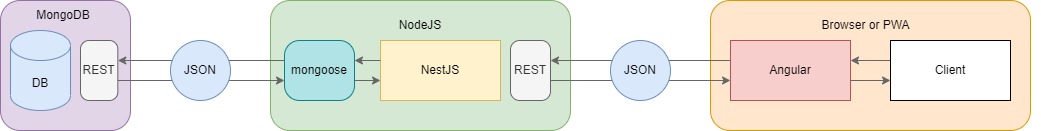
Так як під впливом Інтернету, технологічних інновацій і сервісів архітектура додатку стала більш розмитою, трьохшарова модель додатку розвинулася у багаторівневу архітектуру

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

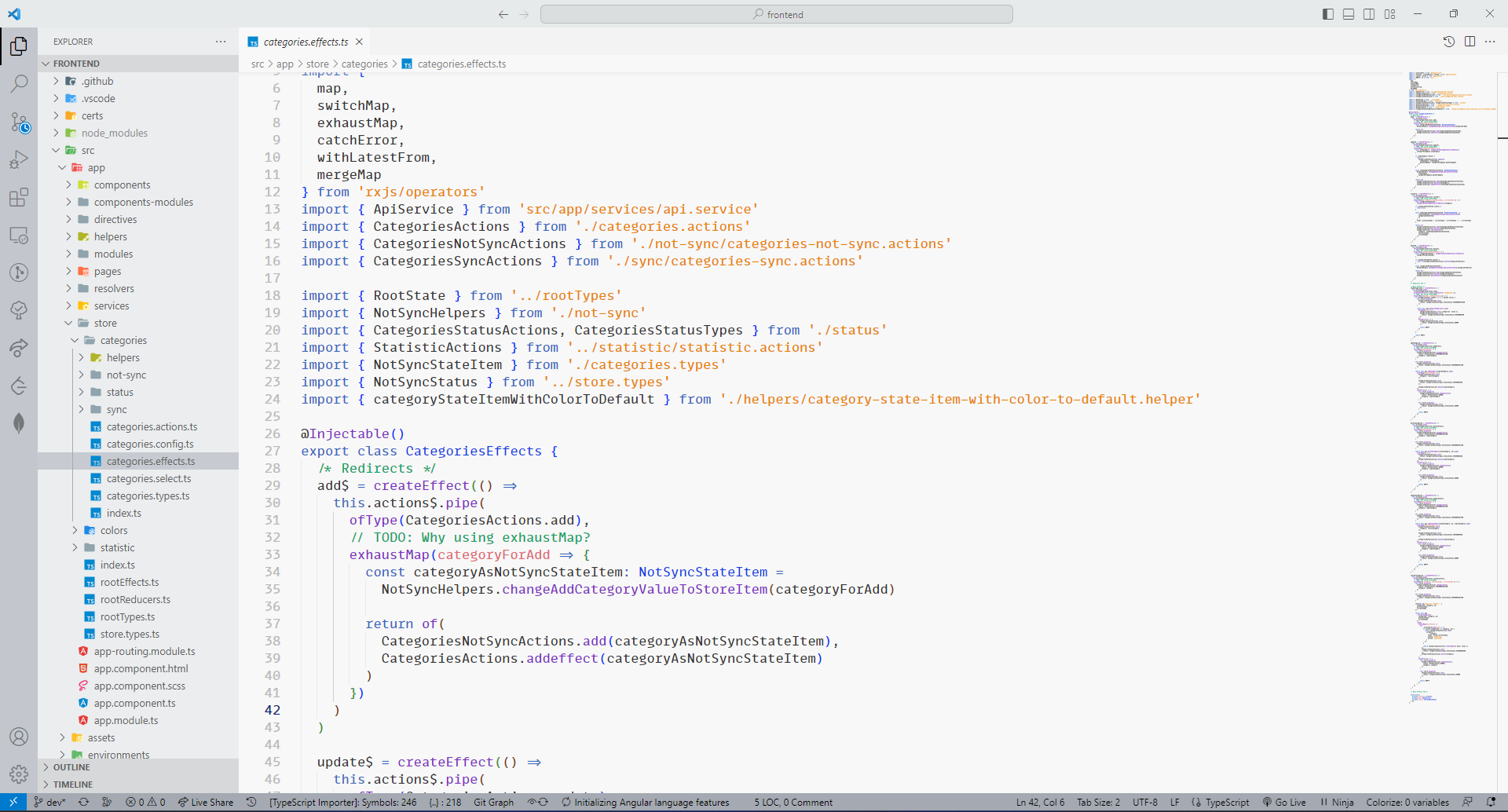
Даний програмний продукт є безкоштовним і доступним за допомогою мережі Інтернет. Щоб мати можливість користуватися програмою, користувачеві необхідно увести в адресний рядок свого браузера наступну адресу: "https://counter-ltlaitoff.vercel.app". Ця адреса є посиланням на веб-сторінку, на якій розміщений 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

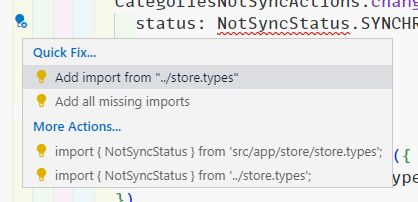
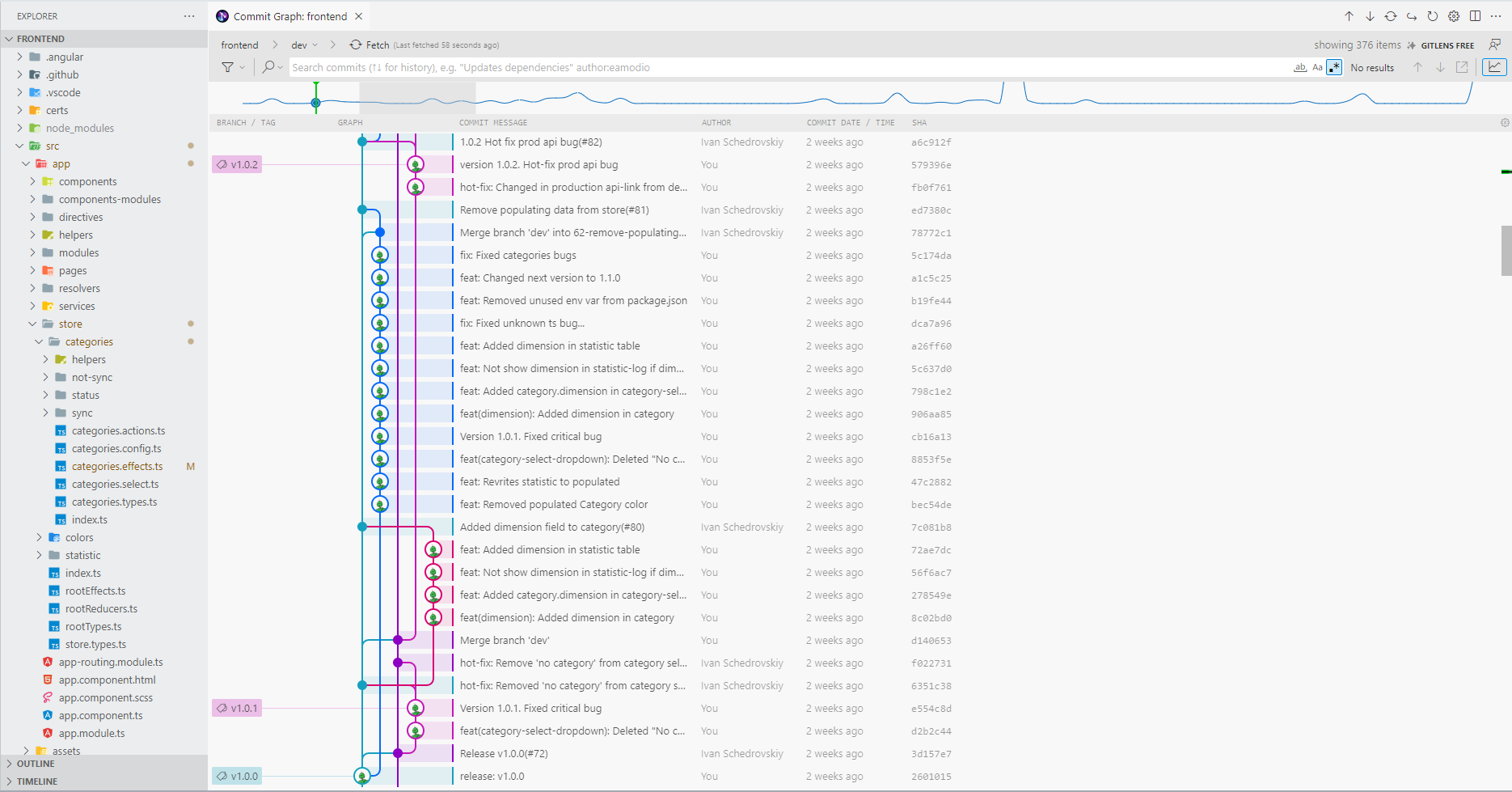


Рисунок 3.3 – Демонстрація контекстних підказок

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]

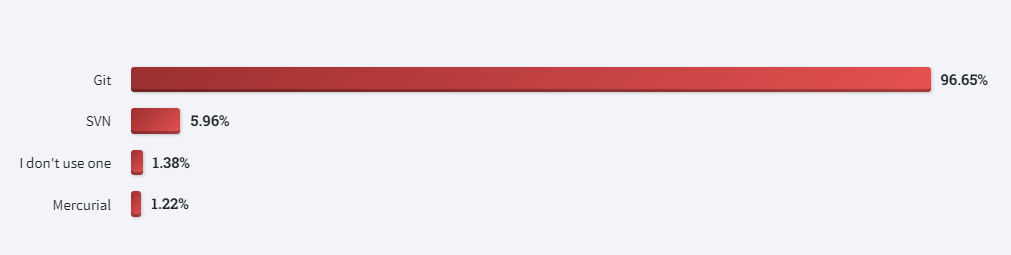
[3.2] - <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

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

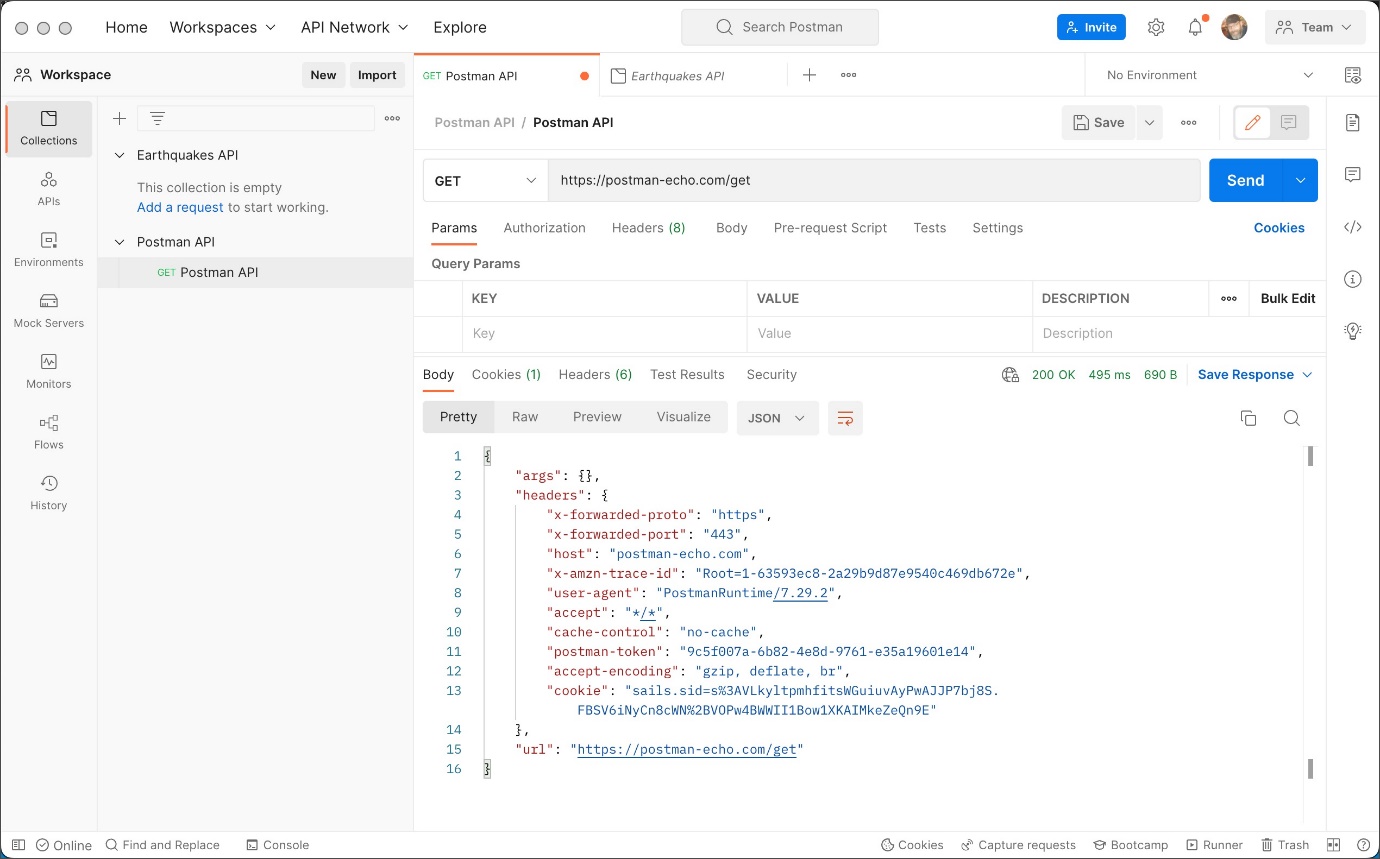


Рисунок 3.6 – Вигляд головного вікна Postman

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

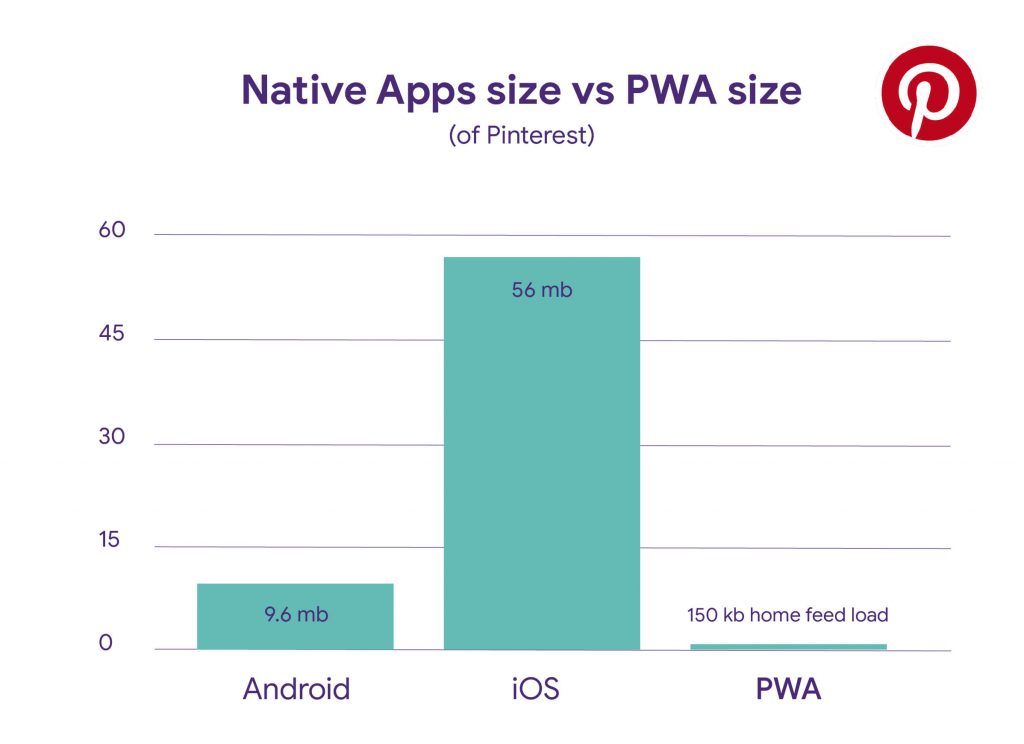


Рисунок 3.7 – Графічне порівняння розмірів застосунку “Pinterest” використовуючи нативні застосунки та PWA

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 представлений приклад документо-орієнтованої моделі

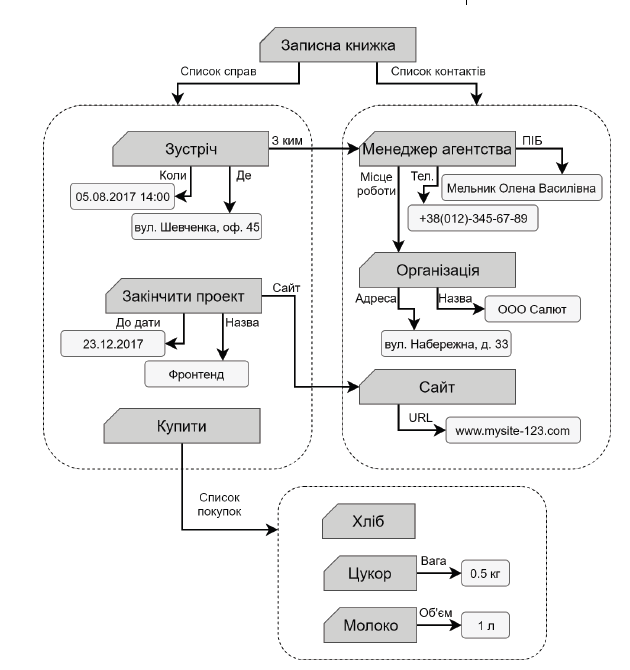


Рисунок - Приклад документо-орієнтованої моделі даних

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

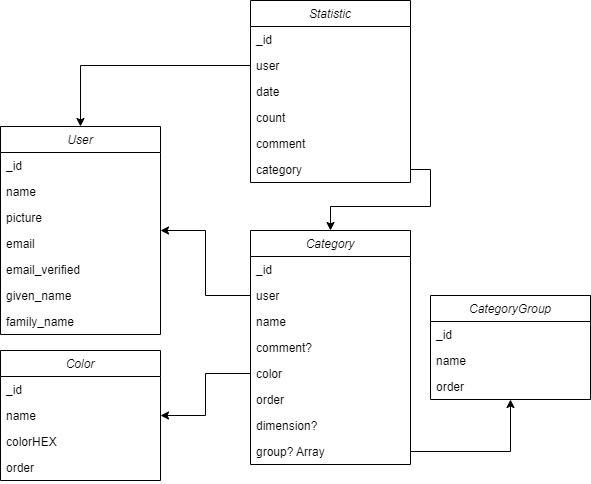


Рисунок - Спроектована модель даних

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 |

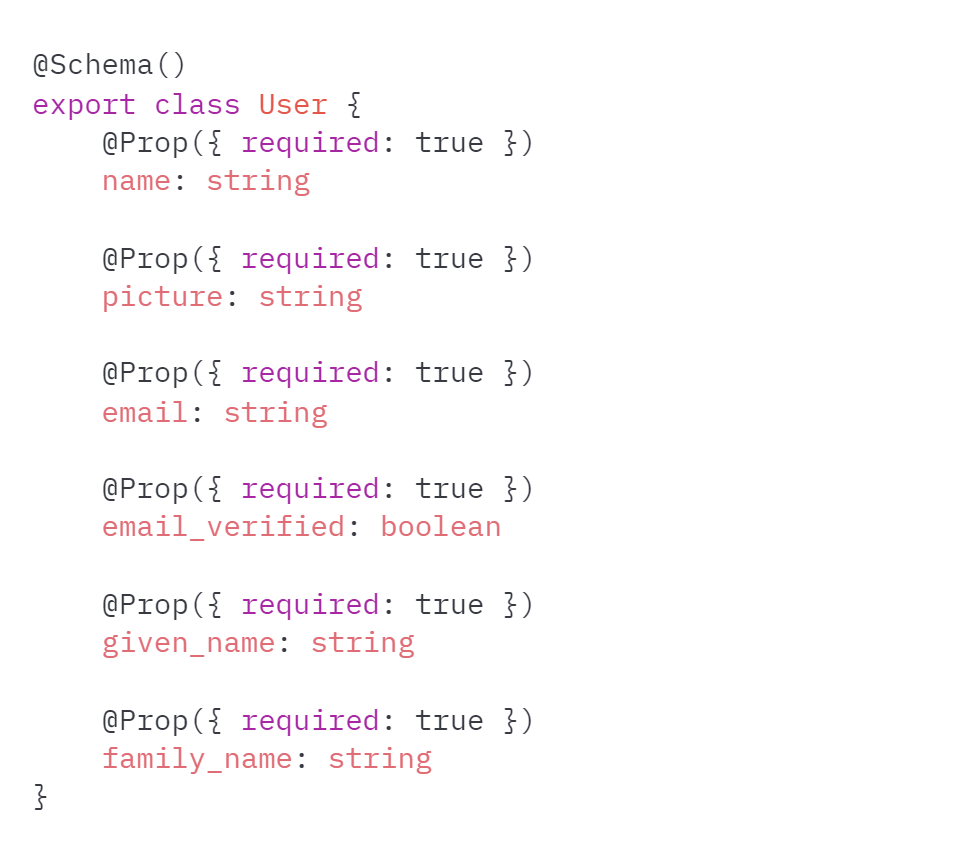


Рисунок x1 – Схема документа «User»

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 |

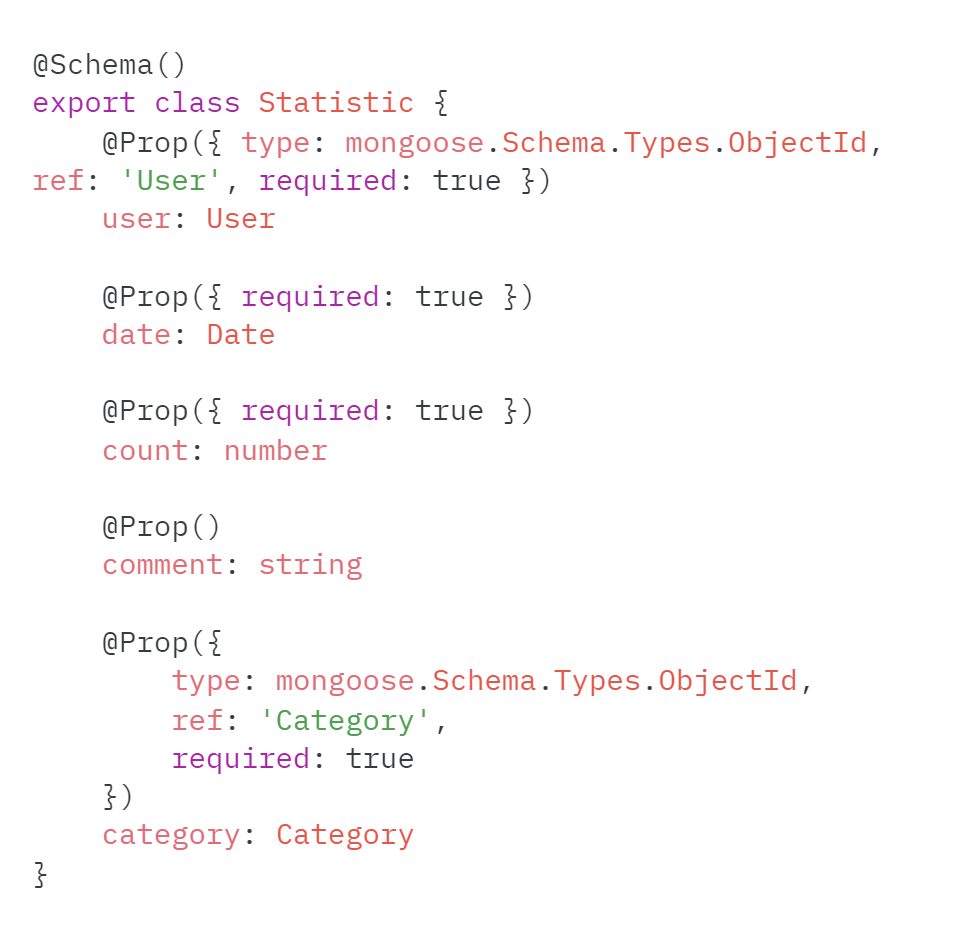


Рисунок x2 – Схема документа «Statistic»

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 |

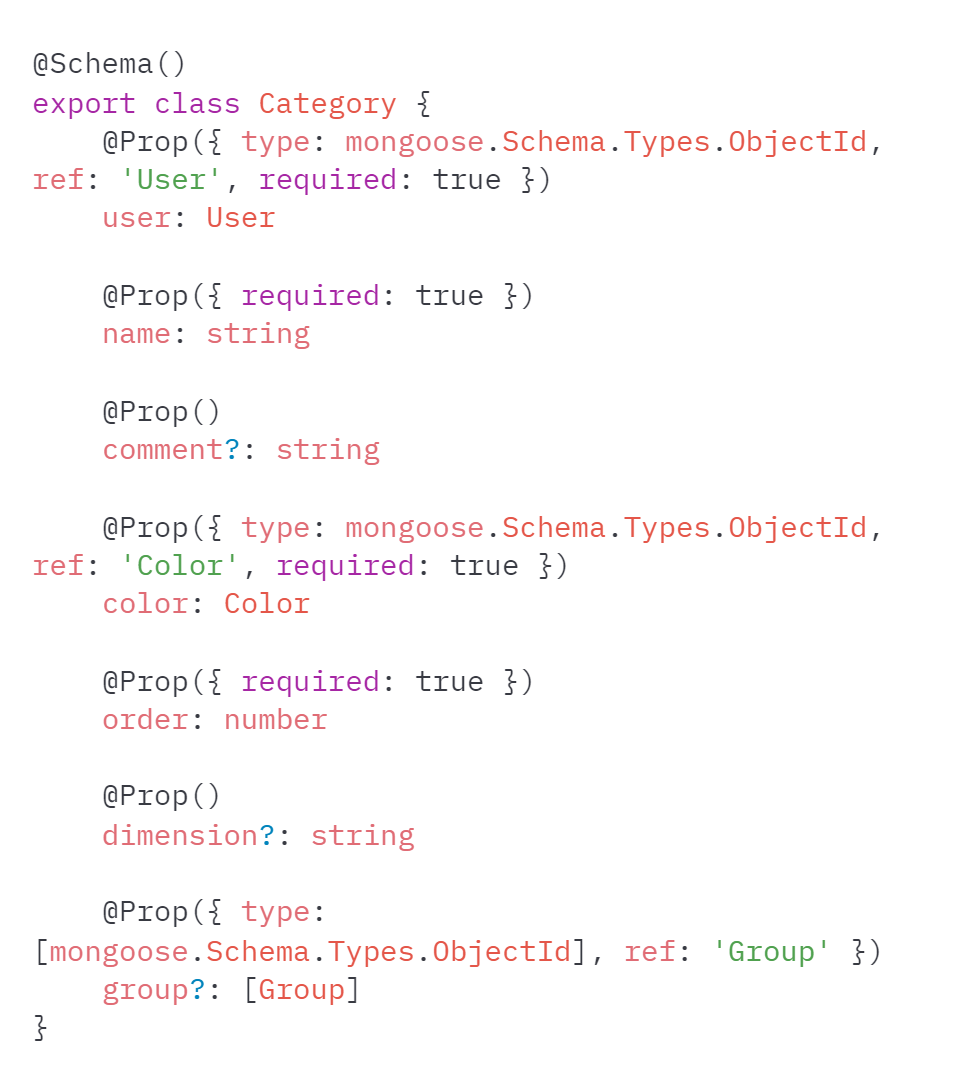


Рисунок x3 – Схема документа «Category»

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 x4 – 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 x4 – Схема документа «Color»

Документ «CategoryGroup» (\_id, name, order) зберігає інформацію про користувачів. Опис документа «CategoryGroup» представлений в таблиці 5, схема документа представлена на рисунку x5

Таблиця 5 – Опис документа «CategoryGroup»

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

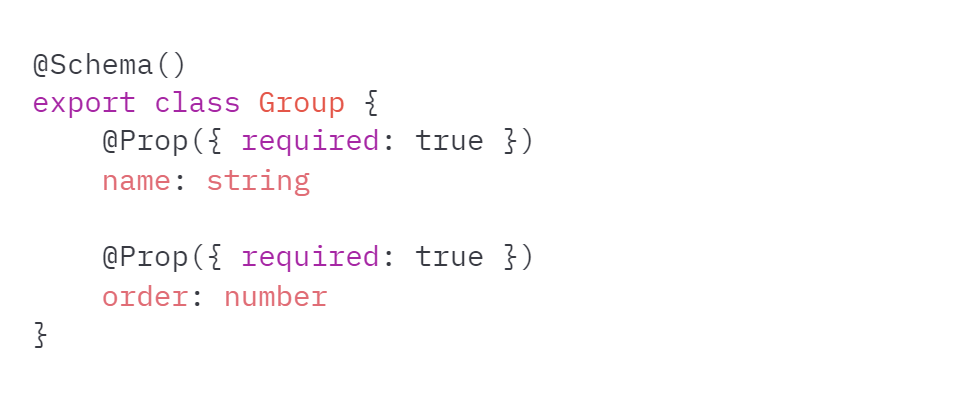


Рисунок x5 – Схема документа «CategoryGroup»

### 3.3.2 Структурна схема програми

Загальна схематична система, зображена на рисунку , складається з:

* MongoDB – База даних
* Сервер – програма, яка приймає та відправляє запити від клієнта, працює з базою даних. Операції між сервером та базою, а також сервером та клієнтом відбуваються за допомогою Rest API
* Клієнт – користувач, який взаємодіючи з сервісом відправляє та отримує запити від сервера

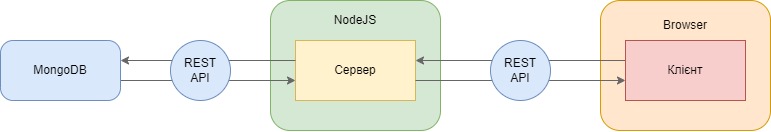


Рисунок – Загальне схематичне зображення роботи сервісу

Повна схематична система, зображена на рисунку складається з:

* MongoDB – База даних
* Mongoose – Пакет для роботи з базою даних
* NestJS – Фреймворк для побудови API
* Angular – Фреймворк для клієнтської частини

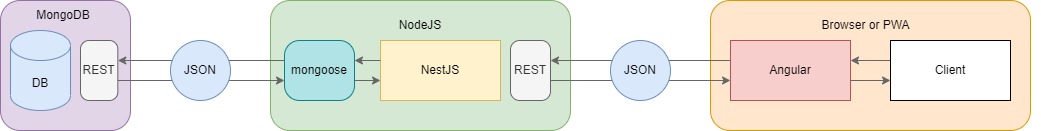


Рисунок – Повне схематичне зображення роботи сервісу

### 3.3.3 Розробка модулів системи

Основною та єдиною мовою програмування доступною в браузері є JavaScript, тому її використання для розробки сервісу є логічним і необхідним

JavaScript – 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

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.

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. [x]

[x] - <https://developer.mozilla.org/ru/docs/conflicting/Web/JavaScript>

Оскільки JavaScript динамічно-типізована мова програмування її не завжди зручно використовувати без доповнень та надбудов в розробці програмних застосунків. Для більш комфортної, продуктивної та якіснішої розробки застосунку було обрано TypeScript

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.

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. [x]

[x] - <https://www.typescriptlang.org/>

В силу повної зворотної сумісності адаптація наявних застосунків з JavaScript на TypeScript може відбуватися поетапно, шляхом поступового визначення типів. Підтримка динамічної типізації 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 [x]

[x] - octoverse.github.com

На рисунку 2.4 представлено графік популярності мов програмування

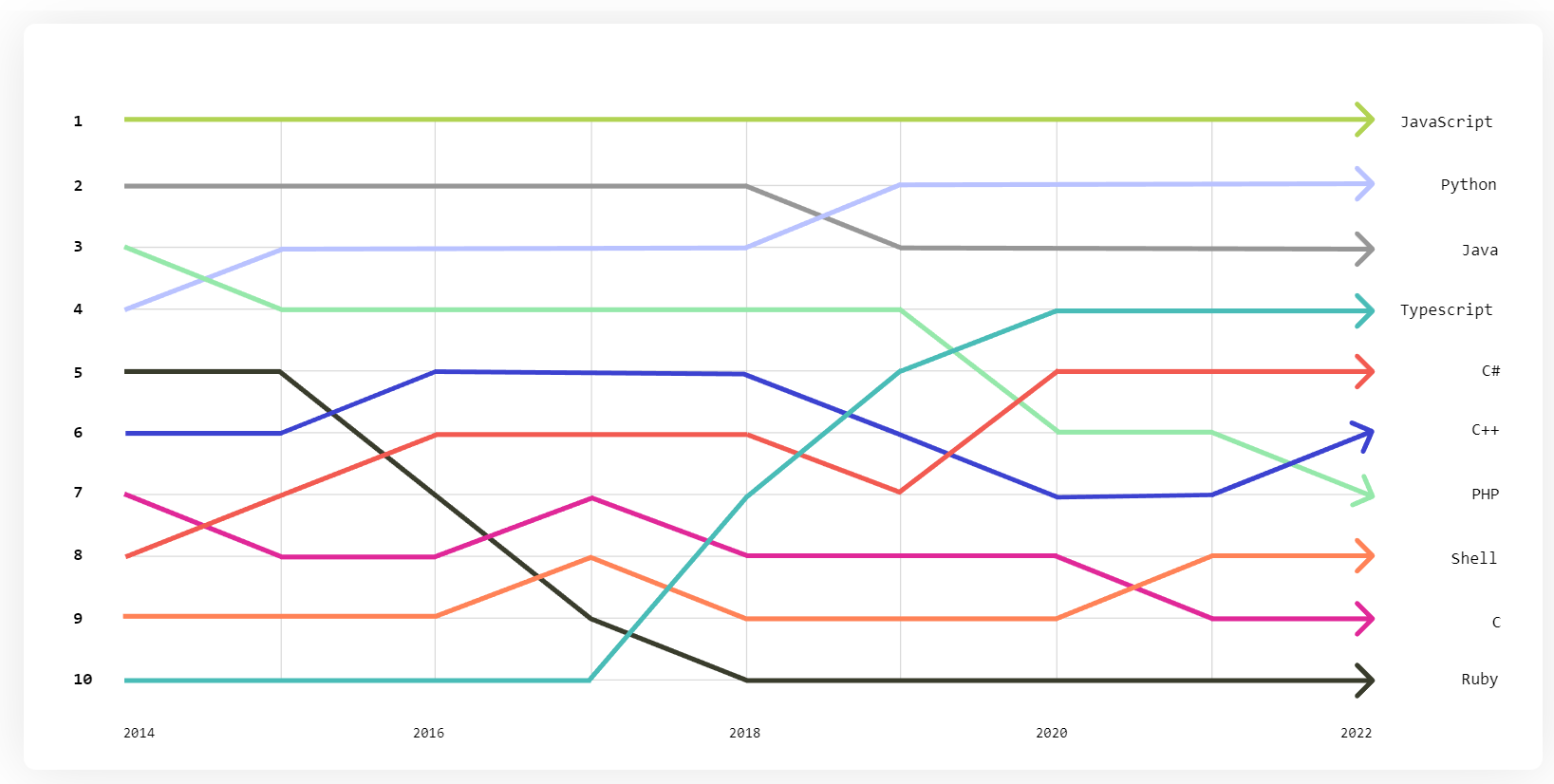


Рисунок 2.4 - Графік трендів мов програмування

Оскільки TypeScript перетворюється на JavaScript, який виконується будь-де, де працює JavaScript, а JavaScript може виконуватись як на клієнті за допомогою браузера, так і на сервері за допомогою, наприклад, NodeJS для більш ефективної розробки та перевикористання коду TypeScript було обрано як мову програмування і для серверної частини також.

Платформою для розробки серверної частини було обрано NodeJS

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. [x]

[x] - https://nodejs.org/en/about/

Для швидкого тестування застосунку в різних версіях NodeJS було обрано Node Version Manager.

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. [x]

[x] - https://github.com/nvm-sh/nvm

Для управління залежностями NodeJS було обрано pnpm який основний над Node Package 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. [x]

[x] - https://www.npmjs.com/about

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 представлено візуалізацію роботи PNPM

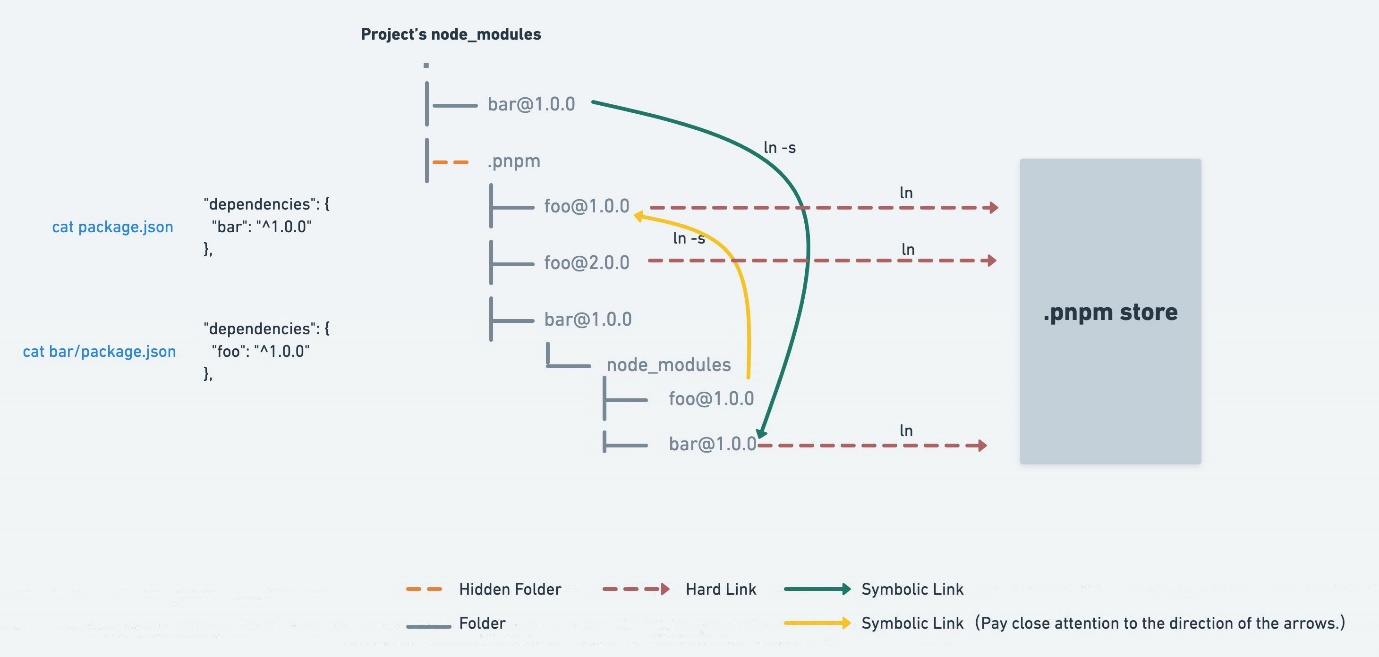


Рисунок - Візуалізація роботи pnpm

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. [x]

[x] - <https://pnpm.io/motivation>

Для написання будь-якого веб-сайту, чи то сервісу або ж додатку в будь-якому випадку використовується мова гіпертекстової розмітки.

Мова гіпертекстової розмітки (далі HTML) – найбільш фундаментальний будівельний блок для всього вебу. HTML визначає значення та структуру веб-контенту. Інші технології окрім HTML, як правило, використовуються для опису вигляду / презентації веб-сторінки (CSS) або функціональності / поведінки (JavaScript).

"Гіпертекст" відноситься до посилань, які з'єднують веб-сторінки між собою, як в межах одного веб-сайту, так і між веб-сайтами. Посилання є фундаментальним аспектом Інтернету. Завантажуючи вміст в Інтернет та посилаючись на сторінки, створені іншими людьми, ви стаєте активним учасником всесвітньої павутини.

HTML використовує "розмітку" для анотування тексту, зображень та іншого контенту для відображення веб-браузером. [x]

[x] - <https://developer.mozilla.org/en-US/docs/Web/HTML>

TODO: CSS, SCSS

Для розробки клієнтської частини сервісу було обрано frontend-фреймворк Angular

Фронтенд (далі frontend) – це публічна частина web-додатків (веб-сайтів), з якою користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. Frontend – це все те, що бачить користувач при відкритті web-сторінки. [x]

[x] - <https://dan-it.com.ua/uk/blog/rozrobka-z-boku-front-end-shho-ce-take-i-chim-vidriznjaietsja-vid-back-end/>

Фреймворк (далі framework) — це програмна оболонка, яка дозволяє спростити і прискорити вирішення типових завдань, властивих конкретній мові програмування. Фреймворк використовують, оскільки це зручно. Набагато простіше створювати проект, оскільки він уже має свою структуру.

Головна ціль фреймворку, дати розробнику комфортне середовище для проекту з великим та масштабованим функціоналом.

Framework використовує патерн проектування MVC (model-view-controller – «Модель-подання-контролер»). MVC – це обов’язкова умова для організації коду чи компонентів. Завдання MVC – вирішити проблему проектування, що виникла у робочому рішенні. На основі MVC створено різні види фреймворків, наприклад Angular або ж Vue.js. [x]

[x] - https://tonyline.com.ua/glossary/framework/

Angular - це фреймворк для дизайну та розробки додатків, який дозволяє створювати ефективні та складні односторінкові(SPA) застосунки. [x]

[x] - https://angular.io/docs

Односторінкові застосунки, або Single Page Application (далі SPA) - це реалізація веб-додатку таким чином, щоб він завантажував лише один веб-документ, а потім оновлював вміст тіла цього одного документа за допомогою JavaScript коли потрібно показати різний вміст.

Це дозволяє користувачам використовувати веб-сайти, не завантажуючи цілком нові сторінки з сервера, що може призвести до покращення продуктивності та більш швидшого відклику сайту, але при цьому виникає деякий недолік, такий як погіршення SEO, більше зусиль для збереження стану застосунку, реалізації навігації та проведення вимірювань продуктивності. [x]

[x] - https://developer.mozilla.org/en-US/docs/Glossary/SPA

На рисунку зображена візуалізація традиційного підходу та SPA щодо життєвого циклу веб-сторінки [x]

[x] - https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya

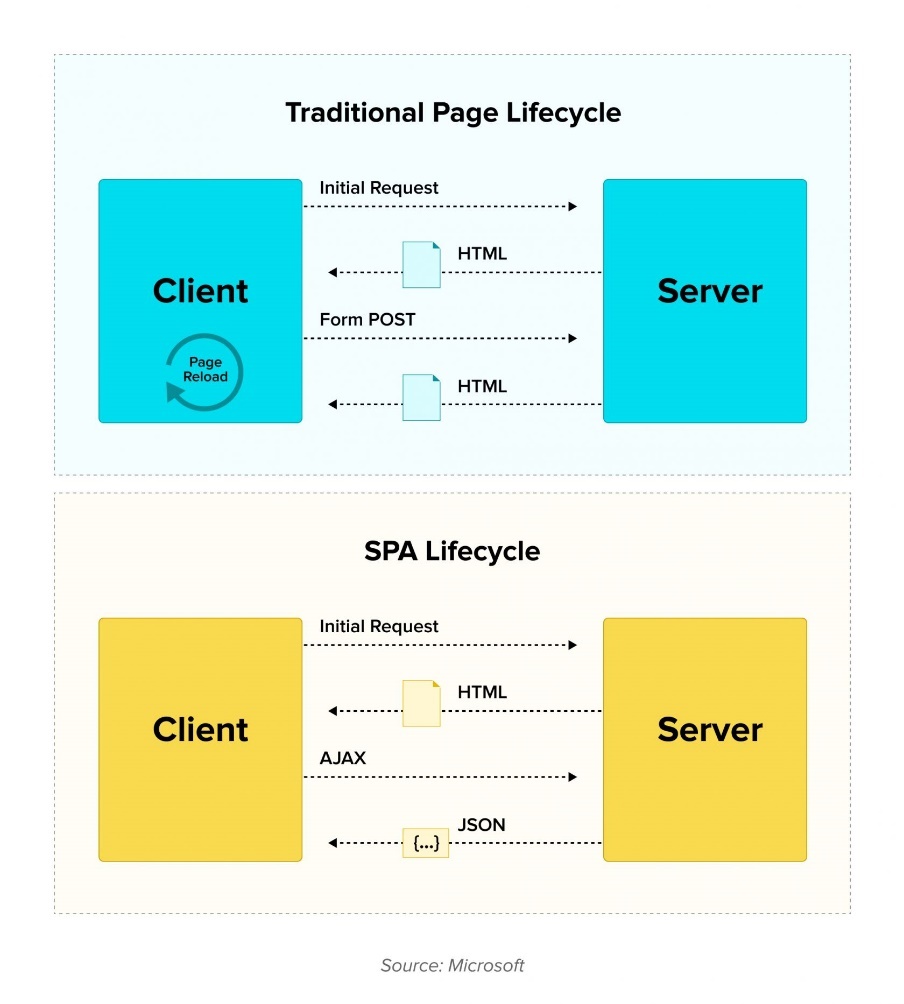


Рисунок – Візуалізація роботи традиційного підходу та SPA щодо життєвого циклу веб-сторінки

Angular має, в порівнянні з React та Vue, деякі відмінності.

React – бібліотека JavaScript для створення користувацьких інтерфейсів. React використовує декларативний, базований на компонентах, підход для створення інтерфейсів. [x]

[x] - https://legacy.reactjs.org/

Vue - це фреймворк, який працює на JavaScript, створений для розробки користувацьких інтерфейсів. Він працює на базі звичайного HTML, CSS та JavaScript, з можливостями декларативно програмувати користувацькі інтерфейси будь-якої складності на основі компонентів. [x]

[x] - <https://ua.vuejs.org/guide/introduction.html#what-is-vue>

На рисунку зображений приклад коду однофайлового компоненту на Vue

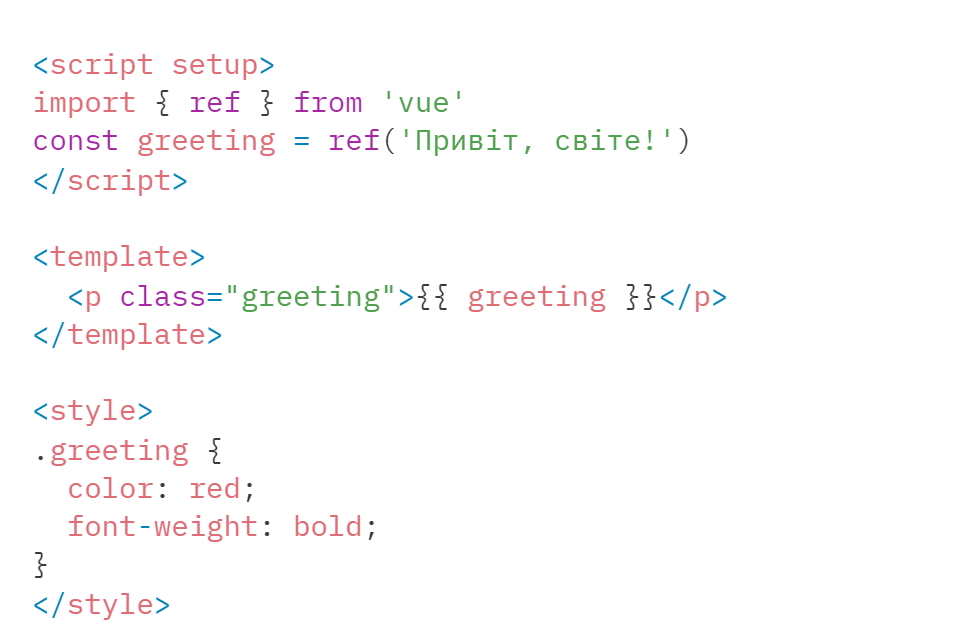


Рисунок – Приклад однофайлового компоненту на фреймворку Vue

JSX — це синтаксичне розширення для JavaScript, яке дозволяє писати HTML-подібну розмітку всередині файлу JavaScript. Хоча існують інші способи написання компонентів, більшість розробників React віддають перевагу лаконічності JSX, і більшість кодових баз використовують його. [x]

[x] - <https://react.dev/learn/writing-markup-with-jsx>

На рисунку 3.x зображений код для створення React компоненту використовуючи JSX



Рисунок – Приклад коду створення компоненту для бібліотеки React використовуючи JSX

Angular є html-first фреймворком, тобто розробник спочатку пише html, а потім вставивє в нього директиви Angular та описує за допомогою TypeScript як воно повинно працювати. Vue також є html-first фреймворком.

Приклад html-коду для Angular зображений на рисунку 3.x.



Рисунок – Приклад html-коду для фреймворку Angular

React же є javascript-first бібліотекою. Доказом цього є приклад коду, який зображений на рисунку 3.x написаного для React без використання JSX. Як видно з зображення це чистий javascript код який викликає функції бібліотеки React напряму без написання html, а лише вказуючи назви тегів які потрібно створити.



Рисунок – Приклад коду для бібліотеки React написаного без використання JSX

Прив'язка даних (Data binding) - це процес зв'язку двох джерел даних та їх синхронізації. З допомогою прив'язки даних, зміна елементу в наборі даних автоматично оновлюється в пов'язаному наборі даних.

Прив'язка з одного боку або одностороння прив’язка (One-way binding) - це відносно простий тип прив'язки даних. Зміни, які відбуваються в постачальнику даних, автоматично оновлюються у споживачах даних, але не навпаки.

Прив'язка з двох боків або двостороння прив’язка (Two-way binding) - це випадок, коли зміни, які відбуваються або в постачальнику даних, або в споживачах даних, автоматично оновлюють інші джерела даних. [x]

[x] - <https://www.techtarget.com/whatis/definition/data-binding>

Angular підтримує обидва види прив’язки даних, та має три типи:

* Одностороння прив’язка даних від джерела даних до цільового перегляду, використовується для передачі параметрів(Inputs) компонентам та для відображення змінних на сторінці
* Одностороння прив’язка даних від відображаємої цілі до джерела даних. Використовується для деяких подій javascript та подій компонентів(Outputs).
* Двостороння прив’язка даних. [x]

[x] - <https://angular.io/guide/binding-syntax#types-of-data-binding>

Vue, як і Angular, підтримує обидва види прив’язки даних

React же підтримує тільки одно-сторонню прив’язку даних, від джерела даних до цільового перегляду або ж компонентам(props). Тому для отримання даних від користувача в React використовують стандартні javascript правила.

Великою перевагою Angular над React та Vue є Angular CLI.

Angular CLI – текстовий інтерфейс користувача, який надає можливість з легкістю створювати додаток, запускати, тестувати та створювати нові компоненти, сервіси, директиви та інше. Це відповідає кращим сучасним тенденціям та є дуже зручним при створені хоч скільки то великого проекту.

На рисунку 3.x представлена консольна команда Angular CLI для генерація модуля Graph



Рисунок – Консольна команда Angular CLI для генерація модуля Graph

Декоратор — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки». [x]

[x] - https://refactoring.guru/uk/design-patterns/decorator

React компонентом є одна функція, або ж класс.

Vue компонентом є єдиний файл.

Angular компонентом є класс з декоратором ‘@Component(…)’ в якому розробник може вказати наступні параметри(основні які найбільш часто використовуються):

* selector - назва за якою інші компоненти зможуть використовувати цей
* template або ж templateUrl – HTML-шаблон або ж посилання на файл в якому цей шаблон знаходиться
* styleUrls або ж styles – посилання на файли стилів або ж масив стилів в вигляді строки

На рисунку представлений приклад стандартного Angular компоненту



Рисунок – Приклад стандартного Angular компоненту

Однією з головних відмінностей Angular від React та Vue є наявність різних типів декораторів.

Якщо в React є функції-компоненти React, які містять в собі JSX та використовують React, а також всі інші звичайні JavaScript функції. Або ж в Vue є одно файлові компоненти та всі інші звичайні JavaScript функції.

Впровадження залежностей (DI) — це частина фреймворку Angular, яка надає компонентам доступ до сервісів та інших ресурсів. Angular надає розробникам можливість вставити службу в компонент, щоб надати цьому компоненту доступ до служби.

Декоратор ‘@Injectable()’ визначає клас як службу в Angular і дозволяє Angular вставляти його в компонент як залежність. Подібним чином декоратор ‘@Injectable()’ вказує, що компонент, клас, канал або NgModule мають залежність від служби.

Для будь-якої залежності, яка вам потрібна у вашій програмі, ви повинні зареєструвати постачальника в інжекторі програми, щоб інжектор міг використовувати постачальника для створення нових екземплярів. Для послуги постачальник зазвичай є самим класом послуги. [x]

[x] - https://angular.io/guide/architecture-services#dependency-injection-di

В Angular є досить велика кількість різних типів класів, а оскільки стандартом в програмуванні є один клас на файл дуже зручно створювати красиву, зручну, а саме головне ефективну для розробки структуру застосування. Наприклад, в Angular є наступні типи класів:

* Module – ‘@NgModule’ – Модулі в Angular – це контейнери для об’єднання коду в блоки, призначеного для домену програми, робочого процесу або тісно пов’язаного набору можливостей. Вони можуть містити компоненти, постачальників послуг та інші кодові файли, область дії яких визначається NgModule, що містить. Вони можуть імпортувати функції, які експортуються з інших NgModules, і експортувати вибрані функції для використання іншими NgModules. [x]

[x] - <https://angular.io/guide/architecture-modules>

* Component – ‘@Component’ - Компоненти є основним будівельним блоком для додатків Angular. [x]

[x] – https://angular.io/guide/component-overview

* Service – ‘@[Injectable](https://angular.io/api/core/Injectable)’ – Дозволяє створити сервіс, який потім можна використовувати в іншому сервісі, компоненті, експортувати в модулі для використання в інших модулях та інше.
* Pipe – ‘@[Pipe](https://angular.io/api/core/Pipe)’ - Клас, якому передує декоратор ‘@Pipe{}’ і який визначає функцію, яка перетворює вхідні значення у вихідні значення для відображення у поданні. Angular визначає різні pipe, і розробник може створити нові. [x]

[x] – https://angular.io/guide/glossary#pipe

* Directive – ‘@Directive’ – Директиви атрибути в Angular дозволяють змінювати зовнішній вигляд або поведінку елементів DOM і компонентів Angular. [x]

[x] – https://angular.io/guide/attribute-directives

В більшості своєму Angular використовується для створення великих багато навантажених проєктів, наприкад Google, які і є розробниками Angular, саме тому цей фреймворк має всі необхідні інструменти для швидкої та зручної розробки.

Для пришвидшення розробки стилізації сервісу було обрано Tailwindcss

Tailwindcss – перший утилітарний  фреймворк CSS наповнений класами, за допомогою яких можна створити будь-який дизайн безпосередньо в мові гіпертекстової розмітки.

Tailwindcss дозволяє швидко створювати сучасні веб-сайти, не відриваючись від HTML [x]

[x] - <https://tailwindcss.com/>

На рисунку відображений приклад використання tailwindcss для оформлення кнопки «Купити зараз»

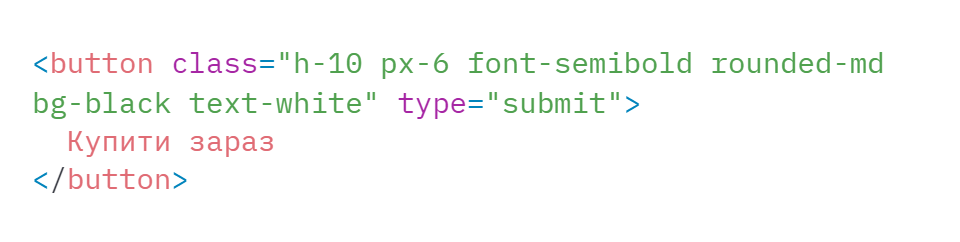


Рисунок – Приклад стандартного Angular компоненту

Для більш зручної роботи з реактивними даними, тобто даними, що можуть асинхронно змінюватись з плином часу, а також для роботи бібліотеки яка займається загальним станом всього сервісу, було взято бібліотеку RxJs.

RxJS — це бібліотека для реактивного програмування з використанням Observables, щоб полегшити створення асинхронного коду або коду на основі зворотного виклику.

RxJS є переписаним Reactive-Extensions/RxJS із кращою продуктивністю, кращою модульністю, кращими стеками викликів із можливістю налагодження, зберігаючи в основному зворотну сумісність із деякими критичними змінами, які зменшують поверхню API.

RxJS — це бібліотека для створення асинхронних і програм які базуються на подіях за допомогою спостережуваних послідовностей.

RxJS надає один основний тип, Observable, супутникові типи (Observer, Schedulers, Subjects) і оператори, натхненні методами Array (map, filter, reduce, every тощо), щоб дозволити обробку асинхронних подій як колекцій.

ReactiveX поєднує шаблон проектування Observer із шаблоном проектування Iterator і функціональне програмування з колекціями, щоб задовольнити потребу в ідеальному способі керування послідовністю подій. Основні концепції в RxJS, які вирішують асинхронне керування подіями:

Observable(Спостережувані) - представляє ідею викликаної колекції майбутніх значень або подій.

Observer(Спостерігач) - це набір зворотних викликів, який знає, як прослуховувати значення, надані Observable.

Subscription(Підписка) - представляє виконання Observable, в першу чергу корисна для скасування виконання.

Operators(Оператори) - це чисті функції, які забезпечують функціональний стиль програмування роботи з колекціями за допомогою таких операцій, як map, filter, concat, reduce тощо.

Subject(Предмет) - еквівалент Angular EventEmitter і єдиний спосіб групової передачі значення або події кільком спостерігачам.

Schedulers(Планувальники) - це централізовані диспетчери для керування паралельністю, що дозволяє нам координувати, коли обчислення відбуваються, наприклад, setTimeout або requestAnimationFrame або інші.[x]

[x] - <http://reactivex.io/rxjs/manual/overview.html>

На рисунку зображений приклад роботи RxJs.



Рисунок – Приклад роботи RxJs

Для зберігання глобального стану сервісу було обрано бібліотеки @ngrx/store разом з @ngrx/effects та допоміжну @ngrx/store-devtools для тестування та відладки.

@ngrx/store забезпечує керування станом для створення супроводжуваних явних додатків за допомогою використання єдиного стану та дій для вираження змін стану.

@ngrx/store — це бібліотека для керування глобальним станом застосунку на базі RxJS для програм Angular, натхненне Redux.

@ngrx/store — це контейнер із контрольованим станом, розроблений, щоб допомогти писати продуктивні, узгоджені програми поверх Angular.

Безпека типів підтримується в усій архітектурі з опорою на компілятор TypeScript для коректності програми. На додаток до цього, суворість безпеки типів у NgRx і використання шаблонів добре підходить для створення коду вищої якості.

@ngrx/store побудовано на єдиній незмінній структурі даних, що робить виявлення змін відносно простим завданням за допомогою Angular OnPush стратегії. @ngrx/store також надає API для створення мемоізованих функцій вибору, які оптимізують отримання даних із стану.

Використовуючи @ngrx/effects and @ngrx/store, будь-які побічні ефекти взаємодії із зовнішніми ресурсами, такими як мережеві запити або веб-сокети, а також будь-яка бізнес-логіка, можуть бути ізольовані від інтерфейсу користувача. Ця ізоляція дозволяє використовувати більш чисті та прості компоненти та підтримує принцип єдиної відповідальності.

Нормалізуючи зміни стану та передаючи їх через спостережувані, NgRx забезпечує можливість серіалізації та забезпечує передбачуване збереження стану. Це дозволяє зберегти стан у зовнішній пам’яті, наприклад localStorage.

Це також дозволяє перевіряти, завантажувати, завантажувати та відправляти дії з @ngrx/store-devtools.

Оскільки @ngrx/store використовує чисті функції для зміни та вибору даних із стану, а також здатність ізолювати побічні ефекти від інтерфейсу користувача, тестування стає дуже простим. NgRx також надає тестові ресурси, такі як provideMockStore і provideMockActions для ізольованих тестів, і загалом кращий досвід тестування. [x]

[x] - https://ngrx.io/guide/store/why

@ngrx/effects — це модель побічних ефектів на основі RxJS для @ngrx/store.

@ngrx/effects використовують потоки, щоб забезпечити [нові джерела](https://martinfowler.com/eaaDev/EventSourcing.html) дій для зменшення стану на основі зовнішніх взаємодій, таких як мережеві запити, повідомлення веб-сокетів і часові події.

У сервісному додатку Angular компоненти відповідають за взаємодію із зовнішніми ресурсами безпосередньо через сервіси. Натомість ефекти забезпечують спосіб взаємодії з цими службами та ізолюють їх від компонентів. Ефекти – це те, де ви виконуєте такі завдання, як отримання даних, довгострокові завдання, які створюють кілька подій, та інші зовнішні взаємодії, де вашим компонентам не потрібні чіткі знання про ці взаємодії.

Ефекти ізолюють побічні ефекти від компонентів, дозволяючи використовувати більш чисті компоненти, які вибирають стан і дії диспетчеризації.

Ефекти — це довготривалі служби, які прослуховують спостережувану кожну дію, що надсилається з store.

Ефекти фільтрують ці дії на основі типу дії, яка їх цікавить. Це робиться за допомогою оператора.

Ефекти виконують завдання, які є синхронними або асинхронними, і повертають нову дію. [x]

[x] - https://ngrx.io/guide/effects

На рисунку зображений приклад коду @ngrx/effect



Рисунок – Приклад коду @ngrx/effect

Для роботи з svg іконками було обрано angular-svg-icon

angular-svg-icon — це сервіс і компонент Angular 15, який надає засоби для вбудовування файлів SVG, щоб можна було легко стилізувати їх за допомогою CSS і коду.

Сервіс надає реєстр значків, який завантажує та кешує SVG, індексований за його url. Компонент відповідає за відображення SVG. Після отримання svg із реєстру він клонує SVGElement і SVG у внутрішній HTML компонента. [x]

[x] - <https://www.npmjs.com/package/angular-svg-icon>

На рисунку зображений приклад використання angular-svg-icon

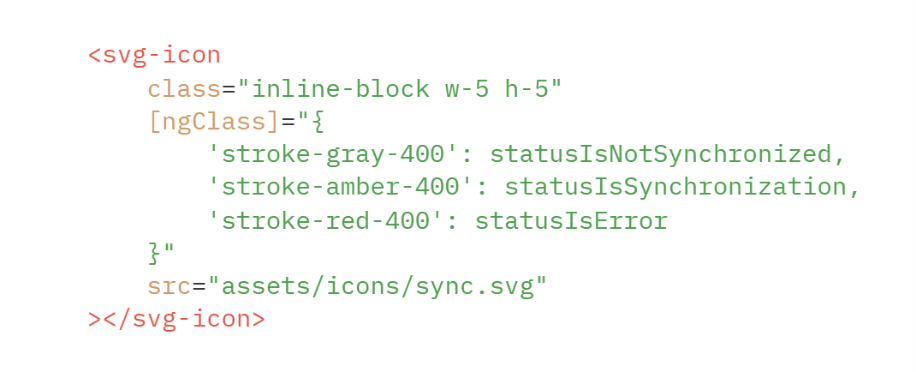


Рисунок – Приклад використання angular-svg-icon

Для відображення анімованих svg було взято бібліотеку lottie-web та зв’язку її з Angular – ngx-lottie

Lottie — це бібліотека для Android, iOS, Web і Windows, яка аналізує анімацію Adobe After Effects, експортовану як JSON за допомогою Bodymovin, і рендерить її на мобільних пристроях і в Інтернеті! [x]

[x] - <https://airbnb.io/lottie/>

Lottie-web – Адаптована версія lottie для написання вебу

Ngx-lottie - Мінімально настроюваний компонент Angular зі стабільною продуктивністю для відтворення анімації After Effects. [x]

[x] - https://www.npmjs.com/package/ngx-lottie

На рисунку 3.x зображений приклад використання lottie в Angular

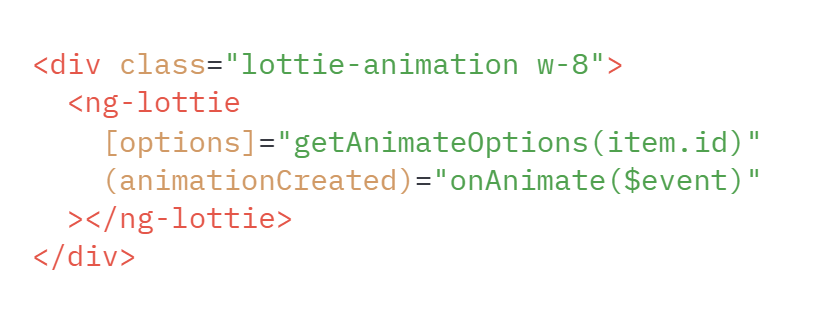


Рисунок – Приклад використання lottie

Для створення функціоналу зміни порядку категорій на сторінці клієнтської частини категорій було використано @angular/cdk/drag-drop

Набір для розробки компонентів, або ж Component Dev Kit (CDK) — це набір примітивів поведінки для створення компонентів інтерфейсу користувача. [x]

[x] - <https://material.angular.io/cdk/categories>

Модуль @angular/cdk/drag-drop надає розробникам спосіб легкого та декларативного створення інтерфейсів перетягування та скидання з підтримкою вільного перетягування, сортування в списку, перенесення елементів між списками, анімації, сенсорних пристроїв, спеціального перетягування ручки, попередні перегляди та заповнювачі, на додаток до горизонтальних списків і блокування вздовж осі. [x]

[x] - https://material.angular.io/cdk/drag-drop/overview

Для створення графіку на сторінці статистики було обрано бібліотеку Chart.js

Chart.js - проста, але гнучка бібліотека діаграм JavaScript для сучасного Інтернету

Серед багатьох бібліотек діаграм для розробників додатків JavaScript Chart.js наразі є найпопулярнішою згідно зі зірками GitHub (~60 000) і завантаженнями npm (~2 400 000 щотижня).

Chart.js був створений і оголошений у 2013 році, але з того часу пройшов довгий шлях. Він має відкритий вихідний код, ліцензований за дуже дозволеною ліцензією MIT і підтримується активною спільнотою.

Chart.js надає набір часто використовуваних типів діаграм, плагінів і параметрів налаштування. Окрім розумного набору вбудованих типів діаграм, ви можете використовувати додаткові типи діаграм, які підтримуються спільнотою. Крім того, можна об’єднати кілька типів діаграм у змішану діаграму (по суті, змішуючи кілька типів діаграм в одну на одному полотні). Chart.js легко налаштовується за допомогою користувальницьких плагінів для створення анотацій, масштабування або функцій перетягування. [x]

[x] - https://www.chartjs.org/docs/latest/

Для створення авторизації за допомогою Google аккаунту було використано Google identity та @abacritt/angularx-social-login

Google Identity Platform(далі Identity Platform) – це платформа керування ідентифікацією та доступом клієнтів (CIAM), яка допомагає організаціям додавати функції керування ідентифікацією та доступом до своїх програм, захищати облікові записи користувачів і впевнено масштабуватись у Google Cloud.

За допомогою платформи Identity Platform розробники можуть легко додати широко поширену, зручну та настроювану службу автентифікації до своїх веб- і мобільних програм, щоб вони могли зосередитися на створенні своєї програми чи послуги.

Платформа Identity Platform може допомогти захистити користувачів додатків та запобігти захопленню облікових записів, пропонуючи багатофакторну автентифікацію (MFA) та інтеграцію з розвідкою Google для захисту облікових записів. [x]

[x] - <https://cloud.google.com/identity-platform>

@abacritt/angularx-social-login – Модуль для соціального входу та автентифікації для Angular. Підтримує автентифікацію в Google, Facebook, Amazon та Microsoft.

Клієнтська частина сервісу має 3 основні сторінки та 1 додаткову

В шапці клієнтської частини сервісу знаходиться логотип при натисненні на який користувач перейде на головну сторінку, меню, а також інформація про користувача, а саме його Ім’я та Призвіще, а також аватар.

На рисунку 3.x продемонстрована шапка сервісу



Рисунок – Демонстрація шапки сервісу

Якщо натиснути на інформацію про користувача з’явиться меню, в якому на даний момент можна вийти з аккаунту

На рисунку 3.x продемонстровано меню користувача

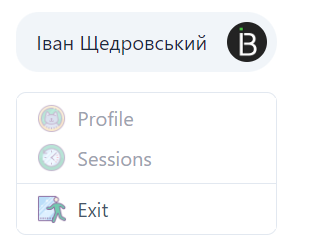


Рисунок – Демонстрація меню користувача

Футор, або ж нижня панель сервісу має інформацію про поточну версію застосунку, посилання на github розробника, а також посилання на tailwindcss та icons8, сервіси які були використані при розробці, та які по ліцензійній угоді потрібно указувати. На рисунку 3.x продемонстрований футер сервісу



Рисунок – Демонстрація футора сервісу

Домашня сторінка представляє собою форму для вводу нових даних статистики обираючи раніше створені категорії, вводячи коментар. Також для форми є додаткові опції, які можна побачити натиснувши на кнопку “Additional”. На рисунку продемонстрована головна сторінка застосунку

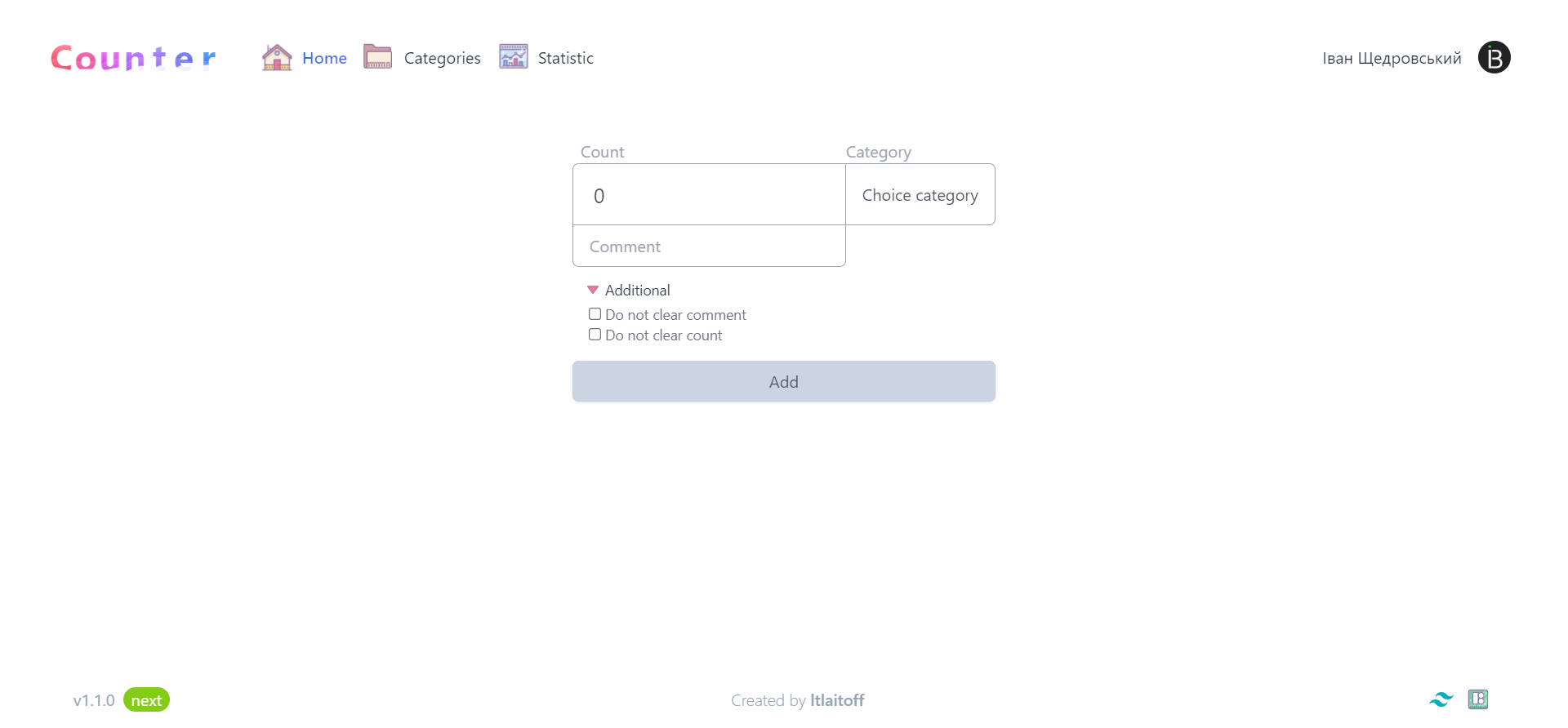


Рисунок – Демонстрація головної сторінки застосунку

При натисненні на кнопку «Choice category» з’являється форма в якій можна вибрати категорію, представлена на рисунку 3.x

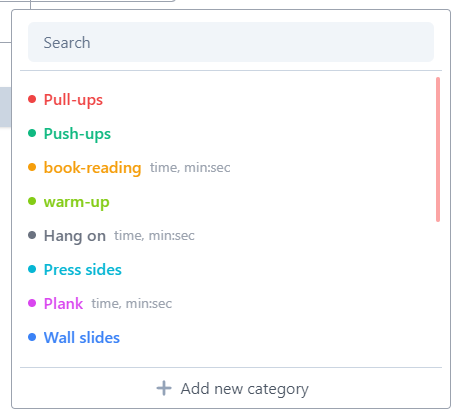


Рисунок 3.x – Форма для вибору категорій

В формі вибору категорій реалізований функціонал пошуку за назвою, який продемонстрований на рисунку 3.x.

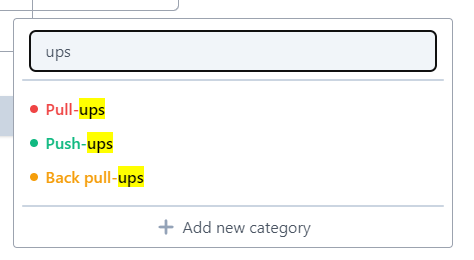


Рисунок 3.x – Демонстрація пошуку категорії за назвою в формі вибору категорій

Також в формі вибору категорій реалізований функціонал додавання нової категорій за допомогою кнопки «Add new category» яка відкриє форму для додавання нової категорії

Сторінка категорій представляє собою дві кнопки та таблицю з категоріями користувача.

Перша кнопка «Add new category» при натисненні відкриває форму для додавання нової категорії. На рисунку продемонстрована форма додавання нової категорії

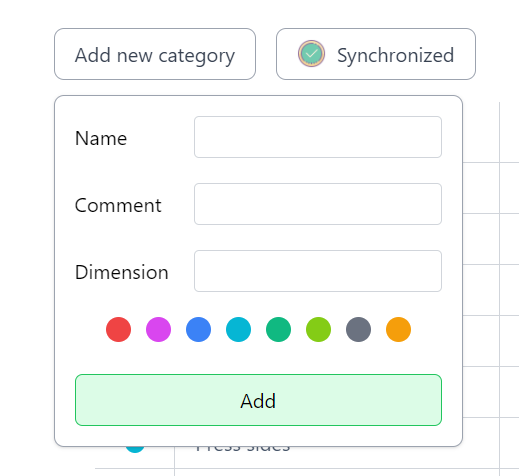


Рисунок – Демонстрація форми для додавання нової категорії

При натисненні на другу кнопку відбудеться примусове перезавантаження даних з API без перезавантаження сторінки. А також ця кнопка показує поточний статус категорій, всього їх 4: Не синхронізовано, синхронізація, помилка та синхронізовано.

Таблиця має в собі 4 стовпця:

* Перший стовпець відповідає за колір категорії
* Другий за її назву
* Третій за коментар
* Четвертий за тип величин для зручності користувачу

На рисунку зображена вся сторінка категорій

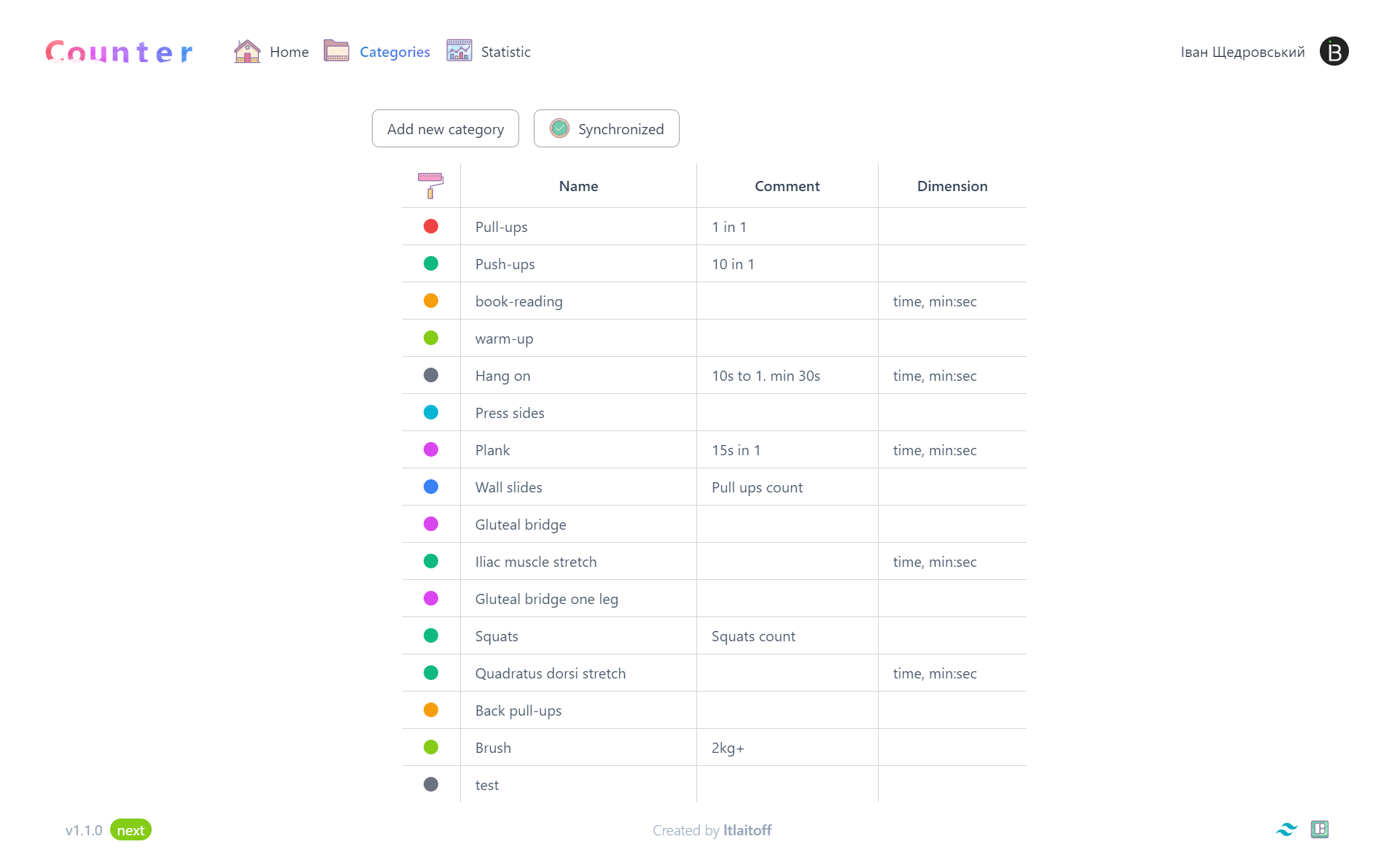


Рисунок – Демонстрація всієї сторінки категорій

При наведені на рядок таблиці рядом з ним з’являється кнопка завдяки якій користувач може змінювати порядок категорії перетаскуванням, змінювати дані категорії або ж видалити її.

При наведенні курсору миші на цю кнопку користувач побачить підказку, в якій говориться, що потрібно зажати мишу та потянути для зміни порядку, або ж клацнути для відкриття меню

На рисунку 3.x представлена підказка для користувача для рядка таблиці категорій

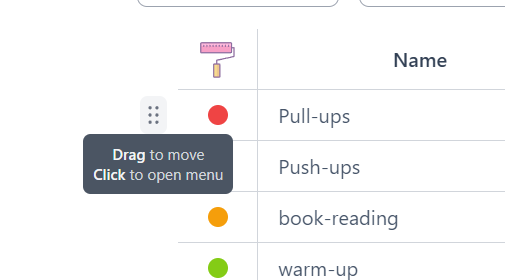


Рисунок – Підказка для користувача для рядка таблиці категорій

При змінні порядку категорій, категорія яка переміщується замальована сірим кольором. На рисунку 3.x представлена категорія, якій змінюють порядок

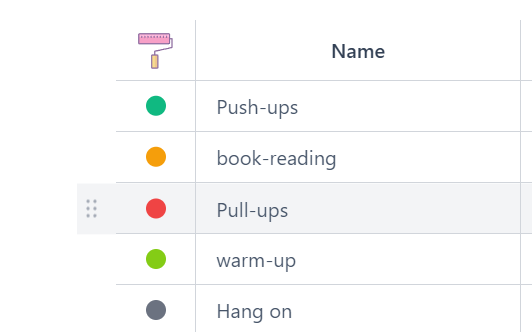


Рисунок – Категорія, якій змінюють порядок

При натисканні з’являється меню в якому користувач може змінити категорію, або ж видалити її.

На рисунку 3.x зображено меню для зміни та видалення категорії

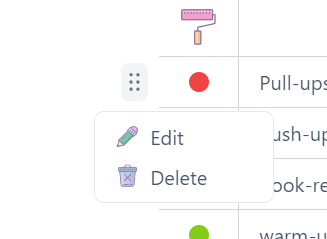


Рисунок – Меню для зміни та видалення категорії

При натисканні на пункт меню «Edit» відкриється форма в якій можна змінити інформацію про категорію.

На рисунку 3.x зображена форма для зміни інформації категорії

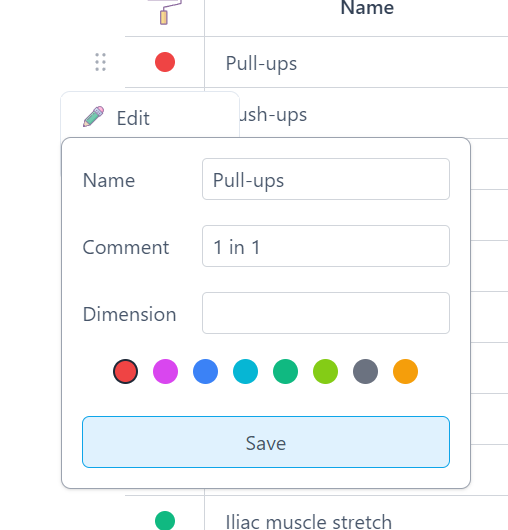


Рисунок – Форма для зміни інформації про категорію

На сторінці статистики зображений графік та таблиця даних. На рисунку 3.x представлений приклад графіку статистики на сторінці статистики

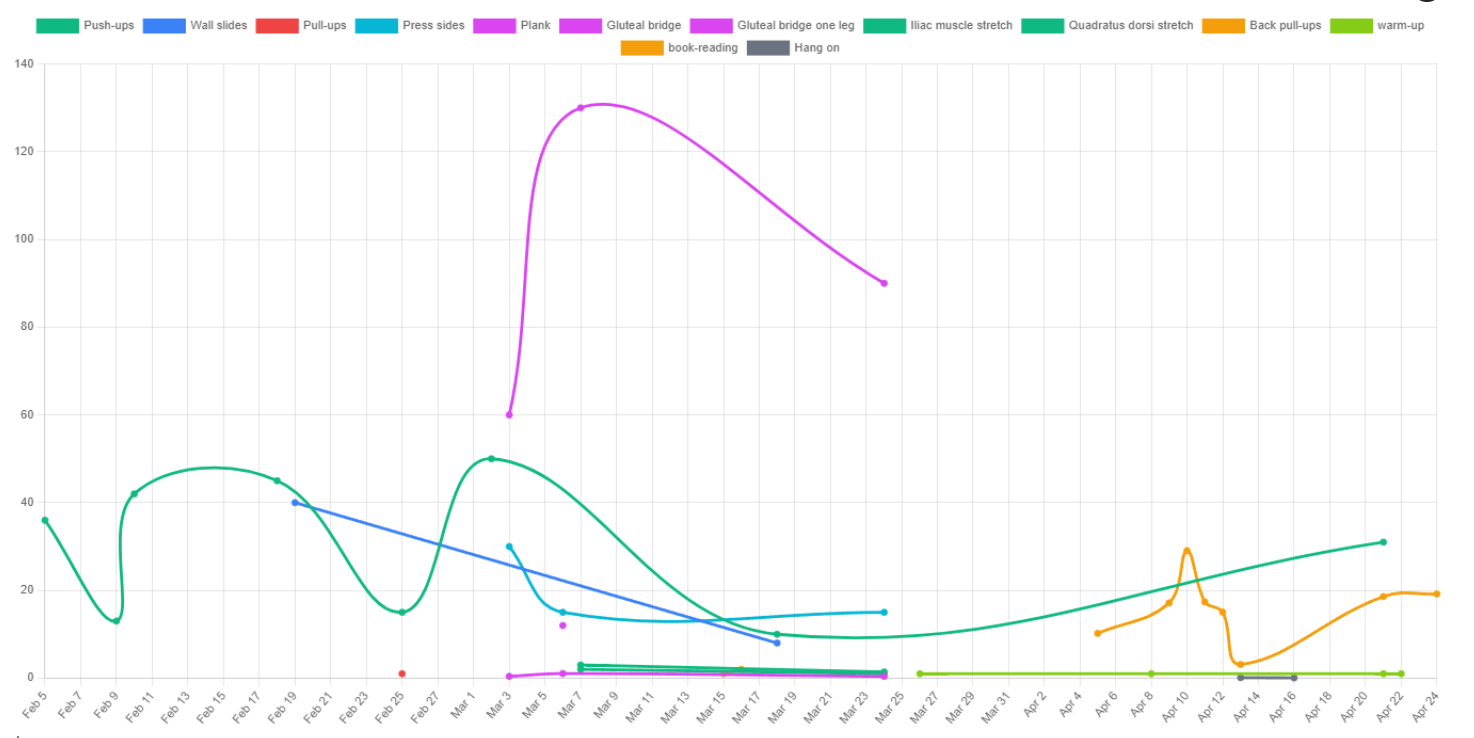


Рисунок 3.x – Приклад графіку статики

При наведені на рядок таблиці статистики з’являються дві кнопки. За допомогою кнопки яка позначена як корзина можна видалити запис. За допомогою кнопки, яка позначена як ручка – можна змінити запис в відповідний фор На рисунку 3.x представлений приклад таблиці статистики з наведенням миші на рядок

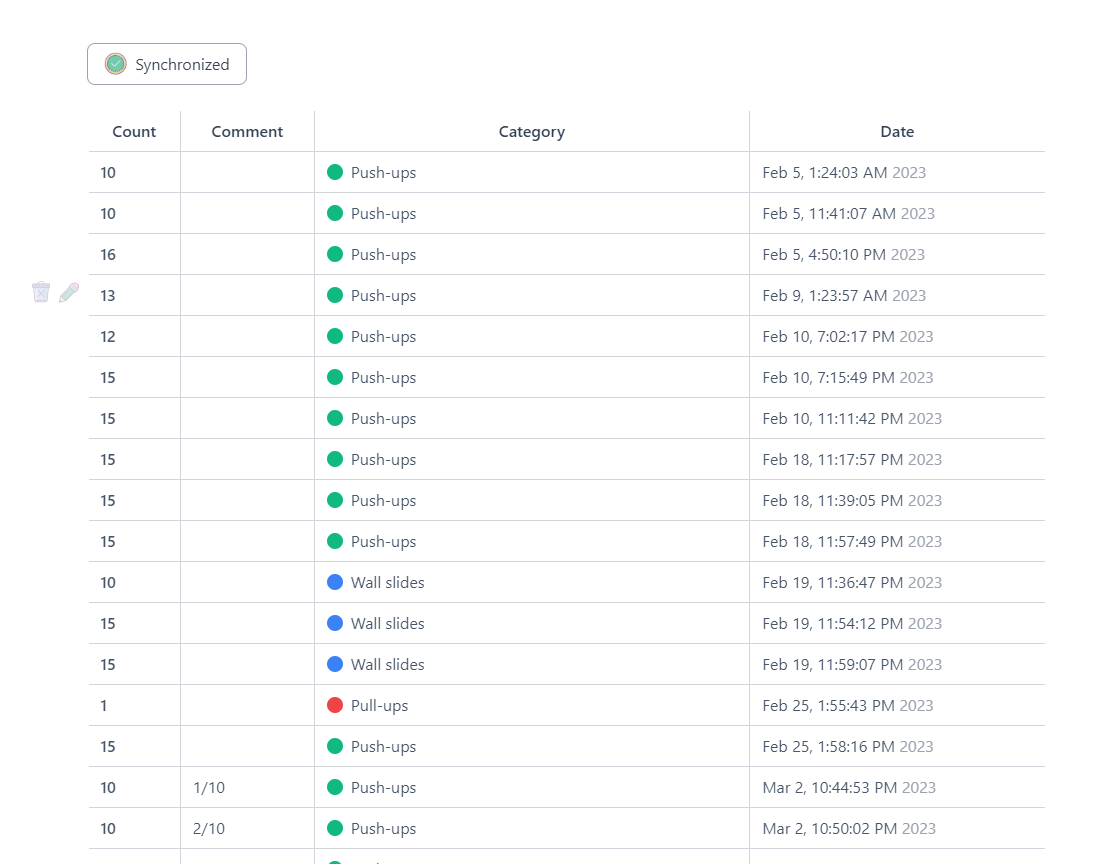


Рисунок 3.x – Приклад таблиці статистики з наведенням миші на рядок таблиці

При натисканні на кнопку позначену ручкою буде викликана форма для редагування даних запису статики яка показана на рисунку 3.x

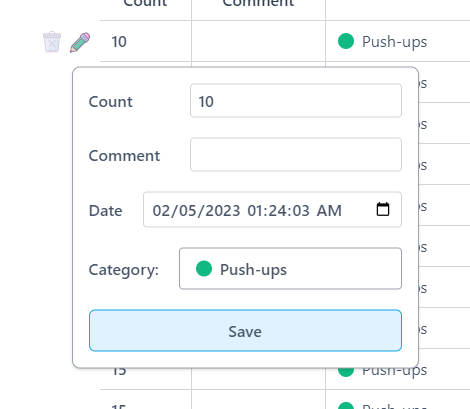


Рисунок 3.x – Форма редагування даних статистики

На сторінці авторизація представлена одна кнопка для входу в аккаунт за допомогою Google Identity

На рисунку 3.x представлений вигляд сторінки авторизації

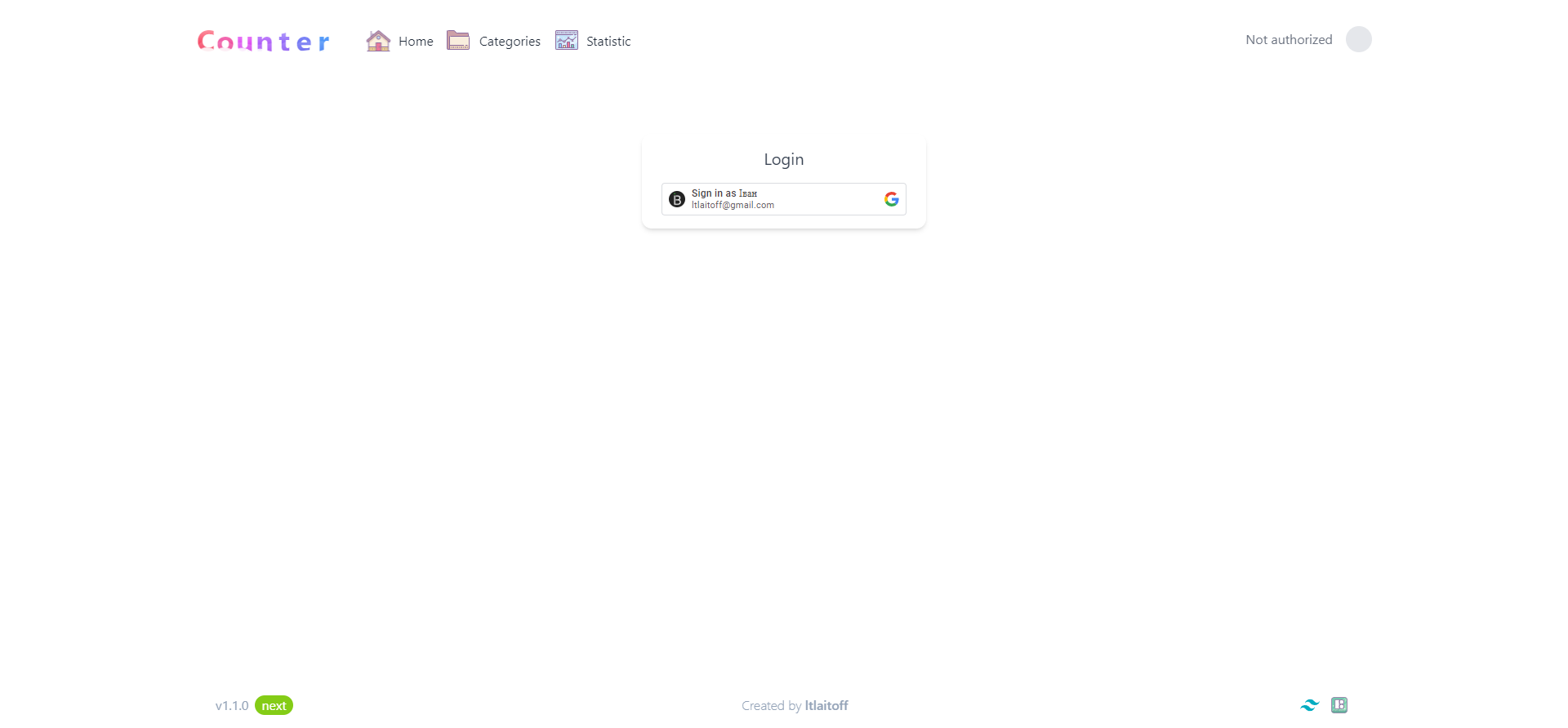


Рисунок 3.x – Вигляд сторінки авторизації

Для створення серверної частини сервісу було обрано фреймворк NestJS

Nest (NestJS) — це фреймворк для створення ефективних, масштабованих серверних програм Node.js. Він використовує прогресивний JavaScript, створений і повністю підтримує TypeScript (все ще дозволяє розробникам кодувати на чистому JavaScript) і поєднує елементи ООП (об’єктно-орієнтоване програмування), FP (функціональне програмування) і FRP (функціональне реактивне програмування).

Під капотом Nest використовує надійні фреймворки HTTP-сервера, такі як Express (за замовчуванням).

Nest забезпечує рівень абстракції, вищий за ці звичайні фреймворки Node.js (Express/Fastify), але також надає їхні API безпосередньо розробнику. Це дає розробникам свободу використовувати незліченну кількість сторонніх модулів, які доступні для основної платформи.

В останні роки, завдяки Node.js, JavaScript став «лінгва франка» Інтернету для зовнішніх і внутрішніх програм. Це призвело до появи чудових проектів, таких як Angular, React і Vue , які покращують продуктивність розробників і дозволяють створювати швидкі, тестовані та розширювані зовнішні програми. Однак, незважаючи на те, що існує безліч чудових бібліотек, допоміжних засобів та інструментів для Node (і серверного JavaScript), жоден із них не вирішує ефективно головну проблему – архітектури .

Nest надає готову архітектуру додатків, яка дозволяє розробникам і командам створювати високоякісні, масштабовані, слабко пов’язані та легко підтримувані додатки. Архітектура значною мірою натхненна Angular. [x]

[x] - <https://docs.nestjs.com/>

Nest

Під капотом Nest використовує надійні фреймворки HTTP-сервера, такі як Express (за замовчуванням).

Express - 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. [x]

[x] - <https://expressjs.com/>

Для розробки серверної частини сервісу було обрано саме NestJS, а не Express бо Nest надає більше можливостей для розробки та його архітектура в більшості натхнена Angular, в порівнянні з Express.

Функції проміжного програмного забезпечення(далі middleware) — це функції, які мають доступ до об’єкта запиту (req), об’єкта відповіді (res) і наступної функції проміжного програмного забезпечення в циклі запит-відповідь програми. Наступна функція проміжного програмного забезпечення зазвичай позначається змінною з іменем next.[x]

[x] - https://expressjs.com/en/guide/using-middleware.html

Життєвий цикл Express представляє собою набір middleware’s. Життєвий цикл Express представлений на рисунку 3.x [x]

[x] - https://iq.opengenus.org/middlewares-in-express/

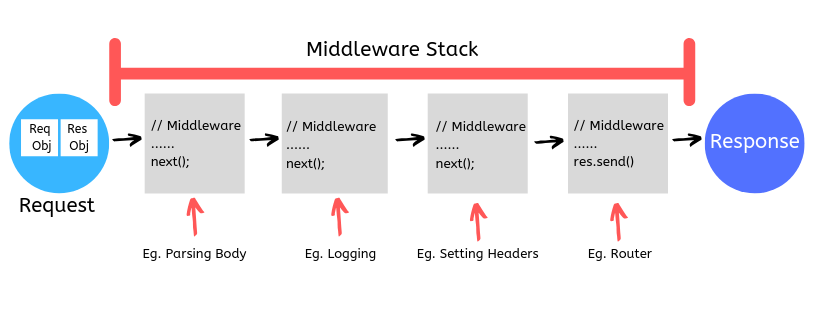


Рисунок 3.x – Демонстрація життєвого циклу Express

Оскільки в Express фактично всі функції через які проходить http-запит є middleware це дуже не зручно та не зрозуміло для розробників, бо потрібно заходити в кожну middleware та дивитись для чого вона потрібна та що робить.

Наприклад, в розробці існують паттерни проєктування для того, щоб розробники говорили на одній мові та не потрібно було пояснювати що робить код.

Життєвий цикл Nest, зображений на рисунку 3.x, є більш складнішим та має більшу кількість елементів.[x]

[x] - https://slides.com/yariv-gilad/nest-js-request-lifecycle/fullscreen

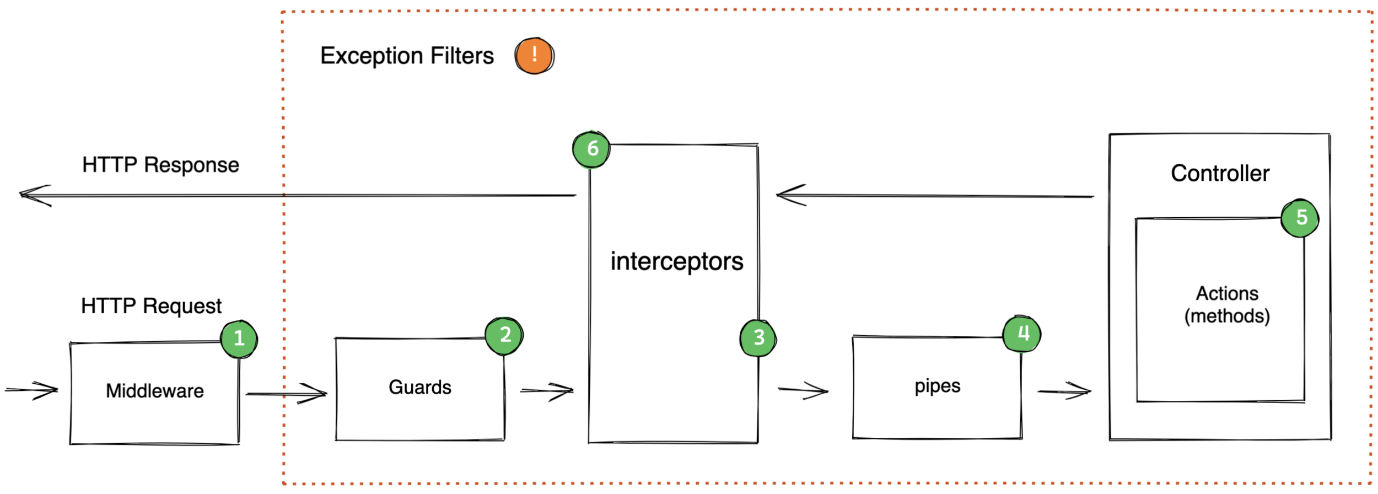


Рисунок 3.x – Життєвий цикл Nest

Загалом життєвий цикл запиту виглядає так:

1. Вхідний запит
2. Middleware глобального зв’язку
3. Middleware пов’язане з модулем
4. Глобальні guards
5. Контролери guards
6. Guards маршруту
7. Глобальні interceptors  (перед-контролер)
8. Перехоплювачі interceptors  (перед-контролер)
9. Перехоплювачі маршрутів (перед-контролер)
10. Глобальні pipes
11. Pipes контролера
12. Pipes маршруту
13. Pipes параметрів маршруту
14. Контролер (обробник методу)
15. Service (якщо є)
16. Interceptor  маршруту (після запиту)
17. Interceptor  контролера (після запиту)
18. Глобальний interceptor  (після запиту)
19. Filters винятків (маршрут, потім контролер, потім глобальний)
20. Відповідь сервера

Middleware - проміжне програмне забезпечення виконується в певній послідовності. Спочатку Nest запускає глобально прив’язане проміжне програмне забезпечення (наприклад, проміжне програмне забезпечення, пов’язане з app.use), а потім запускає проміжне програмне забезпечення, пов’язане з модулем , яке визначається шляхом. Проміжне програмне забезпечення запускається послідовно в порядку їх зв’язування, подібно до того, як працює проміжне програмне забезпечення в Express. У випадку проміжного програмного забезпечення, зв’язаного між різними модулями, спочатку буде працювати проміжне програмне забезпечення, прив’язане до кореневого модуля, а потім проміжне програмне забезпечення — у порядку, у якому модулі додаються до масиву імпорту.

Guards - виконання охорони починається з глобальних охоронців, потім переходить до охоронців контролерів і, нарешті, до охоронців маршрутів. Як і у випадку з проміжним програмним забезпеченням, охоронці працюють у тому порядку, у якому вони зв’язані.

Interceptors - Перехоплювачі, здебільшого, дотримуються тієї самої моделі, що й охоронці, з одним недоліком: коли перехоплювачі повертають RxJS Observables , спостережувані будуть вирішені за принципом «перший, останній, вихід». Отже, вхідні запити проходитимуть через стандартну глобальну розв’язку на рівні контролера, маршруту, але сторона відповіді запиту (тобто після повернення від обробника методу контролера) буде вирішена від маршруту до контролера до глобальної. Крім того, будь-які помилки, викинуті каналами, контролерами або службами, можна прочитати в catchError оператори перехоплювача.

Pipes - Конвеєри дотримуються стандартної глобальної послідовності зв’язування з контролером і маршрутом, з однаковими параметрами @UsePipes(). Однак на рівні параметрів маршруту, якщо у вас запущено кілька каналів, вони працюватимуть у порядку останнього параметра з каналом до першого. Це також стосується труб рівня маршруту та рівня контролера.

Filters, або ж фільтри — це єдиний компонент, який спочатку не вирішує глобальні. Натомість фільтри розв’язуються з найнижчого можливого рівня, тобто виконання починається з будь-яких фільтрів, пов’язаних з маршрутом, і продовжується на рівні контролера, а нарешті – до глобальних фільтрів. Зауважте, що винятки не можна передавати від фільтра до фільтра; якщо фільтр рівня маршруту перехоплює виняток, контролер або фільтр глобального рівня не можуть перехопити той самий виняток. Єдиний спосіб досягти такого ефекту — використовувати успадкування між фільтрами. [x]

[x] - <https://docs.nestjs.com/faq/request-lifecycle#middleware>

В якості бази даних для сервісу було обрано MongoDB.

MongoDB — документо-орієнтована система керування базами даних з відкритим вихідним кодом, 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)). [x]

[x] - https://aws.amazon.com/ru/documentdb/what-is-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 3.x представлено візуалізацію роботи NodeJS та MongoDB через Mongoose

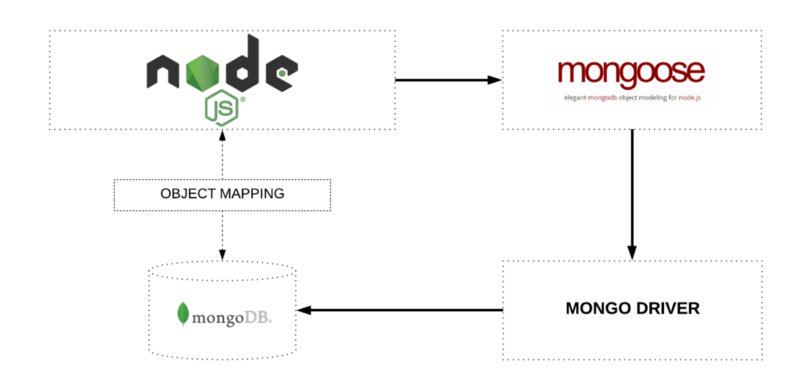


Рисунок 3.x - Візуалізація роботи Node та Mongo через Mongoose

«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. [x]

[x] - https://www.freecodecamp.org/news/introduction-to-mongoose- for-mongodb-d2a7aa593c57

Для зручного використання тіла http-запитів було взято middleware «body-parser».

body-parser – middleware, яке дозволяє розбирайти тіла вхідних запитів перед обробниками, доступними у властивості req.body.

Об’єкт bodyParser відкриває різні фабрики для створення проміжного програмного забезпечення. Усі проміжні програми заповнять властивість req.body проаналізованим тілом, якщо заголовок запиту Content-Type збігається з опцією типу, або порожнім об’єктом ({}), якщо немає тіла для аналізу, Content-Type не відповідає або виникла помилка [x]

[x] - <https://www.npmjs.com/package/body-parser>

Спільне використання ресурсів між джерелами (далі CORS) — це механізм на основі HTTP-заголовків, який дозволяє серверу вказувати будь-які джерела (домен, схему або порт), окрім власного, з якого браузер має дозволити завантажувати ресурси. CORS також покладається на механізм, за допомогою якого веб-переглядачі надсилають «переддруковий» запит на сервер, на якому розміщено ресурс із перехресним джерелом, щоб перевірити, чи сервер дозволить фактичний запит. Під час попередньої перевірки браузер надсилає заголовки, які вказують метод HTTP, і заголовки, які використовуватимуться у фактичному запиті. [x]

[x] - <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

Для роботи з CORS було обрано однойменний NodeJS пакет «cors».

cors — це пакет node.js для надання проміжного програмного забезпечення Connect/Express, яке можна використовувати для ввімкнення CORS із різними параметрами. [x]

[x] - https://www.npmjs.com/package/cors

Для роботи з сеансами в Nest було обрано бібліотеку express-session

HTTP-сеанси забезпечують спосіб зберігання інформації про користувача в кількох запитах, що особливо корисно для програм MVC.

Приклад використання express-session в Nest показаний на рисунку 3.x. [x]

[x] - https://docs.nestjs.com/techniques/session



Рисунок 3.x – Приклад використання express-session в Nest

Для того, щоб зберігати сеанси в MongoDB було обрано пакет connect-mongo

connect-mongo - сховище сеансів MongoDB для Connect і Express, написане на Typescript

На рисунку 3.x зображений приклад використання connect-mongo при створенні express-session [x]

[x] - https://www.npmjs.com/package/connect-mongo



Рисунок 3.x – Приклад використання connect-mongo при створенні express-session

Веб-токен JSON (JWT) — це відкритий стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити та довіряти їй, оскільки вона має цифровий підпис. JWT можна підписати за допомогою секрету (з алгоритмом HMAC) або пари відкритих/приватних ключів за допомогою RSA або ECDSA.

Хоча JWT можна зашифрувати, щоб також забезпечити секретність між сторонами, ми зосередимося на підписаних токенах. Підписані токени можуть підтвердити цілісність претензій, що містяться в ньому, тоді як зашифровані токени приховують ці претензії від інших сторін. Коли токени підписуються за допомогою пар відкритих/приватних ключів, підпис також засвідчує, що лише сторона, яка володіє закритим ключем, є тією, яка його підписала. [x]

[x] - <https://jwt.io/introduction>

Для аутентифікації користувачів було обрано пакети @nestjs/passport, passport та @nestjs/jwt, passport-jwt(для роботи з jwt).

[Passport](https://github.com/jaredhanson/passport) — найпопулярніша бібліотека автентифікації node.js, добре відома спільноті та успішно використовується в багатьох робочих програмах. Інтегрувати цю бібліотеку з додатком Nest за допомогою модуля просто @nestjs/passport. На високому рівні Passport виконує ряд кроків, щоб:

Автентифікувати користувача, підтвердивши його «облікові дані» (наприклад, ім’я користувача/пароль, веб-токен JSON ( [JWT](https://jwt.io/) ) або маркер ідентифікації від постачальника ідентифікаційної інформації)

Керувати автентифікованим станом (шляхом видачі портативного маркера, наприклад JWT, або створення [експрес-сеансу](https://github.com/expressjs/session) )

Додати інформацію про автентифікованого користувача до Requestоб’єкта для подальшого використання в обробниках маршрутів

Passport має багату екосистему [стратегій](http://www.passportjs.org/) , які реалізують різні механізми автентифікації. Незважаючи на просту концепцію, набір стратегій Passport великий і представляє велику різноманітність. Passport абстрагує ці різноманітні кроки в стандартний шаблон, а @nestjs/passportмодуль обертає та стандартизує цей шаблон у знайомі конструкції Nest.

У цьому розділі ми реалізуємо повне рішення наскрізної автентифікації для сервера RESTful API за допомогою цих потужних і гнучких модулів. Ви можете використовувати концепції, описані тут, для реалізації будь-якої стратегії Passport для налаштування схеми автентифікації. Щоб побудувати цей повний приклад, виконайте дії, описані в цьому розділі.

Корисно думати про Passport як про міні-фреймворк сам по собі. Елегантність фреймворку полягає в тому, що він абстрагує процес автентифікації на кілька основних кроків, які розробник налаштовує відповідно до стратегії, яку він реалізує.

Це як фреймворк, тому що розробник налаштовує його, надаючи параметри налаштування (як звичайні об’єкти JSON) і спеціальний код у формі функцій зворотного виклику, які Passport викликає у відповідний час. Модуль @nestjs/passport обгортає цю структуру пакетом у стилі Nest, що полегшує інтеграцію в програму Nest.

Пакет @nestjs/jwt — це пакет утиліт, який допомагає маніпулювати JWT.

Пакет passport-jwt— це пакет Passport, який реалізує стратегію JWT і @types/passport-jwtнадає визначення типів TypeScript. [x]

[x] - <https://docs.nestjs.com/recipes/passport>

Для валідації(або перевірки) даних було обрано пакети class-validator та class-transformer.

Найкращою практикою є перевірка правильності будь-яких даних, надісланих у веб-програму. Для автоматичної перевірки вхідних запитів Nest надає кілька каналів, доступних прямо з коробки, наприклад, ValidationPipe

ValidationPipe дозволяє використовувати потужний пакет class-validator та його декоратори декларативної перевірки.  ValidationPipe надає зручний підхід до застосування правил перевірки для всіх вхідних корисних даних клієнта, де конкретні правила оголошуються з простими анотаціями в оголошеннях локального класу/DTO в кожному модулі.

Для роботи з ValidationPipe потрібно встановити пакети class-validator та class-transformer. [x]

[x] – https://docs.nestjs.com/techniques/validation

class-validator – пакет, який дозволяє використовувати перевірку на основі декоратора та не на основі декоратора. Внутрішньо використовує validator.js для виконання перевірки.

class-validator працює як у браузері, так і на платформах node.js. [x]

[x] - <https://www.npmjs.com/package/class-validator>

class-transformer - дозволяє перетворювати звичайний об'єкт на деякий екземпляр класу і навпаки. Також це дозволяє серіалізувати/десеріалізувати об’єкт на основі критеріїв. Цей інструмент надзвичайно корисний як для інтерфейсу, так і для серверу. [x]

[x] - <https://www.npmjs.com/package/class-transformer>

Всього серверна частина підтримує 5 типів запитів.

Перший тип це запити на авторизацію та вихід з аккаунту:

* POST запит на “/authorization” - для авторизації
* POST запит на “/authorization/logout” – для виходу

Другий тип це запити на категорії:

* GET запит на “/category/all” – Щоб отримати всі категорії користувача
* POST запит на “/category/add” – Щоб створити нову категорію
* DELETE запит на “/category/:id” – Щоб видалити категорію
* PUT запит на “/category/reoder” – Для зміни порядку категорії
* PUT запит на “/category/:id” – Для зміни інформації категорії

Третій тип це запити на інітіалізацію:

* GET запит на “/initialize” - отримання інформації про те, авторизований користувач чи ні

Четвертий тип це запити на колір:

* GET запит на “/color/all” – Отримати всі кольори
* POST запит на “/color/initialize” – Ініціалізувати кольори застосунку за допомогою окремого логіну та паролю

П’ятий тип це запити на статистику:

* GET запит на “/statistic/all” – Отримання всіх даних статистики користувача
* GET запит на “/statistic/:id” – Отримання даних статистичного запису за його унікальним ідентифікатором
* POST запит на “/statistic/add” – Створення нового статистичного запису
* DELETE запит на “/statistic/:id” – Видалення статистичного запису за його унікальним ідентифікатором
* PUT запит на “/statistic/:id” – Оновлення статистичного запису за його унікальним ідентифікатором

Для приведення коду до єдиних стандартів і всьому застосунку було обрано Prettier.

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. [x]

[x] - https://prettier.io/docs/en/index.html

Для відслідковування синтаксичних та помилок типів в коді при написанні застосунку був обраний ESLint

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. [x]

[x] - https://dev.to/shivambmgupta/eslint-what-why-when-how-5f1d

# 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