ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка інформаційної системи банку єВідновлення на базі JavaScript»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Дмитро Шевченко*

Виконав:
здобувач вищої освіти Дмитро Шевченко
група ІСДМ-61

Керівник: Оксана Ткаленко
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім’я, ПРІЗВИЩЕ

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Шевченку Дмитру Ігоровичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Розробка інформаційної системи банку єВідновлення на базі JavaScript»

керівник кваліфікаційної роботи Оксана ТКАЛЕНКО, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «19» жовтня 2023 року № 145.

2. Строк подання кваліфікаційної роботи: 29 грудня 2023 року.

3. Вихідні дані до кваліфікаційної роботи: Методи HTTP GET, POST, PUT, PATCH, DELETE;
Інтегроване середовище розробки;
Науково-технічна література щодо технологій бездротової передачі даних та основ архітектури RESTful;
Програмне забезпечення;
Науково-технічна література щодо архітектури та можливостей вебфреймворків;
Наукова література щодо державної програми "єВідновлення".

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Цілі та перспективи інформаційної системи банку єВідновлення;
2. Дослідження технологій вебфреймворків для серверної частини;
3. Застосування вебфреймворку NestJS;
4. Результати виконаної роботи та висновки.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Актуальність і завдання роботи.
   2. Вибір апаратних та програмних засобів.
   3. Аналіз технічної документації.
   4. Аналіз технологій для розробки інформаційної системи.
   5. Створення вебсерверу для клієнтських запитів.

6. Дата видачі завдання: 19 жовтня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10 – 25.10.23	
2	Аналіз документації програми єВідновлення	26.10 – 02.11.23	
3	Аналіз вебфреймворків	03.11 – 09.11.23	
4	Використання вебфреймворку NestJS	10.11 – 16.11.23	
5	Розробка інформаційної системи згідно документації	17.11 – 21.11.23	
6	Тестування інформаційної системи та виправлення помилок	22.11 – 11.12.23	
7	Оформлення роботи: вступ, висновки, реферат	12.12 – 20.12.23	
8	Розробка демонстраційних матеріалів	21.12 – 28.12.23	

Здобувач вищої освіти _____ Дмитро ШЕВЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 80 стор., 12 рис., 20 табл., 30 джерел.

Мета роботи – аналіз методів прикладного програмного інтерфейсу інформаційної системи єВідновлення, її можливостей, а також проектування та розробка такої системи на основі проаналізованої інформації.

Об'єкт дослідження – процес створення інформаційної системи для надання можливості громадянам та клієнтам банку отримувати допомогу від держави.

Предмет дослідження – методи прикладного програмного інтерфейсу інформаційної системи єВідновлення.

Короткий зміст роботи: В даній магістерській роботі було визначено основні методи прикладного програмного інтерфейсу інформаційної системи єВідновлення. Проаналізовано технології, які дозволяють створити серверну частину моделі, а саме мови програмування та бази даних. Проведено аналіз можливих вебфреймворків, визначено їх переваги та недоліки. Було досліджено параметри та їх типи даних для усіх методів прикладного програмного інтерфейсу інформаційної системи.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС, ПЕРЕДАЧА РЕПРЕЗЕНТАТИВНОГО СТАНУ ЗА ДОПОМОГОЮ ВЕБФРЕЙМВОРКА, ЄВІДНОВЛЕННЯ.

ABSTRACT

Text part of the master`s qualification work: 80 pages, 12 pictures, 20 tables, 30 sources.

The purpose of the work is to analysis of the methods of the applied software interface of the information system is Restoration, its capabilities, as well as the design and development of such a system based on the analyzed information.

Object of research is the process of creating an information system to provide citizens and bank clients with assistance from the state.

Subject of research is methods of the applied programming interface of the eRecovery information system.

Summary of the work: In this master's thesis, the main methods of the applied software interface of the information system are Recovery. The technologies that allow creating the server part of the model are analyzed, namely programming languages and databases. An analysis of possible web frameworks was carried out, their advantages and disadvantages were determined. The parameters and their data types for all methods of the application software interface of the information system were investigated.

KEYWORDS: INFORMATION TECHNOLOGIES, APPLIED PROGRAMMING INTERFACE, REPRESENTATIVE STATE TRANSFER VIA WEB FRAMEWORK, ERECOVERY.

ЗМІСТ

ВСТУП

Актуальність теми: єВідновлення – це послуга, яка допомагає отримати державну допомогу для відновлення свого житла. Банки активно впроваджують інформаційні системи та технології для полегшення банківських операцій, забезпечення безпеки та підвищення якості обслуговування клієнтів. Розробка інформаційної системи, яка допоможе всім громадянам, чиє житло отримало пошкодження внаслідок збройної агресії рф і ще не було відремонтоване, вони зможуть скористатися послугою. Послугу єВідновлення максимально цифровізували та спростили для громадян. У такий непростий час для нашої країни важливо піклуватися про громадян, та надавати допомогу швидко і зручно.

Метою роботи є дослідження та розробка інформаційної системи з використанням мови програмування JavaScript.

Для досягнення поставленої мети необхідно *виконати наступні завдання*:

- проаналізувати методи взаємодії з системою;
- дослідити фреймворки мови програмування JavaScript для її проектування системи;
- визначити вимоги для проведення валідації вхідних та вихідних даних.

Об'єкт дослідження: процес створення інформаційної системи банку для надання можливості громадянам та клієнтам отримувати допомогу від держави в короткі строки.

Предметом дослідження є методи прикладного програмного інтерфейсу інформаційної системи єВідновлення.

Апробація результатів магістерської роботи:

1. МінфінМедіа. "Офіційний прес-реліз приєднання банку до програми єВідновлення". – Київ, 27 серпня 2023. https://minfin.com.ua/ua/2023/08/27/111493805/

1 ДОСЛІДЖЕННЯ І АНАЛІЗ ОБ`ЄКТУ ПРОГРАМУВАННЯ

1.1 Постановка задачі

Основним завданням дипломної роботи є розробити прикладний програмний інтерфейс клієнт–серверну інформаційну систему для обміну інформацією методом відправки POST–повідомлень у форматі JSON стандарту RFC 8259, на тему інформаційної системи банку єВідновлення, використовуючи вебфреймворк NestJS, яка буде містити у собі параметри об’єктів та їх методи. Програма повинна мати вигляд вікна консолі, яка через методи вебфреймворку реалізує обмін інформацією.

На спілкування між серверами не повинно накладатися обмежень, окрім, встановлених політик безпеки та обмежений доступ як із зовнішньої мережі так і з внутрішньої, тільки по адресам і портам, які використовуються для роботи – система обміну повинна бути децентралізованою. Обмін інформацією повинен здійснюватися напряму між клієнтами, використовуючи сервер виключно для збору інформації про клієнта банку, який відкриває або закриває рахунок. Демонстраційна програма повинна працювати в операційних системах Windows та дистрибутивах Ubuntu, CentOS, Debian останніх версій.

Користувачі системи:

– оператор системи – аналізує виниклі помилки;

– вебсервер зі сторони Державного комерційного підприємства «ДІЯ»;

– вебсервер зі сторони банку.

Демонстраційний прикладний програмний інтерфейс має включати в себе такі функціональні можливості:

– прийняття вхідних запитів;

– надсилання вихідних запитів;

– автоматична валідація параметрів вхідних запитів;

– автоматична обробка помилок;

– робота з базою даних;

– логування вхідних/вихідних запитів.

1.2 Вимоги до апаратного та програмного забезпечення

Вимоги до апаратного забезпечення:

– оперативна пам’ять: мінімум 4 Гб або більше;

– диск: рекомендовано SSD від 128 Гб;

– процесор: чотирьохядерний або більше, з частотою більше 2.5 ГГц;

– інше: потужніший процесор та більше оперативної пам’яті можуть бути корисними для обробки великого обсягу запитів.

Вимоги до програмного забезпечення:

– операційна система: Microsoft Windows 10–11, або Ubuntu 22.04, Debian 12.2, CentOS 7.

1.3 Використані програмні засоби

Для розробки дипломного проекту обрано інтегроване середовище розробки Microsoft Visual Studio Code. Це потужне і зручне середовище, яке надає широкий набір інструментів для комфортної та ефективної роботи розробника.

Мовою програмування для розробки обрано JavaScript, яка є однією з найпоширеніших та використовуваних мов у веб-розробці. Для розширення можливостей проекту використовується TypeScript, що є надмножиною JavaScript, дозволяючи використовувати статичні типізації та інші переваги для більш надійного та підтримуваного коду.

Основною платформою для виконання коду є Node.js, що забезпечує виконання JavaScript/TypeScript на серверній стороні. Вибір Node.js дозволяє створювати ефективні та масштабовані серверні застосунки.

Для створення серверної частини проекту використовується фреймворк NestJS. NestJS є модульним та ефективним фреймворком, побудованим на засадах Angular, і надає розробникам зручні інструменти для швидкої та структурованої розробки серверних додатків.

Для забезпечення надійного та ефективного зберігання даних у дипломному проекті використовується Microsoft SQL Server (MSSQL) як система управління базами даних (СУБД). MSSQL є потужним та надійним інструментом для роботи з базами даних, який надає високу швидкість, безпеку та ряд продуктивних можливостей.

Вибір MSSQL обумовлений його широкою популярністю в корпоративному середовищі та можливістю працювати з великим обсягом даних. Інтеграція MSSQL з дипломним проектом дозволяє ефективно управляти базою даних, використовуючи мову запитів T-SQL та забезпечує високий рівень надійності та швидкодії при роботі з великим обсягом інформації.

З використанням MSSQL можна легко забезпечити консистентність та доступність даних, а також проводити ефективні операції збереження та витягування інформації. Це робить базу даних належним компонентом у загальній архітектурі дипломного проекту, забезпечуючи його стабільність та високий рівень функціональності.

Усі ці технологічні компоненти обрані з урахуванням їхньої сумісності та можливості спільної роботи, що сприяє ефективному та успішному виконанню завдань у рамках дипломного проекту.

Microsoft Visual Studio Code — інтегроване середовище розробки програмного забезпечення, що включає себе ряд інструментальних засобів.

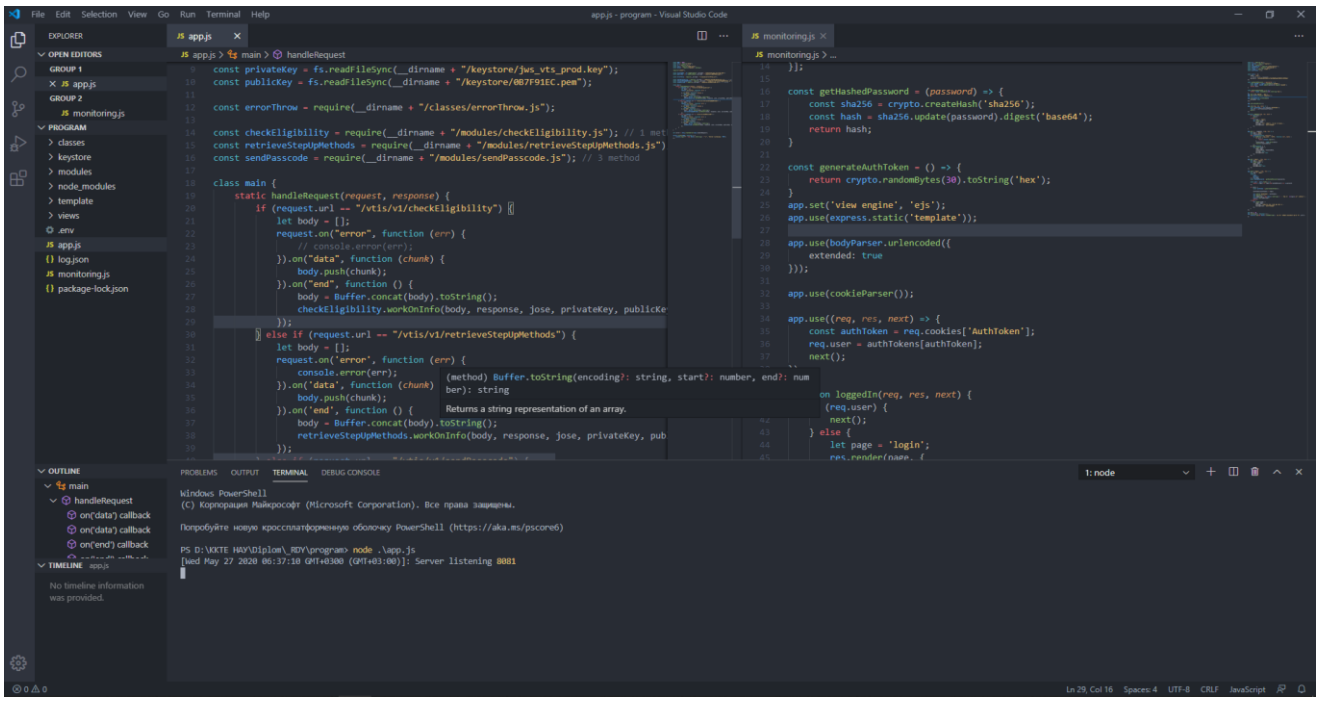


Рисунок 1.1 – Середовище розробки

Середовище розробки розроблено компанією Microsoft.

JavaScript (JS) — динамічна, об'єктно–орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб–сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб–сторінки. Емблема мови зображена на рисунку 1.2.

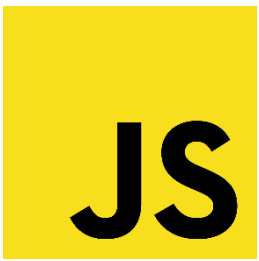


Рисунок 1.2 – Емблема мови програмування JavaScript

Взаємодія між трекерами і сервером буде відбуватися через API по технології REST.

API або інтерфейс прикладного програмування — це набір правил, які визначають, як програми чи пристрої можуть підключатися та спілкуватися один з одним. REST API — це API, який відповідає принципам проектування REST або архітектурному стилю передачі стану репрезентації. З цієї причини API REST іноді називають API RESTful.

API REST спілкуються через HTTP–запити для виконання стандартних функцій бази даних, таких як створення, читання, оновлення та видалення записів (також відомих як CRUD) у ресурсі. Наприклад, REST API використовуватиме запит GET для отримання запису, запит POST для його створення, запит PUT для оновлення запису та запит DELETE для його видалення. Усі методи HTTP можна використовувати у викликах API.

Дані можна надати клієнту практично в будь–якому форматі, включаючи нотацію об’єктів JavaScript (JSON), HTML, Python, PHP або звичайний текст. JSON популярний, оскільки його читають як люди, так і машини, і він не залежить від мови програмування.

Заголовки та параметри запитів також важливі у викликах REST API, оскільки вони містять важливу інформацію про ідентифікатор, таку як метадані, авторизації, уніфіковані ідентифікатори ресурсів (URI), кешування, файли cookie тощо. Заголовки запитів і відповіді, а також звичайні коди статусу HTTP використовуються в правильно розроблених API REST.

Для того, щоб API вважався RESTful, він повинен відповідати наступним критеріям:

- Архітектура клієнт–сервер, що складається з клієнтів, серверів і ресурсів, із запитами, які керуються через HTTP.

- Зв’язок клієнт–сервер без стану, тобто інформація про клієнта не зберігається між запитами на отримання, і кожен запит є окремим і не підключеним.
- Дані з кешуванням, які спрощують взаємодію клієнт–сервер.
- Єдиний інтерфейс між компонентами, щоб інформація передавалася у стандартній формі. Для цього потрібно:
  - запитувані ресурси є ідентифікованими та відокремленими від представлень, надісланих клієнту.
  - Клієнт може маніпулювати ресурсами через представлення, яке вони отримують, оскільки подання містить достатньо інформації для цього.
  - самоописні повідомлення, що повертаються клієнту, мають достатньо інформації, щоб описати, як клієнт повинен її обробити.
  - гіпертекст/гіпермедіа доступний, що означає, що після доступу до ресурсу клієнт повинен мати можливість використовувати гіперпосилання, щоб знайти всі інші доступні на даний момент дії, які він може виконати.
- Багатошарова система, яка організовує кожен тип серверів (відповідальних за безпеку, балансування навантаження тощо), передбачала отримання запитуваної інформації в ієрархії, невидимі для клієнта.
- Код на вимогу (необов’язково): можливість надсилати виконуваний код із сервера клієнту за запитом, що розширює функціональні можливості клієнта.

TypeScript (TS) — мова програмування, яка є надмножиною (суперсетом) JavaScript. Це означає, що весь валідний код JavaScript є також валідним кодом TypeScript, але TypeScript додає до JavaScript додатковий функціонал, зокрема, статичну типізацію. Емблема мови зображена на рисунку 1.3.

Рисунок 1.3 – Емблема мови програмування TypeScript

Основні особливості TypeScript:

- Статична типізація: TypeScript дозволяє вказувати типи даних для змінних, параметрів функцій та інших об'єктів. Це допомагає виявляти помилки на етапі компіляції та полегшує роботу з великими кодовими базами.
- Класи та інтерфейси: TypeScript підтримує об'єктно–орієнтовану парадигму програмування з використанням класів та інтерфейсів. Це дозволяє структурувати код та створювати повторно використовувані компоненти.
- Enums (переліки): TypeScript додає переліки, які дозволяють створювати зручний іменований синтаксис для числових значень, полегшуючи розуміння коду.
- Generics: TypeScript підтримує generics, що дозволяє створювати функції та класи, які можуть працювати з різними типами даних.
- Сумісність із JavaScript: TypeScript використовує синтаксис JavaScript, тому весь вже існуючий код на JavaScript може бути поетапно переведений на TypeScript.

Ці особливості дозволяють покращити підтримку, масштабованість та читабельність коду, зменшити кількість помилок і полегшити роботу з великими проектами.

Всі дані які будуть передаватися між серверами будуть в форматі JSON.

JSON — це текстовий формат даних, який відповідає синтаксису об’єкта JavaScript. Незважаючи на те, що він дуже нагадує літеральний синтаксис об’єкта JavaScript, його можна використовувати незалежно від JavaScript, і багато середовищ програмування мають можливість читати (розбирати) та генерувати JSON. Приклад JSON об’єкта зображено на рисунку 1.4.

Рисунок 1.4 – Приклад JSON об’єкта

Під такі критерії підходить технологія на якій можливо створити вебсервер та працювати з запитами. Далі розглянемо детальніше.

Децентралізована, Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Якщо раніше JavaScript застосовувався для обробки даних в браузері користувача, то Node.js надав можливість виконувати JavaScript–скрипти на сервері та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js має наступні властивості:

– асинхронна одно–нитева модель виконання запитів;
– неблокуючий ввід/вивід;
– система модулів CommonJS;

– рушій JavaScript Google V8;

Для керування модулями використовується пакетний менеджер npm (node package manager).

Як асинхронне подієве JavaScript–оточення, Node.js спроектований для побудови масштабованих мережевих додатків. Готовий додаток може одночасно обробляти багато з'єднань. Для кожного з'єднання викликається функція зворотнього виклику, проте коли з'єднань немає Node.js засинає.

Це контрастує з більш загальною моделлю в якій використовуються паралельні OS потоки. Такий підхід є відносно неефективним та дуже важким у використанні. Більше того, користувачі Node.js можуть не турбуватись про блокування процесів, оскільки немає жодних блокувань. Майже жодна з функцій у Node.js не працює напряму з I/O, тому процес не блокується ніколи. Оскільки нічого не блокується на Node.js легко розробляти масштабовані системи.

Node.js створений під впливом таких систем як Event Machine в Ruby або Twisted в Python. Node.js використовує подієву модель значно ширше, він приймає цикл подій (event loop) за основу оточення, замість того, щоб використовувати його в якості бібліотеки. В інших системах завжди стається блокування виклику, щоб запустити цикл подій.

Зазвичай поведінка визначається через функції зворотнього виклику на початку скрипта і в кінці запускає сервер через блокуючий виклик, як от EventMachine::run(). В Node.js немає нічого подібного на виклик початку циклу подій. Node.js просто входить в подієвий цикл після запуску скрипта на виконання. Node.js виходить з подієвого циклу тоді, коли не залишається зареєстрованих функцій зворотнього виклику. Така поведінка схожа на поведінку браузерного JavaScript: подієвий цикл прихований від користувача.

HTTP є об'єктом першого роду в Node.js, розробленим з потоковістю та малою затримкою. Це робить Node.js хорошою основою для веб–бібліотеки або фреймворку.

Те що Node.js спроектований без багатопоточності, не означає, що ви не можете використовувати можливості кількох ядер у вашому середовищі. Ви можете

створювати дочірні процеси, якими легко керувати з допомогою API child_process.fork(). Модуль cluster побудований на цьому інтерфейсі і дозволяє вам ділитись сокетами між процесами та розподіляти навантаження між ядрами.

Рисунок 1.5 – Емблема платформи Node.js

NestJS — це фреймворк для розробки серверних (backend) додатків на мові програмування TypeScript (або JavaScript). Він побудований на основі ідей, подібних тим, які використовуються в Angular для розробки клієнтських додатків (frontend). Емблема фреймворку зображено на рисунку 1.6.

Рисунок 1.6 – Емблема фреймворку NestJS

Основні характеристики та концепції NestJS:

- Express–подібний: NestJS частково моделює свою архітектуру під популярний фреймворк Express для Node.js, що полегшує зрозуміння та використання фреймворка для розробників, які вже працювали з Express.
- Модульність: NestJS сприяє структуруванню додатків за допомогою модулів. Модулі дозволяють організовувати код у відокремлені функціональні блоки, що полегшує підтримку та розширення.
- Dependency Injection (Ін'єкція залежностей): NestJS використовує ін'єкцію залежностей, що полегшує тестування та забезпечує високу міру модульності в додатках.
- Метапрограмування: NestJS використовує декоратори та метапрограмування для декларативного опису інфраструктури додатка.
- WebSocket: Підтримка WebSocket для реального часу та двостороннього спілкування між клієнтом та сервером.
- Можливість використання TypeScript: NestJS розроблений з урахуванням TypeScript, що дозволяє використовувати статичну типізацію та інші переваги цієї мови програмування.

NestJS широко використовується для розробки сучасних веб–додатків та API на Node.js, надаючи розробникам структурований та ефективний спосіб роботи з серверною частиною проекту.

Фундаментальні поняття:

Клас — визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Класи вносять модульність та структурованість в об'єктно–орієнтовану програму. Як правило, клас має бути зрозумілим для не–програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами.

Об'єкт. Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас «Собака» відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт «Сірко» є одним окремим собакою, окремим варіантом значень характеристик. «Собака» має хутро; «Сірко» має коричнево–біле хутро.

Об'єкт «Сірко» є екземпляром (примірником) класу «Собака». Сукупність значень атрибутів окремого об'єкта називається станом. На основі класу «Собака» можна, також, створити інший об'єкт «Дружок», який відрізнятиметься від об'єкта «Сірко» своїм станом (наприклад кольором хутра). Обидва об'єкта (Сірко і Дружок) є екземплярами класу «Собака».

Метод. Можливості об'єкта. Оскільки «Сірко» — «Собака», він може гавкати. Тому гавкати() є одним із методів об'єкта «Сірко». Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі «Собаки» гавкати, але треба щоб гавкав лише один окремий собака.

Успадкування (наслідування). Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватися цілі дерева успадкування. Наприклад, клас «Собака» може мати підкласи Коллі, Вівчарка і т.п.

Так, «Сірко» може бути екземпляром класу «Вівчарка». Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування.

Приховування інформації (інкапсуляція). Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм

повідомлення. Так, наприклад, клас «Собака» має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт представляє кожному іншому класу певний інтерфейс — члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись.

Це дозволить полегшити внесення змін, тобто, без потреби змінювати і користувачів інтерфейсу. Часто, члени класу позначаються як публічні (англ. public), захищені (англ. protected) та приватні (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено. Деякі мови програмування йдуть ще далі: Java використовує ключове слово private для обмеження доступу, що буде дозволений лише з методів того самого класу, protected — лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, C# та VB.NET відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова internal (C#) або Friend (VB.NET), а Eiffel дозволяє вказувати які класи мають доступ до будь–яких членів.

Абстрагування. Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми.

Поліморфізм. Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по–різному

на однакові повідомлення. На практиці – це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. В залежності від того, що передається і вибирається відповідна підпрограма.

Використовувана система управління базами даних — Microsoft SQL Server.

Microsoft SQL Server — система управління базами даних, яка розробляється корпорацією Microsoft. Як сервер даних виконує головну функцію по збереженню та наданню даних у відповідь на запити інших застосунків, які можуть виконуватися як на тому ж самому сервері, так і у мережі.

Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних.

Microsoft SQL Server як мову запитів використовує версію SQL, що отримала назву Transact–SQL (скорочено T–SQL), яка є реалізацією SQL–92 (стандарт ISO для SQL) з багатьма розширеннями. T–SQL дозволяє використовувати додатковий синтаксис процедур, що зберігаються і забезпечує підтримку транзакцій (взаємодія бази даних з керуючим застосунком). Microsoft SQL Server та Sybase ASE для взаємодії з мережею використовують протокол рівня застосунка під назвою Tabular Data Stream (TDS, протокол передачі табличних даних).

Microsoft SQL Server також підтримує Open Database Connectivity (ODBC)— інтерфейс взаємодії застосунків з СУБД. Версія SQL Server 2005 надає можливість підключення користувачів через вебсервер–сервіси, що використовують протокол SOAP. Це дозволяє клієнтським програмам, не призначеним для Windows, кроссплатформенно з'єднуватися з SQL Server. Microsoft також випустила сертифікований драйвер JDBC, що дозволяє застосункам під керування Java (таким як BEA і IBM Websphere) з'єднуватися з Microsoft SQL Server 2000 і 2005.

SQL Server підтримує дзеркалювання та кластеризацію баз даних. Кластер серверу SQL—це сукупність однаково конфігурованих серверів; така схема

допомагає розподілити робоче навантаження між декількома серверами. Усі сервери мають одне віртуальне ім'я, а дані розподіляються за IP–адресами машин кластеру протягом робочого циклу. Також у разі відмови або збою на одному з серверів кластеру доступне автоматичне перенесення навантаження на інший сервер.

Рисунок 1.7 – Емблема Microsoft SQL Server

SQL Server підтримує надлишкове дублювання даних за трьома сценаріями:

- знімок: виконується «знімок» бази даних, який сервер відправляє одержувачам;
- історія змін: всі зміни бази даних безперервно передаються користувачам;
- синхронізація з іншими серверами: бази даних декількох серверів синхронізуються між собою. Зміни усіх баз даних відбуваються незалежно на кожному сервері, а під час синхронізації відбувається звірка даних. Дублювання такого типу передбачає можливість вирішення протиріч між базами даних.

SQL Server 2005 має вбудовану підтримку .NET Framework. Завдяки цьому, процедури бази даних, що зберігаються, можуть бути написані на будь–якій мові платформи .NET з використанням повного набору бібліотек, доступних для .NET Framework. На відміну від інших процесів, .NET Framework виділяє додаткову

пам'ять і будує засоби керування SQL Server, не використовуючи вбудовані засоби Windows. Це підвищує продуктивність порівняно із загальними алгоритмами Windows, оскільки алгоритми розподілу ресурсів спеціально налагоджені для використання у структурах SQL Server.

1.4 Використані плагіни

Для забезпечення якості коду, його читабельності та консистентності в проекті використовуються два ключові інструменти: ESLint та Prettier.

Для забезпечення високої якості коду та дотримання стандартів стилю у дипломному проекті використовується інструмент ESLint. ESLint є лінтером для мови програмування JavaScript, який дозволяє автоматично виявляти потенційні помилки, невідповідності стилевим правилам та недоліки у коді. Логотип ESLint зображений на рисунку 1.8.

Рисунок 1.8 – Логотип ESLint

Інтеграція ESLint у розробницький процес дозволяє автоматично перевіряти код на виявлення можливих проблем ще на етапі розробки. Це допомагає уникнути

поширення помилок та підвищує загальну якість кодової бази. ESLint також забезпечує можливість налаштовувати власні стилеві правила, що дозволяє адаптувати його до конкретних вимог та стандартів проекту.

З використанням ESLint розробники можуть однаково дотримуватися стилістики коду, а також ефективно співпрацювати в команді, оскільки інструмент допомагає створити єдиний стандарт коду. Це сприяє підтримці консистентності та читабельності усього коду, що є важливим фактором у розробці та підтримці проекту.

Додатково до використання ESLint для виявлення потенційних помилок та невідповідностей стилевим правилам, у дипломному проекті також використовується інструмент Prettier для автоматичного форматування коду. Prettier спрощує процес збереження єдиного стилістичного формату в усьому проекті, роблячи код консистентним та легким для читання.

Інтеграція Prettier дозволяє автоматично вирівнювати код, дотримуючись заданих стандартів форматування, таких як розташування відступів, використання одинарних чи подвійних кавичок, тощо. Це сприяє уніфікації кодової бази та зменшенню можливості спорів між розробниками щодо стилістики. Логотип Prettier зображений на рисунку 1.9.

Рисунок 1.9 – Логотип Prettier

Prettier і ESLint можуть бути використані разом, використовуючи плагіни та налаштування, для того щоб забезпечити якісний та консистентний код. Застосування Prettier у дипломному проекті допомагає не лише уникнути погроз ручного форматування, але й зробити процес розробки більш ефективним та спрощеним, зосереджуючись на основних завданнях розробки замість формальних аспектів стилістики коду.

2 ОПИС МЕТОДІВ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ БАНКУ ІНФОРМАЦІЙНОЇ СИСТЕМИ єВідновлення

2.1 Метод POST /api/v1/create–diia–account

Мета методу – передача клієнтського ідентифікатора від сервера мобільного додатку до API банку, який на основі цього ідентифікатора витягне інформацію з бази даних згідно вимог реалізації.

Параметри запиту наведені у таблиці 2.1.

Таблиця 2.1 – Параметри запиту методу POST /api/v1/create–diia–account

Параметр	Тип	Обов'язковий	Опис
customerId	integer	Так	Ідентифікатор клієнта

Таблиця 2.2 – Параметри успішної відповіді методу POST /api/v1/create–diia–account

Параметр	Тип	Обов’язковий	Опис
message	string	Так	Інформаційне повідомлення

Таблиця 2.3 – Параметри невдалої відповіді методу POST /api/v1/create–diia–account

Параметр	Тип	Обов’язковий	Опис
error	string	Так	Ознака, що вказує на невдачу операції з реєстрації нового банківського рахунку сервером ДІЯ

2.2 Метод DELETE /api/v1/delete–diia–account/:customerId

Мета методу – передача клієнтського ідентифікатора від сервера мобільного додатку до API банку, який на основі цього ідентифікатора передасть серверу ДІЯ команду на видалення акаунту.

Параметри запиту наведені у таблиці 2.4.

Таблиця 2.4 – Параметри запиту методу DELETE /api/v1/delete–diia–account/:customerId

Параметр	Тип	Обов'язковий	Опис
customerId	integer	Так	Ідентифікатор клієнта

Таблиця 2.5 – Параметри успішної відповіді методу DELETE /api/v1/delete–diia–account

Параметр	Тип	Обов’язковий	Опис
message	string	Так	Інформаційне повідомлення

Таблиця 2.6 – Параметри невдалої відповіді методу DELETE /api/v1/delete–diia–account/:customerId

Параметр	Тип	Обов’язковий	Опис
error	string	Так	Ознака, що вказує на невдачу операції з видалення банківського рахунку сервером ДІЯ

3 ОПИС БАНКІВСЬКОЇ БАЗИ ДАНИХ

3.1 Структура бази даних “bank”

База даних, розроблена для потреб банку, визначає сучасний підхід до зберігання та управління фінансовою інформацією. Ця база даних включає п'ять ключових таблиць, які взаємодіють між собою, надаючи повний спектр фінансових послуг клієнтам.

Кожна таблиця має свій унікальний ідентифікатор `Id`, а стовпці містять інші дані, такі як ідентифікатор клієнта, номер карти, тип документу тощо.

На рисунку 3.1 зображено базу даних `bank`.

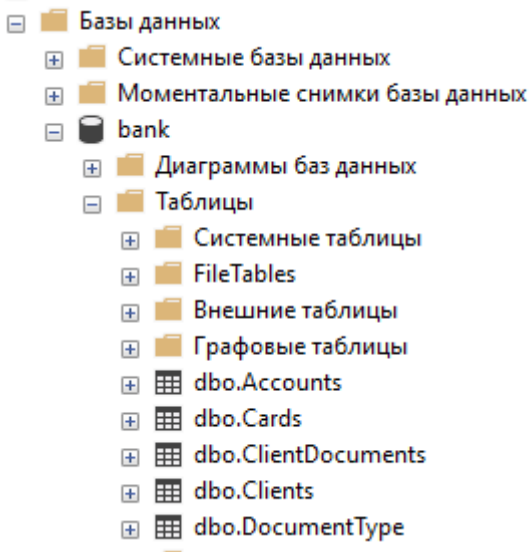


Рисунок 3.1 – База даних `bank`

Таблиця "Accounts" зберігає дані про банківські рахунки клієнтів, включаючи номер рахунку, зв'язок з конкретним клієнтом, міжнародний номер банківського рахунку (IBAN), звичайний номер банківського рахунку та його повну назву.

Таблиця "Cards" дозволяє відстежувати інформацію про кредитні картки, пов'язані з клієнтами. Інформація включає унікальний ідентифікатор карти, номер картки, термін дії та зв'язок з конкретним клієнтом.

Таблиця "ClientDocuments" містить дані про різноманітні документи, пов'язані з клієнтами. Вона включає унікальний ідентифікатор документа, ідентифікатор клієнта, тип документа та його номер.

Таблиця "Clients" зберігає інформацію про самих клієнтів банку. Ідентифікатор клієнта, його код, повне ім'я, код стану та зв'язок із відділенням банку.

"DocumentType" надає перелік різних типів документів, які можуть бути пов'язані з клієнтами.

Взаємодія між цими таблицями створює структуровану базу даних, яка відображає комплексність взаємозв'язків між клієнтами, їх рахунками та документами. Ця база даних служить надійним інструментом для ефективного управління фінансовими ресурсами та забезпечення високої якості обслуговування клієнтів.

3.2 Детальний опис таблиць

Типи даних та довжина зазначені відповідним чином у таблицях нижче:

1. Clients: таблиця, що містить інформацію про клієнтів банку.

Таблиця 3.1 – Структура та опис таблиці Clients

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор клієнта
Code	varchar	32	Код клієнта
FullName	varchar	240	Повне ім'я клієнта
StateCode	varchar	32	РНОКПП користувача
BranchId	integer	NULL	Ідентифікатор відділення, пов'язаний з клієнтом

2. DocumentType: таблиця, що містить інформацію про типи документів.

Таблиця 3.2 – Структура та опис таблиці DocumentType

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор типу документу
TypeId	integer	NULL	Ідентифікатор типу
Name	varchar	50	Назва типу документа

3. ClientDocuments: таблиця, що містить інформацію про документи клієнтів.

Таблиця 3.3 – Структура та опис таблиці ClientDocuments

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор документу
CustomerId	integer	NULL	Ідентифікатор клієнта, пов'язаний з документом
DocTypeId	integer	NULL	Ідентифікатор типу документа
DocNumber	varchar	50	Номер документа

4. Accounts: таблиця, що містить інформацію про рахунки клієнтів.

Таблиця 3.4 – Структура та опис таблиці Accounts

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор рахунку
CustomerId	integer	NULL	Ідентифікатор клієнта, пов'язаний з документом
Iban	varchar	50	Номер міжнародного банківського рахунку
Moniker	varchar	50	Номер банківського рахунку
Name	varchar	250	Повна назва рахунку

5. Cards: таблиця, що містить інформацію про кредитні картки клієнтів.

Таблиця 3.5 – Структура та опис таблиці Cards

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор клієнта
CustomerId	integer	NULL	Ідентифікатор клієнта, пов'язаний з картою
CardNumber	varchar	16	Номер кредитної карти
ExpDate	varchar	7	Термін дії кредитної карти

4 ОПИС МЕТОДІВ ПАРТНЕРСЬКОГО API ДЕРЖАВНОГО КОМЕРЦІЙНОГО ПІДПРИЄМСТВА «ДІЯ»

4.1 Метод POST /api/v1/partner/bank/create–account

Мета методу – реєстрація рахунку для виплати компенсації. Сервер Банку звертається до Сервера Дії із запитом «Реєстрація нового рахунку для виплати компенсації» з вказанням даних створеного банківського рахунку.

Сервер Дія підтверджує, що такий рахунок може бути зареєстрований:

- вказані значення параметрів валідні;
- картки з таким IBAN–кодом ще не зареєстровано;
- сервер Дія реєструє картку, зазначену в запиті;
- сервер Дія формує та надсилає відповідь з ознакою успішної реєстрації картки;
- сервер Дія виходить з виконання прецедента.

В іншому разі:

- сервер Дія підтверджує, що такий рахунок не може бути зареєстрований по одній з перелічених причин;
- сервер Дія формує та надсилає відповідь Серверу Партнера з ознакою НЕуспішної реєстрації картки та вказанням помилки;
- сервер Дія виходить з виконання прецедента.

Параметри запиту наведені у таблиці 4.1.

Таблиця 4.1 – Параметри запиту методу POST /api/v1/partner/bank/create–account

Параметр	Тип	Обов’язковий	Валідація	Опис
docType	string	Так	enum['passport', 'idpassport', 'zpassport', 'ident']	Тип документа, що посвідчує особу, на основі якого відкрито банківський рахунок: passport – паспорт громадянина України; idpassport – ID–картка громадянина України; zpassport – закордонний паспорт (на контролі фін. установи, тому згідно з специфікацією НБУ може бути і паспорт іншої держави); ident – будь–який інший документ, що посвідчує особу
iban	string	Так		IBAN–код рахунку
cardNum	string	Так		Номер банківської картки користувача у скороченому вигляді: <перші 4 цифри>********<останні 4 цифри>
expirationDate	date	Так	yyyy–MM	Дата завершення терміну дії картки (рік та місяць)
service	string	Так	enum['aid', 'recovery']	Тип послуги, в рамках якої створено картку: aid – єПідтримка; recovery – єВідновлення

Таблиця 4.2 – Параметри успішної відповіді методу POST /api/v1/partner/bank/create–account

Параметр	Тип	Обов’язковий	Опис
message	string	Так	Інформаційне повідомлення

Таблиця 4.3 – Параметри невдалої відповіді методу POST /api/v1/partner/bank/create–account

Параметр	Тип	Обов’язковий	Опис
error	string	Так	Ознака, що вказує на невдачу операції з реєстрації нового банківського рахунку сервером ДІЯ

4.2 Метод DELETE /api/v1/partner/bank/delete–account

Мета методу – видалення рахунку для виплати компенсації. Сервер Банку звертається до Сервера Дії із запитом «Закриття зареєстрованого банківського рахунку для виплати компенсації» з вказанням IBAN–коду рахунку, який необхідно закрити.

Сервер Дія підтверджує, що такий рахунок може бути закритий:

- вказані значення параметрів валідні;
- картку з таким IBAN–кодом знайдено;
- сервер Дія видаляє картку, зазначену в запиті;
- сервер Дія формує та надсилає відповідь з ознакою успішного закриття картки;
- сервер Дія виходить з виконання прецедента.

В іншому разі:

- сервер Дія підтверджує, що такий рахунок не може бути закритий по одній з перелічених причин;
- сервер Дія формує та надсилає відповідь Серверу Партнера з ознакою НЕуспішного закриття картки та вказанням помилки;
- сервер Дія виходить з виконання прецедента.

Таблиця 4.4 – Параметри запиту методу DELETE /api/v1/delete–diia–account/:iban

Параметр	Тип	Обов'язковий	Опис
iban	string	Так	IBAN–код банківського рахунку

Таблиця 4.5 – Параметри успішної відповіді методу DELETE /api/v1/delete–diia–account

Параметр	Тип	Обов’язковий	Опис
message	string	Так	Інформаційне повідомлення

Таблиця 4.6 – Параметри невдалої відповіді методу DELETE /api/v1/delete–diia–account/:iban

Параметр	Тип	Обов’язковий	Опис
error	string	Так	Ознака, що вказує на невдачу операції з видалення банківського рахунку сервером ДІЯ

5 ОПИС ПАРТНЕРСЬКОЇ БАЗИ ДАНИХ ДЕРЖАВНОГО КОМЕРЦІЙНОГО ПІДПРИЄМСТВА «ДІЯ»

5.1 Структура бази даних “diia”

Система "єВідновлення" використовує базу даних `diia` для ефективного зберігання та керування інформацією про рахунки користувачів, типи статусів та документів. У наведеному нижче описі докладно висвітлено структуру трьох основних таблиць: Accounts, StatusType, та DocumentType.

Кожна таблиця має свій унікальний ідентифікатор `Id`, а стовпці містять інші дані, такі як ідентифікатор клієнта, номер карти, тип документу, тип статусу тощо.

На рисунку 5.1 зображено базу даних `diia`.

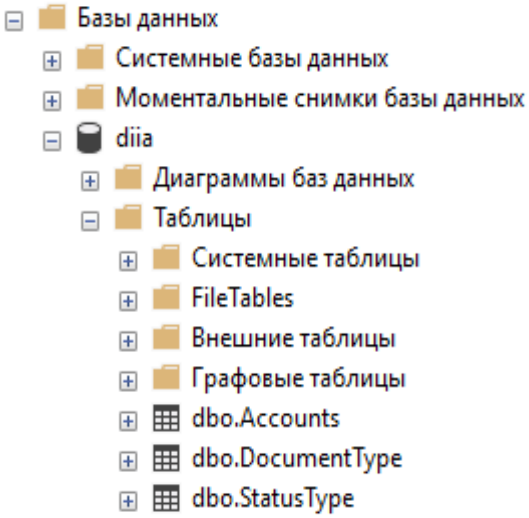


Рисунок 5.1 – База даних `diia`

Таблиця "Accounts" зберігає дані про рахунки клієнтів, включаючи тип документу, міжнародний номер банківського рахунку (IBAN), шифрований номер банківської картки, термін дії картки тощо.

"DocumentType" надає перелік різних типів документів, які можуть бути пов'язані з клієнтами.

"StatusType" надає перелік різних типів статусів, які можуть бути пов'язані з клієнтами.

Взаємодія між цими таблицями створює структуровану базу даних, яка відображає комплексність взаємозв'язків між рахунками, статусами та типами документів. Ця база даних служить надійним інструментом для ефективного

управління фінансовими ресурсами та забезпечення високої якості обслуговування клієнтів.

5.2 Детальний опис таблиць

Типи даних та довжина зазначені відповідним чином у таблицях нижче:

1. Accounts: таблиця, що відображає інформацію про рахунки користувачів у системі 'єВідновлення'.

Таблиця 5.1 – Структура та опис таблиці Accounts

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор клієнта
DocTypeId	integer	NULL	Ідентифікатор типу документа, пов'язаного з рахунком
Iban	varchar	50	Номер міжнародного банківського рахунку
ExpirationDate	varchar	7	Термін дії кредитної карти
CardNum	varchar	16	Номер кредитної карти
Service	varchar	32	Назва послуги або продукту, пов'язаного з рахунком
Status	integer	NULL	Статус рахунку, пов'язаний з таблицею `StatusType`
Created	datetime	NULL	Дата і час створення запису
Changed	datetime	NULL	Дата і час останньої зміни запису

2. DocumentType: таблиця, що містить інформацію про типи документів.

Таблиця 5.2 – Структура та опис таблиці DocumentType

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор типу документу
TypeId	integer	NULL	Ідентифікатор типу документу
Name	varchar	50	Назва типу документа

3. StatusType: таблиця, що містить інформацію про статуси клієнтів.

Таблиця 5.3 – Структура та опис таблиці StatusType

Назва	Тип	Довжина	Опис
Id	integer	NULL	Унікальний ідентифікатор документу
StatusId	integer	NULL	Ідентифікатор типу статусу
Name	varchar	50	Назва типу статусу

6 ПОРІВНЯННЯ ВИКОРИСТАНИХ ПРОГРАМНИХ ЗАСОБІВ

6.1 Порівняння IDE

Враховуючи великий вибір інтегрованих середовищ розробки (IDE) для програмістів, порівняння між Visual Studio Code (VS Code) та PyCharm може визначити, яка із цих платформ краще відповідає потребам.

Visual Studio Code (VS Code):

Плюси:

1. Універсальність та легкість використання: VS Code призначений для роботи з різними мовами програмування та відзначається легкістю використання.
2. Розширюваність: Велика кількість розширень дозволяє вам адаптувати IDE до своїх потреб.
3. Активна спільнота: Завдяки широкій аудиторії користувачів, ви можете швидко знайти відповіді на свої питання або вирішення проблем.

Мінуси:

1. Менше спеціалізованих інструментів: Для певних мов може бути менше вбудованих інструментів порівняно з спеціалізованими IDE.

Альтернатива - PyCharm:

Плюси:

1. Спеціалізація на Python: PyCharm розроблено спеціально для мови Python та має вбудовані інструменти для розробки на цій мові.
2. Великий функціонал для рефакторингу: PyCharm має потужні інструменти для рефакторингу коду та аналізу якості.

Мінуси:

1. Більший обсяг ресурсів: В порівнянні з VS Code, PyCharm може використовувати більше системних ресурсів.
2. Обираючи між ними, враховуйте, що VS Code є загальною та легкою використанні платформою, тоді як PyCharm спеціалізується на Python і надає більше спеціалізованих інструментів для цієї мови.

6.2 Порівняння СУБД

Вибір системи управління базами даних (СУБД) — важливий етап для будь-якого проекту. Один з популярних варіантів - Microsoft SQL Server (MSSQL). Однак існують інші альтернативи, які можуть відповідати різним потребам. Розглянемо порівняльний аналіз MSSQL та однієї з альтернатив.

Microsoft SQL Server (MSSQL):

Плюси:

1. Широкий функціонал: MSSQL надає розширені можливості управління даними та аналітики, що робить його ефективним для різних застосувань.
2. Інтеграція з продуктами Microsoft: Взаємодія з іншими продуктами Microsoft полегшує розробку та впровадження, забезпечуючи єдність у робочому процесі.

Мінуси:

1. Ліцензійні витрати: Вартість ліцензій на MSSQL може бути високою, що важливо врахувати при плануванні бюджету.

Альтернатива - PostgreSQL:

Плюси:

1. Відкритий код: PostgreSQL - це вільна та відкрита СУБД, що дозволяє змінювати її код відповідно до потреб проекту.
2. Широкі можливості розширення: PostgreSQL підтримує розширення, що робить його гнучким для різноманітних вимог.

Мінуси:

Менша інтеграція з Microsoft-продуктами: Деяка несумісність з продуктами Microsoft може виникнути через відмінності в стандартах та протоколах.

6.3 Порівняння мов програмування

JavaScript та його сучасний синтаксис TypeScript широко використовуються у веб-розробці для створення динамічних та інтерактивних інтерфейсів. TypeScript є суперсетом JavaScript, що дозволяє використовувати статичні типи даних, що полегшує розробку складних проектів.

Переваги:

1. Широке застосування у веб-розробці: JS/TS є основною мовою для фронтенд-розробки, а також використовується для створення серверної частини (Node.js).
2. Активна спільнота та екосистема: Велика спільнота та багата екосистема бібліотек та фреймворків для розробки.

Недоліки:

1. Менша придатність для наукового програмування: У порівнянні з Python, JS/TS може бути менш зручним для наукового програмування та обробки даних.

Альтернатива - Python:

Python — це універсальна мова програмування, яка використовується у різних областях, таких як веб-розробка, наукове програмування, штучний інтелект та інші.

Має чистий синтаксис та велику кількість бібліотек.

Переваги:

1. Силка в науковому програмуванні: Python є популярним вибором для наукових досліджень, обробки даних та машинного навчання.
2. Читабельний код: Python славиться своїм зрозумілим та лаконічним синтаксисом, що полегшує написання та розуміння коду.

Недоліки:

1. Швидкість виконання: У порівнянні з JavaScript, Python може бути менш продуктивним в деяких випадках через його інтерпретовану природу.

6.4 Порівняння фреймворків

NestJS і Fastify - це два популярних інструменти для розробки серверних додатків на платформі Node.js. Кожен з них має свої унікальні особливості та підходи, і вибір між ними залежатиме від конкретних потреб та вимог вашого проекту. Давайте розглянемо їх порівняльний аналіз з врахуванням важливих аспектів розробки веб-додатків.

NestJS:

Переваги:

1. Підтримка TypeScript: NestJS базується на TypeScript, що дозволяє вам використовувати статичні типи та отримувати переваги типізації на етапі розробки.
2. Масштабованість та модульність: Заснований на патерні "Dependency Injection", NestJS полегшує створення масштабованих та легко модульних додатків.
3. Архітектурний підхід MVC: Надає структурований підхід до розробки, що робить код більш організованим.

Недоліки:

1. Велика кількість абстракцій: Для деяких розробників може здаватися, що NestJS використовує велику кількість абстракцій, що може збільшити поріг входження.

Fastify:

Переваги:

1. Висока продуктивність: Fastify славиться високою швидкістю та ефективністю, що дозволяє обробляти багато запитів за короткий час.
2. Низький рівень абстракцій: Fastify надає мінімальну кількість абстракцій, що сприяє прозорості та дозволяє розробникам більше контролю над своїм кодом.
3. Широка підтримка middleware: Fastify активно підтримує широкий вибір

middleware для розширення функціональності додатків.

Недоліки:

1. Специфічний для HTTP: Fastify спеціалізується на обробці HTTP-запитів, що може обмежити його застосування для інших варіантів використання.

NestJS надає високорівневий підхід та TypeScript, в той час як Fastify фокусується на швидкості та прозорості.

7 РОЗРОБКА ТА ОПИС ІНФОРМАЦІЙНОЇ СИСТЕМИ БАНКУ

7.1 Схема роботи інформаційної системи банку

Наведена блок-схема, зображена на рисунку 7.1, ілюструє послідовність взаємодій між клієнтом, вебсервером банку, базою даних банку, вебсервером ДІЯ та базою даних ДІЯ в рамках виконання складних запитів.

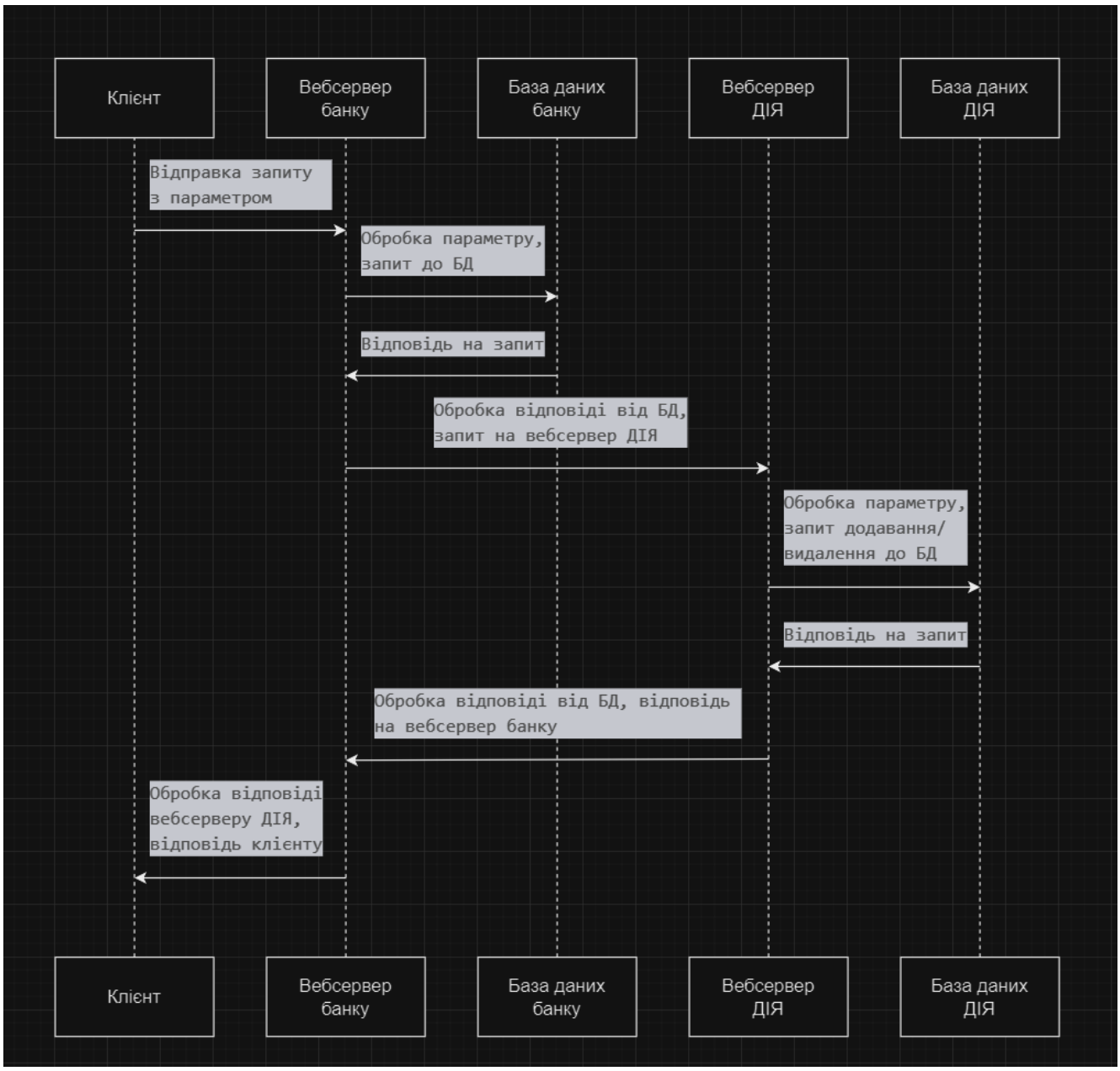


Рисунок 7.1 – Схема роботи системи

За допомогою чітких етапів та взаємодії, представлено процес обробки запиту від моменту відправки клієнтом до отримання відповіді. Детально врахована

обробка параметрів, взаємодії з базами даних, та подальший обмін інформацією між вебсерверами.

Даний процес визначає ефективну роботу системи у відповідь на потреби клієнта та динамічної внутрішньої інфраструктури банку та вебсервера ДІЯ.

7.2 Розробка демонстраційного вебсервісу банку

1. Встановлення Nest CLI:

```
npm install -g @nestjs/cli
```

2. Створення проекту:

```
nest new bank-side
```

3. Перейдемо в директорію проекту:

```
cd bank-side
```

4. Встановлення TypeORM та драйвера для MSSQL:

```
npm install --save @nestjs/typeorm typeorm mssql
```

5. Створення файлу конфігурації для TypeORM:

Створюємо файл `ormconfig.ts`. Після створення файлу, додаємо налаштування для нашої бази даних MSSQL. Параметр `entities` визначає шлях або шаблон шляхів до файлів, які містять описи сутностей (entities) бази даних.

При створенні класів, які представляють таблиці бази даних, ці класи вважаються "сутностями" (entities). Наприклад, якщо ми створили клас `Accounts` для відображення таблиці бази даних "Accounts", цей клас буде вважатися сутністю.

Приклад конфігурації:

```
import { TypeOrmModuleOptions } from '@nestjs/typeorm';
import { ConfigService } from '@nestjs/config';
import { Account } from '../models/accounts.model';
import { Card } from '../models/cards.model';
import { ClientDocument } from '../models/client-documents.model';
import { Client } from '../models/clients.model';
import { DocumentType } from '../models/document-type.model';

export default async (
  configService: ConfigService,
): Promise<TypeOrmModuleOptions> => ({
  type: 'mssql',
  host: configService.get<string>('DB_HOST'),
  port: parseInt(configService.get<string>('DB_PORT'), 10),
  username: configService.get<string>('DB_USERNAME'),
  password: configService.get<string>('DB_PASSWORD'),
  database: configService.get<string>('DB_DATABASE'),
  entities: [Account, Card, ClientDocument, Client, DocumentType],
  synchronize: false,
  logging: false,
  options: {
    trustServerCertificate: true,
  },
});
```

6. Використання `.env` файлу та `ConfigService`:
Впевнюємося, що у нас встановлений необхідний пакет для взаємодії:

```
npm install --save @nestjs/config dotenv
```

Створимо файл з ім’ям `.env` у кореневому каталозі нашого проекту та додамо необхідні змінні оточення.

Приклад:

```
DB_HOST=localhost
DB_PORT=1433
DB_USERNAME=sheva
DB_PASSWORD=1234
DB_DATABASE=bank
DIIA_URL_CREATE=http://localhost:3001/api/v1/partner/bank/create-account
DIIA_URL_DELETE=http://localhost:3001/api/v1/partner/bank/delete-account
```

Створимо модуль конфігурації в файлі `config.module.ts`:

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      envFilePath: ['.env'],
    }),
  ],
})
export class AppConfigModule {}
```

Додамо цей модуль до нашого основного модулю додатку, в ньому ми визначаємо всі компоненти нашого додатку, такі як контролери, провайдери тощо:

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AppConfigModule } from './config/config.module';
import { ConfigService } from '@nestjs/config';
import ormconfig from './config/ormconfig';
import { DiiaAccountController } from './controllers/diia-account.controller';
import { DiiaAccountService } from './services/diia-account.service';
import { Accounts } from './models/accounts.model';
import { Cards } from './models/cards.model';
import { ClientDocuments } from './models/client-documents.model';
import { Clients } from './models/clients.model';
import { DocumentType } from './models/document-type.model';

@Module({
  imports: [
    AppConfigModule,
    TypeOrmModule.forRootAsync({
      useFactory: async (configService: ConfigService) =>
        await ormconfig(configService),
      inject: [ConfigService],
    }),
    TypeOrmModule.forFeature([
      Accounts,
      Cards,
      ClientDocuments,
      Clients,
      DocumentType,
    ]),
  ],
  controllers: [DiiaAccountController],
  providers: [DiiaAccountService],
})
export class AppModule {}
```

Після цих маніпуляцій, ми зможемо отримувати конфігураційні змінні оточення в нашому `ormconfig.ts`.

7. Створення моделей, контролерів, сервісів та DTO (Data Transfer Object):

Створимо файли для наших моделей у відповідному каталозі (`src/models`). Приклад моделі для `Accounts`:

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity({ name: 'Accounts' })
export class Accounts {
  @PrimaryGeneratedColumn()
  Id: number;

  @Column()
  CustomerId: number;

  @Column()
  Iban: string;

  @Column()
  Moniker: string;

  @Column()
  Name: string;
}
```

Приклад моделі для `Clients`:

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity({ name: 'Clients' })
export class Clients {
  @PrimaryGeneratedColumn()
  Id: number;

  @Column()
  Code: string;

  @Column()
  FullName: string;

  @Column()
  StateCode: string;

  @Column()
  BranchId: number;
}
```

Приклад моделі для `Cards`:

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity({ name: 'Cards' })
export class Cards {
  @PrimaryGeneratedColumn()
  Id: number;

  @Column()
  CustomerId: number;

  @Column()
  CardNumber: string;

  @Column()
  ExpDate: string;
}
```

Аналогічно створимо моделі для `ClientDocument` та `DocumentType`.

Далі, необхідно створити DTO для обробки вхідних даних запиту для створення акаунту. Наприклад:

```
import { IsString } from 'class-validator';

export class CreateDiiaAccountDto {
  @IsString()
  customerId: string;
}
```

Аналогічно створимо DTO для вхідних даних запиту для видалення акаунту:

```
import { IsString } from 'class-validator';

export class DeleteDiiaAccountDto {
  @IsString()
  customerId: string;
}
```

Після можемо створити файл для сервісу, який буде містити бізнес-логіку:

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { ConfigService } from '@nestjs/config';
import { Accounts } from '../models/accounts.model';
import { Cards } from '../models/cards.model';
import { ClientDocuments } from '../models/client-documents.model';
import { Clients } from '../models/clients.model';
import { DocumentType } from '../models/document-type.model';
import { DeleteDiiaAccountDto } from '../dto/delete-diia-account.dto';

@Injectable()
export class DiiaAccountService {
  constructor(
    private readonly configService: ConfigService,
    @InjectRepository(Accounts)
    private readonly accountRepository: Repository<Accounts>,
    @InjectRepository(Cards) private readonly cardRepository:
Repository<Cards>,
    @InjectRepository(ClientDocuments)
    private readonly clientDocumentsRepository: Repository<ClientDocuments>,
    @InjectRepository(Clients)
    private readonly clientsRepository: Repository<Clients>,
    @InjectRepository(DocumentType)
    private readonly documentTypeRepository: Repository<DocumentType>,
  ) {}

  async processData(customerId: string): Promise<any> { }
  async sendPostRequest(data: any): Promise<any> { }
  async findIbanByCustomerId(customerId: string): Promise<string | null> { }
  async sendDeleteRequest(iban: string): Promise<any> { }
  async deleteDiiaAccount(dto: DeleteDiiaAccountDto): Promise<any> { }
```

В оголошених методах присутня бізнес-логіка. Кожен метод має свої функціональні особливості. Пояснення про них:

1) processData – приймає ідентифікатор клієнта, та шукає в базі даних відомості для відправки на сервер ДІЯ, через нього контролер отримує відповідь на запит.
2) sendPostRequest – відправляє POST запит до серверу ДІЯ.

3) findIbanByCustomerId – шукає IBAN рахунок по ідентифікатору клієнта.
4) sendDeleteRequest – відправляє DELETE запит до серверу ДІЯ.
5) deleteDiiaAccount – приймає ідентифікатор клієнта, та викликає допоміжні методи, через нього контролер отримує відповідь на запит.

Далі, можемо створити контролер, який обробляє POST запити.

```
import {
  Controller,
  Post,
  Body,
  ValidationPipe,
  Delete,
  Param,
} from '@nestjs/common';
import { DiiaAccountService } from '../services/diia-account.service';
import { CreateDiiaAccountDto } from '../dto/create-diia-account.dto';
import { DeleteDiiaAccountDto } from '../dto/delete-diia-account.dto';

@Controller('api/v1')
export class DiiaAccountController {
  constructor(private readonly diiaAccountService: DiiaAccountService) {}

  @Post('create-diia-account')
  async createDiiaAccount(
    @Body(ValidationPipe) requestBody: CreateDiiaAccountDto,
  ): Promise<any> {
    try {
      const result = await this.diiaAccountService.processData(
        requestBody.customerId,
      );
      const responseFromOtherServer =
        await this.diiaAccountService.sendPostRequest(result);
      // Вертаємо відповідь клієнту
      return responseFromOtherServer;
    } catch (error) {
      console.error(error);

      // Обробка помилок
      return { error: 'An error occurred while processing the request.' };
    }
  }
}
```

Для подальшої роботи контролера з DELETE запитами, потрібно внести корективи в наш класс `DiiaAccountController`, додавши в нього:

```
@Delete('delete-diia-account/:customerId')
  async deleteDiiaAccount(
    @Param('customerId') customerId: string,
  ): Promise<any> {
    try {
      const dto = new DeleteDiiaAccountDto();
      dto.customerId = customerId;

      const result = await this.diiaAccountService.deleteDiiaAccount(dto);
      // Вертаємо відповідь клієнту
      return result;
    } catch (error) {
      // Обробка помилок
      return { error: 'An error occurred while processing the request.' };
    }
  }
```

8. Налаштування `main.ts`:

Для фінального запуску вебсервісу необхідно налаштувати `main.ts` для коректного завантаження усіх конфігурацій:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);
}
bootstrap();
```

9. Запуск демонстраційного вебсервісу

```
npm run start
```

7.3 Опис каталогів та файлів

Загальна структура демонстраційного вебсервісу банку включає в себе назву та опис усіх каталогів, конфігів, контролерів, моделей, сервісів, DTO та інших файлів.

Структура:

- src/controllers: Каталог для контролерів, які обробляють HTTP-запити і визначають, які дії повинні бути виконані для кожного маршруту.
- src/dto: Каталог для DTO (Data Transfer Objects), які використовуються для обробки вхідних даних.
- src/models: Каталог для моделей бази даних.
- src/services: Каталог для сервісів, які містять бізнес-логіку.
- src/app.module.ts: Основний модуль додатку, де імпортуються інші модулі, контролери та сервіси.
- src/main.ts: Файл для запуску додатку.
- src/config: Каталог для конфігураційних файлів, таких як `config.module.ts` та `ormconfig.ts`.
- .env: Файл із змінними середовища, в якому ми зберігаємо конфіденційні дані або налаштування.
- package.json: Файл із залежностями та скриптами для запуску та управління проектом.
- tsconfig.json: Файл конфігурації TypeScript.
- ormconfig.json: Файл конфігурації для TypeORM, який містить параметри для підключення до бази даних.
- .eslintrc.js: Конфігурація ESLint для налаштування стилістики коду та виявлення помилок.
- .prettierrc: Конфігурація Prettier для налаштування форматування коду.

7.4 Приклади запитів та відповідей

В розділі надаються конкретні приклади HTTP-запитів, які можна використовувати для взаємодії з нашим вебсервісом, а також відповіді, які очікуються від сервера. Цей розділ призначений для того, щоб надати розуміння та детальний огляд того, як клієнти (інші сервери) можуть спілкуватися з нашим вебсервісом через його інтерфейс API.

У кожному прикладі запиту не лише синтаксично правильний HTTP-запит, але й пояснення про його призначення. Крім того, відповіді на ці запити розібрані, надаючи зрозумілу інформацію про статус виконання, дані відповіді та будь-які можливі помилки.

Цей розділ допоможе як розробникам, які інтегрують наш вебсервіс, так і тестерам, які перевіряють його функціональність. Він є інструментом для забезпечення чіткості та взаєморозуміння між розробниками та користувачами

нашого вебсервісу.

1) Запит та відповідь на створення акаунту:

Запит:

```
curl --location 'http://localhost:3000/api/v1/create-diia-account' \
--header 'Content-Type: application/json' \
--data '{
    "customerId": "1"
}'
```

Відповідь:

```
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 43
ETag: W/"2b-pGph72lvvzoJq3JX2Nn6ajay2SA"
Date: Wed, 13 Dec 2023 05:38:24 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Account created successfully."}
```

2) Запит та відповідь на створення дублюючого акаунту:

Запит:

```
curl --location 'http://localhost:3000/api/v1/create-diia-account' \
--header 'Content-Type: application/json' \
--data '{
    "customerId": "1"
}'
```

Відповідь:

```
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 52
ETag: W/"34-1yY7f7o1rTnGXcvIRRxnNbMKE8Q"
Date: Wed, 13 Dec 2023 05:46:13 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Account with this IBAN already exists."}
```

3) Запит та відповідь на видалення акаунту:

Запит:

```
curl --location --request DELETE 'http://localhost:3000/api/v1/delete-diia-account/1'
```

Відповідь:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 43
ETag: W/"2b-7ny52eoXTj6TavBzUqTJbYaG6UQ"
Date: Wed, 13 Dec 2023 05:48:43 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Account deleted successfully."}
```

4) Запит та відповідь на видалення неіснуючого акаунту:

Запит:

```
curl --location --request DELETE 'http://localhost:3000/api/v1/delete-diia-account/1'
```

Відповідь:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 52
ETag: W/"34-6yehBob5v5yerz5OgALfBOH3DQQ"
Date: Wed, 13 Dec 2023 05:50:13 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Account with this IBAN does not exist."}
```

8 ДЕМОНСТРАЦІЙНИЙ ВЕБСЕРВІС ДІЯ

8.1 Розробка демонстраційного вебсервісу ДІЯ

1. Встановлення Nest CLI:

```
npm install -g @nestjs/cli
```

2. Створення проекту:

```
nest new diia-side
```

Ця команда створить каталог `diia-side` і встановить базовий проект NestJS в цьому каталозі.

3. Перейдемо в директорію проекту:

```
cd diia-side
```

4. Встановлення TypeORM та драйвера для роботи з MSSQL:

```
npm install –save @nestjs/typeorm typeorm mssql
```

Ця команда встановить необхідні пакети, такі як `@nestjs/typeorm` (інтеграція NestJS з TypeORM), `typeorm` (ORM для TypeScript та JavaScript) та `mssql` (драйвер для підключення до Microsoft SQL Server).

5. Створення каталогу `partner`, пов'язаного з функціональністю, що стосується партнерів Дії:

```
mkdir src/partner
```

6. Створення ентіті, контролеру та сервісу:

На цьому ж етапі створимо `entities` файли (де будуть міститись описи сутностей робочих таблиць нашої БД). Простими словами, ентіті описують сутності, які ми хочемо зберігати у базі даних, та їх зв'язки між собою.

1) Файл /src/partner/account.entity.ts:

```
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity({ name: 'Accounts' })
export class Account {
  @PrimaryGeneratedColumn()
  Id: number;

  @Column({ name: 'DocTypeId' })
  DocTypeId: number;

  @Column({ name: 'Iban' })
  Iban: string;

  @Column({ name: 'ExpirationDate' })
  ExpirationDate: string;

  @Column({ name: 'CardNum' })
  CardNum: string;

  @Column({ name: 'Service' })
  Service: string;

  @Column({ name: 'Status' })
  Status: number;

  @Column({ name: 'Created' })
  Created: Date;

  @Column({ name: 'Changed' })
  Changed: Date;
}
```

2) Файл /src/partner/document-type.entity.ts:

```
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity({ name: 'DocumentType' })
export class DocumentType {
  @PrimaryGeneratedColumn({ name: 'TypeId' })
  DocumentId: number;

  @Column({ name: 'Name' })
  Name: string;
}
```

Використання ентіті дозволяє нам використовувати об'єктно-орієнтований підхід до роботи з даними в базі даних, а не працювати безпосередньо з SQL-запитами чи таблицями. Після створення ентіті, можна перейти до написання бізнес-логіки контролеру, який буде оброблювати та відповідати на запити:

```
import { Controller, Post, Body, Delete, Param } from '@nestjs/common';
import { AccountService } from './account.service';

@Controller('api/v1/partner/bank')
export class PartnerController {
  constructor(private readonly accountService: AccountService) {}

  @Post('create-account')
  async createAccount(
    @Body()
    data: {
      docType: string;
      iban: string;
      expirationDate: string;
      cardNum: string;
      service: string;
    },
  ): Promise<{ message: string }> {
    const existingAccount = await this.accountService.findAccountByIban(
      data.iban,
    );
```

Продовження коду:

```
    if (existingAccount) return { message: 'Account with this IBAN already exists.',
};
    await this.accountService.createAccount(data);
    return { message: 'Account created successfully.', };
  }

  @Delete('delete-account/:iban')
  async deleteAccount(
    @Param('iban') iban: string,
  ): Promise<{ message: string }> {
    const existingAccount = await this.accountService.findAccountByIban(iban);

    if (!existingAccount) return { message: 'Account with this IBAN doesn`t exist.'
};

    await this.accountService.deleteAccount(iban);
    return { message: 'Account deleted successfully.' };
  }
}
```

Далі, створюється сам сервіс, в ньому міститься основна логіка обробки вхідної інформації:

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Account } from './account.entity';
import { DocumentType } from './document-type.entity';

@Injectable()
export class AccountService {
  constructor(
    @InjectRepository(Account)
    private readonly accountRepository: Repository<Account>,
    @InjectRepository(DocumentType)
```

Продовження коду:

```
  private readonly documentTypeRepository: Repository<DocumentType>,
) {}

async createAccount(data: {
  docType: string;
  iban: string;
  expirationDate: string;
  cardNum: string;
  service: string;
}): Promise<Account> {
  // Пошук номеру DocumentType за його назвою
  const documentType = await this.documentTypeRepository.findOne({
    where: { Name: data.docType },
  });

  if (!documentType) {
    // Обробка випадку коли DocumentType не знайдений
    throw new NotFoundException(
      `DocumentType with Name ${data.docType} not found.`,
    );
  }

  // Створення акаунту
  const account = new Account();
  account.DocTypeId = documentType.DocumentId;
  account.Iban = data.iban;
  account.ExpirationDate = data.expirationDate;
  account.CardNum = data.cardNum;
  account.Service = data.service;
  account.Status = 1; // Статус 1 за замовчуванням - активний
  account.Created = new Date();
  account.Changed = new Date();

  return this.accountRepository.save(account);
}

async findAccountByIban(iban: string): Promise<Account | undefined> {
  return this.accountRepository.findOne({ where: { Iban: iban, Status: 1 } });
}
```

Продовження коду:

```
  async deleteAccount(iban: string): Promise<void> {
    const existingAccount = await this.findAccountByIban(iban);

    if (!existingAccount) throw new NotFoundException('Account not found.');

    existingAccount.Status = 0;
    existingAccount.Changed = new Date();

    await this.accountRepository.save(existingAccount);
  }
}
```

7. Оновлення `app.module.ts` та підключення до БД:

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Account } from './partner/account.entity';
import { AccountService } from './partner/account.service';
import { PartnerController } from './partner/partner.controller';
import { DocumentType } from './partner/document-type.entity';

@Module({
  imports: [
    TypeOrmModule.forRootAsync({
      useFactory: () => ({
        type: 'mssql',
        host: process.env.DB_HOST,
        port: parseInt(process.env.DB_PORT, 10) || 1433,
        username: process.env.DB_USERNAME,
        password: process.env.DB_PASSWORD,
        database: process.env.DB_DATABASE,
        synchronize: false,
        entities: [Account, DocumentType],
        logging: false,
```

Продовження коду:

```
        options: {
          trustServerCertificate: true,
        },
      }),
    }),
    TypeOrmModule.forFeature([Account, DocumentType]),
  ],
  controllers: [PartnerController],
  providers: [AccountService],
})
export class AppModule {}
```

8. Інсталяція `dotenv` та налаштування `.env` у корені проекту:

```
npm install –save dotenv
```

Ця команда викачує модуль, який дозволяє завантажувати значення змінних середовища з файлу `.env`. Створимо та заповнимо його:

```
DB_HOST=localhost
DB_PORT=1433
DB_USERNAME=sheva
DB_PASSWORD=1234
DB_DATABASE=diia
```

Цей файл використовується для зберігання конфігураційних параметрів, таких як дані підключення до бази даних.

9. Актуалізуємо `main.ts` згідно нашого оновленого коду:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as dotenv from 'dotenv';

dotenv.config();

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3001);
}

bootstrap();
```

10. Запускаємо демонстраційний вебсервіс:

```
npm run start
```

8.2 Опис каталогів та файлів

Загальна структура демонстраційного вебсервісу Дії включає в себе назву та опис усіх каталогів, контролерів, ентіті, сервісів та інших файлів.

Структура:

- src/app.module.ts: Цей файл визначає головний модуль нашого вебсервісу. Він вказує, які модулі повинні бути завантажені та які сервіси повинні бути доступні в усьому додатку.
- src/main.ts: Це основний файл, який викликається при запуску додатка. Він використовується для створення екземпляра додатка та його запуску.
- src/partner/: Це каталог, де організовуємо код, пов'язаний із партнерами Дії.
- src/partner/account.entity.ts: Описує сутність (ентіті) Account, яка відображає структуру таблиці Accounts у базі даних.
- src/partner/document-type.entity.ts: Описує сутність (ентіті) DocumentType, яка відображає структуру таблиці DocumentType у базі даних.

- src/partner/account.service.ts: Містить логіку, пов'язану із операціями бази даних для обробки даних.
- src/partner/partner.controller.ts: Містить контролер, який визначає ендпоїнти (шляхи API) та обробляє HTTP-запити для операцій над даними.
- src/: Загальний каталог проекту, в якому зберігається весь код вебсервісу.
- node_modules/: Каталог, де зберігаються всі залежності проекту, які встановлюються за допомогою npm install.
- .env: Файл із змінними середовища, в якому ми зберігаємо конфіденційні дані або налаштування.
- package.json: Файл із залежностями та скриптами для запуску та управління проектом.
- tsconfig.json: Файл конфігурації TypeScript.
- .eslintrc.js: Конфігурація ESLint для налаштування стилістики коду та виявлення помилок.
- .prettierrc: Конфігурація Prettier для налаштування форматування коду.

ВИСНОВКИ

У сучасному світі, де технологічний прогрес швидко розвивається, банківська сфера не може залишатися осторонь від інноваційних та інформаційних технологій. Розробка та впровадження інформаційних систем стає важливою складовою для забезпечення ефективного функціонування банківських установ. Дана магістерська дипломна робота присвячена розробці інформаційної системи для банку "єВідновлення" на базі мови програмування JavaScript.

Обрана тема дозволяє врахувати не лише сучасні вимоги до банківської діяльності, але й використовувати потужності сучасних веб-технологій для покращення обслуговування клієнтів та оптимізації внутрішніх банківських процесів. За допомогою інформаційної системи банку "єВідновлення" можливо надавати постраждалим громадянам державні виплати для відновлення житла або бізнесу, які постраждали від збройної агресії рф. Також, ця система є надійною, швидкою та безпечною.

В ході роботи була використана сучасна технологічна база, а саме мова програмування JavaScript, що дозволило створити вебсервіс з високою продуктивністю. Розроблену інформаційну систему можна вважати актуальною та конкурентоспроможною на фінансовому ринку.

Особлива увага була приділена аналізу та валідації вхідних/вихідних даних. Головне, що використана стратегія із заходами забезпечення кібербезпеки, яка спрямована на захист інформації, де на рівні мережі існує правило з пулом білих адрес, яким доступна взаємодія між серверами.

У подальших дослідженнях можливо розширити функціонал інформаційної системи, додавши нові модулі або покращити існуючі сервіси відповідно до змін в банківській сфері та технологічних тенденцій.

В цілому, розробка інформаційної системи банку "єВідновлення" на базі JavaScript є актуальною та перспективною задачею, яка може сприяти покращенню ефективності та конкурентоспроможності банківської установи в умовах сучасного фінансового ринку.

В результаті було розроблено повнофункціональну інформаційну систему, яку можна вдосконалити, додавши деталізоване логування запитів і відповідей, можливість формування добової звітності, авторизацію між серверами за токенами та записувати в базу унікальний ідентифікатор банку, від якого створений акаунт.

ПЕРЕЛІК ПОСИЛАНЬ

1. Вимоги до Порядку інформаційної взаємодії єВідновлення [Роздрукована документація]: Міністерство цифрової трансформації України. Державне підприємство «ДІЯ».
2. Node.js v18.19.0 documentation [Електронний ресурс]: https://nodejs.org/docs/latest-v18.x/api/index.html
3. Документація JavaScript [Електронний ресурс]: https://developer.mozilla.org/en-US/docs/Web/JavaScript
4. Офіційна документація NestJS [Електронний ресурс]: https://docs.nestjs.com/
5. Документація MSSQL (Microsoft SQL Server) [Електронний ресурс]: https://docs.microsoft.com/en-us/sql/
6. GitHub Репозиторій dotenv [Електронний ресурс]: https://github.com/motdotla/dotenv
7. Документація REST API на MDN Web Docs [Електронний ресурс]: https://developer.mozilla.org/en-US/docs/Web/HTTP
8. Документація Prettier [Електронний ресурс]: https://prettier.io/docs/en/
9. Документація ESLint [Електронний ресурс]: https://eslint.org/docs/user-guide/getting-started
10. NPM Пакет mssql [Електронний ресурс]: https://www.npmjs.com/package/mssql
11. Документація TypeScript [Електронний ресурс]: https://www.typescriptlang.org/docs/
12. Документація JSON на MDN Web Docs [Електронний ресурс]: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON
13. Документація Типи даних JavaScript на MDN Web Docs [Електронний ресурс]: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
14. GitHub Репозиторій Prettier [Електронний ресурс]: https://github.com/prettier/prettier
15. GitHub Репозиторій ESLint [Електронний ресурс]: https://github.com/eslint/eslint
16. Документація mssql на GitHub [Електронний ресурс]: https://github.com/tediousjs/node-mssql
17. GitHub Репозиторій TypeScript [Електронний ресурс]: https://github.com/microsoft/TypeScript
18. W3Schools JSON Tutorial [Електронний ресурс]: https://www.w3schools.com/js/js_json_intro.asp
19. Розділ Типи даних JavaScript на W3Schools [Електронний ресурс]: https://www.w3schools.com/js/js_datatypes.asp
20. REST API Introduction [Електронний ресурс]: https://restfulapi.net/
21. JSON.org [Електронний ресурс]: https://www.json.org/json-en.html

22. NPM Пакет Prettier [Електронний ресурс]: https://www.npmjs.com/package/prettier
23. NPM Пакет ESLint [Електронний ресурс]: https://www.npmjs.com/package/eslint
24. TypeScript Handbook на GitHub [Електронний ресурс]: https://github.com/microsoft/TypeScript/blob/main/doc/spec.md
25. Документація Node.js на GitHub [Електронний ресурс]: https://github.com/nodejs/node
26. GitHub Репозиторій NestJS [Електронний ресурс]: https://github.com/nestjs/nest
27. Wikipedia REST API [Електронний ресурс]: https://en.wikipedia.org/wiki/Representational_state_transfer
28. Документація MSSQL (Microsoft SQL Server) на W3Schools [Електронний ресурс]: https://www.w3schools.com/sql/
29. NPM Пакет dotenv [Електронний ресурс]: https://www.npmjs.com/package/dotenv
30. Документація TypeORM для NestJS [Електронний ресурс]: https://docs.nestjs.com/techniques/database

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

Державний університет інформаційно-комунікаційних технологій
Кафедра Інженерії програмного забезпечення автоматизованих систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
“Розробка інформаційної системи банку єВідновлення на базі JavaScript”

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр. ІСДМ-61
Дмитро ШЕВЧЕНКО
Керівник: к.т.н., доцент кафедри ІПЗАС
Оксана ТКАЛЕНКО

Київ - 2023

Актуальність теми: єВідновлення – це послуга, яка допомагає отримати державну допомогу для відновлення свого житла. Банки активно впроваджують інформаційні системи та технології для полегшення банківських операцій, забезпечення безпеки та підвищення якості обслуговування клієнтів. Розробка інформаційної системи, яка допоможе всім громадянам, чиє житло отримало пошкодження внаслідок збройної агресії рф і ще не було відремонтоване, вони зможуть скористатися послугою. Послугу єВідновлення максимально цифровізували та спростили для громадян. У такий непростий час для нашої країни важливо піклуватися про громадян, та надавати допомогу швидко і зручно.

Об’єкт дослідження: процес створення інформаційної системи банку для надання можливості громадянам та клієнтам отримувати допомогу від держави в короткі строки.

Предмет дослідження: методи прикладного програмного інтерфейсу інформаційної системи єВідновлення.

Мета і завдання дослідження: розробка інформаційної системи з використанням мови програмування JavaScript. У роботі передбачено проаналізувати методи взаємодії з системою, фреймворки мови програмування JavaScript для її проектування, визначити вимоги для проведення валідації вхідних та вихідних даних.

Методика дослідження: аналітична – ознайомлення з існуючою документацією, та практична – створення тестових запитів до системи, для інсценування різних випадків.

Наукова новизна: інформаційна система, яка автоматично передає усі необхідні дані для отримання допомоги з відновлення житла.

Практична значущість результатів: за пошкоджене росіянами майно вже виплатили 1,5 млрд грн. Кошти отримали майже 19 тисяч родин. Завдяки цій програмі наша країна може залишатися економічно стійкою державою. Так як, відновлюються постраждалі підприємства, збільшується кількість робочих місць, зберігається конкурентоспроможність України на світовому ринку. Це також про збереження людей в Україні, про те, щоб громадяни нашої країни не виїжджали, а залишалися.

Використані програмні засоби








Microsoft SQL Server

ESLint

Опис методу

POST /api/v1/partner/bank/create–account

Мета методу – реєстрація рахунку для виплати компенсації. Сервер Банку звертається до Сервера Дії із запитом «Реєстрація нового рахунку для виплати компенсації» з вказанням даних створеного банківського рахунку.

Сервер Дія підтверджує, що такий рахунок може бути зареєстрований:

- вказані значення параметрів валідні;
- картки з таким IBAN–кодом ще не зареєстровано;
- сервер Дія реєструє картку, зазначену в запиті;
- сервер Дія формує та надсилає відповідь з ознакою успішної реєстрації картки;
- сервер Дія виходить з виконання прецедента.

В іншому разі:

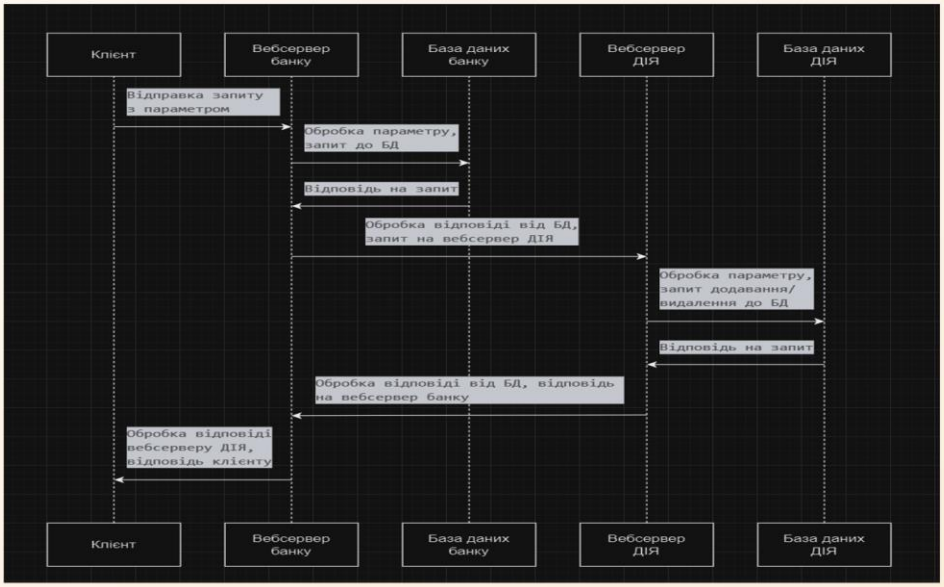
- сервер Дія підтверджує, що такий рахунок не може бути зареєстрований по одній з перелічених причин;
- сервер Дія формує та надсилає відповідь Серверу Партнера з ознакою НЕуспішної реєстрації картки та вказанням помилки;
- сервер Дія виходить з виконання прецедента

Параметри запиту методу

POST /api/v1/partner/bank/create–account

Параметр	Тип	Обов'язковий	Валідація	Опис
docType	string	Так	enum['passport', 'idpassport', 'zpassport', 'ident']	Тип документа, що посвідчує особу, на основі якого відкрито банківський рахунок: passport – паспорт громадянина України; idpassport – ID–картка громадянина України; zpassport – закордонний паспорт (на контролі фін. установи, тому згідно з специфікацією НБУ може бути і паспорт іншої держави); ident – будь–який інший документ, що посвідчує особу.
iban	string	Так		IBAN–код рахунку
cardNum	string	Так		Номер банківської картки користувача у скороченому вигляді: <перші 4 цифри>********<останні 4 цифри>
expirationDate	date	Так	yyyy–MM	Дата завершення терміну дії картки (рік та місяць)
service	string	Так	enum['aid', 'recovery']	Тип послуги, в рамках якої створено картку: aid – єПідтримка; recovery – єВідновлення

Схема роботи інформаційної системи



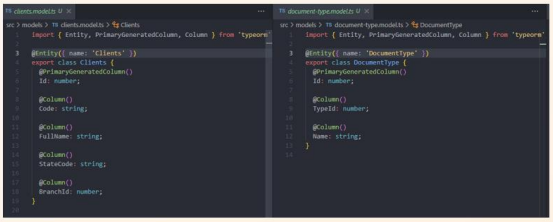
Параметри успішної відповіді методу POST /api/v1/partner/bank/create–account

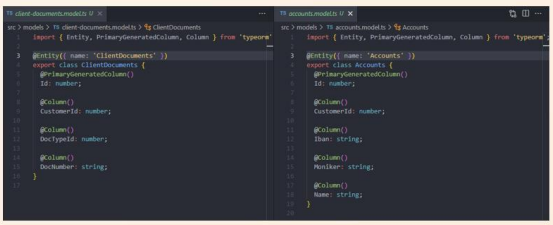
Параметр	Тип	Обов'язковий	Опис
message	string	Так	Інформаційне повідомлення

Параметри невдалої відповіді методу POST /api/v1/partner/bank/create–account

Параметр	Тип	Обов'язковий	Опис
error	string	Так	Ознака, що вказує на невдачу операції з реєстрації нового банківського рахунку сервером ДІЯ

Приклади валідації даних





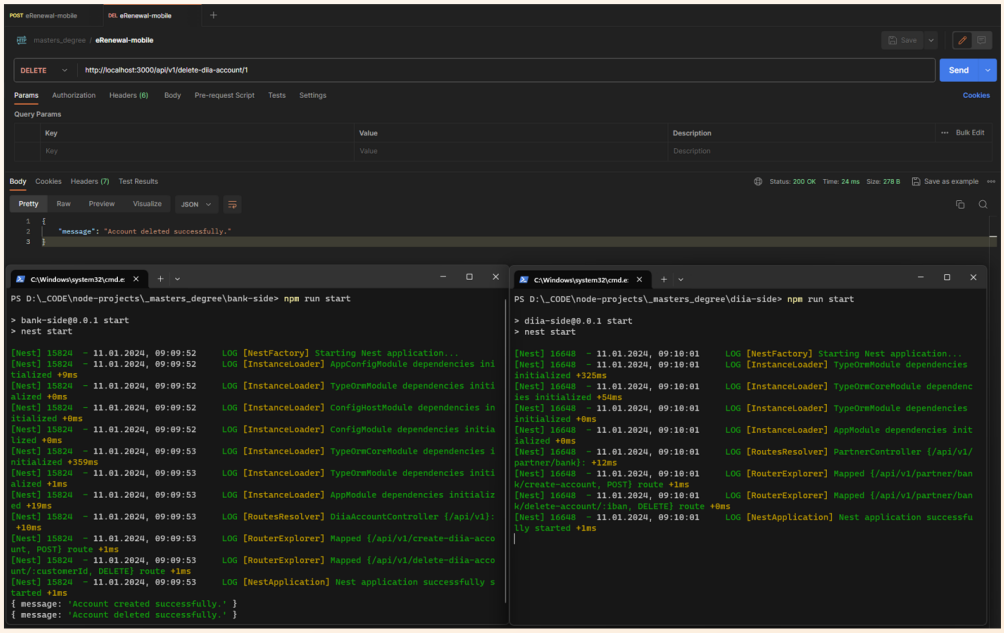
Статистика

Кількість створених рахунків

640,509

bank	COUNT(*)	%count
[redacted]	52,752	70.56%
[redacted]	12,755	17.06%
[redacted]	3,086	4.13%
[redacted]	2,689	3.60%
[redacted]	1,533	2.05%
[redacted]	1,196	1.60%
[redacted]	626	0.84%
[redacted]	69	0.09%
[redacted]	15	0.02%
[redacted]	12	0.02%
[redacted]	10	0.01%
[redacted]	9	0.01%
[redacted]	9	0.01%
[redacted]	4	0.01%
[redacted]	1	0.00%
Totals	74,766	

Приклад відпрацювання системи



Перспективи

Для Банку є дуже важливою участь у такій програмі, як єВідновлення. Сьогодні ми всі підтримуємо та допомагаємо один одному, бо маємо одну біду на всіх. І такі програми покращують передумови для швидкого відновлення пошкодженого житла, дають розуміння, що ти не залишаєшся сам на сам зі своїми проблемами, що тебе підтримають.

Допоможуть друзі, близькі, знайомі й незнайомі, допоможе держава і весь цивілізований світ. Банк, зі свого боку, для одержувачів виплати за програмою єВідновлення також долучається з допомогою. Додатково до коштів, які клієнт отримає за програмою, банк нараховуватиме бонуси та відсотки на залишок коштів на рахунках, передбачено також вручення подарункових сертифікатів на будівельні матеріали.

Висновки

У сучасному світі, де технологічний прогрес швидко розвивається, банківська сфера не може залишатися осторонь від інноваційних та інформаційних технологій. Розробка та впровадження інформаційних систем стає важливою складовою для забезпечення ефективного функціонування банківських установ.

Обрана тема дозволяє врахувати не лише сучасні вимоги до банківської діяльності, але й використовувати потужності сучасних веб-технологій для покращення обслуговування клієнтів та оптимізації внутрішніх банківських процесів. За допомогою інформаційної системи банку "єВідновлення" можливо надавати постраждалим громадянам державні виплати для відновлення житла або бізнесу, які постраждали від збройної агресії рф. Також, ця система є надійною, швидкою та безпечною.

В ході роботи була використана сучасна технологічна база, а саме мова програмування JavaScript, що дозволило створити вебсервіс з високою продуктивністю. Розроблену інформаційну систему можна вважати актуальною та конкурентоспроможною на фінансовому ринку.

Особлива увага була приділена аналізу та валідації вхідних/вихідних даних. Головне, що використана стратегія із заходами забезпечення кібербезпеки, яка спрямована на захист інформації, де на рівні мережі існує правило з пулом білих адрес, яким доступна взаємодія між серверами.

У подальших дослідженнях можливо розширити функціонал інформаційної системи, додавши нові модулі або покращити існуючі сервіси відповідно до змін в банківській сфері та технологічних тенденцій.

В результаті було розроблено повнофункціональну інформаційну систему, яку можна вдосконалити, додавши деталізоване логування запитів і відповідей, можливість формування добової звітності, авторизацію між серверами за токенами та записувати в базу унікальний ідентифікатор банку, від якого створений акаунт.

Апробація результатів дослідження

1. МінфінМедіа. “Офіційний прес-реліз приєднання банку до програми єВідновлення”. – Київ, 27 серпня 2023. https://minfin.com.ua/ua/2023/08/27/111493805/
2. ТОВ «Файненс.юа». “єВідновлення в Дії: 3 причини отримати допомогу через Банк”. – Київ, 5 вересня 2023. https://news.finance.ua/ua/yevidnovlennya-v-dii-3-prychyny-otrymaty-dopomohu-cherez-bank

Дякую за увагу!