Міністерство освіти і науки України

Херсонський державний університет

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК, ФІЗИКИ ТА МАТЕМАТИКИ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Моделювання та розроблення вебсервісу "Пільгові стипендії"

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу 12-441 групи
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми «Інженерія програмного забезпечення»
Дадаєв Владислав Олегович

Керівник: старший викладач Черненко І.Є.

Рецензент: Захарченко Р.М. кандидатка технічних наук, доцентка, доцентка кафедри програмних засобів і технологій Херсонського національного технічного університету

Івано-Франківськ – 2024

ЗМІСТ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТОК

ВСТУП

У сучасному світі, де освіта стає все більш цифровою та доступною, важливість ефективного управління освітніми ресурсами, зокрема стипендіями, набуває нового значення. З цієї перспективи, розробка веб-сервісу для управління пільговими стипендіями є не тільки актуальною, але й необхідною задачею.

Актуальність дослідження полягає в потребі оптимізації процесів призначення та управління пільговими стипендіями, що є ключовим аспектом підтримки обдарованих студентів та студентів, які потребують фінансової підтримки.

Мета роботи полягає у створенні та впровадженні ефективного, зручного у використанні веб-сервісу для управління пільговими стипендіями, який би задовольняв потреби студентів, навчальних закладів та стипендіальних фондів.

Завдання дослідження включають: аналіз існуючих систем управління стипендіями; розробку концептуальної моделі веб-сервісу; програмування та тестування веб-сервісу; аналіз його ефективності та пропозиції щодо подальшого розвитку.

Предмет дослідження веб-сервіс “Пільгові стипендії”.

Об'єкт — методи та технології розроблення веб-сервісів.

Практичне значення дослідження полягає в тому, що розроблений веб-сервіс може бути використаний навчальними закладами для ефективного управління процесом призначення стипендій, забезпечуючи прозорість та зручність для усіх учасників процесу.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ МОДЕЛЮВАННЯ ТА РОЗРОБЛЕННЯ ВЕБ-СЕРВІСІВ

1.1 Основні концепції веб-сервісів

У цьому розділі розглядаються ключові концепції та терміни, які є фундаментальними для розуміння веб-сервісів. Основна увага приділяється клієнт-серверній архітектурі, RESTful сервісам та мовам опису ресурсів, таким як WSDL та OpenAPI.

Клієнт-серверна архітектура є однією з найбільш фундаментальних парадигм у розробці мережевих застосунків та сервісів. Вона визначає систему, де існують два основних учасники: клієнт та сервер, які взаємодіють один з одним. Клієнт це програма або пристрій, який ініціює запит до сервера[1]. Цей запит може бути пов'язаний з отриманням, зміною, видаленням або оновленням даних. Сервер, у свою чергу, є програмою або машиною, що приймає та обробляє ці запити, виконує необхідні дії і повертає результати клієнту. Часто сервери відповідають за значну кількість даних, їх обробку та збереження.

Однією з ключових характеристик клієнт-серверної архітектури є чітке розмежування ролей. Клієнти відповідають за візуалізацію інформації та взаємодію з користувачами, тоді як сервери фокусуються на більш складних завданнях, таких як обробка даних, управління ресурсами та виконання бізнес-логіки. Ця модель дозволяє централізувати та ефективно управляти ресурсами, що важливо для забезпечення цілісності, безпеки та доступності даних[19].

Клієнт-серверні системи можуть бути легко масштабовані, шляхом додавання або оновлення серверів, без необхідності внесення змін до клієнтських програм. Це робить їх відмінним вибором для різних застосунків, від простих веб-сервісів до складних розподілених систем.

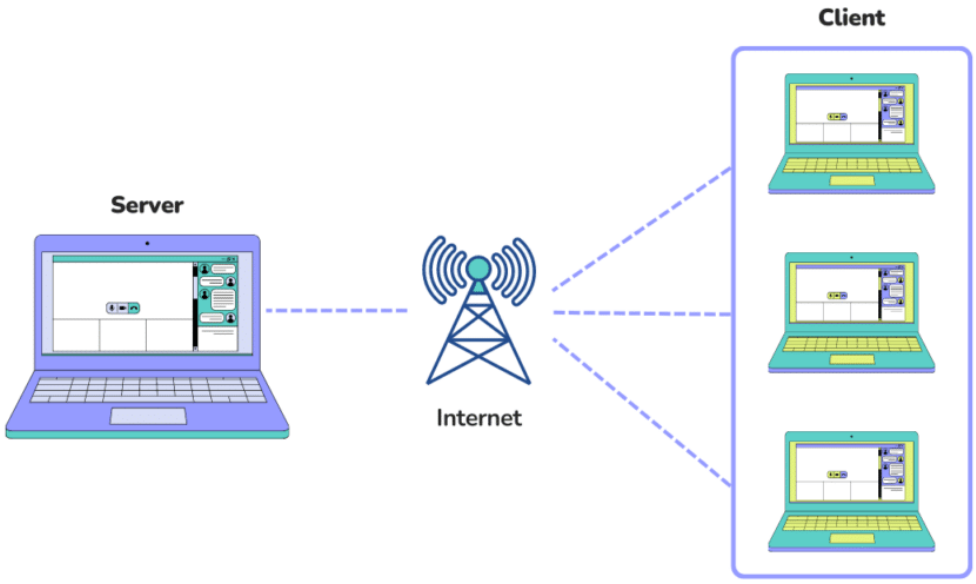


Рис.1.1 – Клієнт серверна архітектура

Найбільш поширеним прикладом використання клієнт-серверної архітектури є взаємодія між веб-браузером (клієнтом) та веб-сервером. Браузер запитує веб-сторінки, а сервер їх надає. Інший приклад - це електронна пошта, де клієнтські програми взаємодіють з серверами пошти для надсилання та отримання повідомлень[2]. Також, у контексті баз даних, клієнтське програмне забезпечення виконує запити до серверів баз даних для отримання, оновлення або видалення інформації.

Клієнт-серверна архітектура лежить в основі багатьох сучасних мережевих застосунків та сервісів, пропонуючи гнучкість, масштабованість та ефективність. Це робить її ідеальною для широкого спектру застосувань, від базових веб-сервісів до складних розподілених систем і додатків.

RESTful сервіси базуються на архітектурному стилі REST (Representational State Transfer), який визначає підхід до розробки веб-сервісів із акцентом на простоту та ефективність мережевих взаємодій. Головна ідея REST полягає у використанні стандартних HTTP методів для обміну даними між клієнтом і сервером[3]. Ці методи включають GET для

отримання даних, POST для створення нових записів, PUT для оновлення існуючих даних та DELETE для їх видалення.

Однією з основних переваг RESTful сервісів є їх простота та зрозумілість. Це досягається завдяки використанню стандартного HTTP протоколу та чітким визначенням ресурсів за допомогою URL. Такий підхід забезпечує легкість у розробці, тестуванні та інтеграції веб-сервісів.

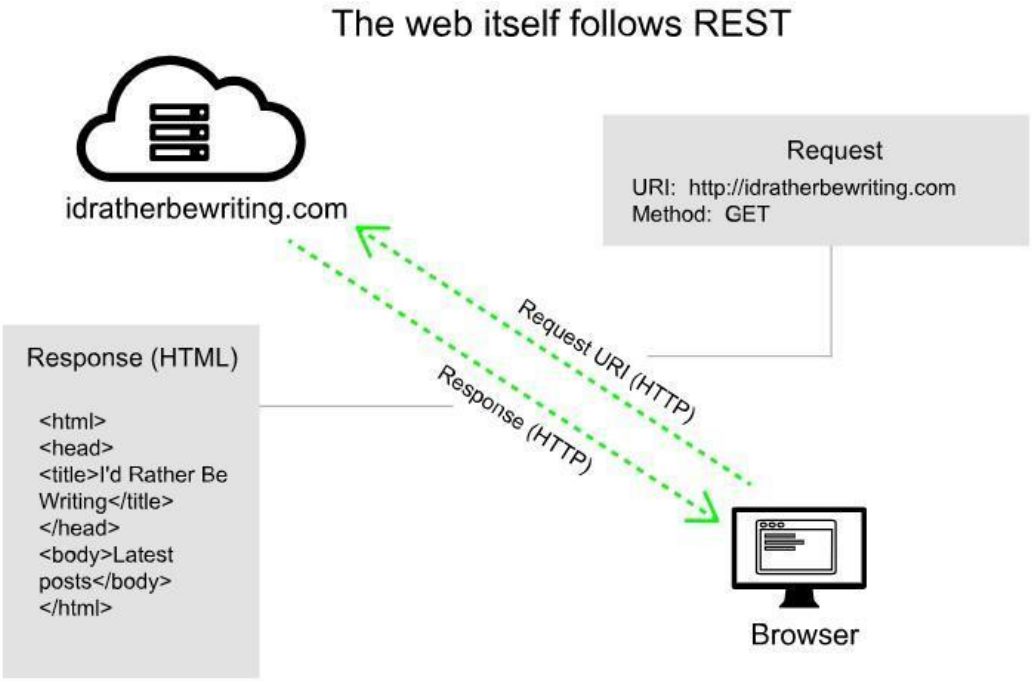


Рис.1.2 – REST архітектура

Масштабованість є ще однією важливою перевагою REST. Через те, що REST використовує безстатусові з'єднання, кожен запит від клієнта до сервера містить усю необхідну інформацію для його обробки, це дозволяє легко масштабувати сервіси, додаючи більше серверів без необхідності змінювати клієнтські додатки[20].

Уніфіковані інтерфейси є ще однією ключовою характеристикою RESTful сервісів. Це означає, що всі операції над різними ресурсами виконуються за допомогою одних і тих же методів HTTP. Такий підхід спрощує розуміння та використання API, а також забезпечує більшу гнучкість при проєктуванні системи.

Разом з тим, RESTful архітектура має деякі обмеження та виклики. Наприклад, управління станом сесії повністю залежить від клієнта, оскільки сервер не зберігає стан між запитами. Це може ускладнити реалізацію деяких функцій, таких як транзакції або підтримка постійних з'єднань[4].

Загалом, RESTful сервіси пропонують гнучку, масштабовану та ефективну платформу для розробки сучасних веб-сервісів, що дозволяє легко впроваджувати та інтегрувати різні веб-додатки.

WSDL, або Web Services Description Language, є мовою, заснованою на XML, яка використовується для опису функціональності та інтерфейсу веб-сервісів. Ця мова є ключовим елементом для веб-сервісів, які працюють на основі SOAP (Simple Object Access Protocol). WSDL дозволяє точно визначити, які операції веб-сервіс може виконувати, які повідомлення він приймає та повертає, а також деталізує структуру цих повідомлень[5].

Однією з головних переваг WSDL є здатність надавати докладний, формалізований опис веб-сервісів. Це означає, що клієнти, які хочуть взаємодіяти з веб-сервісом, можуть точно зрозуміти його інтерфейс та структуру без необхідності глибокого аналізу вихідного коду сервісу. WSDL визначає операції, які можна виконувати з веб-сервісом, а також формати запитів та відповідей, що робить можливим автоматизоване створення клієнтських програм для взаємодії з веб-сервісом.

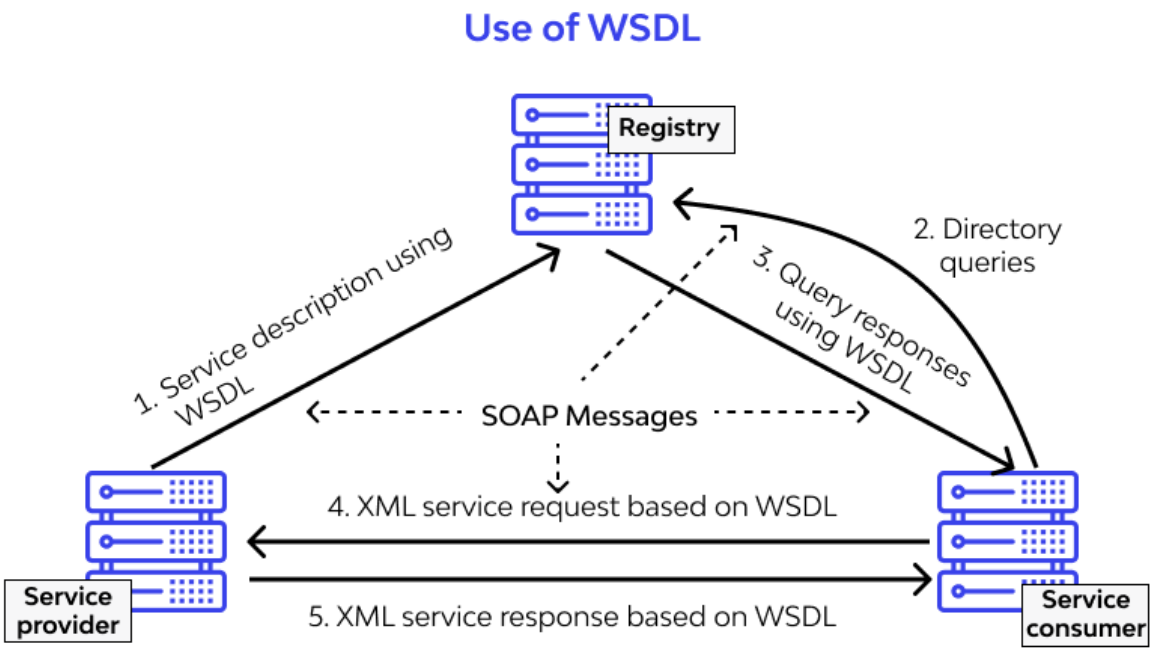


Рис.1.3 – WSDL схема використання

WSDL тісно інтегрована з SOAP, протоколом обміну повідомленнями, який дозволяє програмам різних платформ ефективно спілкуватися через мережу[21]. Використання WSDL у поєднанні з SOAP надає можливість створювати надійні, добре структуровані веб-сервіси, які можуть бути легко інтегровані в різноманітні застосунки.

Важливим аспектом WSDL є її здатність до автоматизації. Завдяки формалізованому опису, інструменти для розробки програмного забезпечення можуть автоматично генерувати вихідний код або створювати необхідні з'єднання для взаємодії з веб-сервісами. Це спрощує процес інтеграції та розширення функціоналу програм.

Загалом, WSDL відіграє критичну роль у стандартизації та формалізації взаємодії з веб-сервісами. Вона надає засоби для точного опису інтерфейсів та функціоналу веб-сервісів, що є незамінним для розробників, які прагнуть створювати надійні та інтегровані мережеві рішення.

OpenAPI — це всебічна специфікація, призначена для детального опису RESTful API. Ця специфікація відіграє важливу роль у процесі розробки веб-сервісів, оскільки вона надає стандартизований та зрозумілий спосіб опису інтерфейсів API[6]. Завдяки OpenAPI, розробники можуть детально визначати шляхи (endpoints) своїх API, операції, які можуть бути виконані, а також структуру вхідних та вихідних даних. Крім того, ця специфікація включає описи методів аутентифікації та авторизації, що є ключовими для забезпечення безпеки веб-сервісів.

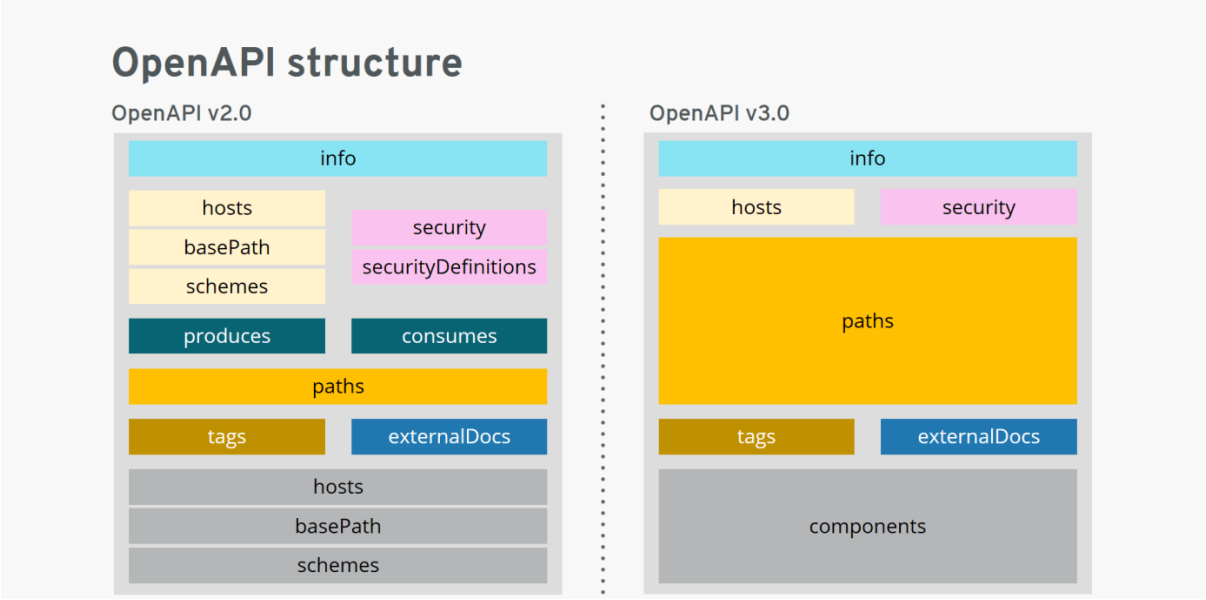


Рис.1.4 – Структура специфікації OpenAPI

Основна перевага використання OpenAPI полягає в тому, що вона спрощує процес проєктування та інтеграції API. Розробники отримують можливість створювати чіткі та зрозумілі документації для своїх API, що полегшує співпрацю та інтеграцію сервісів у різних проєктах. Також OpenAPI сприяє автоматизації багатьох процесів, зокрема, у генерації клієнтського коду, тестування API та створенні технічної документації.

Додатково, OpenAPI забезпечує уніфікованість в описі API, що важливо для забезпечення сумісності між різними системами. Це особливо важливо у великих та складних проєктах, де множина API взаємодіють

один з одним. Стандартизований опис дозволяє легше управляти цими взаємодіями, зменшуючи ризик помилок та проблем сумісності[22].

Завершуючи, OpenAPI є потужним інструментом для розробників, який забезпечує стандартизацію, зрозумілість та ефективність у процесі створення та використання RESTful API. Ця специфікація не тільки полегшує розробку та інтеграцію API, але й сприяє кращому розумінню та використанню API серед розробників і кінцевих користувачів.

У цілому, розуміння цих ключових концепцій є необхідним для ефективного проєктування та розробки сучасних веб-сервісів. Знання клієнт-серверної архітектури, RESTful сервісів, а також вміння використовувати мови опису ресурсів, такі як WSDL та OpenAPI, є фундаментальними для створення надійних, масштабованих та зручних у використанні веб-сервісів.

1.2 Технології розробки веб-сервісів

Розробка веб-сервісів охоплює широкий спектр технологій, від серверних мов програмування до баз даних та фронтенд-фреймворків. Кожна з цих технологій відіграє важливу роль у створенні ефективних та взаємодіючих веб-сервісів.

Серверні мови програмування є критичною складовою в арсеналі будь-якого веб-розробника[7]. Вони використовуються для реалізації серверної логіки, обробки запитів від клієнтів, управління відповідями, взаємодії з базами даних та виконання різних фонових завдань, пов'язаних із веб-сервісами.

- **Node.js** – це потужне середовище виконання JavaScript, що дозволяє розробникам використовувати JavaScript на стороні сервера. Основною перевагою Node.js є його асинхронна, неблокуюча модель вводу-виводу, що робить його ідеальним для створення швидких і масштабованих мережевих додатків[23]. Node.js широко використовується

для розробки RESTful API, реалізації серверних частин веб-додатків та мікросервісів, а також для роботи з різноманітними базами даних.

- **Django** – це високорівневий фреймворк на Python, спеціально призначений для спрощення та прискорення процесу веб-розробки. Він надає високий рівень абстракції для баз даних, потужні механізми для обробки URL, систему шаблонів і багато іншого. Django відомий своєю "батарейкою-включеною" філософією, що означає, що він містить багато вбудованих функцій для негайного використання, включаючи аутентифікацію, адміністрування та RSS-стрічки.
- **ASP.NET** – Розроблений Microsoft, ASP.NET є частиною ширшої платформи .NET і дозволяє створювати динамічні веб-сайти, веб-додатки та веб-сервіси. ASP.NET особливо відомий своєю масштабованістю, безпекою та продуктивністю, завдяки тісній інтеграції з іншими компонентами .NET[8]. Він надає розробникам гнучкість, дозволяючи використовувати різні мови програмування, такі як C# або VB.NET, і підтримує багатий набір функцій, включаючи широкі можливості для розробки RESTful сервісів та веб-API.

Ці серверні мови та фреймворки надають розробникам потужні інструменти для створення веб-сервісів, кожен з яких має свої унікальні характеристики та переваги. Вибір конкретної технології часто залежить від вимог проєкту, особистих переваг розробника та конкретного контексту використання.

Бази даних відіграють вирішальну роль у розробці веб-сервісів, надаючи засоби для зберігання, управління та обробки даних. Вони є необхідним елементом для забезпечення функціональності та ефективності веб-додатків.

- **MySQL** – це одна з найпопулярніших систем управління реляційними базами даних (RDBMS). Вона широко використовується в інтернет-проєктах різного масштабу, від невеликих веб-сайтів до великих онлайн-платформ[24]. MySQL пропонує широкий спектр функціональностей, включаючи підтримку різноманітних типів даних,

потужні інструменти для оптимізації запитів та індексації, а також можливості для реплікації та масштабування. Її головні переваги - це стабільність, висока продуктивність та легкість у використанні, що робить MySQL відмінним вибором для багатьох веб-додатків.

- **MongoDB** – це NoSQL база даних, орієнтована на документи, яка стала популярною завдяки своїй гнучкості та високій продуктивності. Вона дозволяє зберігати дані у форматі, схожому на JSON, що робить її ідеальною для веб-додатків, які інтенсивно використовують дані у форматі JSON[9]. MongoDB підтримує динамічні схеми даних, що дозволяє легко змінювати структуру даних. Крім того, вона пропонує можливості для горизонтального масштабування та розподілу даних, що є важливим для роботи з великими обсягами даних та реалізації високодоступних систем.

Обираючи між MySQL і MongoDB, розробники зазвичай враховують специфіку проєкту та вимоги до обробки даних. MySQL є відмінним вибором для проєктів, що потребують суворої структури даних та складних запитів з великою кількістю з'єднань. MongoDB, у свою чергу, ідеально підходить для сценаріїв, які вимагають гнучкості у схемах даних, швидкого прототипування та роботи з великими обсягами неструктурованих даних.

Фронтенд-фреймворки відіграють ключову роль у розробці сучасних веб-сервісів, надаючи розробникам потужні інструменти для створення інтерактивних та привабливих користувацьких інтерфейсів[10]. Ці фреймворки забезпечують гнучкість та продуктивність, необхідні для розробки надійних та ефективних веб-додатків.

- **React** – Розроблена Facebook, React є однією з найпопулярніших JavaScript бібліотек для створення користувацьких інтерфейсів. Основною перевагою React є його компонентний підхід, який дозволяє розробникам створювати великі веб-додатки, що можуть легко змінюватися та масштабуватися. React використовує декларативний стиль програмування, що сприяє зручності коду та його ефективності. Бібліотека

також забезпечує ефективне управління станом інтерфейсу та взаємодією з користувачем, що робить її відмінним вибором для розробки інтерактивних веб-додатків.

- **Angular** – Розроблений Google, Angular є міцним та повнофункціональним MVC (Model-View-Controller) фреймворком для створення динамічних веб-додатків. Angular використовує TypeScript, що додає додаткову строгості та інструменти для розробки. Однією з головних особливостей Angular є його двостороннє зв'язування даних, що дозволяє автоматично синхронізувати моделі та вигляди, роблячи розробку більш інтуїтивною. Angular також надає розширені можливості для модульності, тестування та впровадження залежностей, роблячи його популярним вибором для розробки складних та великомасштабних веб-додатків.

Користуючись цими фронтенд-фреймворками, розробники можуть створювати інтуїтивно зрозумілі, відгукові та візуально привабливі веб-інтерфейси[11]. React та Angular кожен надають унікальні переваги та підходи до розробки веб-додатків, дозволяючи розробникам вибирати той фреймворк, який найкраще відповідає потребам їхнього проєкту.

1.3 Безпека веб-сервісів

Безпека веб-сервісів - це комплексне завдання, яке включає захист від різноманітних кіберзагроз, а також забезпечення конфіденційності, цілісності та доступності даних[12]. Розглянемо більш детально ключові аспекти безпеки веб-сервісів.

Захист від атак

- **Атаки типу "відмова в обслуговуванні" (DDoS)** – Ці атаки спрямовані на перевантаження сервера великою кількістю запитів, що може призвести до його недоступності для легітимних користувачів. Для

захисту від DDoS атак використовуються спеціалізовані рішення для моніторингу та фільтрації трафіку, які можуть виявляти та блокувати аномальні запити.

- **SQL-ін'єкції** – Це атаки, при яких зловмисник вводить шкідливі SQL команди через веб-форми або URL-параметри. Захист від таких атак включає в себе валідацію та санітізацію вхідних даних, використання параметризованих запитів та ORM (Object-Relational Mapping) систем[13].
- **Кросс-сайтовий скриптинг (XSS)** – При цьому типі атак зловмисники вводять шкідливі скрипти у веб-сторінки, які потім виконуються іншими користувачами. Захист полягає у фільтрації та екрануванні вхідних даних та використанні безпечних функцій для відображення даних на стороні клієнта.

- **Використання вогненних стін та інших захисних механізмів** – Веб-додаткові вогненні стіни (WAF) допомагають виявляти та блокувати шкідливі запити до веб-сервісу. Вони можуть бути налаштовані для фільтрації трафіку за різними критеріями, включаючи IP-адреси, строки запитів та типи HTTP-запитів[26].
- **Регулярне оновлення програмного забезпечення** – Однією з ключових практик у захисті веб-сервісів є своєчасне оновлення всіх компонентів системи, включаючи операційну систему, веб-сервер, бази даних та всі використовувані бібліотеки. Регулярні оновлення дозволяють виправляти відомі уразливості та забезпечувати захист від нових загроз[14].
- **Моніторинг та аудит безпеки** – Ефективна система моніторингу може вчасно виявляти підозрілу активність та потенційні атаки. Використання інструментів для логування та аналізу логів допомагає відстежувати нестандартну поведінку та швидко реагувати на інциденти безпеки.

Шифрування даних

- **HTTPS та SSL/TLS** – Основною метою шифрування даних при передачі через Інтернет є запобігання їх перехопленню та несанкціонованому доступу третіми сторонами. Протокол HTTPS, який включає шифрування SSL (Secure Sockets Layer) або TLS (Transport Layer Security), є стандартом для безпечного обміну даними[15]. Використання SSL/TLS дозволяє створити зашифрований канал між веб-браузером користувача та сервером, забезпечуючи конфіденційність та цілісність переданих даних. Це особливо важливо для транзакцій, що включають обробку конфіденційної інформації, такої як дані кредитних карт або особисті дані користувачів.

Аутентифікація та авторизація

- **Аутентифікація користувачів**: Аутентифікація - це процес визначення особистості користувача, що входить у систему. Традиційно це відбувається за допомогою комбінації імені користувача та пароля. Проте, для забезпечення більшої безпеки, часто використовуються додаткові методи, такі як багатофакторна аутентифікація (MFA)[16], де користувачеві потрібно надати два або більше доказів своєї особистості (наприклад, пароль плюс код із SMS).
- **Авторизація та контроль доступу**: Після аутентифікації користувача наступним кроком є авторизація, яка визначає, які ресурси чи дані користувач може використовувати. Це може бути реалізовано через рольовий контроль доступу (Role-Based Access Control, RBAC) або через більш гнучкі системи контролю доступу, засновані на політиках та дозволах.
- **Токени доступу та сесії**: Часто для управління сесіями користувачів та авторизацією використовуються токени доступу, такі як JSON Web Tokens (JWT)[27]. JWT - це безпечний спосіб представлення

клеймів між двома сторонами, що може включати інформацію про особистість користувача та його дозволи.

Використання цих практик та технологій дозволяє створити міцну основу безпеки для веб-сервісів. Важливо не лише імплементувати ці заходи, але й регулярно оновлювати їх та адаптувати до нових загроз та викликів у сфері кібербезпеки. Розуміння потенційних ризиків, реалізація ефективних захисних механізмів та регулярне оновлення та оцінка безпеки є ключовими для забезпечення безпеки веб-сервісів.

1.4 Сучасні тенденції у розробці веб-сервісів

Сучасний світ веб-розробки постійно еволюціонує, пропонуючи нові технології та підходи, які радикально змінюють спосіб створення та використання веб-сервісів. Однією з ключових тенденцій останніх років стала контейнеризація, з Docker на чолі цієї революції[17]. Контейнеризація дозволяє розробникам ізолювати додатки в контейнерах з усіма необхідними залежностями, що гарантує їх консистентність у різних середовищах. Це полегшує розгортання та масштабування веб-додатків, особливо у хмарних середовищах.

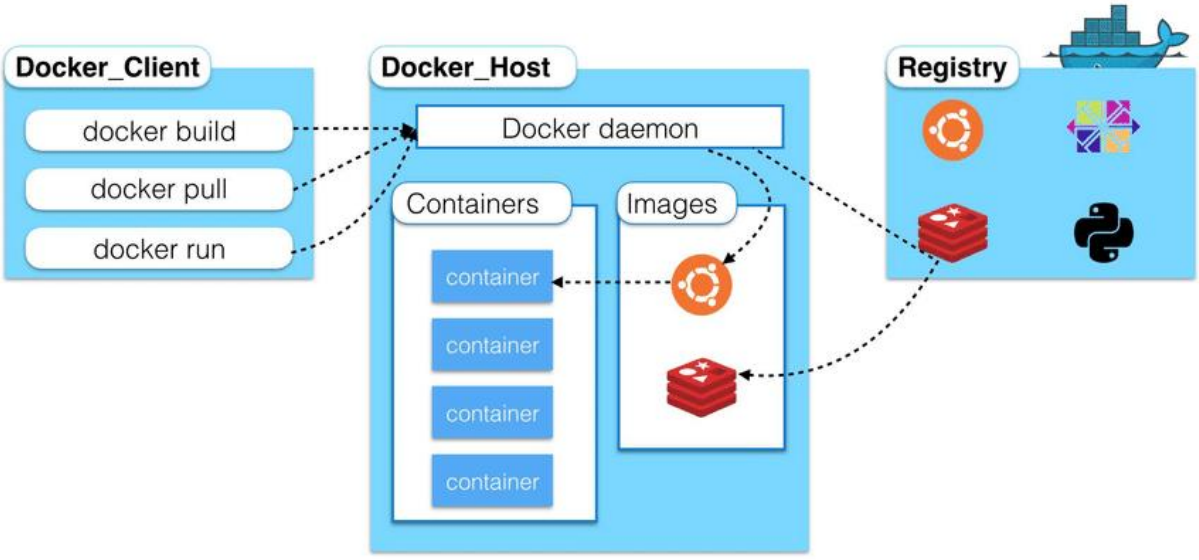


Рис.1.5 – Контейнерізація Docker

Іншою значною тенденцією є перехід до мікросервісної архітектури. Відхід від монолітних структур до дрібнозернистих, незалежних

мікросервісів значно спрощує процес управління великими додатками. Кожен мікросервіс відповідає за певний аспект функціональності та може бути розроблений, оновлений та розгорнутий незалежно від інших. Це сприяє більшій гнучкості та прискорює випуск нових оновлень і функціоналу.

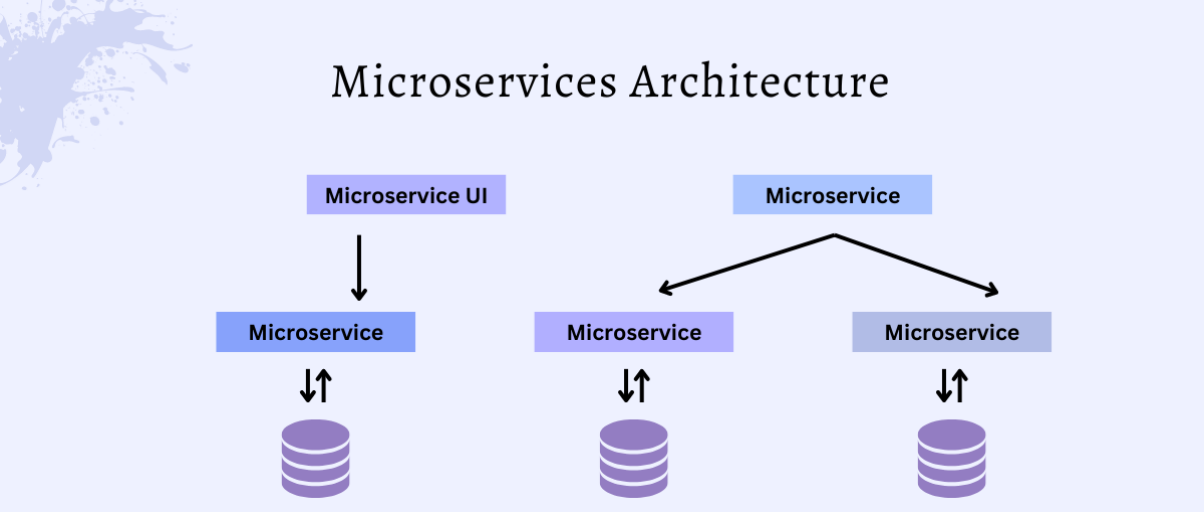


Рис.1.6 – Мікросервісна архітектура

Зростання популярності хмарних сервісів, таких як AWS, Azure та Google Cloud, також має величезний вплив на розвиток веб-сервісів. Хмарні платформи надають розробникам масштабовані, гнучкі та ефективні рішення для розгортання та управління їхніми веб-додатками[18]. Вони дозволяють використовувати ресурси більш економно та ефективно, надаючи при цьому високий рівень надійності та доступності.

У сукупності ці тенденції вказують на зміщення акцентів у розробці веб-сервісів від традиційних підходів до більш гнучких, масштабованих та ефективних рішень. Це відкриває нові можливості для розробників та бізнесів, прагнучих максимально використати потенціал сучасних технологій для створення інноваційних веб-додатків.

1.5 Аналіз вимог

1.5.1 Функціональні вимоги

Визначення основних функціональних можливостей веб-сервісу, враховуючи потреби користувачів:

1. **Реєстрація та авторизація користувачів:** Забезпечення можливості створення особистого аккаунта, який дозволить користувачам зберігати особисту інформацію та переглядати статус заявок на стипендії.

2. **Додавання та облік стипендій:** Ця функція дозволяє користувачам подавати заявки на стипендії, включаючи деталі та вимоги до пільгової стипендії. Користувачі можуть переглядати доступні стипендії, подавати заявки та отримувати інформацію про статус заявки.

3. **Створення розділів з контролем виплат, заборгованостей тощо для конкретного користувача:** Розробка системи, що дозволяє користувачам контролювати свої фінансові операції, включаючи виплати стипендій, нарахування та історію транзакцій. Також система має забезпечувати можливість перегляду існуючих заборгованостей та ведення їх обліку.

Ці функціональні можливості спрямовані на створення зручного та інтуїтивно зрозумілого веб-сервісу, який дозволить студентам ефективно управляти своїми фінансовими питаннями, пов’язаними з отриманням стипендій, та підтримувати прозорість усіх фінансових процесів.

1.5.2 Нефункціональні вимоги

Вивчення та визначення нефункціональних вимог, таких як продуктивність, доступність та безпека:

1. **Продуктивність –** Веб-сервіс повинен забезпечувати високу швидкість обробки даних та здатність обслуговувати велику кількість користувачів одночасно. Важливо забезпечити мінімальний час відгуку

системи, щоб користувачі могли ефективно взаємодіяти з сервісом без затримок.

2. **Доступність –** Сервіс повинен бути доступний 24/7 з будь-якої точки світу. Важливо забезпечити стабільну роботу сервісу, мінімізуючи час простоїв та забезпечуючи високий рівень надійності.

3. **Безпека –** Оскільки сервіс зберігає та обробляє особисті та фінансові дані користувачів, важливо забезпечити захист цих даних. Це включає в себе захист від несанкціонованого доступу, шифрування даних та використання сучасних методів аутентифікації.

4. **Масштабованість –** Сервіс повинен бути гнучким до масштабування, щоб відповідати зростаючій кількості користувачів та обсягів даних без втрати продуктивності.

5. **Зручність користування –** Інтерфейс сервісу повинен бути інтуїтивно зрозумілим та простим у використанні для користувачів з різним рівнем технічних навичок. Важливо забезпечити легку навігацію та доступність всіх необхідних функцій.

6. **Підтримка та оновлення –** Сервіс повинен мати надійну підтримку для вирішення будь-яких технічних проблем та регулярно оновлюватися для виправлення помилок, покращення функціональності та впровадження нових можливостей.

Ці нефункціональні вимоги є фундаментальними для забезпечення високої якості сервісу та задоволення потреб користувачів. Вони відіграють ключову роль у підтримці стабільності, безпеки та ефективності веб-сервісу

РОЗДІЛ 2
РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

2.1 Проєктування системи

2.1.1 Архітектурний план

Організація коду та структура проєкту. Організація коду в нашому проєкті базується на модульному підході, де кожен компонент, від UI елементів до сервісів обробки даних, розробляється та визначається як незалежний модуль. Це значно спрощує процес додавання нових функцій, внесення змін або виправлення помилок в системі. Кожен модуль має чітке призначення, що сприяє легкості управління кодом і його розширенню.

*Фронтенд-архітектура***.** У фронтенд-частині проєкту застосовується патерн "компонентного дизайну" у React. Цей підхід дозволяє створювати перевикористовувані, легко тестовані компоненти, що забезпечує ефективність розробки та підтримку коду. Для управління загальним станом додатку ми можемо використовувати Redux або Context API. Це забезпечує централізоване управління даними і дозволяє уникнути складнощів з передачею даних між компонентами.

*Бекенд-архітектура***.** Для бекенд-розробки обрано підхід REST або GraphQL для створення ефективного API. Це забезпечує гнучкість та оптимізоване взаємодію з фронтендом. Використання патерну мікросервісів дозволяє розділити бізнес-логіку на дрібні, незалежні модулі. Такий підхід полегшує процес розробки, підтримки та масштабування системи, дозволяючи кожному мікросервісу працювати та оновлюватися незалежно.

*База даних та зберігання даних***.** Важливим аспектом є використання нормалізації даних, що допомагає в ефективному зберіганні та управлінні даними, запобігаючи їх дублюванню. Структура бази даних планується

таким чином, щоб забезпечити логічний зв'язок між різними сегментами даних, що сприятиме легкості їх обробки та вилучення.

*Безпека та захист даних***.** Безпека системи є пріоритетом, особливо з огляду на зберігання та обробку особистих та фінансових даних користувачів. Реалізація механізмів аутентифікації та авторизації, захист від різних видів атак (наприклад, CSRF та XSS), а також шифрування даних є ключовими для забезпечення безпеки. Також планується регулярне оновлення безпеки та проведення аудиту системи.

*Тестування та якість коду***.** Впровадження автоматизованого тестування забезпечить високу надійність та якість коду. Це включає розробку модульних, інтеграційних та системних тестів. Використання лінтерів та форматерів допоможе підтримувати чистоту та послідовність коду.

*Розгортання та моніторинг***.** Застосування CI/CD піплайнів для автоматизації процесів тестування та розгортання важливо для підтримки стабільності та ефективності роботи сервісу. Моніторинг системи та логування діяльності дозволять відслідковувати роботу сервісу та швидко реагувати на можливі проблеми.

Цей детальний архітектурний план створює міцну основу для розробки веб-сервісу "Пільгові стипендії", забезпечуючи його надійність, ефективність та масштабованість.

2.1.2 Проєктування баз даних

Розробка бази даних для веб-сервісу вимагає вибору відповідної системи управління базами даних, що відповідає сучасним вимогам та забезпечує ефективність, масштабованість та гнучкість. Вибір пав на MongoDB - одну з найпопулярніших NoSQL баз даних, що активно використовується у розробці веб-додатків.

Основною перевагою MongoDB є її документо-орієнтований підхід, який ідеально підходить для обробки великих обсягів розподілених даних. Відсутність жорсткої схеми даних дозволяє нам з легкістю вносити зміни

у структуру бази, не вдаючись до складного процесу міграції даних. Це особливо важливо для динамічного середовища веб-додатків, де вимоги можуть змінюватися відповідно до потреб користувачів.

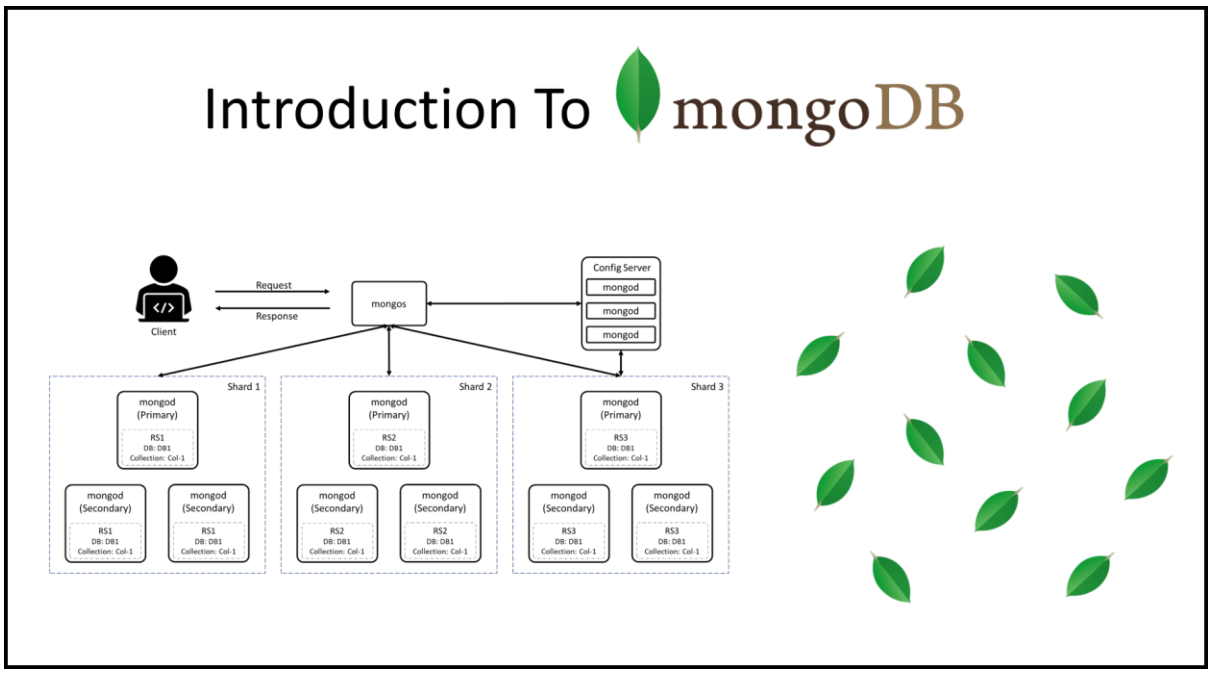


Рис.2.1 – MongoDB

MongoDB також відома своєю високою продуктивністю і швидкістю обробки запитів, що є критично важливим для веб-сервісів, які обслуговують велику кількість користувачів та транзакцій. Інтеграція MongoDB з веб-сервісом дозволить забезпечити швидкий доступ до даних, що підвищить загальну продуктивність та якість обслуговування користувачів.

Масштабованість є ще однією важливою перевагою MongoDB. Вона підтримує горизонтальне масштабування, що дозволяє системі ефективно справлятися зі зростаючим навантаженням і обсягами даних. Це забезпечує гнучкість та можливість розвитку нашого веб-сервісу без обмежень з боку системи зберігання даних.

Безпека даних є ще одним критичним аспектом, який було враховано при виборі MongoDB. Вона пропонує розширені можливості щодо аутентифікації, авторизації та шифрування даних, що дозволяє нам забезпечити захист конфіденційної інформації наших користувачів.

Завдяки цим перевагам, MongoDB є ідеальним рішенням для нашого проєкту, забезпечуючи потрібний рівень гнучкості, продуктивності, масштабованості та безпеки.

Розробка детальних схем баз даних, враховуючи надані приклади документів та потреби додатку

1. *Колекція Користувачів (Users):*
   - Кожен документ у цій колекції містить інформацію про окремого користувача.
   - Поля документа включають: *_id* (унікальний ідентифікатор), *username, password (зашифрований), firstname, lastname, email.*

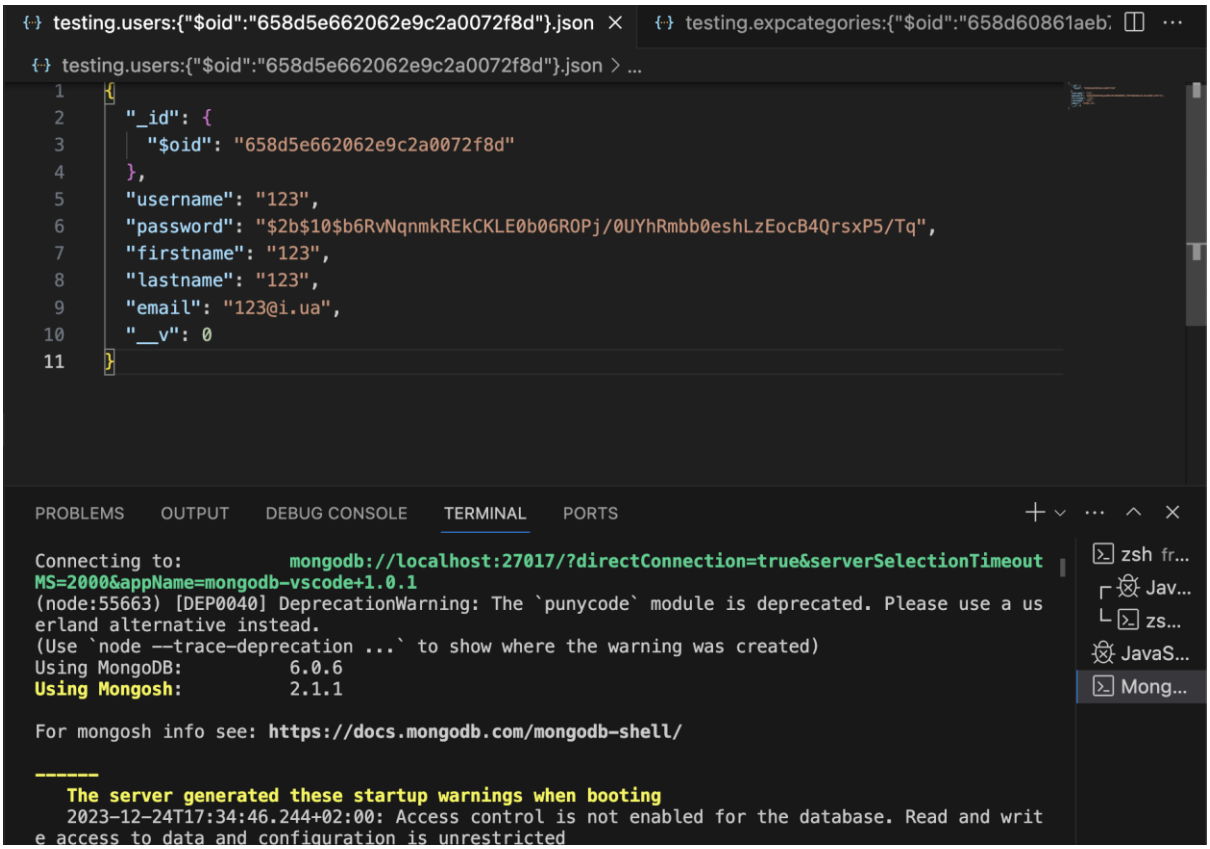


Рис.2.2 – Приклад створеного документа з колекцією користувачів

   - Ця структура дозволяє зберігати основну інформацію про користувачів та їх аутентифікаційні дані.

2. **Колекція Фінансових Транзакцій (Financial Transactions):**

- Документи у цій колекції зберігають інформацію про фінансові операції користувачів.
- Структура документа включає: **_id**, **username** (посилання на користувача), **amount** (сума транзакції), **selectedcategory** (категорія транзакції, наприклад "Student fee" або "Student benefits"), **option** (тип транзакції, наприклад "Savings" або "Income"), **createdAt**, **updatedAt**.
- Це дозволяє відслідковувати та аналізувати фінансові потоки кожного користувача.

3. **Колекція Категорій (Categories):**
   - В цій колекції зберігаються можливі категорії фінансових транзакцій.
   - Кожен документ містить: **_id**, **username**, **category** (назва категорії), **option** (тип транзакції), **color** (колір, асоційований з категорією).
   - Такий підхід дозволяє користувачам кастомізувати категорії та забезпечує зручність візуалізації фінансових даних.

При проєктуванні бази даних слід звернути увагу на наступні аспекти:

- **Індексація:** Індексація полів, які часто використовуються для пошуку (наприклад, **username**), для підвищення ефективності запитів.
- **Безпека:** Застосування шифрування для чутливих даних, таких як паролі, та реалізація політик доступу для захисту інформації про користувачів.
- **Оптимізація:** Аналіз і оптимізація схем для забезпечення швидкого доступу до даних, особливо при великих обсягах інформації.

Ця структура бази даних є гнучкою та масштабованою, дозволяючи ефективно зберігати, обробляти та аналізувати дані, що є критично важливим для функціонування веб-сервісу.

2.1.3 Вибір технологій

При проєктуванні веб-сервісу, важливо вибрати технології, які забезпечать не тільки високу продуктивність та надійність, але й дадуть можливість гнучкої розробки та масштабування. Основні технології, що були обрані для фронтенду та бекенду, детально визначають структуру та можливості майбутнього додатку.

Фронтенд

Фронтенд-частина веб-сервісу побудована з використанням низки передових технологій, які відіграють ключову роль у створенні ефективного та інтуїтивно зрозумілого інтерфейсу користувача.

React та його екосистема – Основою фронтенду є React - сучасна бібліотека JavaScript для розробки динамічних інтерфейсів користувачів. React використовує компонентний підхід, що дозволяє розбити інтерфейс на повторно використовувані частини, полегшуючи розробку та тестування. Компонентна архітектура забезпечує легкість управління станом інтерфейсу та дозволяє створювати більш швидкі та відповідні додатки.

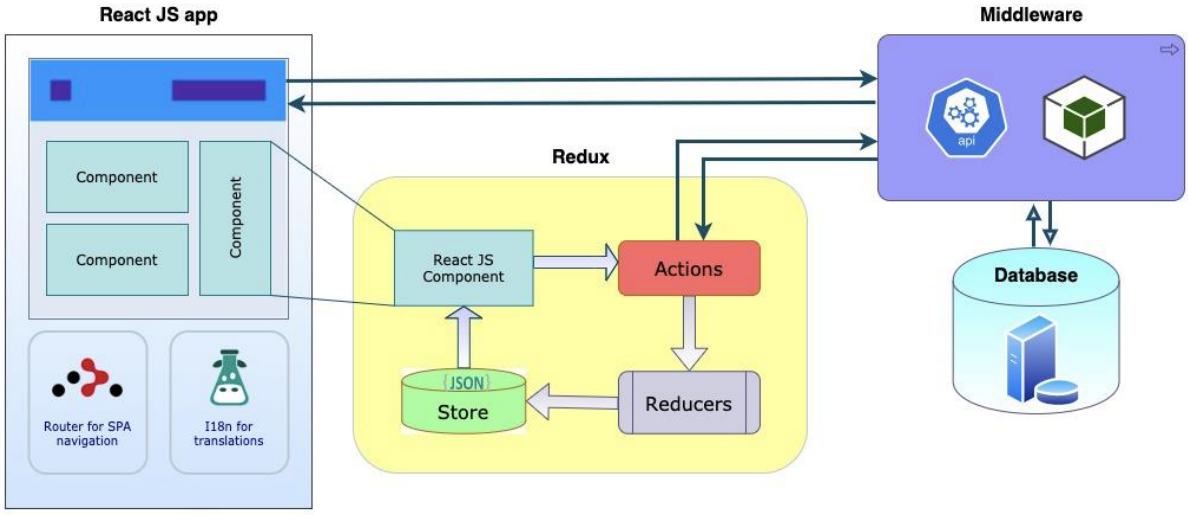


Рис.2.3 – Архітектура React та його компонентів

Ant Design для стилізації – Для стилізації використовується Ant Design (antd), бібліотека UI компонентів для React. Ant Design надає великий набір готових до використання компонентів, таких як форми, кнопки, меню, які дозволяють створювати красиві та консистентні інтерфейси з мінімальними зусиллями. Це істотно прискорює процес розробки та допомагає підтримувати єдиний стиль дизайну по всьому додатку.

Рис.2.4 – Бібліотека Ant Design

Візуалізація даних з Chart.js та React-Chartjs-2 – Для візуалізації даних використовується інтеграція Chart.js та React-Chartjs-2, що дозволяє легко створювати інтерактивні графіки та діаграми. Це важливо для аналітичних функцій нашого веб-сервісу, де візуалізація фінансових показників і статистики забезпечує зручність та ефективність в сприйнятті інформації користувачами.

Навігація та запити до сервера – Для навігації у веб-додатку використовується React Router, який дозволяє організувати взаємодію між різними сторінками та компонентами без перезавантаження сторінки. Axios, бібліотека для виконання HTTP-запитів, використовується для зв'язку з бекендом. Вона забезпечує зручний інтерфейс для відправки запитів та обробки відповідей сервера.

Vite у проєкті – Vite, інструмент сучасної збірки для фронтенд-проєктів, використовується у нашому проєкті для швидкого старту розробки та гарячого перезавантаження. Завдяки Vite розробка стає надзвичайно швидкою і ефективною, забезпечуючи миттєве оновлення модулів у режимі реального часу.

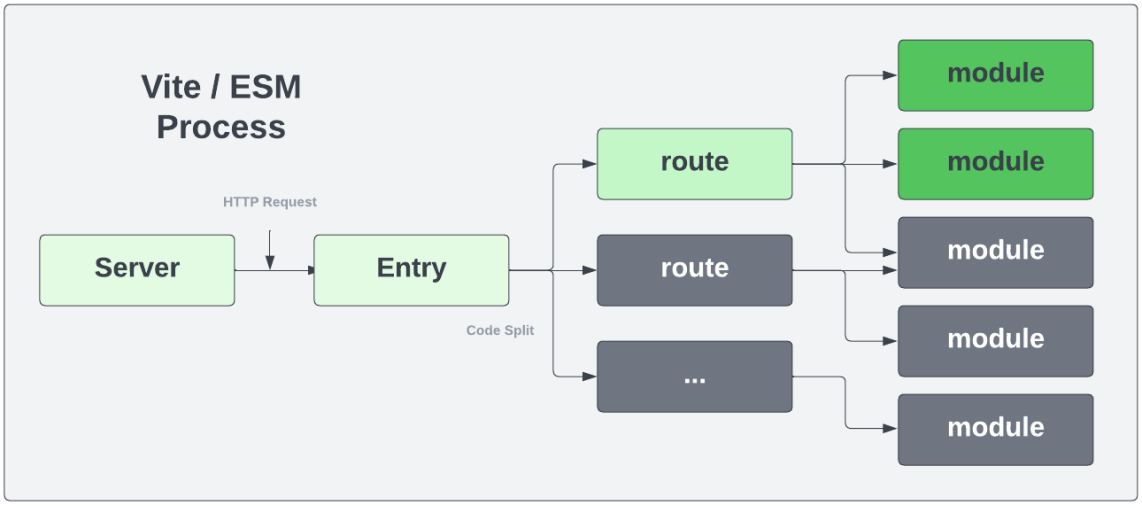


Рис.2.5 – Схема роботи Vite

Ці технології у сукупності створюють потужний та гнучкий інструментарій для розробки інтуїтивно зрозумілого, продуктивного та візуально привабливого фронтенду для веб-сервісу

Бекенд

Бекенд-частина веб-сервісу є фундаментом для обробки запитів, зберігання даних та взаємодії з фронтендом. Вибір технологій для бекенду був здійснений з метою забезпечення високої продуктивності, безпеки та гнучкості в розробці.

Node.js та Express – Node.js є вибором для створення серверної частини веб-додатку завдяки своїй асинхронній архітектурі та неблокуючому вводу/виводу. Це забезпечує високу продуктивність обробки запитів та дозволяє ефективно управляти одночасними процесами, що є критично важливим для додатків з великим обсягом трафіку.

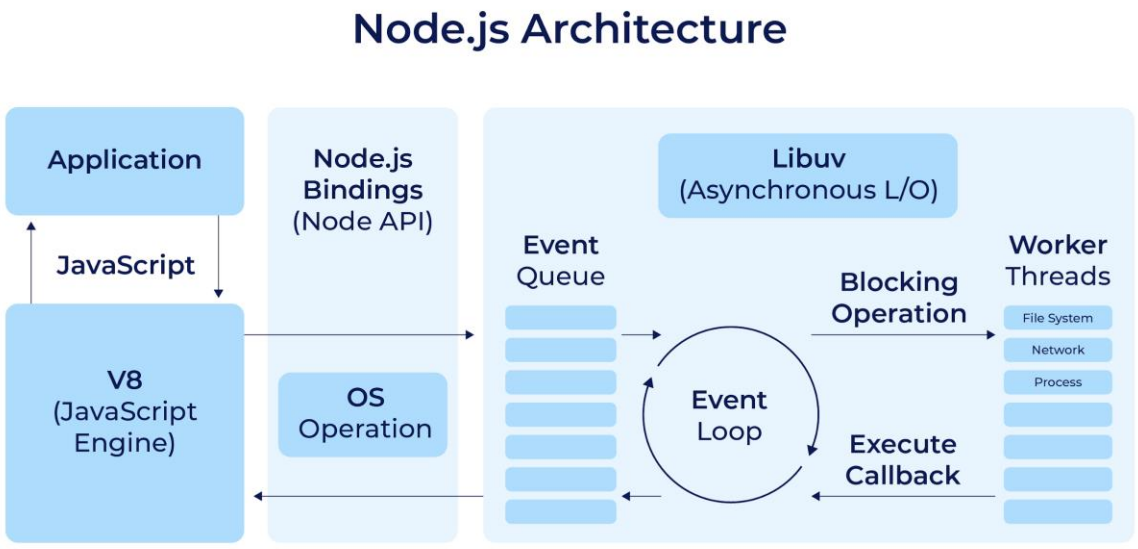


Рис.2.6 – Архітектура NodeJs

Express, фреймворк для Node.js, використовується для спрощення процесу розробки веб-додатків. Він надає гнучкий набір інструментів для створення маршрутів, обробки запитів та відповідей, і значно спрощує розробку RESTful API. Використання Express сприяє швидкому та легкому розгортанню серверної логіки.

Безпека та зберігання даних – Безпека даних користувачів є пріоритетом у нашому веб-сервісі. Використання bcrypt для хешування паролів є ключовим у забезпеченні безпеки користувацьких акаунтів. Bcrypt дозволяє надійно зберігати паролі, що захищає від потенційних загроз, таких як витік даних.

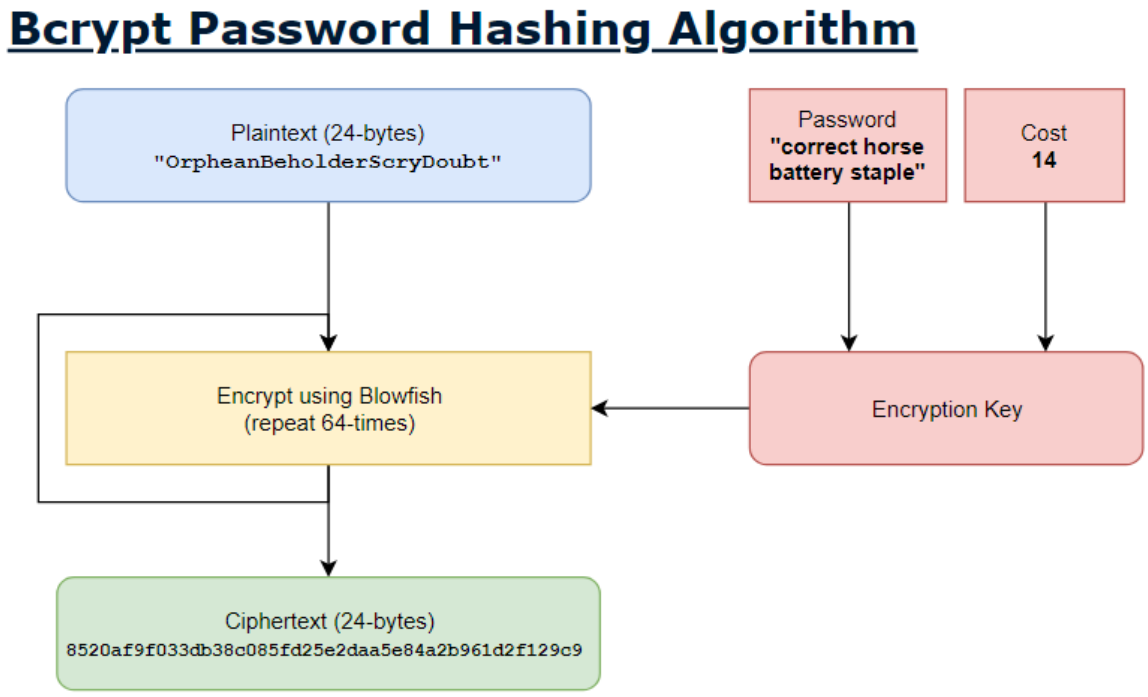


Рис.2.7 – Алгоритм хешування паролів Bcrypt

Інтеграція з MongoDB через Mongoose є важливою для ефективної роботи з базою даних. Mongoose надає зручний інтерфейс для роботи з MongoDB, дозволяючи легко створювати моделі даних, здійснювати запити до бази даних та обробляти результати. Це спрощує взаємодію з базою даних і підвищує ефективність розробки.

Загальна архітектура бекенду – Архітектура бекенду розроблена з метою забезпечення високої гнучкості та масштабованості. Це дозволяє легко адаптувати систему до змінюваних вимог та легко інтегрувати нові функції. Комбінація Node.js, Express, bcrypt та Mongoose створює потужний та надійний фундамент для нашого веб-сервісу, забезпечуючи безпечну та ефективну роботу бекенду.

2.2 Реалізація серверної частини

2.2.1 Бізнес-логіка

Розробка бізнес-логіки для веб-сервісу є ключовим елементом, який забезпечує виконання всіх функціональних вимог проєкту. Бізнес-логіка включає в себе ряд компонентів і механізмів, які разом формують основу для обробки даних та взаємодії з користувачами.

Основні аспекти бізнес-логіки

1. **Аутентифікація та авторизація користувачів:** Система аутентифікації дозволяє користувачам безпечно увійти в систему, використовуючи свій логін та пароль. Авторизація визначає, які дії користувач може виконувати всередині сервісу, відповідно до його ролі та прав доступу.

2. **Обробка фінансових транзакцій:** Реалізація логіки для обробки та зберігання інформації про фінансові транзакції користувачів, включаючи реєстрацію платежів, нарахувань та інших фінансових операцій.

3. **Управління даними користувачів:** Створення механізмів для зберігання та обробки особистих даних користувачів, включаючи персональну інформацію, історію транзакцій та налаштування аккаунту.

4. **Взаємодія з базою даних:** Інтеграція серверної частини з базою даних для забезпечення ефективного зберігання, пошуку та обробки даних. Використання Mongoose як ODM (Object Data Modeling) бібліотеки для MongoDB спрощує роботу з документо-орієнтованою базою даних.

5. **Логування та аудит:** Реалізація системи логування для відстеження дій користувачів та системних подій. Це дозволяє забезпечити прозорість діяльності веб-сервісу та спростити процес виявлення та виправлення помилок.

6. **API для взаємодії з фронтендом:** Розробка RESTful API для забезпечення зв'язку між серверною та клієнтською частинами додатку.

API дозволяє ефективно обмінюватися даними та командами, що забезпечує гнучкість та швидкість взаємодії компонентів системи.

Розробка цієї бізнес-логіки вимагає глибокого розуміння потреб користувачів та технічних можливостей, щоб забезпечити надійність, безпеку та високу продуктивність веб-сервісу. Кожен аспект бізнес-логіки вимагає ретельного планування та тестування, щоб гарантувати, що веб-сервіс відповідає всім вимогам та очікуванням користувачів.

2.2.2 Розробка серверної частини

Структура backend частини проєкту побудована за принципами чистоти коду та модульності, що дозволяє підтримувати та розширювати систему з легкістю. Основний серверний файл, **server.js**, є входом до сервера та координує взаємодію між різними модулями і мідлварами.

Структура проєкту

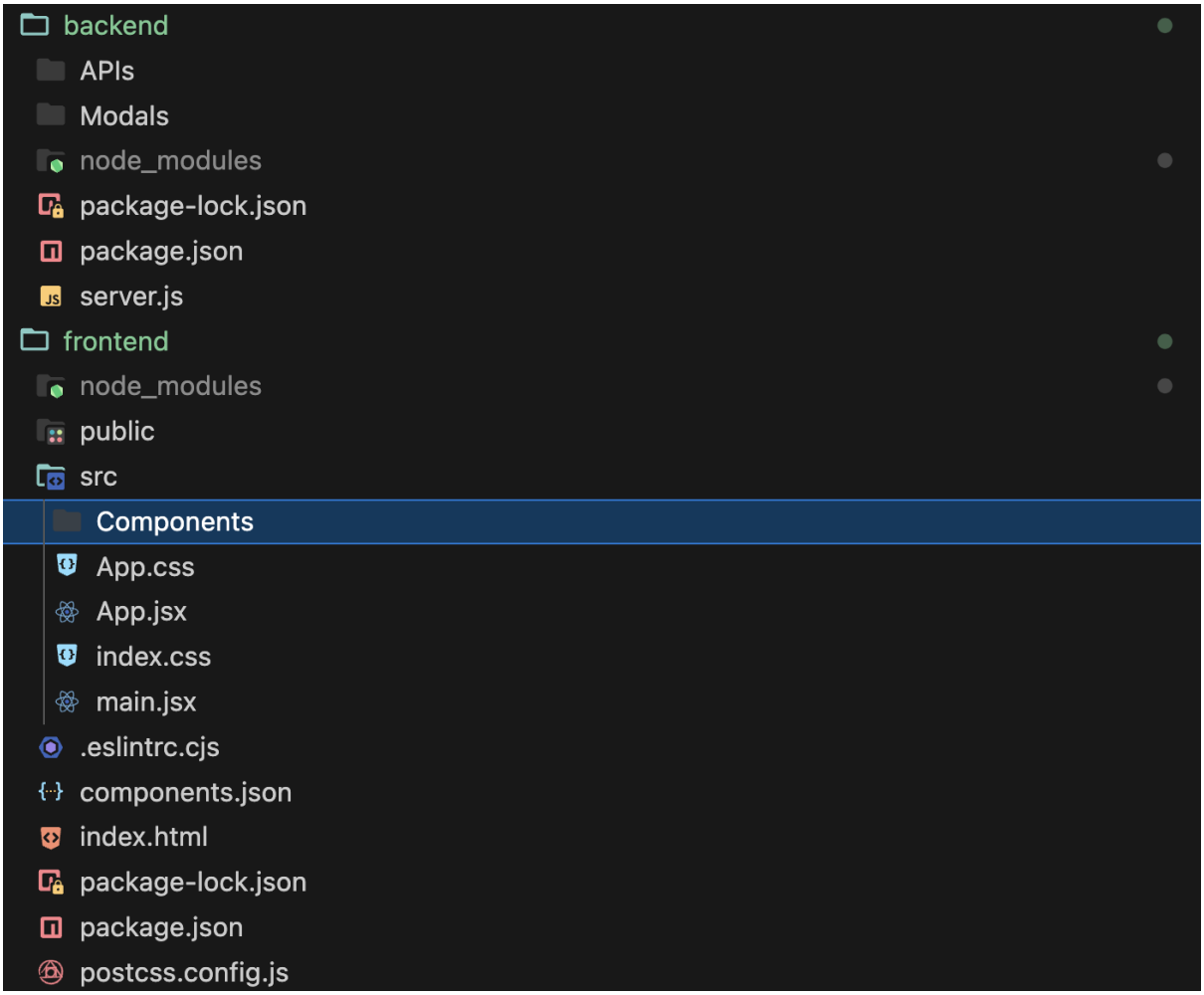


Рис.2.8 – Файлова структура проетку

- APIs: Папка APIs містить різні модулі, кожен з яких відповідає за конкретний набір функціональностей, таких як логін, реєстрація, обробка транзакцій та управління категоріями. Це дозволяє ізолювати кожен аспект бізнес-логіки в окремий файл, що полегшує розуміння та тестування коду.
- Modals: Папка Modals використовується для визначення схем бази даних та моделей Mongoose, що репрезентують структуру даних і взаємодію з MongoDB. Кожна модель відображається на конкретну колекцію у базі даних і визначає методи для взаємодії з цією колекцією.

Ключові компоненти backend частини

- Express Framework: В якості основи сервера використовується Express, потужний фреймворк, що надає зручні інструменти для створення веб-серверів, обробки запитів, маршрутизації, роботи з мідлварами і API.

- Маршрутизація: Сервер визначає маршрути для обробки запитів користувачів, кожен з яких звертається до відповідного API модуля. Наприклад, маршрут для реєстрації (/userRegister) ініціює модуль Register, який займається створенням нових користувачів у системі.
- Безпека: За допомогою бібліотеки bcrypt здійснюється хешування та забезпечення безпеки паролів користувачів, перш ніж вони будуть збережені у базі даних.
- Обробка Даних: Модулі, такі як transaction і addCategory, виконують операції з даними, включаючи створення, оновлення, та видалення записів у базі даних, забезпечуючи важливу бізнес-логіку, пов'язану з фінансовими транзакціями та управлінням категоріями.
- Взаємодія з базою даних: За допомогою mongoose встановлюється зв'язок між Express-додатком та MongoDB, дозволяючи легко та ефективно здійснювати CRUD-операції з даними.

Кожен компонент бекенду є частиною єдиної системи, яка забезпечує стабільність та відповідність функціональним вимогам веб-сервісу. Розуміння того, як ці компоненти взаємодіють, є ключем до розробки надійного та масштабованого рішення.

2.2.3 Розробка API

Перш ніж глибше зануритися в технічні деталі, важливо розуміти загальну картину того, як ключові компоненти серверної частини нашого веб-сервісу "Пільгові стипендії" взаємодіють між собою. Особлива увага приділяється API, яке діє як міст між клієнтською частиною та сервером, дозволяючи обробляти запити та передавати відповідні відгуки.

API логіна є одним із найважливіших компонентів аутентифікації користувачів в системі. Воно виконує критичну функцію перевірки правильності введених даних та надає доступ до користувацького інтерфейсу веб-сервісу. Тепер давайте розглянемо код та його реалізацію більш детально:

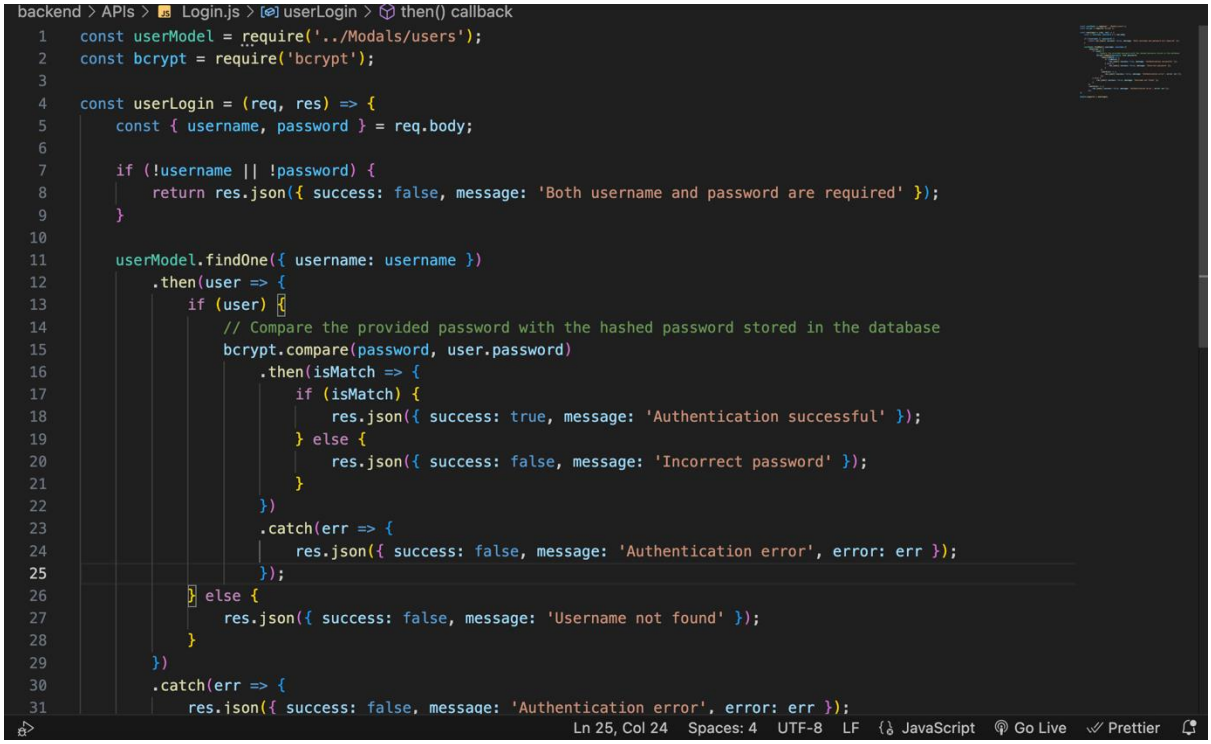


Рис.2.9 – Реалізація API логіну

Код, що відповідає за логін, починається з імпортування необхідних модулів: **userModel** для взаємодії з базою даних користувачів та **bcrypt** для роботи з хешуванням паролів. Функція **userLogin** отримує об'єкт запиту **req** та відповіді **res** як параметри.

Код перевіряє наявність імені користувача та пароля у тілі запиту. Якщо один з параметрів відсутній, відправляється відповідь з повідомленням про помилку. Якщо всі дані присутні, виконується пошук користувача в базі даних за іменем користувача.

У разі успішного знаходження користувача, **bcrypt** порівнює введений пароль з хешем, збереженим у базі даних. Якщо паролі збігаються, користувачу надається доступ. В протилежному випадку, відправляється повідомлення про помилку.

Кожен крок процесу логіна супроводжується відловлюванням можливих помилок та відправкою відповідних відгуків, що дозволяє забезпечити належну обробку виняткових ситуацій та інформувати користувача про стан процесу аутентифікації.

Така структура та реалізація API логіна забезпечує безпечне та надійне підтвердження особи користувачів, що є фундаментальним для забезпечення цілісності веб-сервісу

У фокусі огляду тепер перебуває API, яке було спроєктовано для всебічного моніторингу фінансових операцій користувачів у веб-сервісі "Пільгові стипендії". Цей API відіграє надзвичайно важливу роль, оскільки воно дозволяє не лише контролювати надходження пільгових стипендій, але й надає користувачам інструменти для управління їхніми збереженнями, нарахуваннями та іншими фінансовими транзакціями. Така багатоаспектна функціональність є ключовою для забезпечення користувачам глибокого та зручного управління особистими фінансами.

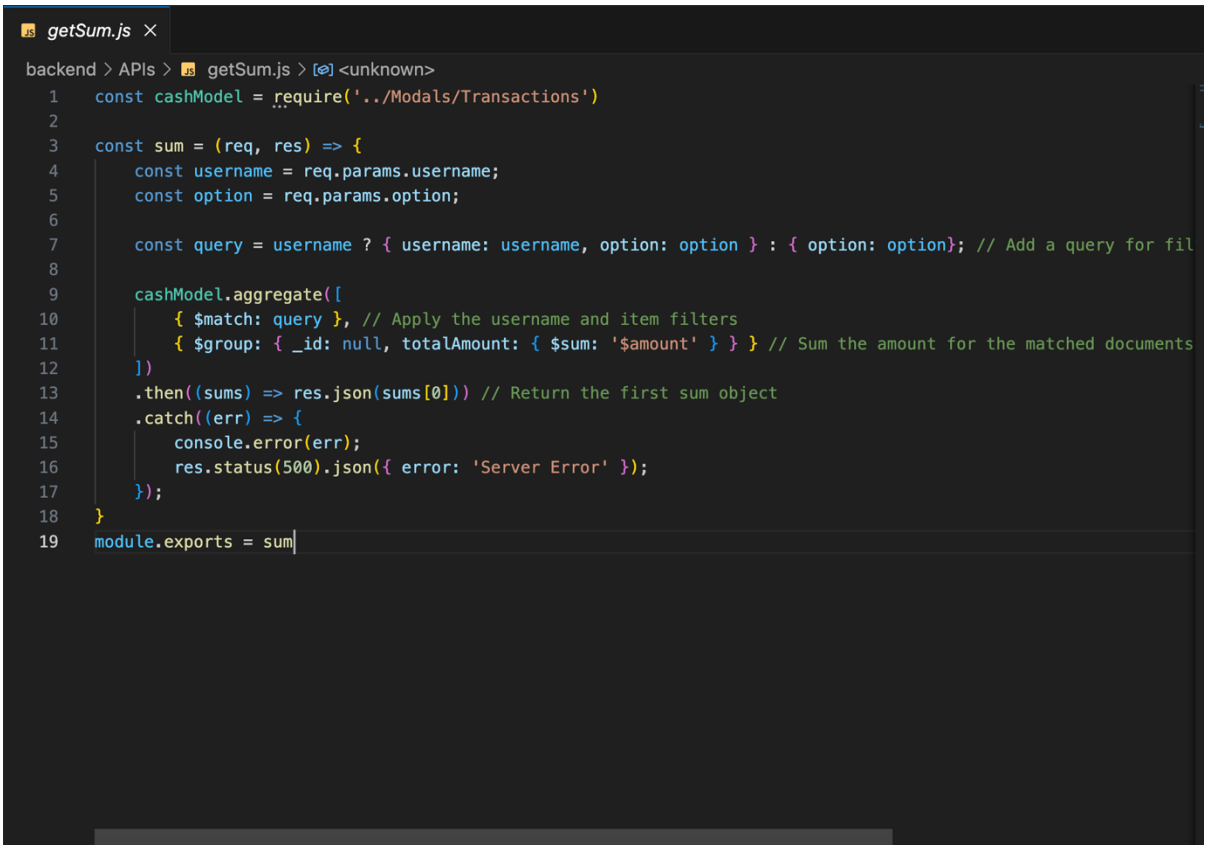


Рис.2.10 – getSum API

Центральний компонент API – це модуль **cashModel**, який інтегрований із колекцією транзакцій у базі даних. Функція **sum**, яка є частиною цього модуля, виконує обробку запитів на отримання сумарної інформації по транзакціях користувача. Запит включає параметри

username та **option**, котрі використовуються для відбору транзакцій за вказаними критеріями.

Побудований запит **query** здатен динамічно адаптуватися до вимог користувача, дозволяючи фільтрувати дані за іменем та типом операції, надаючи широкий спектр можливостей для аналізу фінансових дій. Використовуючи могутність агрегаційних функцій Mongoose, система виконує групування та підсумовування даних, що надає користувачу цілісну картину його фінансової активності.

При успішному завершенні запиту, результати повертаються користувачу у зрозумілому форматі. У випадку помилок у процесі агрегації, сервер відповідає повідомленням про помилку, забезпечуючи зворотній зв'язок та можливість корекції дій користувача.

Такий комплексний підхід до розробки API для розрахунку фінансових операцій є важливим для належного функціонування веб-сервісу "Пільгові стипендії". Він не тільки надає користувачам потужний інструмент для управління особистими фінансами, але й відкриває шляхи для подальших розширень та вдосконалень системи.

Заключним етапом огляду серверної частини розробки веб-сервісу "Пільгові стипендії" стане детальний огляд файлу **server.js**. Цей файл є вирішальним, оскільки він ініціює сервер та встановлює ключові зв'язки між різними API, моделями та мідлварами.

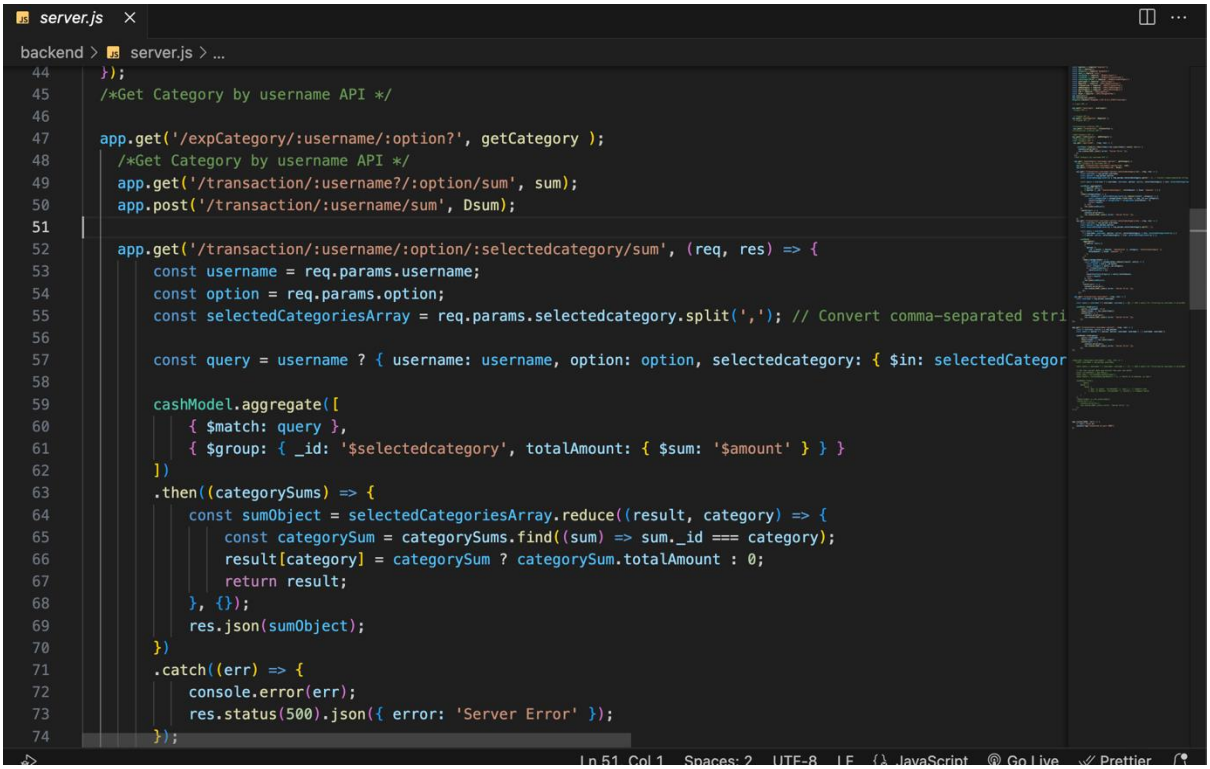


Рис.2.11 – server js

Файл **server.js** стартує з імпортування необхідних модулів та налаштування основних конфігурацій. Використання **express** як основи для сервера, інтеграція з MongoDB через **mongoose**, та встановлення **cors** для обробки міждоменних запитів є фундаментальними для функціонування веб-сервісу. Моделі, такі як **userModel**, **cashModel**, та **expCategoryModel**, представляють структуру даних із якою сервер буде взаємодіяти.

Кожен ендпойнт, такий як **/userLogin** або **/userRegister**, асоційований з відповідною логікою обробки запитів, що реалізована у модулях API. Ці ендпойнти слугують як точки входу для виконання аутентифікації, реєстрації, створення транзакцій та управління категоріями. Маршрутизація та логіка обробки API є відображенням бізнес-логіки, і вони відіграють ключову роль у забезпеченні необхідних функціональних можливостей сервісу.

Окрім основних ендпойнтів, **server.js** також містить додаткові маршрути для специфічних операцій, таких як агрегація даних для візуалізації, що дає змогу користувачам отримувати аналітичні дані за їх

фінансовими операціями. Всі ці елементи разом формують складну і гнучку систему, що може швидко адаптуватися до змінюваних потреб користувачів.

Завершально, сервер ініціює прослуховування на порту 3000, що є стандартним для локальних розробок. Повідомлення про успішне підключення є індикатором того, що сервер готовий до обробки запитів.

2.2.4 Розробка клієнтської частини

Розробка клієнтської частини веб-сервісу займає важливе місце в загальному процесі створення додатку. Клієнтська частина, або "фронтенд", є тим, що користувачі бачать і з чим взаємодіють безпосередньо, тому її розробка вимагає особливої уваги до деталей інтерфейсу, користувацького досвіду та оптимізації.

Основні аспекти розробки фронтенд складової додатку

1. *Створення інтерфейсу користувача:* Фронтенд розробляється з використанням React, що дозволяє створювати компоненти, які легко можуть бути повторно використані та взаємодіяти між собою. Це сприяє створенню динамічного та адаптивного інтерфейсу, який може бути оптимізований для різних пристроїв та розмірів екрану.

2. *Забезпечення взаємодії з сервером:* Взаємодія з серверною частиною здійснюється через API запити, які організовані за допомогою бібліотеки Axios. Це дозволяє клієнтській частині відправляти та отримувати дані від сервера, реалізовувати функціонал реєстрації, авторизації, управління транзакціями та багато іншого.

3. *Управління станом додатку:* Для управління станом додатку використовуються контексти React або бібліотека управління станом, така як Redux, яка дозволяє зберігати стан користувацького інтерфейсу та доступ до нього з різних частин додатку.

4. *Маршрутизація:* Маршрутизація в клієнтській частині веб-додатку важлива для забезпечення користувачів зручною навігацією по додатку. React Router використовується для контролю відображення компонентів відповідно до URL в браузері.

5. *Оптимізація продуктивності:* Оптимізація продуктивності фронтенду включає зменшення часу завантаження сторінок, оптимізацію зображень та використання лінивої загрузки (lazy loading) для компонентів та маршрутів, що не потрібні відразу.

6. *Тестування та забезпечення якості:* Ретельне тестування фронтенду проводиться для забезпечення його надійності та відсутності помилок. Використовуються як ручні, так і автоматизовані тести для різних компонентів, а також інтеграційні тести для перевірки взаємодії між фронтендом та бекендом.

Завершивши розробку серверної частини та перевіривши її працездатність, ми готові до переходу до клієнтської частини. Тут ми зосередимо наші зусилля на створенні інтуїтивно зрозумілого користувацького інтерфейсу, який забезпечить користувачам комфортне та ефективне взаємодію з веб-сервісом.

Огляд компонентів клієнтської частини

Компонент MainAppRoutes

В основу навігації по клієнтській частині веб-сервісу закладено компонент **MainAppRoutes**, який використовує можливості бібліотеки **react-router-dom** для організації маршрутизації всередині додатку. Даний компонент відіграє роль маршрутизатора, визначаючи структуру шляхів (routes) і відповідні компоненти, які будуть відображені при доступі до певних URL.

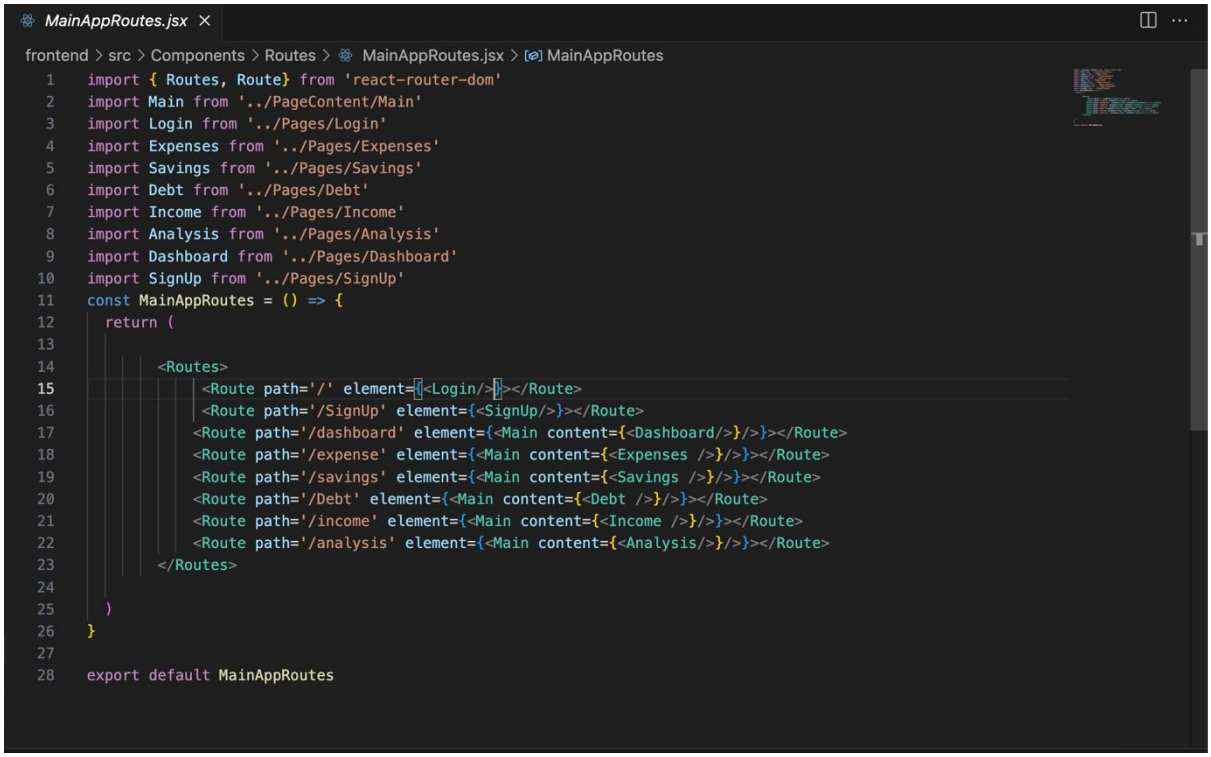


Рис.2.12 – Маршрутизація додатку з компонентом MainAppRoutes

Структура MainAppRoutes включає наступні маршрути

- **Основна сторінка (/):** Призначена для входу в систему, цей маршрут відображає компонент **Login**, який є вхідною точкою для аутентифікації користувачів.
- **Реєстрація (/SignUp):** Шлях для нових користувачів, що бажають створити акаунт, веде до компонента **SignUp**, де користувачі можуть заповнити форму реєстрації.
- **Панель управління (/dashboard):** Цей маршрут дозволяє користувачам отримати доступ до головного інтерфейсу додатку, представленого через компонент **Dashboard**, який вбудований у шаблон **Main**.
- **Витрати (/expense):** Для керування та аналізу витрат користувачів є відведений окремий шлях, що відкриває компонент **Expenses** в контексті **Main**.
- **Заощадження (/savings):** Шлях до компонента **Savings**, де користувачі можуть керувати своїми заощадженнями.

- **Доходи (/income):** Секція **Income** дозволяє користувачам відстежувати свої доходи.
- **Аналіз (/analysis):** Компонент **Analysis** надає функціональність для детального аналізу фінансових операцій.

Кожен з цих маршрутів асоціюється з відповідним компонентом, який надає потрібну функціональність. Використання загального компонента **Main** для обгортання кожної сторінки дозволяє уніфікувати загальний вигляд додатку і забезпечити консистентність користувацького досвіду.

MainAppRoutes є важливим компонентом у структурі фронтенду, оскільки саме через нього проходить навігація користувача по всьому додатку, забезпечуючи плавний і логічний перехід між різними секціями та функціоналом веб-сервісу.

Компонент Navbar

Навігаційна панель є важливою частиною користувацького інтерфейсу веб-сервісу, що забезпечує користувачів зручним доступом до різних секцій додатку. **Navbar** використовує компоненти з бібліотеки **antd**, які сприяють створенню візуально привабливого та інтуїтивно зрозумілого інтерфейсу.

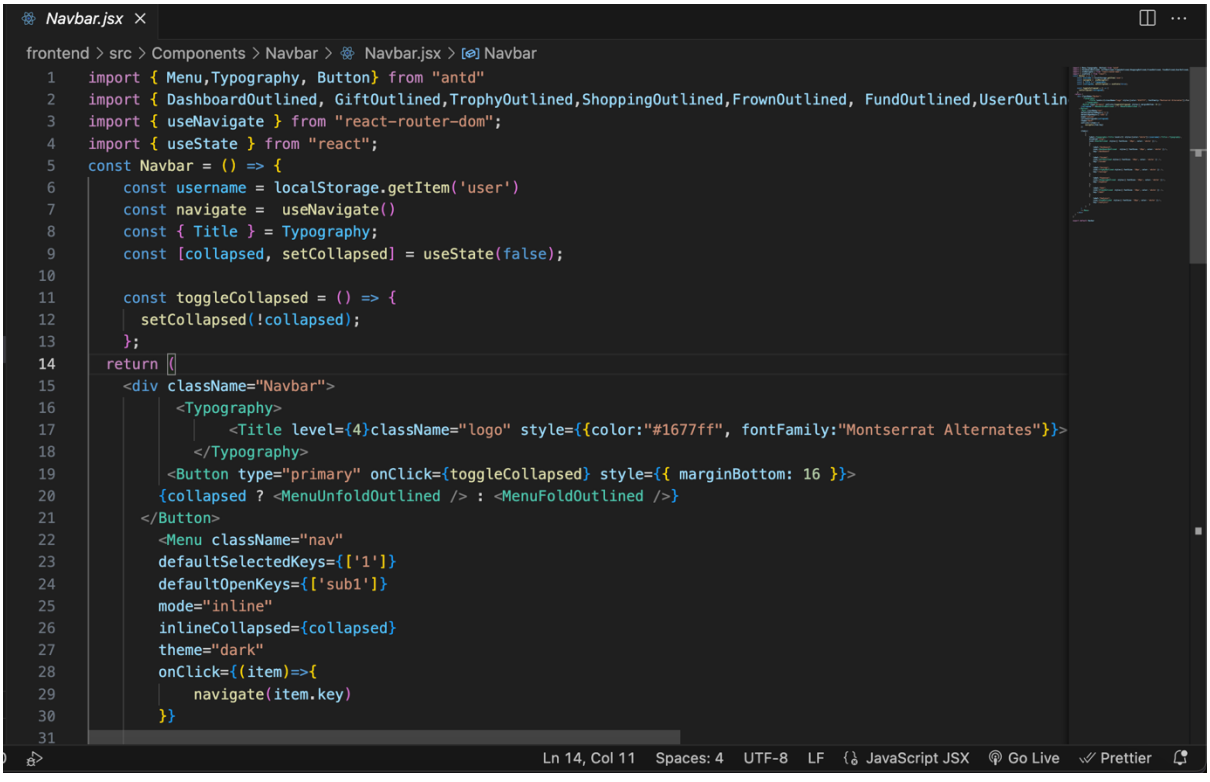


Рис.2.13 – Код реалізація навігаційної панелі

Деталі реалізації Navbar

- **Структура та стилізація –** Компонент **Navbar** включає логотип **FinRec**, який стилізований за допомогою **Typography** та **Title** для надання унікальної візуальної ідентичності. Кнопка для згортання або розгортання меню дозволяє оптимізувати простір на екрані, а також підвищує зручність використання на пристроях з обмеженим дисплеєм.
- **Динамічна інтерактивність –** Стан **collapsed** відповідає за стан згорнутості меню та управляється через хук **useState**. Функція **toggleCollapsed** змінює стан цього параметра, що дозволяє користувачу згортати або розгортати меню.
- **Маршрутизація та навігація –** За допомогою хука **useNavigate** з **react-router-dom** здійснюється навігація до різних секцій додатку. Кожен пункт меню асоційований з відповідним шляхом (**key**), і при виборі користувача, вони перенаправляються до вказаної секції.
- **Іконографія та доступність –** Іконки, такі як **DashboardOutlined**, **GiftOutlined**, **TrophyOutlined** та інші, забезпечують зрозумілу візуальну індикацію для кожної секції, роблячи навігацію більш інтуїтивною. Іконки та текстові мітки підсвічуються для кращої видимості.
- **Персоналізація –** Ім'я користувача, отримане з **localStorage**, відображається у верхній частині меню, що надає елемент персоналізації та вітає користувача в системі.

Компонент Dashboard

Компонент **Dashboard** є центральною частиною веб-додатка і призначений для відображення основних фінансових показників та статистики стосовно пільгових стипендій. Він складається з різних елементів, які відображаються на сторінці інтерфейсу користувача.

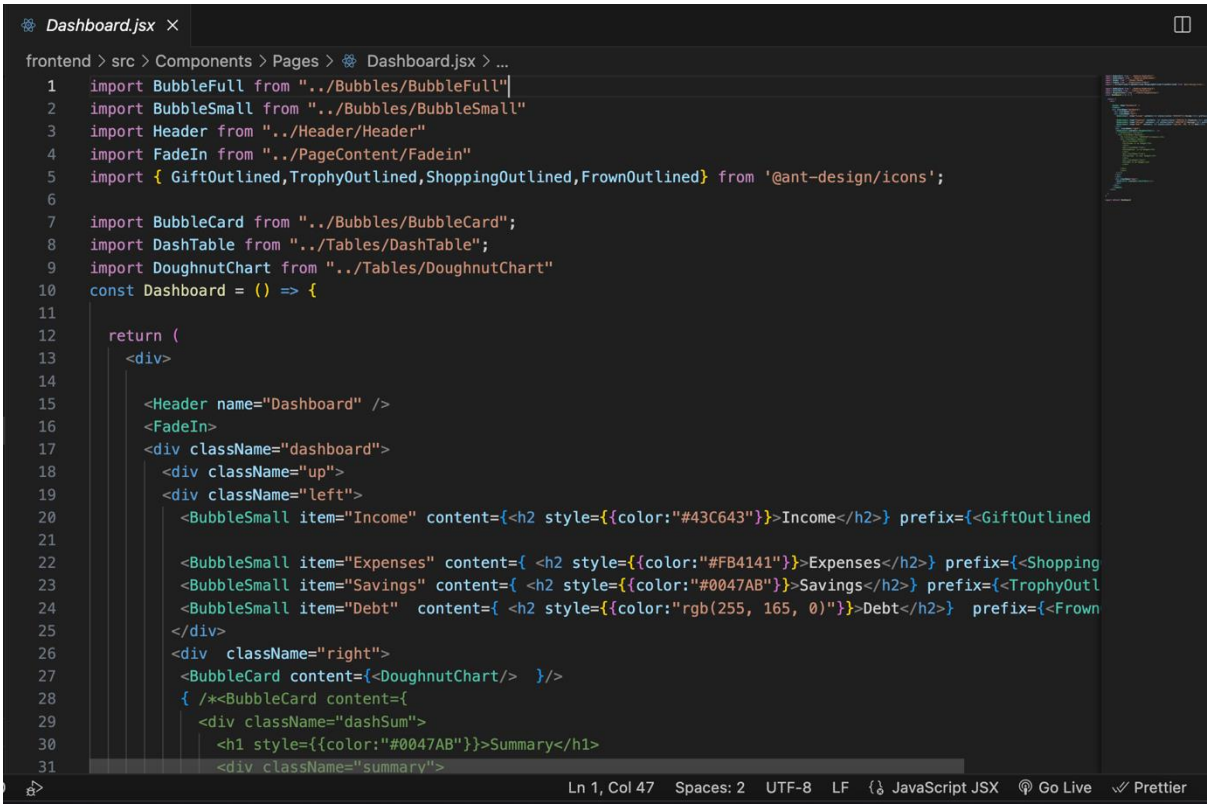


Рис.2.14 – Реалізація компоненту Dashboard

Починаємо з імпорту необхідних компонентів та іконок, які будуть використовуватися у компоненті Dashboard. Код містить імпорти таких компонентів, як **BubbleFull**, **BubbleSmall**, **Header**, **FadeIn**, а також іконок **GiftOutlined**, **TrophyOutlined**, **ShoppingOutlined** та **FrownOutlined**. Це допомагає організувати структуру компонента та відобразити необхідний інтерфейс.

Слід звернути увагу, що компонент **Dashboard** має таку структуру:

1. **Заголовок та анімація:** Перший рядок коду відображає заголовок "Dashboard" на верхній частині сторінки. Анімація, яку надає компонент **FadeIn**, може бути використана для створення ефекту плавного відображення вмісту сторінки.

2. **Відображення фінансової інформації:** Верхня частина сторінки (відсутність **div** з класом "up" та "left" може бути просто об'єднана в один контейнер) призначена для відображення фінансової інформації про дохід, витрати, заощадження та борг. Цю інформацію

представлено за допомогою компонентів **BubbleSmall**, кожен з яких має іконку та текст, який позначає статус кожного фінансового показника.

3. **Графічне представлення даних:** Права частина верхньої секції містить **BubbleCard**, в якому може відображатися **DoughnutChart**. **DoughnutChart** служить для графічного відображення фінансової інформації у вигляді кругової діаграми, яка дозволяє користувачу зрозуміти розподіл різних фінансових показників.

4. **Таблиця фінансових даних:** Нижня частина сторінки (відсутність **div** з класом "down" може бути об'єднана в один контейнер) використовується для відображення докладної інформації у вигляді таблиці. Компонент **DashTable** призначений для відображення даних цієї таблиці у контейнері **BubbleFull**.

2.3 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача веб-сервісу вимагає уважного підходу до кожного елемента, забезпечуючи ефективну інтеракцію користувачів з системою. Сучасний дизайн, інтуїтивна навігація та чітка презентація інформації є ключовими для забезпечення позитивного користувацького досвіду.

Головна сторінка (Dashboard)

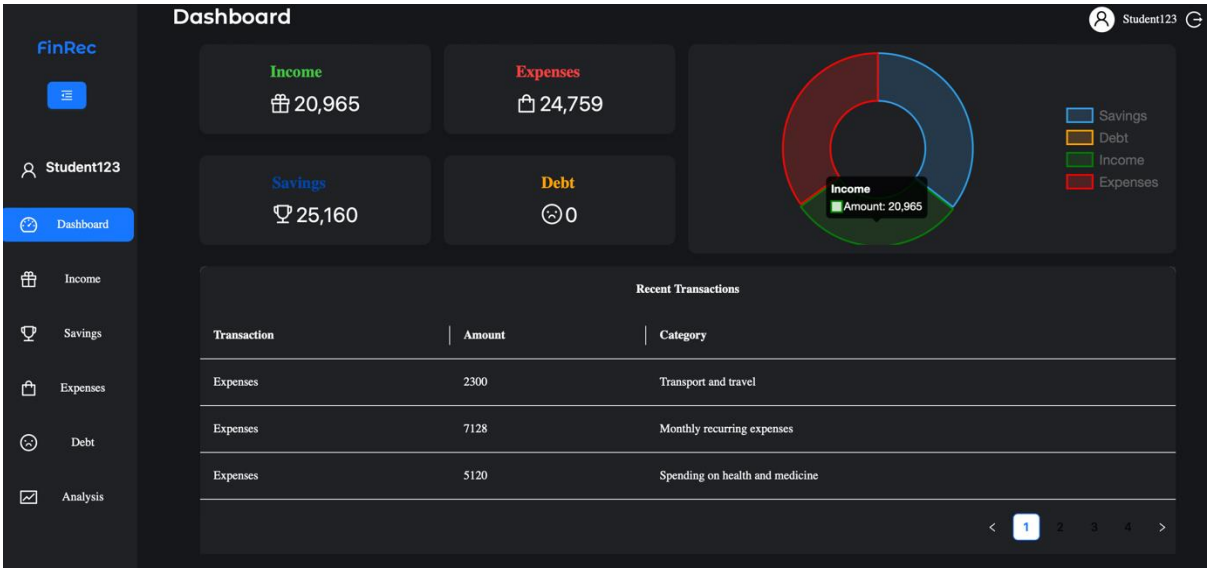


Рис.2.15 – Головна сторінка веб-сервісу

Головна сторінка є осередком користувацького досвіду, надаючи централізований огляд фінансової інформації. Вона представляє собою збалансовану композицію інформаційних блоків, таких як доходи, витрати, заощадження та борги, кожен із яких відображає актуальні дані в режимі реального часу. Кольорова кодифікація та графічні віджети, як-от кільцеві діаграми, надають користувачеві швидке розуміння його фінансового стану.

Сторінка доходів (Income)

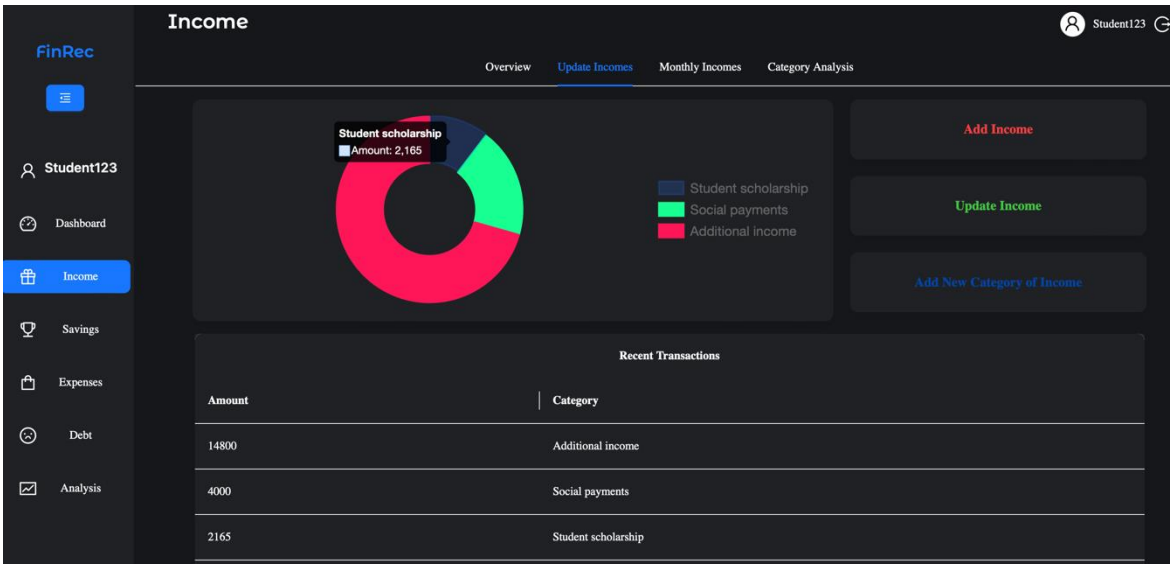


Рис.2.16 – Сторінка доходів

На сторінці доходів користувачі мають можливість додавати та аналізувати різні джерела доходу. Візуальні елементи, такі як яскраві бари та кільцеві діаграми, дозволяють легко ідентифікувати розподіл доходів за категоріями. Віджети для швидкого введення нових транзакцій забезпечують простоту додавання нових записів, роблячи процес максимально ефективним.

Сторінка витрат (Expenses)

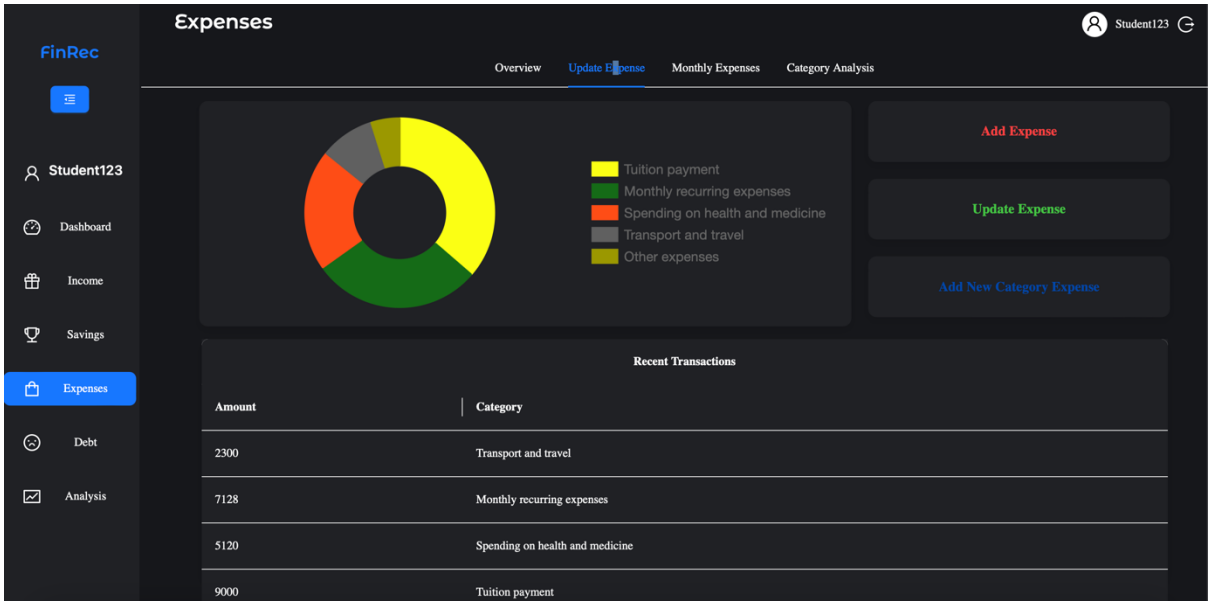


Рис.2.17 – Сторінка витрат

Ця секція призначена для моніторингу та керування витратами. Вона демонструє деталізований розбір витрат за категоріями та динаміку їх зміни. Інтерактивні графіки та діаграми надають користувачам змогу глибше зануритися в аналіз своїх фінансів, дозволяючи краще планувати бюджет.

Сторінка заощаджень (Savings)

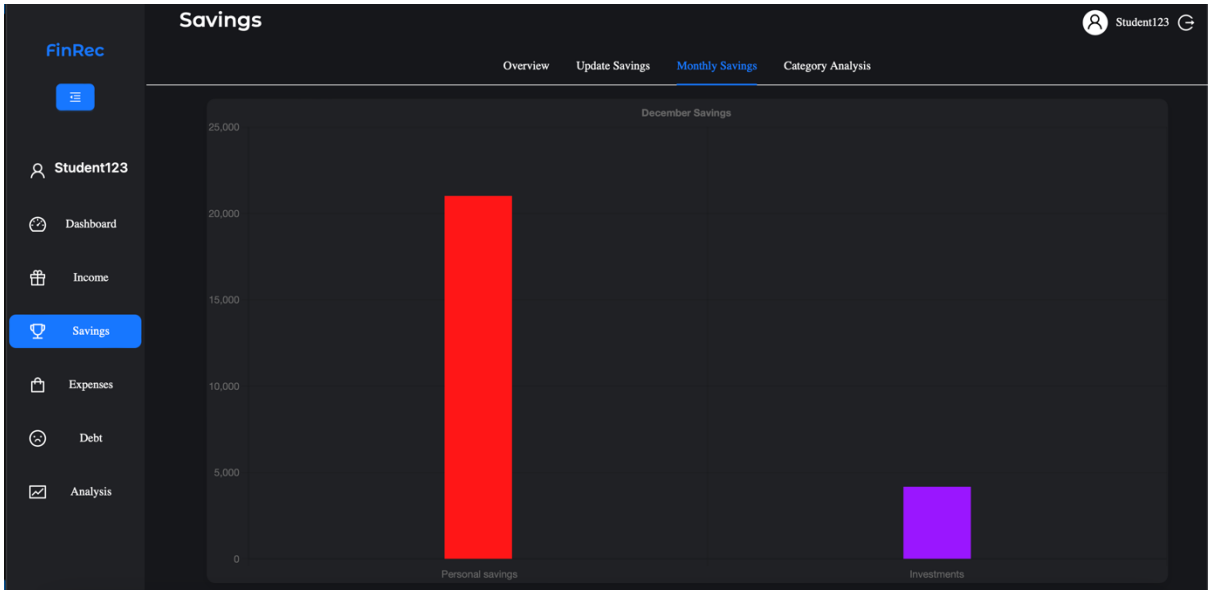


Рис.2.18 – Сторінка заощаджень

На цій сторінці користувачі можуть відслідковувати свої заощадження, вносити зміни та визначати цілі заощадження. Прості форми для введення даних та відображення прогресу заощаджень у вигляді діаграм дають користувачам чітке уявлення про стан їхніх фінансових резервів.

Аналітична сторінка (Analysis)

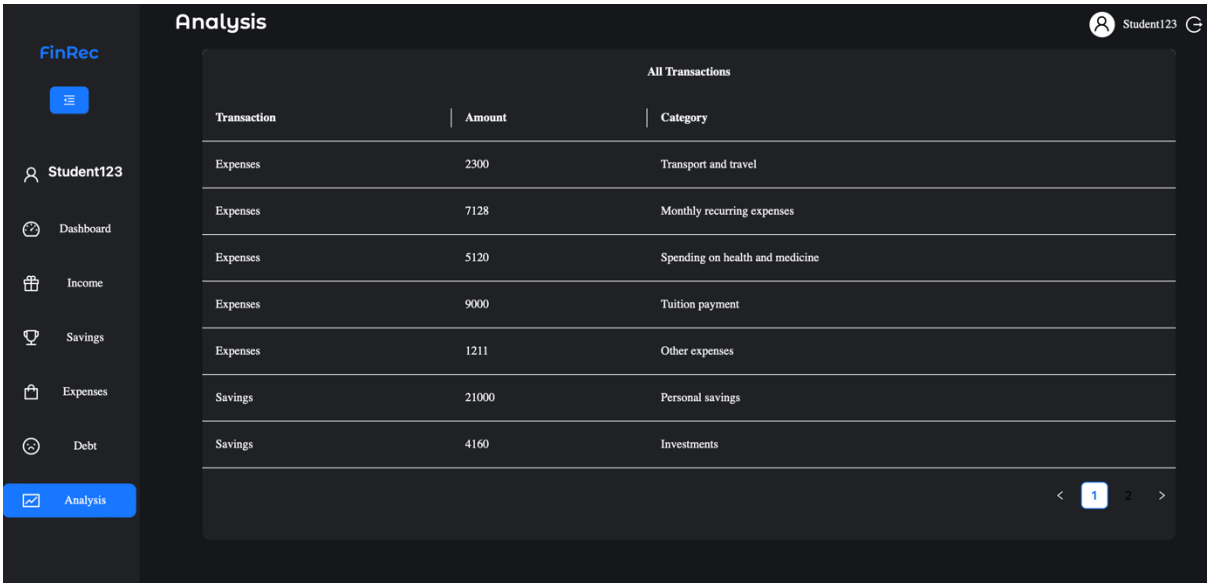


Рис.2.19 – Аналітична сторінка

Аналітична сторінка забезпечує глибоке розуміння фінансових звичок користувача. Вона включає різноманітні інструменти для аналізу, включаючи фільтри, графіки та таблиці, що дозволяють виявляти тенденції та патерни у витратах та доходах.

Загальний вигляд інтерфейсу характеризується чіткими лініями, сучасними шрифтами та контрастною кольоровою палітрою, що забезпечує високий рівень читабельності та зручності. Плавні переходи між сторінками та швидка відповідь на дії користувача роблять взаємодію з веб-сервісом приємною та беззусильною.

Дизайн веб-сервісу спрямований на те, щоб надати користувачам не тільки потрібні інструменти для керування фінансами, але й зробити цей процес візуально привабливим і зрозумілим, забезпечуючи високу залученість та задоволеність користувачів.

ВИСНОВКИ

В ході виконання дипломної роботи було досягнуто ключових цілей проєктування та розробки веб-сервісу, призначеного для управління пільговими стипендіями. Завдяки аналізу потреб користувачів та вимог до системи, вдалося створити інструмент, який відповідає сучасним технічним стандартам.

Проєкт також включав розробку додаткових модулів для аналітики та звітності, що дозволяє відслідковувати поточний стан справ.

Робота над дипломним проєктом виявила ряд можливостей для подальшого розвитку системи. Напрацьовані пропозиції та рекомендації можуть бути використані для оновлень майбутніх версій веб-сервісу.

У цілому, робота підтвердила, що розроблений веб-сервіс є ефективним інструментом для управління пільговими стипендіями. Висока адаптивність системи та можливість її інтеграції з іншими інформаційними системами надають потенціал для її широкого застосування в освітніх інституціях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Strazzullo F. Frameworkless Front-End Development: Do You Control Your Dependencies Or Are They Controlling You? – Apress, 2019. – 248 с. – ISBN: 9781484249673.

2. Ardito C., Ko I.-Y., Schedl M., Di Noia T., ред. Web Engineering: 22nd International Conference, ICWE 2022, Bari, Italy, July 5–8, 2022, Proceedings. – Springer International Publishing, 2022. – 510 с. – ISBN: 9783031099175.

3. Fields D. K., Kolb M. A., Bayern S. Web Development with JavaServer Pages. – Manning, 2002. – 759 с. – ISBN: 9781930110120.

4. Fain Y., Rasputnis V., Tartakovsky A., Gamov V. Enterprise Web Development: Building HTML5 Applications: From Desktop to Mobile. – O'Reilly Media, 2014. – 642 с. – ISBN: 9781449357061.

5. Локхарт Д. Современный Python. Нові можливості та сучасний досвід. – літРес, 2022. – ISBN: 9785040460182.

6. Snyder C., Southwell M. Pro PHP Security. – Apress, 2005. – 528 с. – ISBN: 9781590595084.

7. Da Silva D. L. Framework Flask Python 8 & SqlAlchemy. – Clube de Autores, 2022. – 439 с. – ISBN: 9786500436891.

8. Abdulloh R. Membuat Aplikasi Point of Sale dengan Laravel dan AJAX. – Elex Media Komputindo, 2017. – 262 с. – ISBN: 9786020441092.

9. Ahmed R. Laravel A-Z. – Rumel Ahmed, 2022. – 2 016 с. – ISBN: 9781799812968.

10. Rezaei S., ред. Apps Management and E-Commerce Transactions in Real-Time. – IGI Global, 2017. – 379 с. – ISBN: 9781522524502.

11. Shou D. T., Fazalbhoy S., Gochhait S., ред. Cloud Computing Applications and Techniques for E-Commerce. – IGI Global, 2019. – 185 с. – ISBN: 9781799812968.

12. Zhang J., Wang K., Ying K., Fan Z., Zhao Z., ред. Innovation of Digital Economy: Cases from China. – Springer Nature Singapore, 2023. – 473 с. – ISBN: 9789819917419.

13. Phang C. L. Web Coding Bible (HTML, CSS, Javascript, PHP, SQL, XML, SVG, Canvas, WebGL, Java Applet, ActionScript, jQuery, WordPress, SEO and many more): An Accelerated Course. – Chong Lip Phang, 2015. – 474 с. – ISBN: 9789671317518.

14. McFedries P. HTML, CSS, & JavaScript All-in-One For Dummies. – Wiley, 2023. – 848 с. – ISBN: 9781119824903.

15. Saini S. K. Web Development Building Websites with HTML, CSS, and JavaScript: From Basics to Advanced Techniques for Building Beautiful and User-Friendly Websites. – Sunil Kumar Saini, 2023. – 43 с.

16. Meloni J. C., Kyrnin J. HTML, CSS, and JavaScript All in One: Covering HTML5, CSS3, and ES6, Sams Teach Yourself. – Pearson Education, 2018. – 720 с. – ISBN: 9780135167076.

17. Cybellium Ltd. Mastering HTML and CSS. – Cybellium Ltd, 2023. – 249 с. – ISBN: 9798859157907.

18. Phang C. L. HTML, Bootstrap, CSS, Tailwind, & Cordova: A Complete Guide. – Chong Lip Phang, 2022. – 155 с.

19. AMC College. UI/UX Design (Adobe XD CC). – Advanced Micro Systems Sdn Bhd, 2022.

20. Khandavilli A. 100 Dos and Dont's UI/UX Design Tips eBook-Professional research based. – Akash Khandavilli, 2020. – 78 с.

21. Conta A. The Art and Science of UX Design: A Step-by-step Guide to Designing Amazing User Experiences. – Pearson Education, 2023. – ISBN: 9780138060459.

22. Ghosh A. Mastering UX Design with Effective Prototyping: Turn Your Ideas Into Reality with UX Prototyping. – BPB Publications, 2023. – 314 с. – ISBN: 9789355515346.

23. Hodent C., Isbister K., ред. Game Usability: Advice from the Experts for Advancing UX Strategy and Practice in Videogames. – CRC Press, 2022. – 452 с. – ISBN: 9781000523485.

24. McCombs K. Advanced Programming and Design. – Cavendish Square Publishing, 2016. – 128 с. – ISBN: 9781502619457.

25. Buono S. A. C# and Game Programming: A Beginner's Guide. – CRC Press, 2019. – 532 с. – ISBN: 9781351989343.

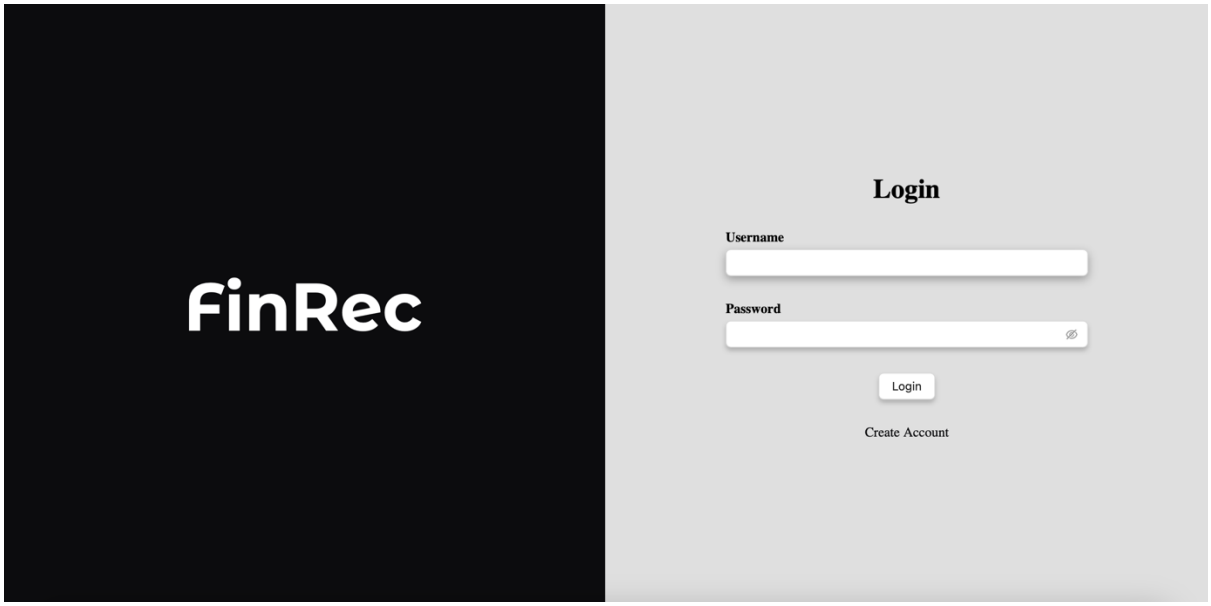
26. Tatroe K., MacIntyre P. Programming PHP: Creating Dynamic Web Pages. – O'Reilly Media, 2020. – 544 с. – ISBN: 9781492054085.

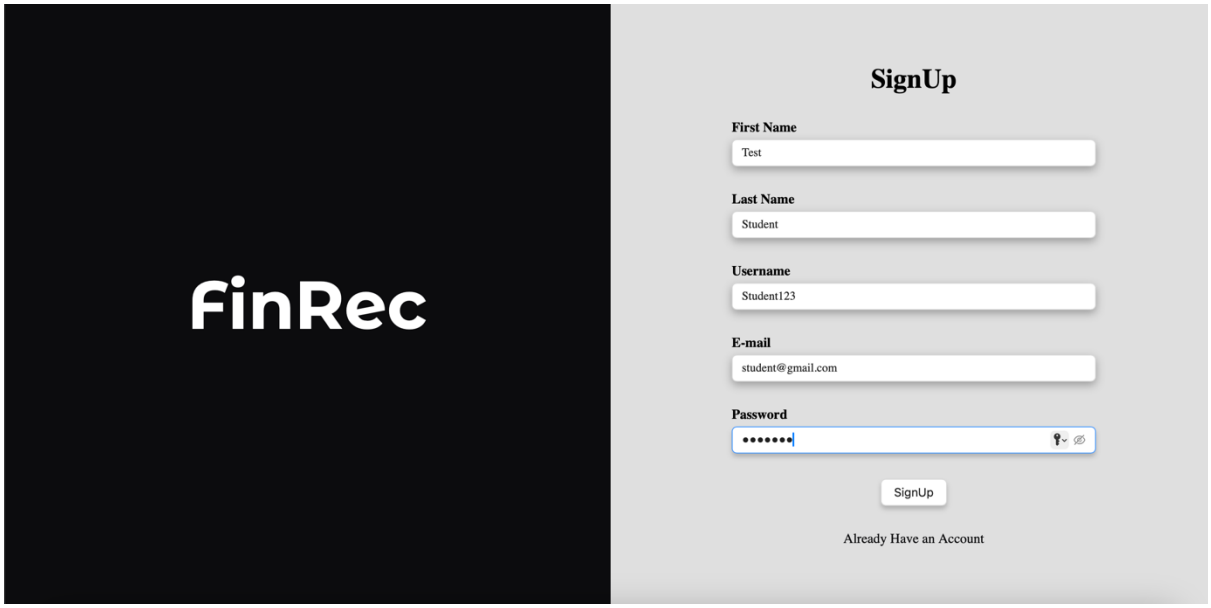
27. Simon A. R., Wheeler T. Open Client/Server Computing and Middleware. – Elsevier Science, 2014. – 288 с. – ISBN: 9781483214276.

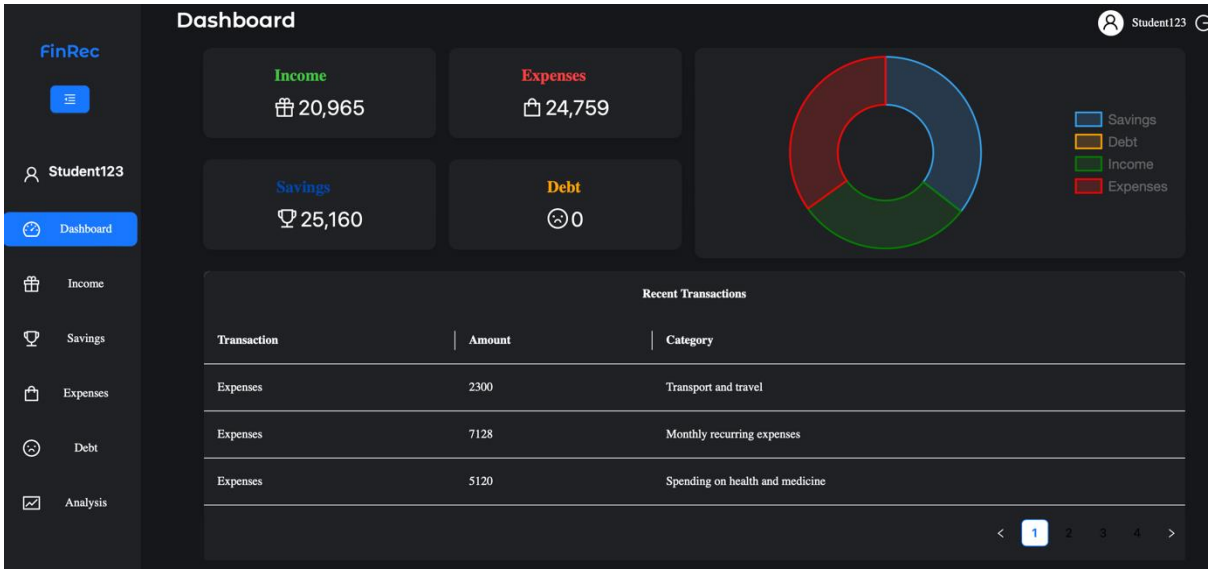
28. Saternos C. Client-Server Web Apps with JavaScript and Java: Rich, Scalable, and RESTful. – O'Reilly Media, 2014. – 260 с. – ISBN: 9781449369316.

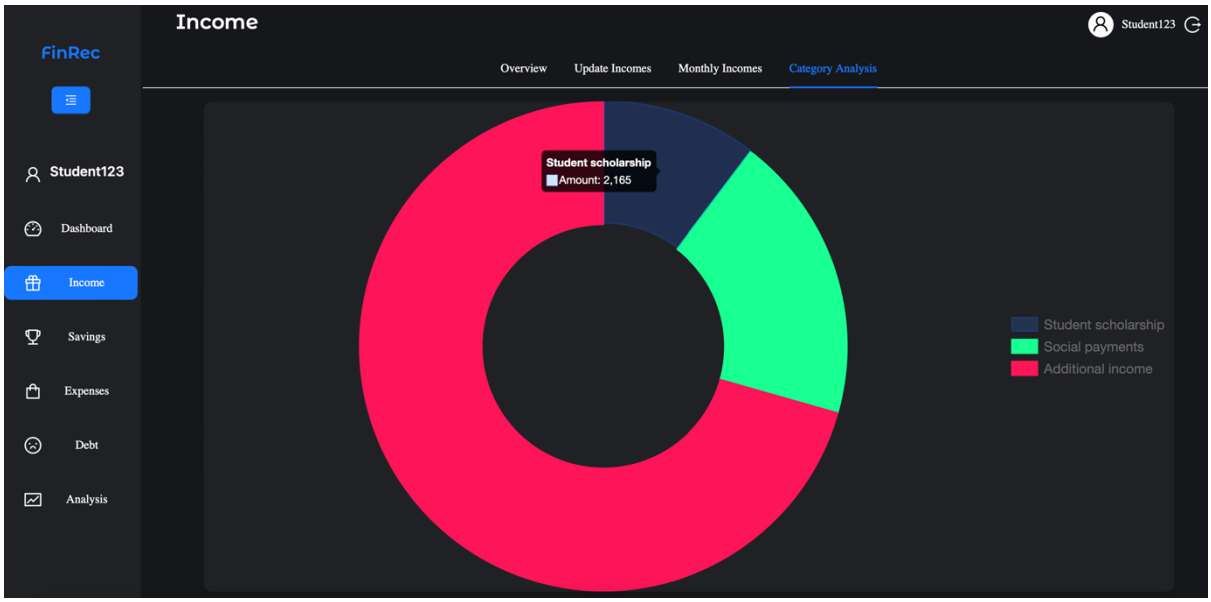
ДОДАТОК

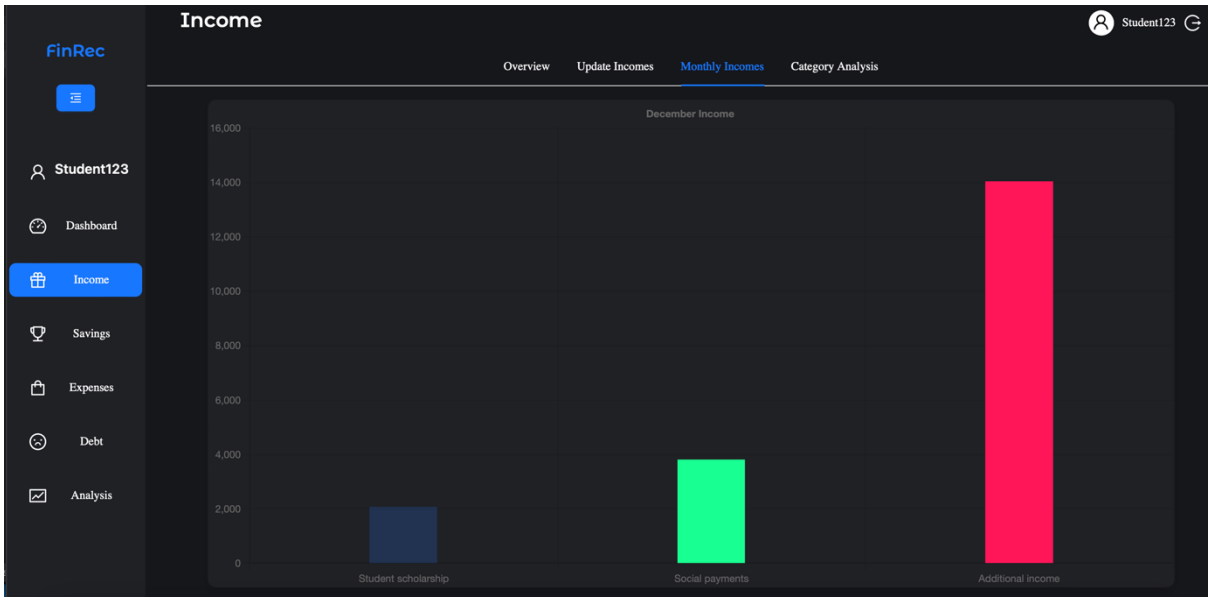
Скріншоти роботи веб-додатку

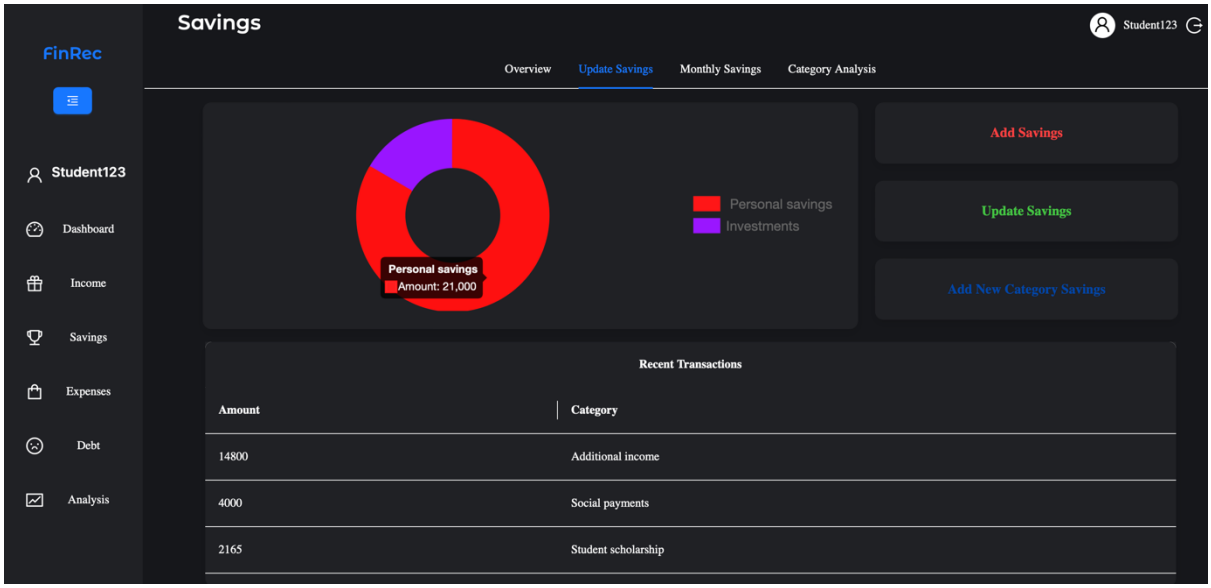


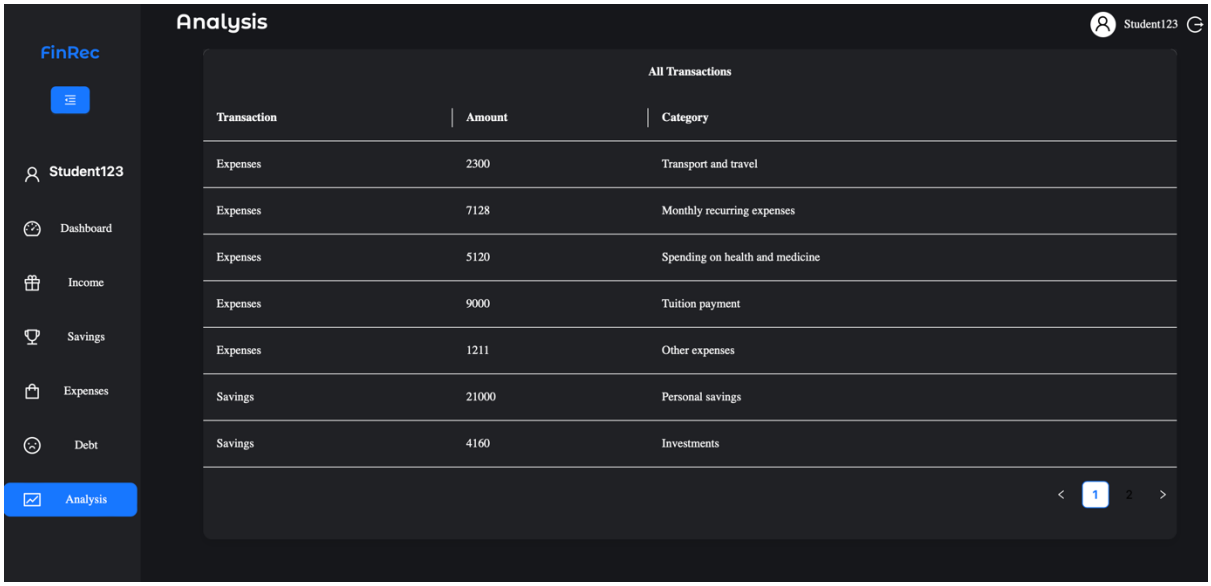



FinRec
Dashboard
Student123
Income
20,965
Expenses
24,759
Savings
25,160
Debt
0
Savings
Debt
Income
Expenses
Student123
Dashboard
Income
Savings
Expenses
Debt
Analysis
Recent Transactions
Transaction
Amount
Category
Expenses
2300
Transport and travel
Expenses
7128
Monthly recurring expenses
Expenses
5120
Spending on health and medicine
1
2
3
4


FinRec
Income
Student123
Overview
Update Incomes
Monthly Incomes
Category Analysis
Student scholarship
Amount: 2,165
Student123
Dashboard
Income
Savings
Expenses
Debt
Analysis
Student scholarship
Social payments
Additional income


FinRec
Income
Student123
Overview
Update Incomes
Monthly Incomes
Category Analysis
December Income
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
Student scholarship
Social payments
Additional income
Student123
Dashboard
Income
Savings
Expenses
Debt
Analysis


FinRec
Savings
Student123
Overview
Update Savings
Monthly Savings
Category Analysis
Student123
Dashboard
Income
Savings
Expenses
Debt
Analysis
Personal savings
Amount: 21,000
Personal savings
Investments
Add Savings
Update Savings
Add New Category Savings
Recent Transactions
Amount
Category
14800
Additional income
4000
Social payments
2165
Student scholarship


FinRec
Analysis
Student123
Student123
Dashboard
Income
Savings
Expenses
Debt
Analysis
All Transactions
Transaction
Amount
Category
Expenses
2300
Transport and travel
Expenses
7128
Monthly recurring expenses
Expenses
5120
Spending on health and medicine
Expenses
9000
Tuition payment
Expenses
1211
Other expenses
Savings
21000
Personal savings
Savings
4160
Investments
1
2