ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-ДОДАТКУ ДЛЯ ГЕНЕРАЦІЇ SQL КОДУ МОВОЮ JAVASCRIPT»**

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

О.Ю. Котул
(прізвище та ініціали)

Керівник С.В. Поперешняк
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Нормконтроль _____
(прізвище та ініціали)

Київ-2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

Котулу Олександру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка web-додатку для генерації SQL коду мовою JavaScript»

Керівник роботи: Поперешняк С.В. к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «16» лютого 2022 року №22.
2. Строк подання студентом роботи «6» червня 2022 року
3. Вхідні дані до роботи
   3.1Методи управління БД;
   3.2 Існуючі інструменти та системи для управління БД;
   3.3Науково-технічна література;
4 Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
   4.1 Аналіз існуючих засобів для генерації SQL коду.
   4.2 Моделювання архітектури додатку для генерації SQL коду.
   4.3 Програмна реалізація веб-додатку.
   4.4 Опис використаних технологій.
   4.5 Висновки

5 Перелік демонстраційного матеріалу (назва основних слайдів)

5.1 Титульний слайд

5.2 Мета, об’єкт та предмет дослідження

5.3 Аналоги

5.4 Порівняння з аналогами

5.5 Технічне завдання

5.6 Програмні засоби реалізації

5.7 Архітектура додатку

5.8 Діаграма діяльності

5.9 Діаграма прецедентів

5.10 Діаграма потоків

5.11 Діаграма класів

5.12 Модель авторизації

5.13 Перспективи дослідження

5.14 Апробація результатів дослідження

5.15 Висновки

6 Дата видачі завдання «11» квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04-15.04	Виконано
2	Аналіз предметної області	16.04-20.04	Виконано
3	Дослідження існуючих інструментів для генерації SQL коду	20.04-23.04	Виконано
4	Моделювання архітектури системи	23.04-30.04	Виконано
5	Розробка системи для генерації SQL коду	30.04-17.05	Виконано
6	Висновки, оформлення роботи	18.05-.23.05	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	23.05-30.05	Виконано
8	Попередній захист роботи	03.06	Виконано
9	Здача роботи	06.06	

Студент _____ Котул О.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Поперешняк С.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 49 с., 36 рис., 2 таб., 18 джерел.

Об'єктом дослідження: автоматизація процесу додавання партнера в БД

Предметом дослідження: система для автоматизація процесу додавання партнера в БД шляхом генерації SQL коду

Мета роботи: прискорення зворотнього зв'язку з клієнтом, завдяки зменшенню обсягу роботи та вилучення з ланцюжка співробітників які вручну пишуть SQL код для внесення параметрів в БД.

Завдання роботи: розробка системи для автоматизації процесу додавання партнера в БД шляхом генерації SQL коду

Методика дослідження: емпіричні,системні та методи аналізу.

Для дослідження поставленної задачі потрібно сформулювати вимоги до ПЗ, змоделювати та розробити його архітектуру. Після аналізу аналогів проекту,а саме додатків які виконують функції CRUD(create,read,update,delete) системи, знайдено декілька систем, які можуть допомогти прискорити написання коду, або допомогти в автоматизації виконання коду. Загальною проблемою цих додатків, являється складність та незрозумілість інтерфейсу та платне використання.

Для вирішення цих проблем було розроблено веб-додаток, з простим інтерфейсом та потрібним функціоналом. Для розробки було використано популярний комплекс програмного забезпечення для веб-розробки MEAN(MongoDB, Express, Angular, NodeJS)

Галузь викоистання: розроблений додаток зможуть використовувати інженери або адміністратори БД для прискорення додавання клієнтів

ЗМІСТ

ВСТУП

Обґрунтування вибору теми та її актуальність: напевно кожна людина виконує якусь одноманітну роботу з певною періодичністю. Для вирішення таких проблем якраз і використовується автоматизація процесів.

Одне з таких завдань, це написання SQL коду за певними шаблонами, для внесення партнерів компанії в базу даних. Даний SQL код пишеться мною та моїми колегами, а потім виконується адміністратором бази даних.

Для прискорення виконання роботи, зменшення часу зворотнього зв’язку із клієнтом та заощадження коштів компанії необхідно розробити систему, яка буде генерувати SQL код, після внесення даних про партнера

Ступінь вивчення проблеми: на сьогодні існує небагато рішень для генерації SQL коду. Деякі системи управління базами даних вміють це робити, але вони мають певні недоліки, наприклад не зручний та складний інтерфейс для недосвідченого користувача або платний функціонал,

Об’єктом дослідження: автоматизація процесу додавання партнера в БД

Предметом роботи: система для автоматизація процесу додавання партнера в БД шляхом генерації SQL коду

Мета роботи: прискорення зворотнього зв’язку з клієнтом, завдяки зменшенню обсягу роботи та вилучення з ланцюжка співробітників які вручну пишуть SQL код для внесення параметрів в БД.

Завдання роботи: розробка системи для автоматизації процесу додавання партнера в БД шляхом генерації SQL коду

Методика дослідження: емпіричні,системні та методи аналізу.

Для дослідження поставленної задачі потрібно сформулювати вимоги до ПЗ, змоделювати та розробити його архітектуру. Після аналізу аналогів проекту,а саме додатків які виконують функції CRUD(create,read,update,delete) системи, не знайдено жодного рішення яке б допомогло забезпечити функціональне вирішення розглянутої проблеми.

Практична значущість результатів: розроблений додаток зможуть використовувати інженери або адміністратори БД для прискорення додавання клієнтів.

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Кожна компанія так чи інакше орієнтована на клієнта, а її головною ціллю є прибуток який він може принести. Маркетинг та реклама основні інструменти для приваблювання клієнтів або партнерів, але чим їх більше, тим складніше їх підтримувати. За рахунок цього збільшується кількість вхідної інформації і як наслідок навантаження на співробітників. Далі існує два варіанта, збільшення персоналу (або їх заробітньої плати), щоб мати можливість вчасно та якісно виконувати свої обов’язки, або автоматизація тих процесів, які безпосередньо приймають участь в обслуговуванні. Найбільш правильний варіант це використання автоматизації, що звільняє людину від участі у складних процесах, або істотно зменшує міру цієї участі чи трудомісткість виконуваних операцій.

Одне з таких завдань, це написання SQL коду за певними шаблонами, для внесення параметрів партнерів компанії в базу даних. Коли клієнт звертається до банку, для того щоб отримати послуги електронної комерції(AccountToCard, CardToAccount, PeerToPeer), співробітник банку створює заявку з необхідними параметрами які необхідно внести в БД. Для внесення даних пишеться SQL код, мною та моїми колегами інженерами, далі перевіряється на правильність внесених даних та погоджується керівником, а потім виконується адміністратором бази даних. Після виконання коду в БД адміністратор повинен дати інженеру зворотній зв'язок, а інженер в свою чергу повинен передати співробітнику який створив заявку дані, які були внесені в БД (ID який отримав партнер, та ім’я яке було внесено в БД). Враховуючи що середній час проходження кожного етапу приблизно 1 година, то фінальний час виконання завдання може досягати близько 5 годин.

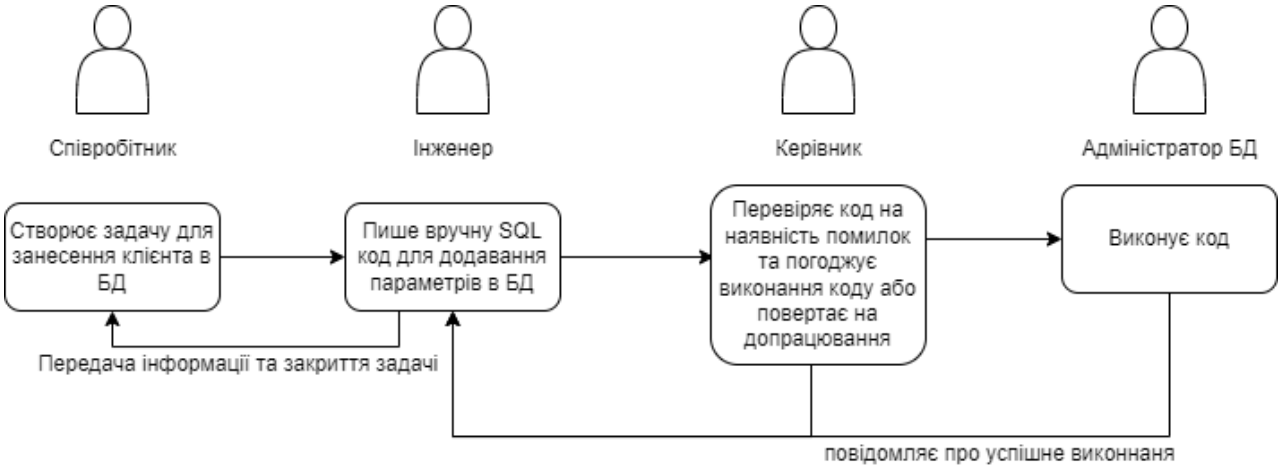


Рис. 1.1 - Бізнес-процес додавання клієнта

Головною метою цього завдання є розробка системи для автоматизації процесу додавання партнера в БД шляхом генерації SQL коду. Також в функціонал системи входить підключення до тестової БД та безпосереднє збереження в неї даних. В цьому дослідженні використовується саме тестова БД, тому що продова БД компанії дуже добре захищена фаєрволами, і доступ в неї здійснюється лише через віртуальну приватну мережу. Результатом цього дослідження в найкращому випадку очікується спрощення бізнес процесу, шляхом видалення з ланцюжка інженера та керівника, що в результаті дозволить зменшити наступне:

1) час очікування клієнта;
2) робочі завдання інженера;
3) час керівника на перевірку та погодження;
4) витрати підприємства.

Рис. 1.2 - Спрощений бізнес-процес додавання клієнта

Також допускається, що керівник може не допустити змінити увесь бізнес процес. Це можна пояснити з точки зору безпеки внесення даних в БД, та контролю якості. В цьому випадку систему буде використовувати не адміністратор БД, а інженер. Розроблена система звільнить інженера від ручного написання SQL коду, йому доведеться тільки заповнити поля даними із заявки співробітника, та передати код далі на керівника та адміністратора БД.

Як ми бачимо, перший варіант більш швидкий та вигідніший, що спрощує життя всім учасникам процесу, але найголовніше що в обох варіантах вдається вирішити проблему з найбільш громіздкою частиною роботи, а саме написання коду вручну. Це в будь якому випадку зменшить час, особливо враховуючи той факт, що інженери часто допускають помилки, що змушує перероблювати код.

1 АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ ГЕНЕРАЦІЇ SQL КОДУ

На сьогодні існує багато засобів які пропонують функціонал для адміністрування БД. Нас цікавлять засоби, що зможуть будь яким методом прискорити написання SQL коду для додавання параметрів клієнта в базу даних. Для прикладу розглянемо наступні інструменти:

– phpMyAdmin

– dbForge Studio for SQL Server

– Valentina Studio

– HeidiSQL

1.1 **PhpMyAdmin** досить поширений веб-додаток відкритим кодом, та написаний на мові програмування PHP для адміністрування СУБД MySQL. PhpMyAdmin дозволяє через браузер і не тільки здійснювати адміністрування сервера MySQL, запускати команди SQL та переглядати вміст таблиць та баз даних. Програма користується великою популярністю у веб-розробників, тому що дозволяє управляти СУБД MySQL без безпосереднього введення SQL команд.

Додаток поширюється під ліцензією GNU General Public License і тому багато інших розробників інтегрують його у свої розробки, наприклад XAMPP , Denwer , AppServ , Open Server .

PhpMyAdmin може підтримувати наступні можливості:

– створювати, переглядати, редагувати та видаляти бази даних, таблиці, стовпці та індекси

– виконувати, редагувати і робити закладки будь якого SQL-оператора, навіть пакетних запитів

– експорт даних в різні формати: CSV, XML, PDF, ISO/IEC 26300 - OpenDocument текст і електронна таблиця, Microsoft Word 2000, і LATEX формати

– імпорт даних і MySQL структури з OpenDocument електронних таблиць, як і XML, CSV, і SQL файлів

– додавати, редагувати та видаляти облікові записи та права користувачів

– підтримка таблиць InnoDB та зовнішніх ключів

– створитювати PDF графічний макет бази даних

– глобальний пошук в базі даних чи її підмножинах

– адміністрування декількох серверів

Переваги:

• оптимальний для повсякденних завдань – набір функціоналу;

• працює на стороні сервера (через браузер).

Недоліки:

• тонка настройка вимагає знань та вмінь;

• не надає такого ж повного набору можливостей, як конкуренти

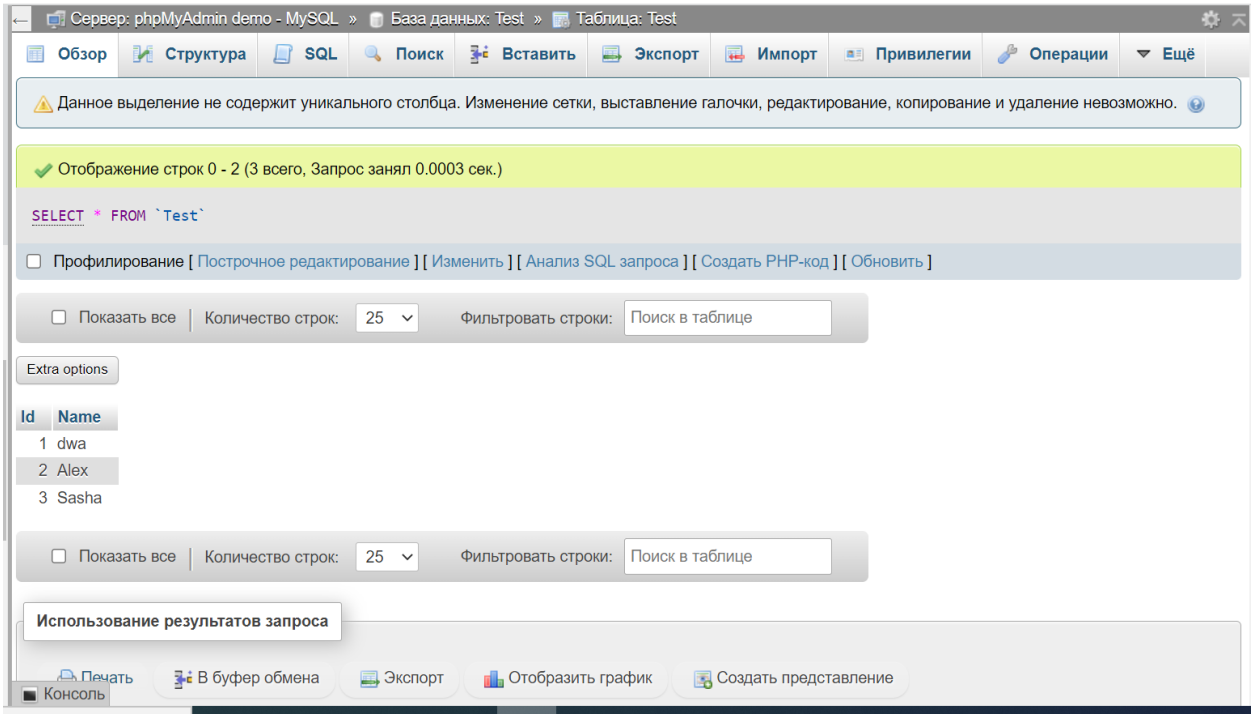


Рис. 1.3 - Панель керування PhpMyAdmin

1.2 **dbForge Studio for SQL Server** це середовище розробки баз даних, для формування звітів за даними, їх аналізу та виконання основних завдань адміністрування. dbForge Studio прискорює вирішення повсякденних завдань та дозволяє вносити комплексні зміни до баз даних. Можливості dbForge Studio for SQL Server:

– Прискорення написання SQL-коду у зручному середовищі створення сценаріїв.

– Генерація та повторне створення таблиць без втрати даних.

– Порівняння баз даних, синхронізація схем та даних.

– Аналіз взаємозалежностей об'єктів за зміни баз даних зі складною структурою.

– Автоматичне розгортання бази даних на робочому сервері.

– Підготовка звітів та автоматизація їх розсилки.

– Швидке та ефективне управління безпекою в базах даних.

Існують види редакцій Standard, Data та Professional. Версія Standard включає компонент налагодження T-SQL, профільник запитів, функції допомоги написання SQL-коду і опції пошуку по базах даних. До складу Standard входять покращені інструменти побудови запитів та експорту/імпорту даних. Версія Data (раніше Data Studio for SQL Server) є спеціальним набором інструментів для перегляду, редагування та аналізу даних, генерації звітів та автоматизації їх доставки. Версія Professional є найбільш повнофункціональною, додаючи можливості синхронізації БД, звітності, побудови зведених таблиць. Підтримує інтерфейс командного рядка для ряду компонентів, що дозволяють автоматизувати операції, що повторюються.

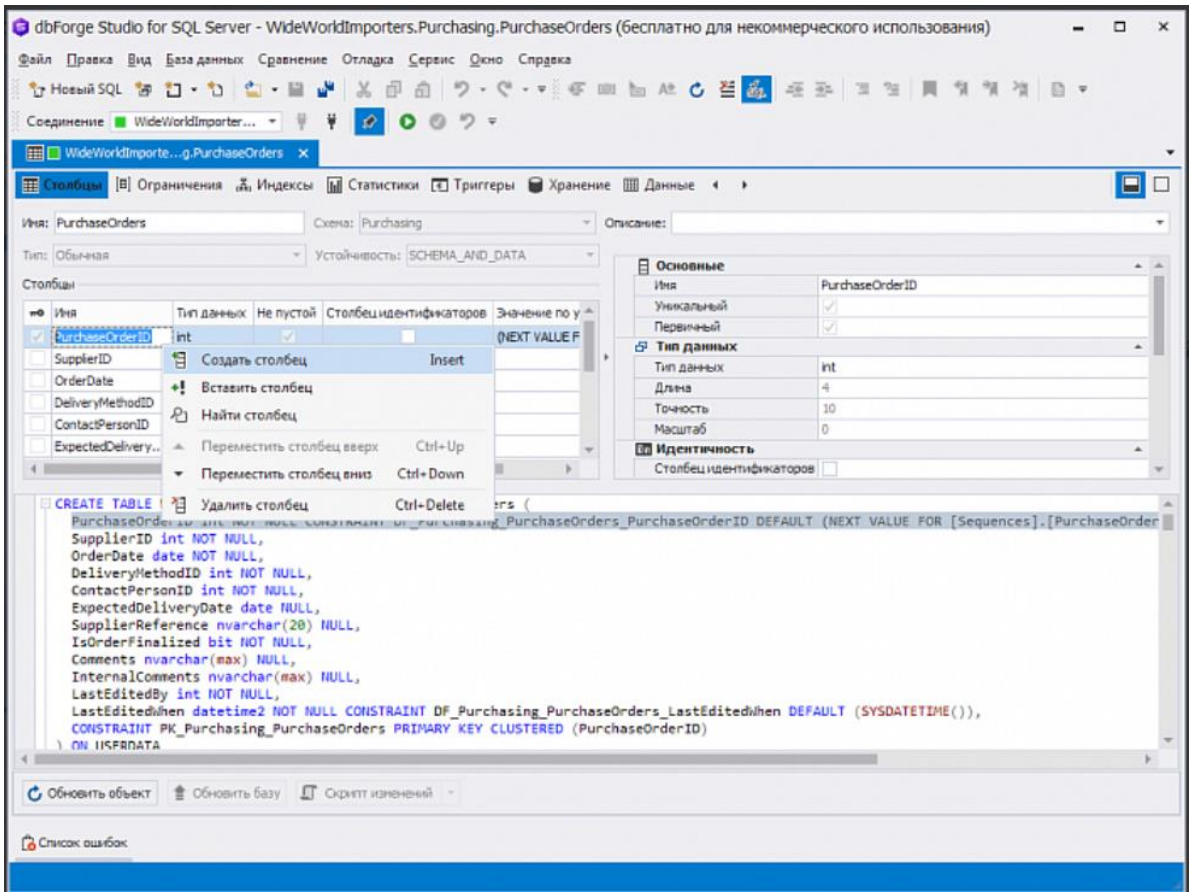


Рис. 1.4 - Інтерфейс dbForge Studio for SQL Server

1.3 **Valentina Studio** – це універсальний інструмент для архітекторів баз даних,

розробників та адміністраторів, що працюють з найбільш популярними СУБД, включаючи бази даних Valentina DB, MySQL, PostgreSQL, SQLite та MS SQL Server.

Він забезпечує візуальне моделювання даних за допомогою діаграм, розробку схеми бази даних, управління записами, розробку та запити SQL, адміністрування сервера тощо.

Безкоштовна версія надає потужні візуальні редактори, включаючи:

– Редактор діаграм – створення нової бази даних, потім ганерація таблиці та інші об'єкти бази даних з цієї діаграми.

– Редактор схеми – створення/зміна об'єктів схеми: таблиці, види, поля, перерахування, посилання, обмеження, тригери, індекси, збережені процеси.

– Редактор даних – легко переглядати записи таблиці у сітці, змінювати записи, є вбудоване редагування. Легко сортувати, фільтрувати записи та зберігати вибрані фільтри. Попередній перегляд зображень.

– Редактор пов'язаних даних – легко вивчати та керувати пов'язаними записами двох таблиць, з'єднувати/відкріплювати записи одним клацанням миші, виконувати операції над пов'язаними записами.

– Редактор SQL – з кольоровим синтаксисом, автозаповненням, збереженими обраними запитами, шаблонами, консоллю зі звітами про помилки/попередження.

– Server Admin – де можна керувати користувачами, вивчати журнали та інші параметри серверів.

PRO-версія додає більше інструментів:

– Valentina Project – збереження звітів, анкет, javascripts

– Редактор форм – дизайн форм

– Редактор звітів – дизайн звітів

– Query Builder – створення SQL-запитів всього кількома клацаннями миші

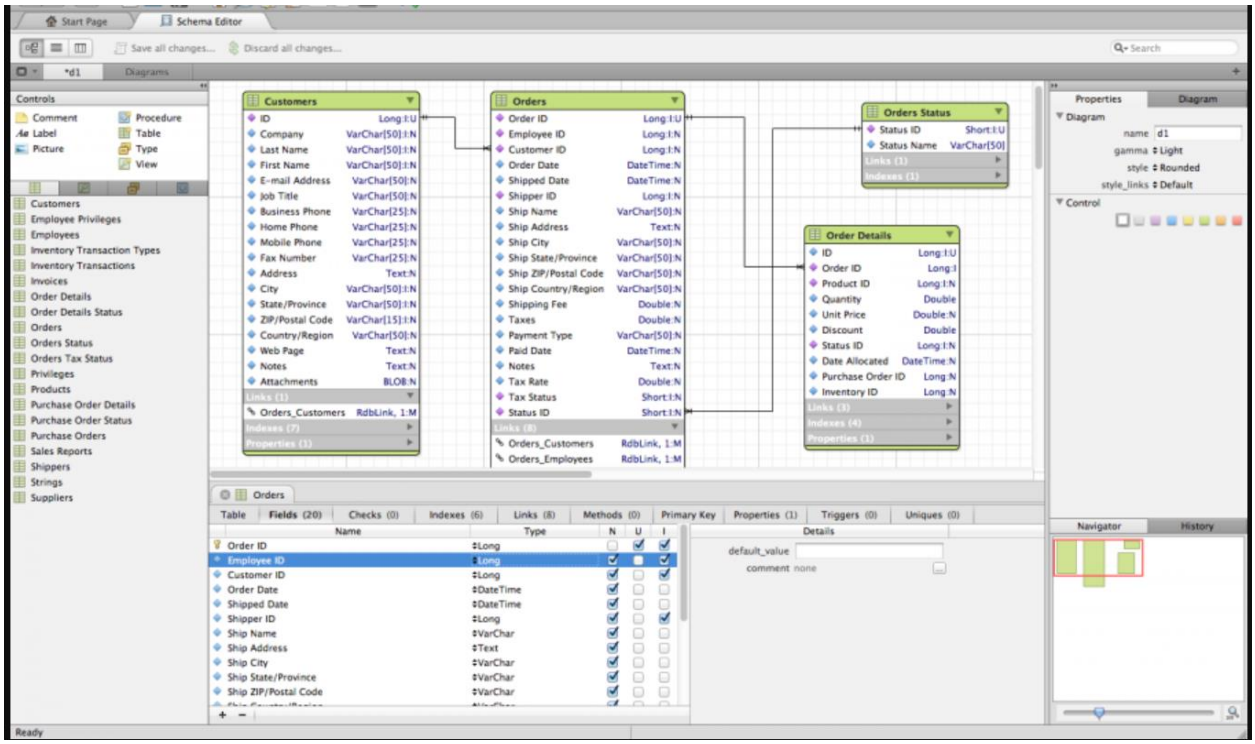


Рис. 1.5 - Редактор діаграм Valentina Studio

1.4 **HeidiSQL** - це програма управління базами даних MySQL і Microsoft SQL.

За допомогою цього інструмента користувач зможе переглядати та редагувати дані, створювати та редагувати таблиці, процедури, тригери та заплановані події. Структуру бази та самі дані можна експортувати в SQL файли, буфер обміну або інші сервери.

Ключові особливості та функції:

– програма безкоштовна – OpenSource;

– з'єднується з кількома серверами в одному вікні;

– можна з'єднуватися із сервером за допомогою командного рядка;

– SSH шифрування з'єднання;

– прямий експорт структури даних та самих даних з одного сервера на інший;

– встановлення прав користувачів;

– імпорт текстових файлів;

– експорт таблиць у формати CSV, HTML, XML, SQL, LaTeX, Wiki

Markup та PHP Array;

– перегляд та редагування табличних даних у зручній графічній оболонці;

– пакетний імпорт ASCII та бінарних файлів у таблиці;

– написання запитів з функціями підсвічування синтаксису та автозавершення рядка коду;

– пошук певного тексту у всіх таблицях, всіх баз даних одному сервері;

– оптимізація та відновлення таблиць у пакетному режимі;

– запуск паралельного вікна командного рядка mysql.exe з використанням поточних налаштувань з'єднання

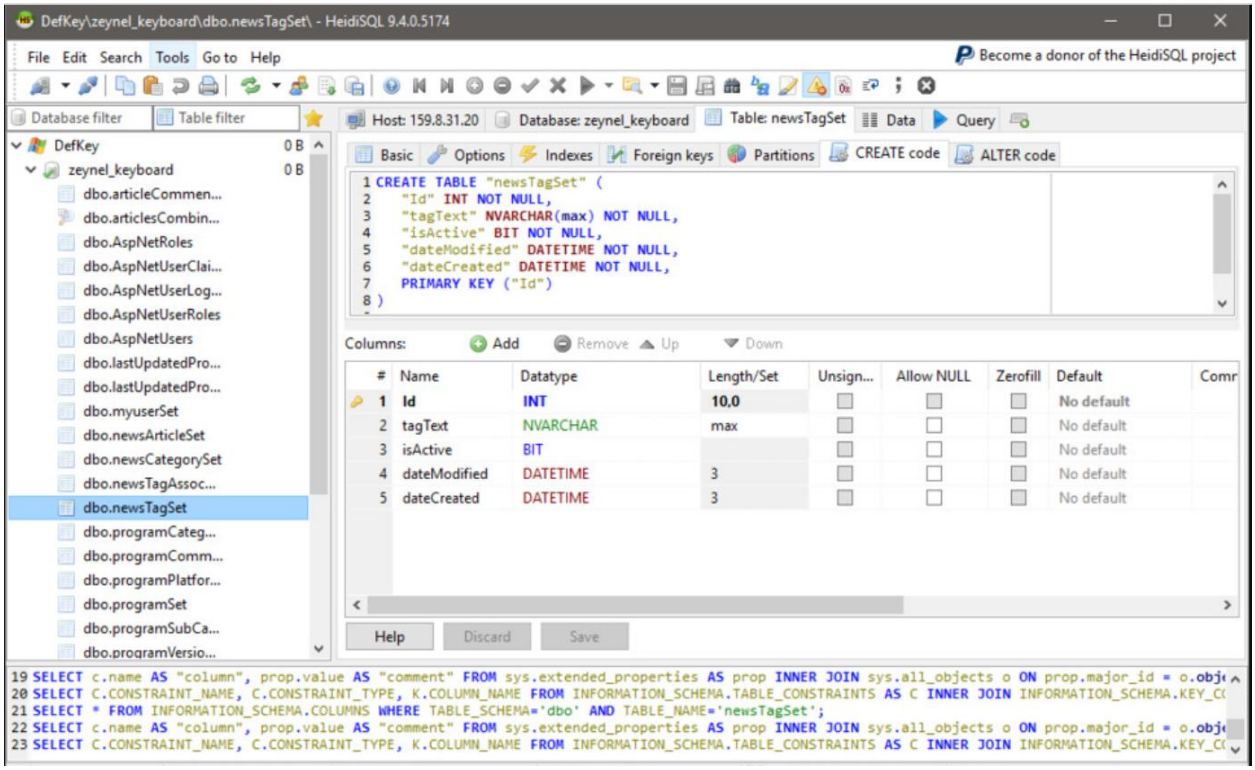


Рис. 1.6 - Перегляд табличних даних HeidiSQL

Теоретично кожен з вище наведених інструментів своїм функціоналом може допомогти у пришвидшенні виконання завдання. Наприклад PhpMyAdmin

після побудови точної копії тестової таблиці допоможе генерувати SQL код для додавання в неї даних. Але проаналізувавши користувацький інтерфейс всіх

інструментів, можна зробити висновки що він у всіх інтуітивно відрізняється та доволі складний, що створює дискомфорт для недосвітченого фахівця. Саме для вирішення цих проблем вирішено розробити свою систему, яка має хоч і набагато менше, зате найнеобхідніші інструменти, та має приємний та зрозумілий інтерфейс для фахівця будь-якого рівня

Таблиця 1.1 - Порівняння існуючих інструментів

	PhpMyAdmin	dbForge Studio for SQL Server	Valentina Studio	HeidiSQL
Вартість ПО	–	+	– +	–
Відкритий код	+	–	–	+
Імпорт\Експорт даних	+	+	+	+
Зручність інтерфейсу користувача	+	–	–	–
Моделювання діаграм	–	+	+	–
Генерація SQL коду	+	– +	+	–
Адміністрування користувачів	+	–	+	+

2 ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ ДЛЯ ГЕНЕРАЦІЇ SQL КОДУ

2.1 Розробка SPA(single page application) з використанням REST API

Для реалізації додатку було вирішено використати стиль односторінкового додатку. Односторінкові програми (SPA, Single-Page Application) — це веб-додатки, що складаються з однієї HTML-сторінки. Вони підключаються до сервера лише один раз, а потім просто динамічно підвантажують та оновлюють дані. Ключові елементи інтерфейсу сторінки залишаються незмінними, оновлюються лише блоки, які використовує користувач. Головна перевага односторінкових програм : не потрібно перезавантажувати всю сторінку, щоб оновити контент. Це дозволяє збільшити швидкість завантаження та покращити досвід взаємодії з продуктом (UX). Зараз все більше компаній обирає односторінкові додатки через їх швидкість роботи та терміни розробки, для прикладу Gmail,Facebook та Netflix це також SPA.

Основними елементами, що використовуються при побудові SPA є:

- Фреймворки для JavaScript, зокрема MVC та MVVM –фреймворки;
- Роутінг: навігація між представленням, проводиться у фронтенді;
- Шаблонізатор;
- HTML5;
- API для бекенда , наприклад, у стилі REST;
- Ajax

Потрібно розуміти, що односторінкові програми (SPA) є досить актуальною темою сьогодні. Існує багато фреймворків MVC(Model-view-controller), таких як AngularJS, React JS, Knockout JS, тощо, щоб швидко розробляти веб-інтерфейси, але суть кожної з них досить проста, усі фреймворки MVC допомагають нам реалізувати один шаблон проектування. Цей шаблон не робить запиту веб-сторінки, лише REST API. REST є архітектурним стилем для побудови API, який має наступні переваги:

- надійність (за рахунок відсутності необхідності зберігати інформацію про стан клієнта, яка може бути втрачена);
- Продуктивність (за рахунок використання кешу);
- Масштабованість;
- Прозорість системи взаємодії (особливо необхідна для додатків обслуговування мережі);
- Простота інтерфейсів ;
- Портативність компонентів;
- Легкість внесення змін;
- Здатність еволюціонувати , пристосовуючись до нових вимог

SPA має як переваги так і недоліки, один з яких це оптимізація SEO. Але проблему індексації пошуковими системами також можна вирішити за допомогою Server Side Rendering.SSR (Server Side Rendering, серверний рендеринг) - спосіб рендерингу односторінкового додатка на стороні сервера, коли в браузер користувача надсилається вже повністю відмальована сторінка.

Вже давно стало зрозуміло, що швидкість дохід. Дослідження показали, що більше половини всього веб-трафіку припадало на мобільні пристрої, але конверсія на них була нижчою, ніж з десктопів. Статистика говорить, що час завантаження сторінки безпосередньо пов'язаний з відсотком відмов (bounce rate) , коли користувачі йдуть без покупки. Наприклад, якщо час завантаження збільшується з 1 до 5 секунд, то відмови збільшуються на 90%. А якщо з 1 до 10 секунд, то шанси втратити клієнта збільшуються на 123%. Односторінкові програми - це зручне рішення для розробки MVP. Воно дозволяє створити простий і чуйний інтерфейс, який швидко завантажуватиметься, при цьому не потрібно буде витрачати мільйони і роки на розробку рішення.

2.2 Порівняння та вибір засобів для розробки веб-додатку

2.2.1 Вибір засобів для серверної частини

Для розробки серверної частини було вирішено обрати платформу NodeJS. Як і JavaScript, NodeJS запускається в V8, це середовище що переводить написаний код розробником в машинний код. Таким чином низькорівневий код стає швидким та зручним для розпізнавання ПК. Основними перевагами NodeJs є:

- модель вводу-виводу що не блокує(допомагає впоратися з навантаженням);
- можливість застосовувати одну мову на клієнті та сервері;
- швидкість;
- синтаксис javascript;
- технологія швидко покращується;
- велике ком'юніті;

Завдяки цьому інструменту можна створити як веб-додаток, так і програму для Linux, OS X або Windows. Але NodeJS найдоцільніше використовувати для веб-додатків, які потребують насиченого обміну даними з користувачами, тому що основні переваги відносяться саме цієї області. Разом з NodeJS ми будемо викоистовувати фреймворк Express.js — програмний каркас розробки веб-застосунків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроектований для створення веб-застосунків і API.

2.2.2 Вибір засобів для клієнтської частини

NodeJS та Express входять до популярного комплексу програмного забезпечення для веб-розробки MEAN(MongoDB, Express, Angular, NodeJS), тому для розробки клієнтської частини вирішено обрати фреймворк Angular. Плюсом MEAN є те, що він використовує в зв’язці ряд найкращих в своїй ніші веб-технологій. Та основну роль відіграє використання спільної мови програмування

JavaScript. Angular —фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону Модель-вид-контролер (MVC), а також спрощення їх тестування та розробки.

Для зберігання даних будемо використовувати NoSQL базу даних MongoDB. MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів та багатьох інших речей, які притаманні об'єктно-реляційним базам даних. На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому працює швидше, має кращу масштабованість, її легше використовувати. Також MongoDB підтримує зберігання документів в JSON-подібному форматі та має досить гнучку мову для формування запитів, Для підключення та використання MongoDB будемо використовувати Mongoose, спеціальну ODM-бібліотеку (Object Data Modelling), яка дозволяє зіставляти об'єкти класів та документи колекцій із бази даних.

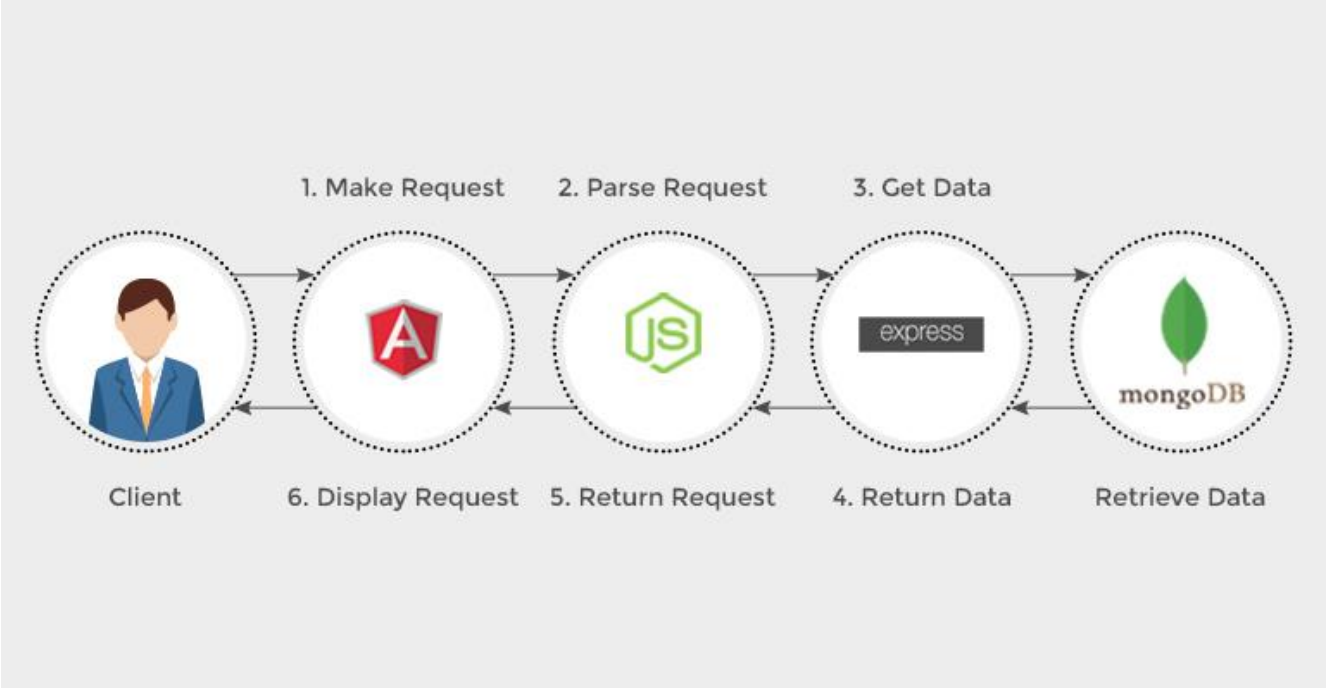


Рис. 2.1 - Стек MEAN

Отже серверна частина буде розроблена на платформі NodeJS та фреймворку Express , а клієнтська за допомогою фреймворку Angular з використанням MongoDB.

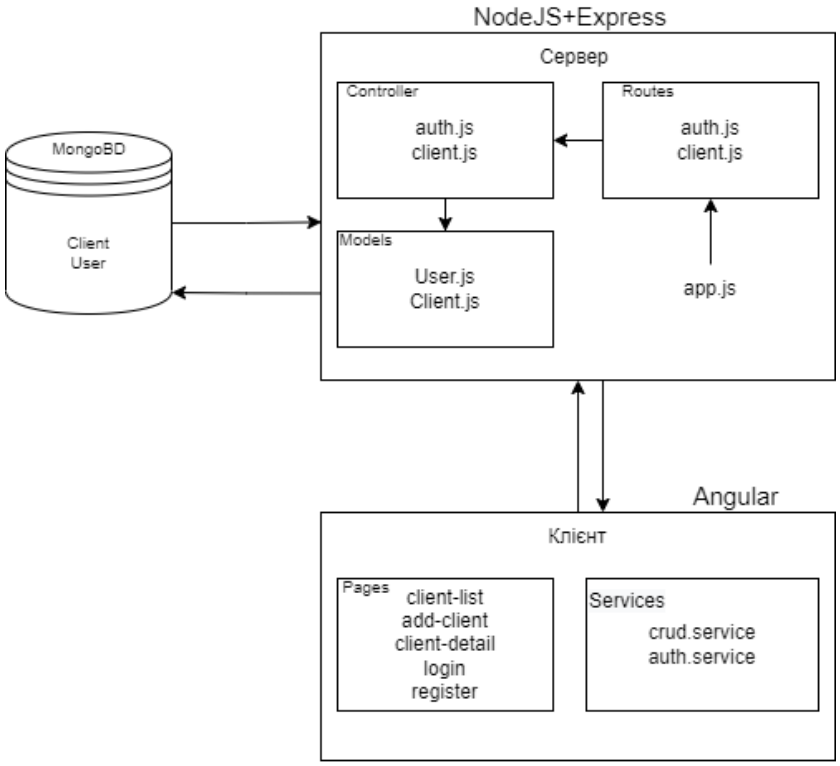


Рис. 2.2 - Архітектура додатку

3 МОДЕЛЮВАННЯ СИСТЕМИ

3.1 Діаграма прецедентів

Оскільки ПЗ має перелік функцій описаних вище, які викликаються та запускаються користувачем, щоб показати загальні межі предметної області, загальних вимог до функціональної поведінки та взаємодії ПЗ з користувачем використаємо графічний спосіб специфікування вимог – діаграму прецедентів UML(Use Case Diagram). Мета встановлення вимог у тому, щоб надати розгорнуте визначення вимог, які учасники проекту мають отримати у системі, яка буде реалізовуватися

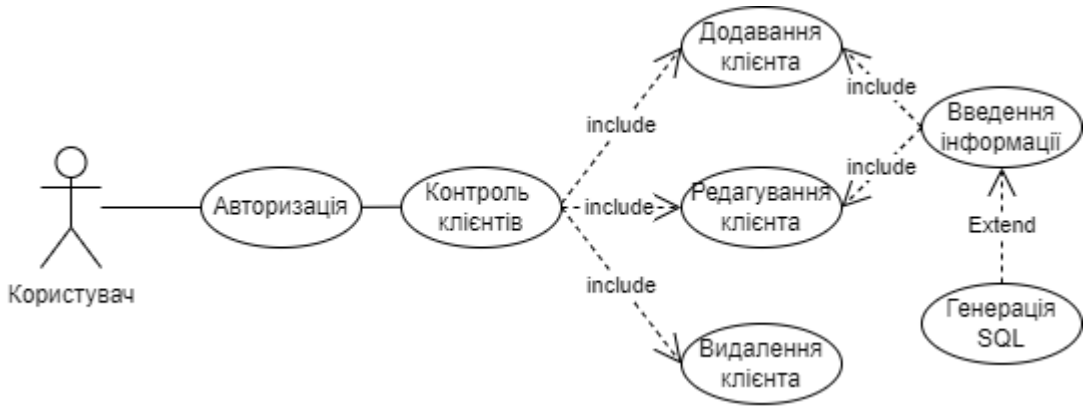


Рис. 3.1 Діаграма прецедентів

3.1.1 Вербальні специфікації прецедентів

Опишемо усі основні прецеденти за допомогою документально зафіксованого потоку подій.

Специфікація прецеденту: “Додавання клієнта”

1. Короткий опис

Оператор додає клієнта у таблицю клієнтів. Система приймає введену інформацію про товар та додає його у БД та таблицю.

2. Суб’єкт – Оператор

3. Передумова

Оператор натискає кнопку додавання клієнта.

4. Основний потік

4.1 Система відображує форму контролю клієнта. Форма має наступні елементи контролю:

4.1.1 Текстові поля для введення інформації про клієнта

4.1.2 Кнопка Додавання клієнта

4.1.3 Кнопка Генерація SQL коду

4.2 Якщо оператор натискає кнопку Додавання, інформація про клієнта відправляється у БД.

4.2.1 Якщо оператор не заповнив поля виконується А1.

4.3 Якщо оператор натискає кнопку Генерація SQL, інформація про клієнта підставляється в шаблон SQL коду для внесення в релятивну БД.

5. Альтернативні потоки

А1 Поле не може бути пустим. Якщо оператор не заповнив всі поля з’явиться повідомлення з помилкою. Прецедент продовується

6. Постумови

Створюється новий клієнт, який вноситься у БД.

Оновлюється таблиця клієнтів.

Специфікація прецеденту: “Видалення клієнта”

1. Короткий опис

Оператор обирає клієнта у таблиці клієнтів. Система приймає

ID обраного клієнта та видаляє його з БД та таблиці.

Суб’єкт – Оператор

2. Передумова

Оператор натискає кнопку видалення клієнта.

3. Основний потік

3.1 Система відображує список клієнтів в БД . Форма має наступні елементи контролю.

3.1.1 Кнопка Редагування клієнта

3.1.2 Кнопка Видалення клієнта

3.1.3 Таблиця клієнтів

3.2 Якщо оператор натискає кнопку Видалення, посилається запит на видалення клієнта з БД та таблиці.

4. Альтернативні потоки

Немає альтернативних потоків так як у прецеденті виключено помилки користувача.

5. Постумови

Обраний клієнт видаляється з БД та таблиці. Оновлюється таблиця клієнтів.

Специфікація прецеденту: “Редагування товару ”

1. Короткий опис

Оператор обирає клієнта у таблиці клієнтів. Система приймає ID обраного клієнта та відкриває форму для введення нових данних.

2. Суб’єкт – Оператор

3. Передумова

Оператор натискає кнопку Редагування та вводить інформацію про клієнта.

4. Основний потік

4.1 Система відображує форму для введення даних про клієнта. Форма має наступні елементи контролю.

4.1.1 Текстові поля для введення інформації

4.1.2 Кнопка Редагування клієнта

4.1.3 Кнопка Генерація SQL коду

4.2 Якщо оператор натискає кнопку Редагування, інформація про клієнта змінюється у БД.

4.2.1 Якщо оператор не заповнив поля виконується А1.

4.3 Якщо оператор натискає кнопку Генерація SQL, інформація про клієнта підставляється в шаблон SQL коду для внесення в релятивну БД.

5. Альтернативні потоки

А1 Поле не може бути пустим. Якщо оператор не заповнив всі поля з’явиться повідомлення з помилкою. Прецедент продовується

6. Постумови

Оновлюється клієнт у БД.

Оновлюється таблиця клієнтів.

3.2 Діаграма діяльності

Діаграма прецедентів дає «вигляд зверху» на функціональність системи, діаграма діяльності UML, навпаки, дозволяє детально ілюструвати окремий варіант використання і його сценарії.

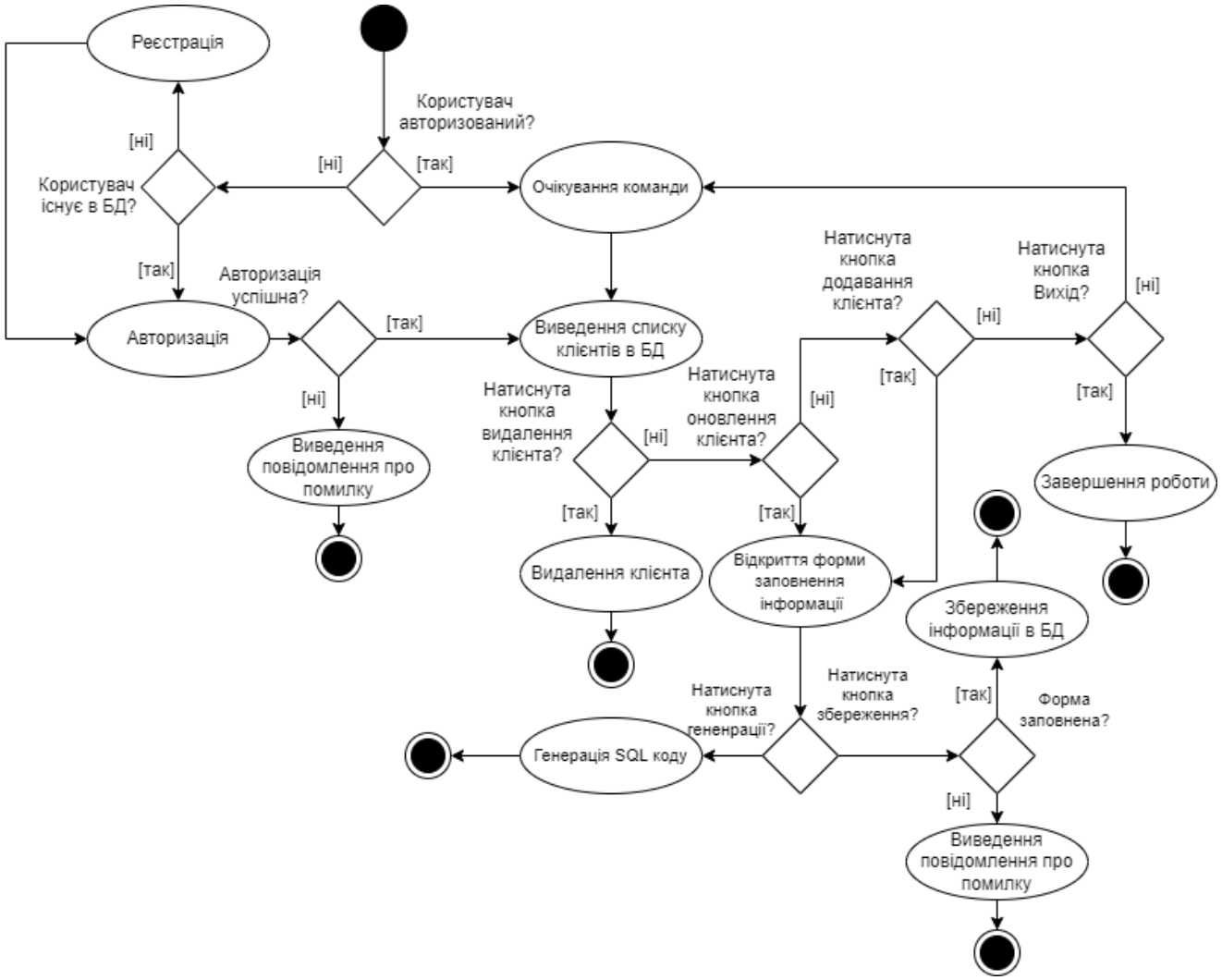


Рис. 3.2 - Діаграма діяльності

3.2.1 Специфікація діаграми діяльності

Таблиця 3.1 - Специфікація діаграми діяльності

Формулювання прецеденту	**Стан виду діяльності**
Очікування команди	Передбачає очікування натискань кнопок інтерфейсу користувачем. Кожна нажата кнопка обробляється та запускає певну команду.
Авторизація	Передбачає введення даних користувача та перевірка чи існує користувач в БД
Реєстрація	Передбачає створення аккаунта для оператора та додавання його в БД
Виведення повідомлення про помилку.	Передбачає виведення повідомлення про помилку.
Видалення клієнта	Передбачає видалення запису про клієнта, що містить інформацію про клієнта з БД.
Виведення списку клієнтів в БД	Передбачає виведення списку клієнтів у таблиці.
Генерація SQL коду	Передбачає додавання даних клієнта в шаблон SQL та виведення коду
Збереження інформації у БД	Передбачає додавання запису про клієнта, що містить: id , та інформацію про клієнта у БД.
Завершення роботи	Передбачає запуск процедури завершення роботи яка включає в себе перехід на сторінку авторизації

3.3 Діаграма потоків даних

Оскільки програмне забезпечення умовно поділене на модулі, зручним способом представлення потоків даних, що переміщуються, є діаграма потоків даних:

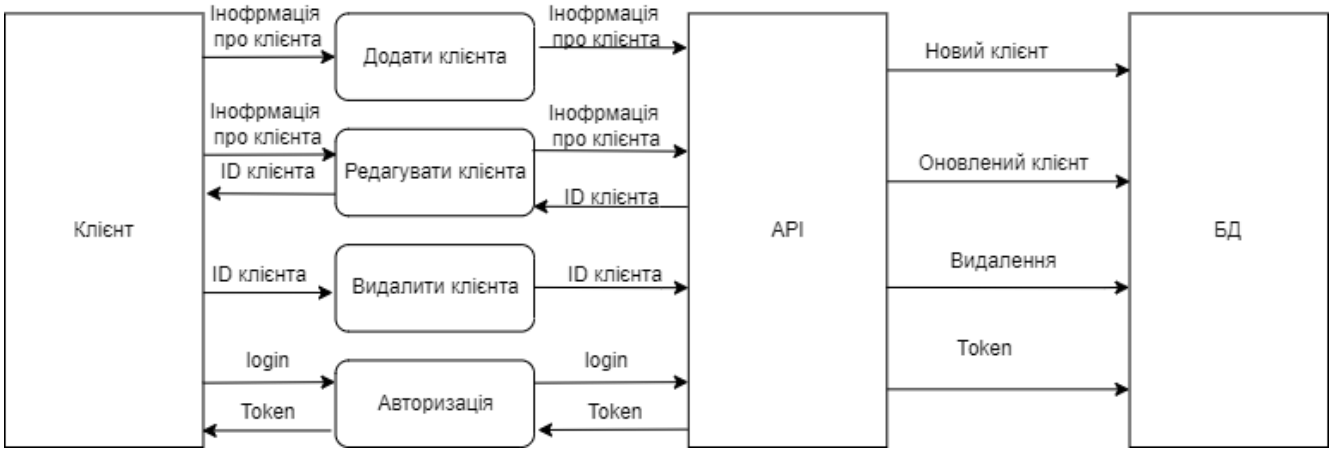


Рис. 3.3 - Діаграма потоків даних

4 РОЗРОБКА СИСТЕМИ ДЛЯ ГЕНЕРАЦІЇ SQL КОДУ

4.1 Розробка серверної частини веб-додатку

Серверна частина являється мозком нашого додатку. Вона відповідає за:

1 Підключення до БД;

2 Створення користувача;

3 Авторизація користувача;

4 Маршрутизацію запитів;

5 CRUD операції з клієнтом;

6 Захист роутів.

Тож розглянемо реалізацію кожного пункту.

4.1.1 Підключення до бд

Підключення до БД виконується у головному файлі додатку app.js за допомогою методу «mongoose.connect()». Mongoose представляє спеціальну ODM-бібліотеку (Object Data Modelling) для роботи з MongoDB, яка дозволяє зіставляти об'єкти класів та документи колекцій із бази даних.

```
mongoose.connect(keys.mongoURI, options: {useNewUrlParser: true, useUnifiedTopology: true })

const db = mongoose.connection;
db.on( eventName: 'error', console.error.bind(console, argArray: 'connection error:'))
db.once( eventName: 'open', listener: function() {
    console.log(`MongoDB connected!`)
})
```

Рис. 4.1 - Підключення до MongoDB

Змінна “keys.mongoURI” представляє строку підключення до БД, що знаходиться у файлі keys.js. Після успішного підключення отримаємо повідомлення в консолі «MongoDB connected», у разі виникнення помилок вони також з’являться у консолі.

Дані, що використовуються у Mongoose, описуються певною схемою. Схема містить метадані об'єктів. Зокрема, тут встановлюємо, які властивості матиме об'єкт і який буде тип даних. Схеми описуються окремо для кожного об’єкту в папці Models.

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const clientSchema= new Schema({
    acqid: {
        type: Number,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    caid: {
        type: Number,
        required: true
    },
    cata: {
        type: String,
        required: true
    }
})

module.exports = mongoose.model( name: 'clients', clientSchema)
```

Рис. 4.2 - Модель клієнта

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const userSchema = new Schema({
    email: {type: String, required: true, unique: true},
    password: {type: String, required: true},

})

module.exports = mongoose.model( name: 'users', userSchema)
```

Рис. 4.3 - Модель користувача

4.1.2 Створення користувача

Для того щоб пройти авторизацію в системі, необхідно ввести email та пароль користувача який існує в БД. Якщо користувач виконує вхід вперше, йому необхідно створити обліковий запис. Для цього він переходить на сторінку реєстрації, вводить свої дані, і відправляє запит на сервер.

```
router.post( path: '/login',controller.login)
router.post( path: '/register',controller.register)

module.exports=router
```

Рис. 4.4 - Файл /routes/auth.js

При авторизації існує два маршрути, зі сторінки /login та /register . Так як користувач реєструється та відправив дані із сторінки /register, то йде звернення до файлу /controllers/auth.js, та методу register().

```
module.exports.register = async function (req, res) {
    const candidate = await User.findOne( filter: {email: req.body.email})

    if (candidate) {
        res.status(409).json({message: 'Такой email уже занят'})
    } else {
        const salt = bcrypt.genSaltSync( rounds: 10)
        const user = new User( doc: {
            email: req.body.email,
            password: bcrypt.hashSync(req.body.password, salt)
        })

        try {
            await user.save()
            res.status(201).json(user)
        } catch(e) {
            error(res, e)
        }
    }
}
```

Рис. 4.5 - Метод register()

Даний метод перевіряє чи не існує такого користувача в БД, генерує хеш, для того щоб не зберігати відкритий пароль в БД, та зберігає користувача.

4.1.3 Авторизація користувача

Після того як було створено користувача, з’являється можливість пройти авторизацію в системі. Для цього користувач переходить на сторінку авторизації та вводить дані які мають існувати в БД. Після відправлення запиту на сервер із сторінки /login, запит йде по маршруту до файлу /controllers/auth.js, та методу login().

```
module.exports.login = async function (req, res) {
    const candidate = await User.findOne( filter: {email: req.body.email})

    if (candidate) {
        const result = bcrypt.compareSync(req.body.password, candidate.password)
        if (result) {
            const token = jwt.sign( payload: {
                email: candidate.email,
                userId: candidate._id
            }, keys.jwt,  options: {expiresIn: 60 * 60})
            res.status(200).json({token: `Bearer ${token}`})
        } else {
            res.status(401).json({message: 'Пароль неверный'})
        }
    } else {
        res.status(404).json({message: 'Пользователь не найден'})
    }

}
```

Рис. 4.6 - Метод login()

Даний метод перевіряє чи існує користувач в БД, розшифровує хеш пароля та перевіряє його, та генерує токен, за допомогою якого користувачу стають доступні інші сторінки додатку. Токен дійсний протягом 1 години, потім потрібно проходити авторизацію заново.

4.1.4 Маршрутизація запитів

При зверненні до веб API використовуються запити HTTP. Ці запити можуть містити в собі наступні методи:

GET — використовується для отримання або читання даних. Запити з використанням цього методу можуть тільки отримувати дані;

PUT — використовується для оновлення ресурсу, шляхом заміни даних запиту;

POST — зазвичай використовується для створення нового ресурсу або використовується для відправки сутностей до певного ресурсу;

DELETE — видаляє дані;

PATCH — використовується для часткової зміни ресурсу.

В залежності від того яка саме дія виконується, формується маршрут запиту. Спочатку запит потрапляє в файл app.js де визначається з якої сторінки відправився запит, якщо це авторизація то запит прямує по маршруту //host:5000/api/auth/. Далі запит обробляє файл /routes/auth.js, де визначається який метод необхідно використати.

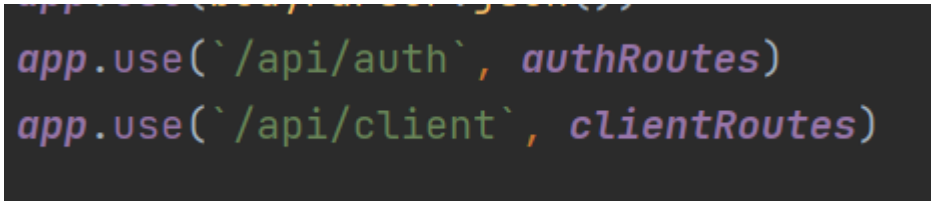


Рис. 4.7 - Підключення роутінгу до сторінки

Це нагадує дерево папок, де в кінцевому результаті викликається необхідний метод для обробки запиту. Для прикладу при авторизації користувача формується посилання //host:5000/api/auth/login , а для реєстрації //host:5000/api/auth/register. Як формується роутінг представлено на рисунку 15. Для представлення взаємодії компонентів між собою, було розроблено діаграму класів.

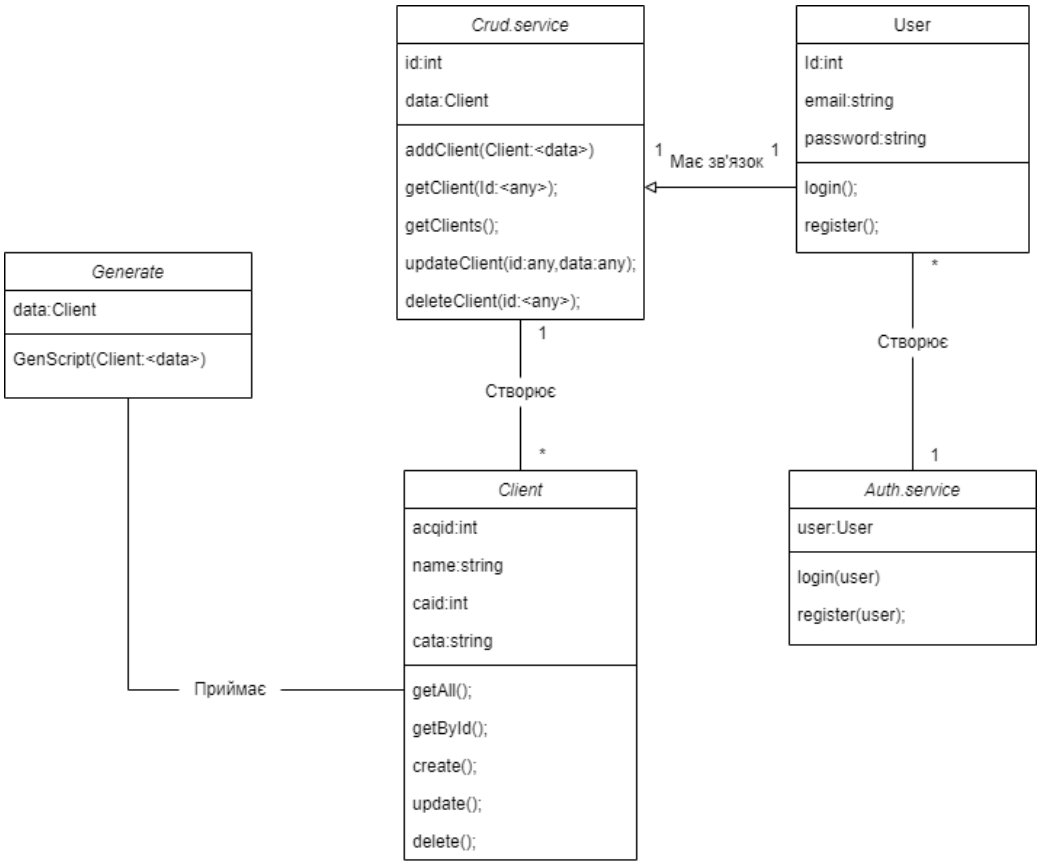


Рис. 4.8 - Діаграма класів

4.1.5 Crud операції з клієнтом

За аналогією авторизації, також формуються запити для управління клієнтами. Для того щоб здійснити якусь операцію, запит прямує за посиланням //host:5000/api/client, потім потрапляє до файлу /routes/client.js

```
router.get( path: '/',passport.authenticate( strategy: 'jwt', options: {session:false}),controller.getAll)
router.get( path: '/read-client/:id',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.getById)
router.post( path: '/add-client',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.create)
router.delete( path: '/delete-client/:id',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.remove)
router.put( path: '/update-client/:id',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.update)
module.exports=router
```

Рис. 4.9 - Файл /routes/client.js

На данному скриншоті, можна побачити співвідношення методів до посилань. Наприклад якщо користувач натискає кнопку «додати клієнта», яка має посилання //host:5000/api/client/add-client , то викликається метод controller.create, який бере значення для клієнта із форми та зберігає до БД.

```
module.exports.create = async function(req, res) {

    try {
        const client = await new Client( doc: {
        acqid:req.body.acqid,
        name: req.body.name,
        caid:req.body.caid,
        cata:req.body.cata
    }).save()
        res.status(201).json(client)
    } catch (e) {
        errorHandler(res, e)
    }
}
```

Рис. 4.10 - Метод створення клієнта create()

4.1.6 Захист роутів

В клієнт-серверних додатках для автентифікації та створення токенів доступу набув популярності відкритий стандарт RFC 7519, який заснований на форматі JSON. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи. Як правило, при використанні JSON-токенів у клієнт-серверних додатках реалізована така схема:

- Клієнт проходить автентифікацію в додатку (наприклад, з використанням логіну та паролю).
- У разі успішної автентифікації сервер надсилає клієнту access-і refresh-токени.

- У разі подальшого звернення до сервера клієнт використовує access-токен. Сервер перевіряє токен на валідність та надає клієнту доступ до ресурсів.
- У випадку, якщо access-токен стає невалідним, клієнт відправляє refresh-токен, у відповідь на який сервер надає два оновлені токени.
- Якщо refresh-токен стає невалідним, клієнт знову повинен пройти процес аутентифікації

JWT має ряд переваг у порівнянні з підходом зберігання виданих сесій на сервері та в cookie на стороні клієнта:

- При використанні JWT - не потрібно зберігати додаткові дані про видані сесії, все що повинен зробити сервер - це перевірити підпис.
- Сервер може не займатися створенням токенів, а надати це зовнішнім сервісам.
- У JSON-токенах можна зберігати додаткову корисну інформацію про користувачів. Як наслідок – більш висока продуктивність. У разі використання cookie іноді необхідно здійснювати запити для отримання додаткової інформації. З використанням JWT ця інформація може бути передана у самому токені.
- JWT робить можливим надання одночасного доступу до різних доменів та сервісів.

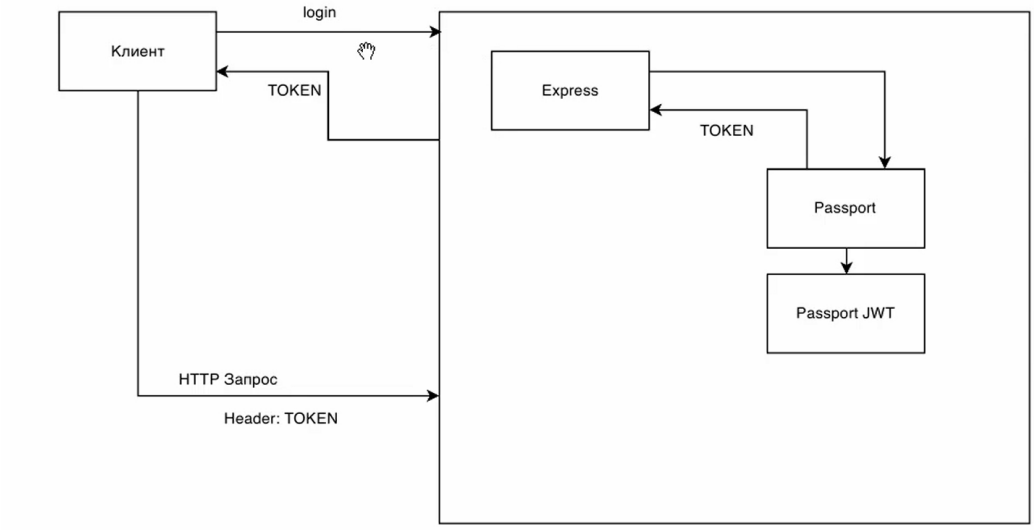


Рис. 4.11 - Модель авторизації

Для того щоб неавторизований користувач не мав доступу до всіх інших сторінок, крім авторизації та реєстрації, потрібно обмежити до них доступ.

Passport — чудовий приклад бібліотеки. У нашому додатку ми використовуємо модуль passport який використовує наступну стратегію авторизації:

```
module.exports = passport =>{
    passport.use(
        new JwtStrategy(options, verify: async (payload, done)=>{
            try {
                const user = await User.findById(payload.userId).select( arg: 'email id')
                if(user){
                    done(null, user)
                }else {
                    done(null,false)
                }
            }catch (e){
                console.log(e)
            }
        })
    )
```

Рис. 4.12 - Стретегія PassportJS

Як тільки findUser повертається з нашим об'єктом користувача, залишається лише порівняти введений користувачем та реальний пароль, щоб побачити, чи є збіг. Якщо вони збігаються, ми дозволяємо користувачеві увійти (повертаючи об'єкт done(null, user)), якщо ні - повертаємо помилку авторизації (шляхом повернення done(null, false)).

Роути які обмежені без авторизації користувача використовують метод passport.authenticate.

```
router.get( path: '/',passport.authenticate( strategy: 'jwt', options: {session:false}),controller.getAll)
router.get( path: '/read-client/:id',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.getById)
router.post( path: '/add-client',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.create)
router.delete( path: '/delete-client/:id',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.remove)
router.put( path: '/update-client/:id',passport.authenticate( strategy: 'jwt', options: {session:false}),
    controller.update)
module.exports=router
```

Рис. 4.13 - Захист роутів від не авторизованих користувачів

4.2 Розробка клієнтської частини веб-додатку

Клієнтська частина це на сам перед обличчя додатку, саме з нею доведеться працювати користувачу. Вона відповідає за представлення інтерфейсу та функціональних можливостей додатку. Розробка інтерфейсу здійснювалася за допомогою бібліотеки Materialize. Materialize – це бібліотека компонентів інтерфейсу користувача, створена з використанням CSS, JavaScript і HTML. Компоненти матеріалізації інтерфейсу допомагають створювати привабливі, узгоджені та функціональні веб-сторінки, в той же час дотримуючись сучасних принципів веб-дизайну, таких як підтримка браузера. Це допомагає у створенні більш швидких та красивих сайтів. Він базується на Google Material Design.

Так як у нас односторінковий веб-додаток, спочатку було розроблено дві основних частини представлення:

- auth-layouts – представлятиме сторінку авторизації та реєстрації;
- site-layouts – представлятиме всю іншу структуру додатку.

Наступним кроком була розробка компонентів додатку які будуть динамічно завантажуватися в auth-layouts та site-layouts, а саме:

- Login-page.component – частина сторінки для авторизації користувача;
- Register-page.component – частина сторінки для реєстрації користувача;
- Add-client.component – частина сторінки для додавання клієнта та генерації SQL коду;

- Client-list.component – частина сторінки для перегляду списку клієнтів, з кнопкою редагування та видалення;
- Client-detail.component – частина сторінки для редагування клієнта та генерації SQL коду;

```
const routes: Routes = [
  {path:'',component:AuthLayoutsComponent,children:[
      {path: '',redirectTo:'login',pathMatch:'full'},
      {path:'login',component:LoginPageComponent},
      {path:'register',component:RegisterPageComponent}

    ]},
  {path:'',component:SiteLayoutsComponent,canActivate:[AuthGuard],children:[
      {path: 'edit-client/:id',component: ClientDetailComponent},
      {path: 'client-list',component: ClientListComponent},
      {path: 'add-client',component: AddClientComponent}
    ]}
];
```

Рис. 4.14 - Роутінг між запитом та компонентами

4.2.1 Розробка сервісу для авторизації

Для того щоб провести авторизацію або реєстрацію на фронтенді веб-додатку, потрібно створити сервіс, який буде звертатися до API на серверній частині додатку. Для цього було розроблено сервіс auth.service.ts, який за допомогою метода POST, відправляє дані користувача для обробки за допомогою API.

```
export class AuthService {

  private token = null

  constructor(private http: HttpClient) {
  }

  login(user: User): Observable<{token: string}> {
    return this.http.post<{token: string}>( url: '/api/auth/login', user)
      .pipe(tap( next: ({token}) => {
        localStorage.setItem('auth-token', token)
        this.setToken(token)
      }))
  }

  register(user: User): Observable<User> {
    return this.http.post<User>( url: '/api/auth/register', user)
  }

  isAuthenticated(): boolean {
    return !!this.token
  }
```

Рис. 4.15 - Сервіс auth.service.ts

Цей сервіс буде викликатися в компонентах login-page.component.ts та register-page.component.ts для маршрутизації введених користувачем даних та обробки на сервері.

```
onSubmit() {
  this.form.disable()
  this.aSub = this.auth.register(this.form.value).subscribe(
    next: ()=> {
      this.router.navigate( commands: ['/login'], extras: {
        queryParams: {
          'registered': true
        }
      })
    },
    error: error => {
      console.warn(error)
      MaterialService.toast(error.error.message)
      this.form.enable()
    }
  )
}
```

Рис. 4.16 - Виклик сервісу в register-page.component.ts

```
onSubmit() {
  this.form.disable()
  this.aSub = this.auth.login(this.form.value).subscribe(
    next: () =>  this.router.navigate( commands: ['/add-client']),
    error: error => {
      MaterialService.toast(error.error.message)
      this.form.enable()
    }
  )
```

Рис. 4.17 - Виклик сервісу в login-page.component.ts

4.2.2 Розробка сервісу для crud операцій

Аналогічно з сервісом авторизації, потрібно розробити сервіс для створення, читання, оновлення та видалення клієнтів з БД. Цей сервіс буде також викликатися в компонентах, та буде відправляти дані введені користувачем для обробки на сервер. Він буде містити наступні методи:

- AddClient – для додавання клієнта;
- GetClient – для отримання клієнта;
- GetClients – для отримання всіх клієнтів;
- UpdateClient – для оновлення клієнта;
- DeleteClient – для видалення клієнта;
- HandleError – для обробки помилок.

Для прикладу розглянемо метод AddClient, який викликається в компоненті add-client.component.ts.

```
AddClient(data: Client): Observable<any> {
  let API_URL = `${this.REST_API}/add-client`;
  return this.httpClient.post(API_URL, data)
    .pipe(
      catchError(this.handleError)
    )
}
```

Рис. 4.18 - метод AddClient в сервісі crud.service.ts

```
onSubmit(): any {
  this.crudService.AddClient(this.clientForm.value)
    .subscribe( next: () => {
      console.log('Data added successfully!')
      this.ngZone.run( fn: () => this.router.navigateByUrl( url: '/client-list'))
    },  error: (err) => {
      console.log(err);
    });
}
```

Рис. 4.19 - Виклик методу AddClient в add-client.component.ts.

4.2.3Розробка генерації sql коду

Генерація SQL коду буде відбуватися методом підстановки введених користувачем даних в шаблон. Можливість згенерувати SQL код буде на компонентах add-client.component та client-detail.component. Біля кнопки додавання чи редагування клієнта, буде кнопка GenScript, яка буде викликати метод, що буде брати дані із заповнених форм, підставляти в шаблон, та виводити на екран.

```
generateScript() {

  this.script = ` INSERT
  INTO
  ITM22D.VMTACQSETT(
    ACQID,
    CAID,
    CATA,
    MERCHNAME
    )
  VALUES(
    ${this.acqid},
    '${this.caid}',
    '${this.cata}',
    '${this.name}'
    )
  ;`
```

Рис. 4.20 - Метод generateScript()

```
<p><div class="form-group">
  <div class="col-md-4">
  <button type=button (click)="generateScript()" id="btnGenScr" name="btnGenScr"
          class="btn btn-primary">Gen scr</button>
</div>
div><p>
```

Рис. 4.21 - Виклик методу натисканням на кнопку

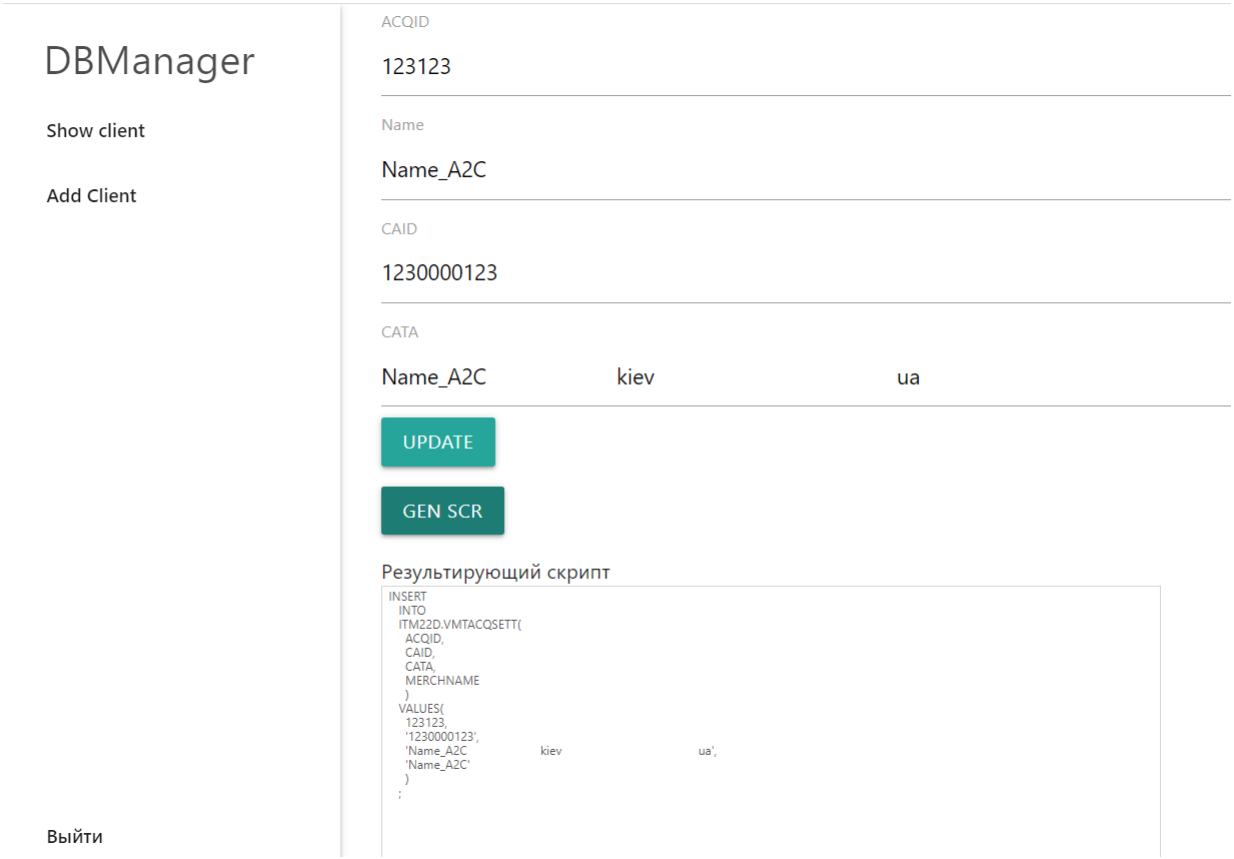


Рис. 4.22 - Виведення згенерованного SQL коду

4.3 Запуск додатку на віддаленому сервері

Для того щоб додаток могли використовувати інші користувачі, його потрібно завантажити на загальнодоступний сервер. В якості сервера було обрано хмарний сервіс Heroku, тому що він забезпечує швидке та ефективне створення,

розгортання та масштабування веб-додатків. Щоб запустити додаток на сервері, потрібно створити клон репозиторію проекту в папці додатку , провести індексацію змін командою «git add .» , потім виконати команду «git commit –m ‘’» для внесення проіндексованих змін, та «git push heroku main» для завантаження проекту на сервер. Далі сервер виконує команди для будування та запуску аплікації, які ми внесли в файл package.json: «ng build –prod» для клієнта, та «NPM_CONFIG_PRODUCTION=false npm run client-install && npm run build --prefix client» для сервера. Після запуску можна перевірити логи сервера, та перейти на сторінку додатку.

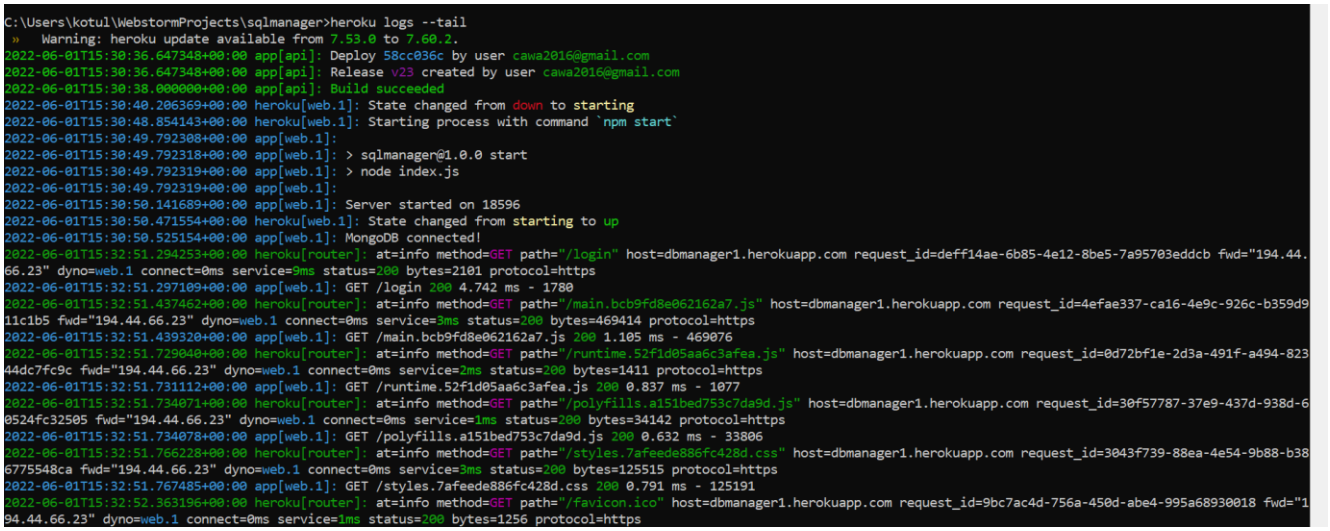


Рис. 4.23 - Старт проекту

4.4 Тестування розробленої системи

Оскільки додаток розроблявся власноруч та поступово, паралельно здійснювалося ad-hoc тестування. В більшості використовувався вид Monkey testing – довільне тестування продукту з метою якомога швидше, використовуючи різні варіації вхідних даних, щоб порушити роботу програми або викликати її зупинку. Таким чином було протестовано: авторизацію користувача, CRUD операції з клієнтом, роботу з БД, виведення помилок, генерацію SQL коду. Основні переваги ad-hoc testing:

- немає необхідності витрачати час на підготовку документації;
- найважливіші дефекти найчастіше виявляються на ранніх етапах;
- можливість знайти складно відтворювані та важковловимі дефекти, які неможливо було б знайти, використовуючи стандартні сценарії перевірок.

Також для уникнення помилок було розроблено валідацію форм на введення коректних даних, та перевірки.

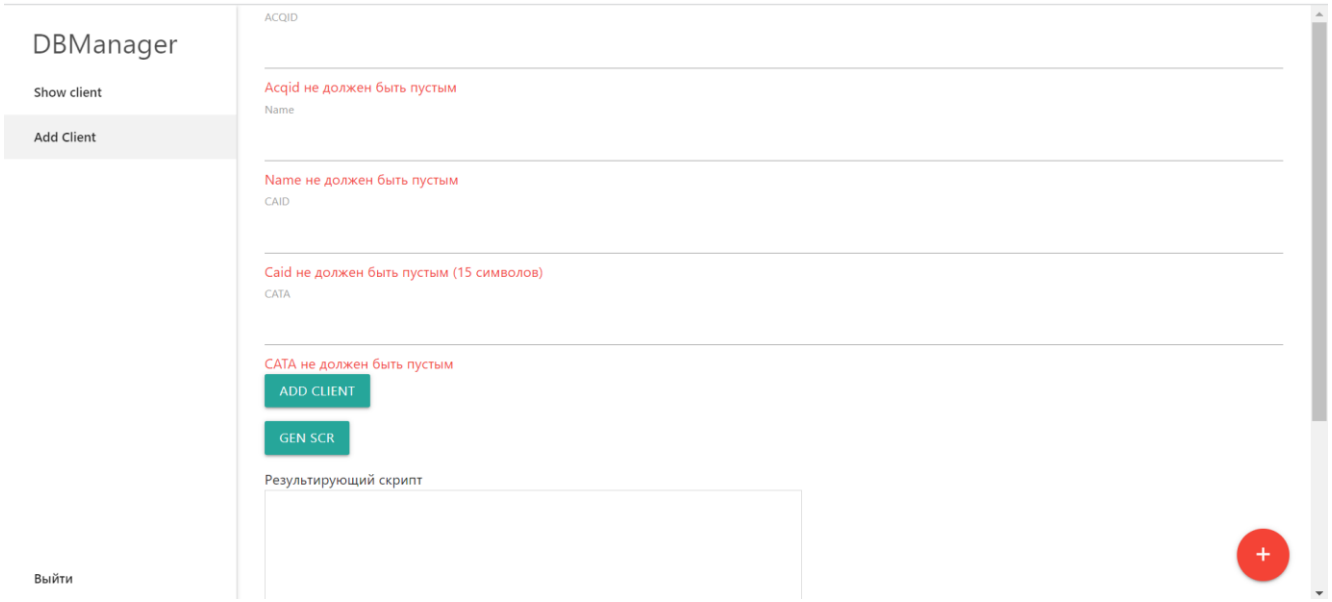


Рис. 4.24 - Обробка пустих форм

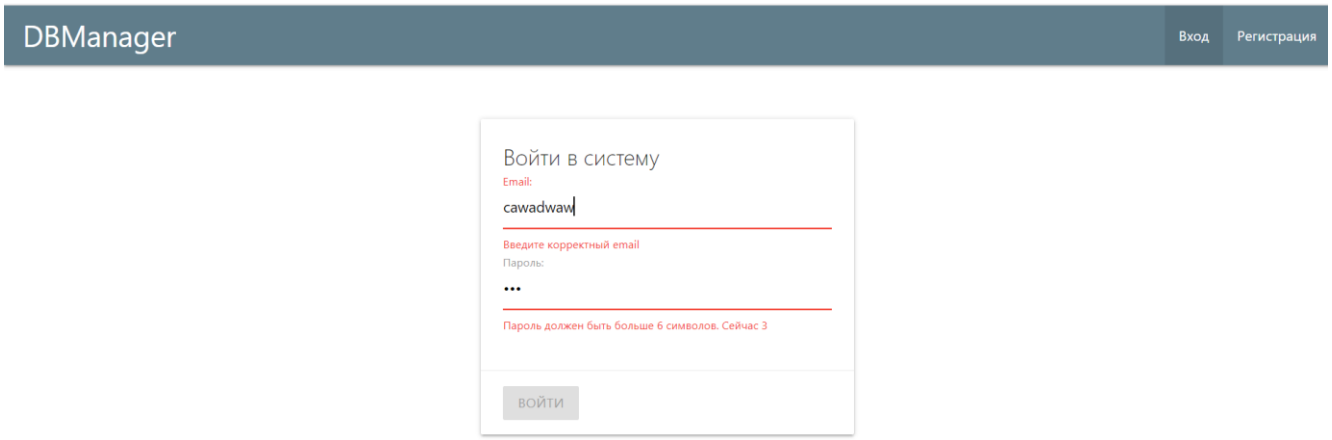


Рис. 4.25 - Обробка введених даних

ВИСНОВКИ

Під час аналізу та виконання індивідуального завдання, було досліждено ключові проблеми предметної області та реалізовано їх вирішення:

1. Проведено аналіз процесу додавання клієнта в БД. Ключові проблеми, які можна виділити, пов’язані з складністю бізнес процесу. Це призводить до збільшення часу на виконання завдання.

2. Визначено вимоги до системи за допомогою створення UML діаграм

3. Визначено та побудовано архітектуру додатку: серверну частину на платформі Node.JS за допомогою фреймворку Express, та клієнтську частину за допомогою фреймворку Angular.

4. Розроблено програмне забезпечення,. яке може бути використано для додавання клієнта в БД. Ключовими недоліками існуючих систем є складність інтерфейсу користувача, та відсутність необхідного функціоналу. Таким чином, розроблений програмний продукт містить наступні ключові функції:

- можливість додавати, видаляти, оновлювати внесені дані
- можливість реєстрації та авторизації користувача
- можливість генерації SQL коду
- можливість переглядати всіх раніше доданих клієнтів

ПЕРЕЛІК ПОСИЛАНЬ

1. Five easy steps to understand JSON Web Tokens (JWT) [Електронний ресурс] – Режим доступу: https://medium.com/cyberverse/five-easy-steps-to-understand-json-web-tokens-jwt-7665d2ddf4d5
2. JSON Web Token [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/JSON_Web_Tok
3. Roy Fielding. Architectural Styles and the Design of Network-based Software Architectures . [Електронний ресурс]. – Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
4. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. [Електронний ресурс]. – Режим доступу: http://www.jopera.org/docs/publications/2008/restws
5. Інструменти адміністрування БД [Електронний ресурс]. – Режим доступу: https://uk.soringpcrepair.com/database-software
6. ТОП 5 фреймворков для Node.js [Електронний ресурс]. – Режим доступу: https://igorosa.com/top-5-node-js-frameworks
7. Alex Rodriguez. RESTful Web services: The basics [Електронний ресурс]. – Режим доступу: https://developer.ibm.com/articles/ws-restful/
8. Todd Fredrich. REST API Tutorial [Електронний ресурс]. – Режим доступу: https://www.restapitutorial.com/
9. Коротко про API та його тестування [Електронний ресурс]. – Режим доступу: https://qagroup.com.ua/publications/korotko-pro-ari-ta-jogo-testuvannia/
10. Материализация Учебник [Електронний ресурс]. – Режим доступу: https://coderlessons.com/tutorials/veb-razrabotka/uchitsia-materializuiutsia/materializatsiia-uchebnik
11. В чем преимущества Node.js? [Електронний ресурс]. – Режим доступу: https://artjoker.ua/ru/blog/v-chem-preimushchestva-nodejs/
12. Почему Node Js: особенности и преимущества [Електронний ресурс]. – Режим доступу:

https://senior.ua/articles/pochemu-node-js-osobennosti-i-preimuschestva

13. Модель-вид-контролер(MVC) [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80

14. Козак О.Л. Опорний конспект лекцій з курсу ―Аналіз вимог до програмного забезпечення‖ для студентів напрямку підготовки ―Програмна інженерія‖ / О.Л. Козак. – Тернопіль, 2011. – 56 с.

15. ПРОФЕСІЙНА ПРАКТИКА ПРОГРАМНОЇ ІНЖЕНЕРІЇ. Лабораторний практикум / уклад. С. В. Поперешняк – К.: Вид-во «Друк», 2019. – 57 с.

16. ДСТУ ISO/IEC 12119:2003. Пакети програм. Тестування і вимоги до якості

17. AD-HOC ТЕСТУВАННЯ) [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/what-is-ad-hoc-testing/

18. Фаулер Скотт До. UML в короткому викладі. Застосування стандартної мови об'єктного моделювання: Пер. з англ. – М.:Мир, 1999. – 191 с.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення

Розробка web-додатку для генерації SQL коду мовою JavaScript

Виконав студент 4 курсу
групи ПД-43
Котул Олександр Юрійович.
керівник роботи
к.т.н., доцент Поперешняк Світлана.Володимирівна.

Київ 2022

Аналоги

dbForge Studio for SQL Server

phpMyAdmin

–HeidiSQL

Valentina Studio

Порівняння аналогів

Аналоги	PhpMyAdmin	dbForge Studio for SQL Server	Valentina Studio	HeidiSQL
Вартість ПО	–	+	– +	–
Відкритий код	+	–	–	+
Імпорт\Експорт даних	+	+	+	+
Зручність інтерфейсу користувача	+	–	–	–
Моделювання діаграм	–	+	+	–
Генерація SQL коду	+	– +	+	–
Адміністрування користувачів	+	–	+	+

3

ОБ’ЄКТ, ПРЕДМЕТ, МЕТА РОБОТИ

Об’єкт дослідження: автоматизація процесу додавання клієнта в БД

Предмет дослідження: система для автоматизації процесу додавання клієнта в БД

Мета роботи: прискорення виконання та зменшення обсягу роботи оператора шляхом генерації SQL коду для додавання клієнта в БД

4

Технічне завдання

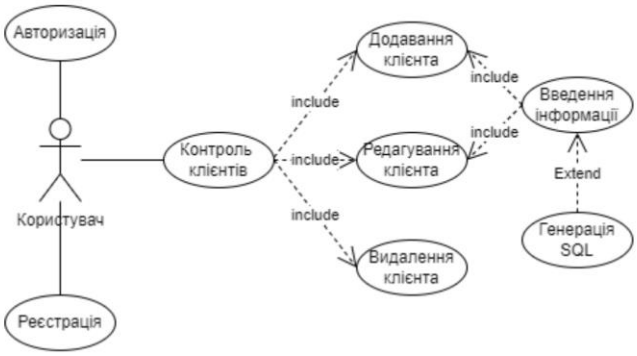
1.Розробити веб-додаток для генерації SQL коду
2.Визначити ключові елементи архітектури
3.Реалізувати можливість додавати, видаляти, оновлювати внесені дані
4.Реалізувати можливість реєстрації та авторизації користувача
5.Реалізувати можливість генерації SQL коду
6.Реалізувати можливість переглядати всіх раніше доданих клієнтів

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

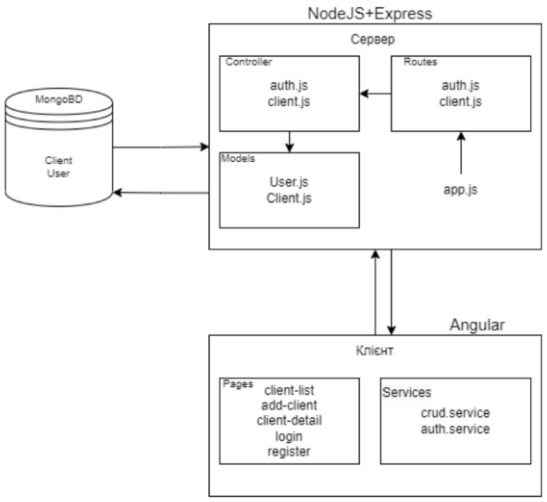
6

Use-case діаграма



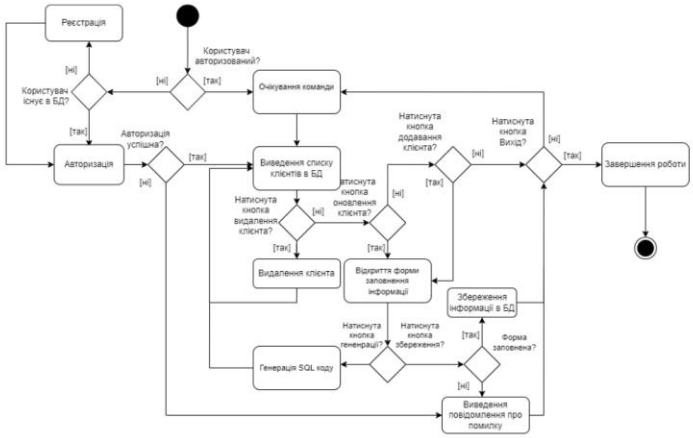
7

Архітектура додатку



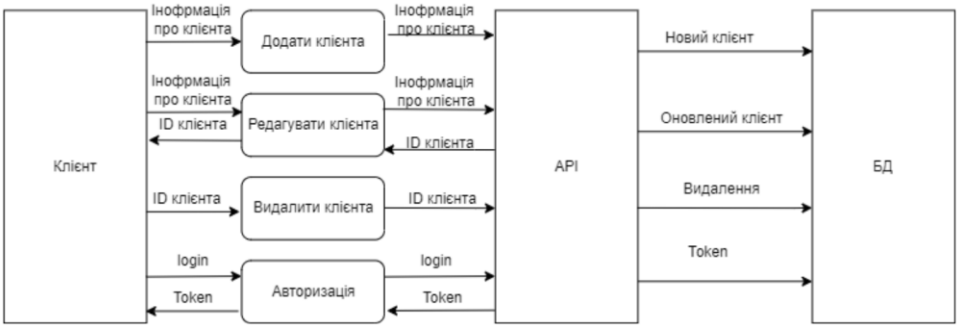
8

Діаграма діяльності



9

Діаграма потоків

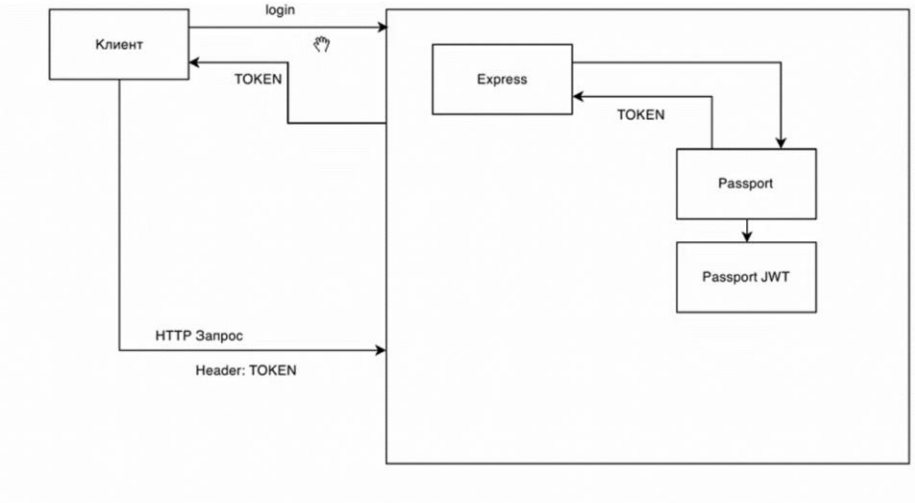


10

Діаграма Класів

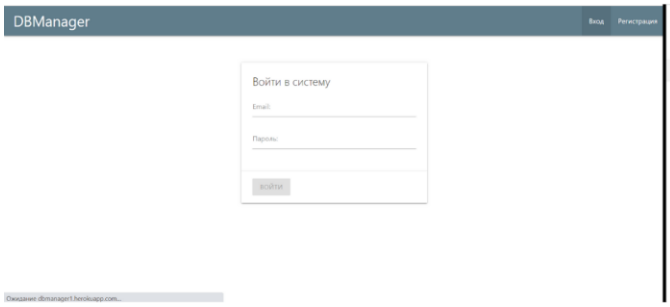
11

Модель авторизації за допомогою JWT

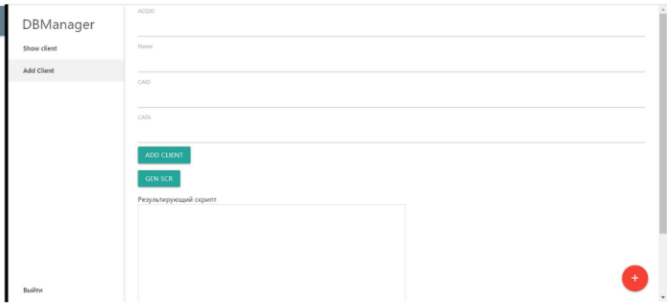


12

Екранні форми



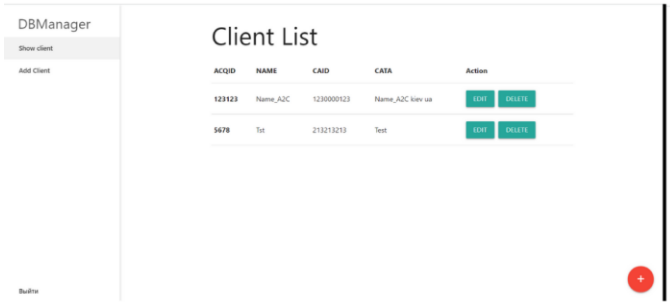
Авторизація



Додавання клієнта
та генерація SQL коду

13

Екранні форми



Список клієнтів

Недагування клієнта та генерація SQL коду

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Котул О.Ю. Застосування JWT(JSON Web Tokens) для автентифікації користувача в веб додатку / О.Ю. Котул // Застосування програмного забезпечення в інфокомунікаційних технологіях: Матеріали науково-технічної конференції. Збірник тез. 20.04.2022, ДУТ, м. Київ — К.: ДУТ, 2022. — С. 49

Котул О.Ю. Переваги та недоліки розробки SPA(single page application) з використанням REST API / О.Ю. Котул // XIV Науково-технічна конференцію студентів та молодих вчених «Сучасні інфокомунікаційні технології» Збірник тез. К.ДУТ, 2022 — подано до друку

15

Висновки

1. Проведено аналіз процесу додавання клієнта в БД. Ключові проблеми, які можна виділити, пов’язані з складністю бізнес процесу. Це призводить до збільшення часу на виконання завдання.
2. Визначено та побудовано архітектуру додатку: серверну частину на платформі Node.JS за допомогою фреймворку Express, та клієнтську частину за допомогою фреймворку Angular.
3. Ключовими недоліками існуючих систем є складність інтерфейсу користувача, та відсутність необхідного функціоналу. Таким чином, розроблено програмне забезпечення,. яке може бути використано для додавання клієнта в БД.
4. Розроблено можливість додавати, видаляти, оновлювати внесені дані.
5. Розроблено можливість реєстрації та авторизації користувача.
6. Розроблено можливість генерації SQL коду.
7. Розроблено можливість переглядати всіх раніше доданих клієнтів.

16

Дякую за увагу!