НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)

“___” червня 20___ р.

Дипломний проєкт

на здобуття ступеня бакалавра

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Засоби аутентифікація та авторизація у Web API _____
_____
_____

Виконав (-ла): студент (-ка) IV курсу, групи КВ-72
(шифр групи)

Коваль Іван Олександрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Наливайчук Микола Васильович _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) __ (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
«___» червня 2021 р.

ЗАВДАННЯ
на дипломний проєкт студента

Коваль Іван Олександрович _____
(прізвище, ім’я, по батькові)

1. Тема проєкту: Засоби аутентифікація та авторизація у Web API _____
_____
_____ ,
керівник проєкту: старший викладач Наливайчук Микола Васильович _____ ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 20__ р. №_____

2. Термін подання студентом проєкту: _____

3. Вихідні дані до проєкту: див. Технічне завдання_____
_____
_____
_____

4. Зміст пояснювальної записки: аналіз існуючих рішень та обгрунтування теми дипломного проекту, обгрунтування вибору засобів реалізації, опис архітектури програмного комплексу _____
_____
_____
_____

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): Взаємозв'язки компонентів програмного комплексу (структурна схема), Структура бази даних (структурна схема), Алгоритм авторизації за допомогою OAuth 2.0 та авторизаційного сервера (схема алгоритма), Взаємозв'язки компонентів серверної частини (структурна схема) _____
_____
_____
_____

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н		

7. Дата видачі завдання «9» грудень 2020_____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проекту	15.04.2021	
2.	Розроблення та узгодження технічного завдання	30.04.2021	
3.	Аналіз існуючих рішень	05.05.2021	
4.	Підготовка матеріалів першого розділу дипломного проекту	10.05.2021	
5.	Підготовка матеріалів другого розділу дипломного проекту	18.05.2021	
6.	Підготовка графічної частини дипломного проекту	20.05.2021	
7.	Оформлення документації дипломного проекту	25.05.2021	
8.	Попередній огляд матеріалів диплому на кафедрі	30.05.2021	

* Консультантом не може бути зазначено керівника дипломного проекту.

Студент _____ Коваль І.О._____
(підпис) (ініціали, прізвище)

Керівник проєкту _____ Наливайчук М.В._____
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Предметом даного дипломного проекту є дослідження методів авторизації та аутентифікації, що використовуються при розробці сучасних Web API.

Об’єктами розробки є створення серверної частини, що виконує функції збереження даних, виконання процесу авторизації та аутентифікації, а також розробка клієнтської частини для користувацького вводу авторизаційних даних і перевірки стану користувача в системі.

Основні частини програмного комплексу:

- серверна частина, що побудована на базі технологій .NET 5, ASP.NET, Docker;
- база даних SQLite;
- клієнтська частина, що розроблена за допомогою технологій Angular 11, Node.js 14, NPM 6.14.

В ході розробки було:

- проведено аналіз існуючих способів авторизації та аутентифікації;
- проведено аналіз сучасних технологій для розробки серверної та клієнтської частин;
- розроблено централізований веб-сервер з базою даних та веб-додаток;
- протестовано роботу всіх частин програмного комплексу.

Даний проект дає змогу порівняти методи авторизації та аутентифікації у сучасних Web API та дозволить обрати метод, що вирішує проблеми конкретної реалізації веб-додатку.

Ключові слова:

Авторизація, аутентифікація, Web API, веб-сервер, веб-додаток, база даних, .NET, ASP, Docker, SQLite, Angular, Node, NPM.

ANNOTATION

The subject of this diploma project is the research of authorization and authentication methods used in the development of modern Web APIs.

The objects of development are the realization of a server part, which performs the functions of data storage, execution of the authorization and authentication process, as well as the development of the client part for user input of authorization data and user status verification in the system.

The main parts of the software package:

- server side, which is built on the basis of technologies .NET 5, ASP.NET, Docker;
- SQLite database;
- client part, which is developed by means of technologies Angular 11, Node.js 14, NPM 6.14.

During development was made:

- the analysis of existing methods of authorization and authentication;
- the analysis of modern technologies for development of server and client parts;
- a centralized Web server with a database and a Web application;
- test of the work of all parts of the software package.

This project allows you to compare the methods of authorization and authentication in modern Web API and choose a method that solves the problems of specific implementation of the Web application.

Keywords:

Authorization, authentication, Web API, Web server, Web application, database, .NET, ASP, Docker, SQLite, Angular, Node, NPM.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045492.002 ТЗ	Засоби аутентифікація та	4		
			авторизація у Web API			
			Технічне завдання			
	А4	ІАЛЦ.045492.003 ТП	Засоби аутентифікація та	1		
			авторизація у Web API			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045492.004 ПЗ	Засоби аутентифікація та	61		
			авторизація у Web API			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Взаємозв'язки компонентів	1		
			програмного комплексу			
			Схема структурна			
	А4	ІАЛЦ.045492.006 Д2	Структура бази даних	1		
			Схема структурна			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА		
Розробив		Коваль І.О.			Засоби аутентифікація та авторизація у Web API **Опис альбому**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Наливайчук М.В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-72	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045492.007 Д3	Алгоритм авторизації за	1		
			допомогою OAuth 2.0 та			
			авторизаційного сервера			
			Схема алгоритма			
	А4	ІАЛЦ.045492.008 Д4	Взаємозв'язки компонентів	1		
			серверної частини			
			Схема структурна			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

					ІАЛЦ.045492.001 ОА	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		2

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Засоби аутентифікація та авторизація у Web API».

Галузь застосування: реалізація аутентифікації та авторизації для використання при реалізації практично будь-якого сучасного Web API.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є розробка серверної частини разом з базою даних для проведення авторизації та аутентифікації, а також веб додатку для вводу користувацьких авторизаційних даних і відслідковування стану користувача в системі.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з будь-яким сучасним браузером з підтримкою HTML5 (Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari);
- можливість реєстрації;
- можливість логіна;
- можливість логаута;
- можливість відслідковувати дані користувача при відповідних методах аутентифікації;
- можливість зберігати дані користувача в системі;
- наявність сховища для зберігання данних;

5.2. Вимоги до апаратного забезпечення

- Процесор: 2-ядерний;
- Оперативна пам’ять: 1 Гб;
- Наявність доступу до мережі Internet (Ethernet, EDGE, 3G, 4G);

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Наявність сучасного браузера з підтримкою HTML5 (Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari).

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.04.2021
2.	Розроблення та узгодження технічного завдання	30.04.2021
3.	Аналіз існуючих рішень	05.05.2021
4.	Підготовка матеріалів першого розділу дипломного проекту	10.05.2021
5.	Підготовка матеріалів другого розділу дипломного проекту	18.05.2021
6.	Підготовка графічної частини дипломного проекту	20.05.2021
7.	Оформлення документації дипломного проекту	25.05.2021
8.	Попередній огляд матеріалів диплому на кафедрі	30.05.2021

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045492.004 ПЗ	Засоби аутентифікація та	61		
			авторизація у Web API			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Взаємозв'язки компонентів	1		
			програмного комплексу			
			Схема структурна			
	А4	ІАЛЦ.045492.006 Д2	Структура бази даних	1		
			Схема структурна			
	А4	ІАЛЦ.045492.007 Д3	Алгоритм авторизації за	1		
			допомогою OAuth 2.0 та			
			авторизаційного сервера			
			Схема алгоритма			
	А4	ІАЛЦ.045492.008 Д4	Взаємозв'язки компонентів	1		
			серверної частини			
			Схема структурна			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ТП		
Розробив		Коваль І.О.			Засоби аутентифікація та авторизація у Web API **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Наливайчук М.В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-72	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Автентифікація – процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора.

Авторизація – керування рівнями та засобами доступу до певного захищеного ресурсу, як у фізичному розумінні, так і в галузі цифрових технологій та ресурсів системи залежно від ідентифікатора і пароля користувача або надання певних повноважень (особі, програмі) на виконання деяких дій у системі обробки даних.

URL – Uniform Resource Locator – стандартизована адреса певного ресурсу (такого як документ, чи зображення) в інтернеті.

ORM – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

CLI – command-line interface – різновид текстового інтерфейсу користувача й комп'ютера, в якому інструкції комп'ютеру можна дати тільки введенням із клавіатури текстових рядків (команд).

Фреймворк – інфраструктура програмних рішень, що полегшує розробку складних систем.

DOM – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами.

SQL – Structured query language – декларативна мова програмування для взаємодії користувача з базами даних.

MVC – Model-View-Controller – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

СУБД – система управління базами даних.

REST – Representational State Transfer ("передача репрезентативного стану") – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

SPA – Single Page Application (односторінковий додаток).

HTML – HyperText Markup Language (мова розмітки гіпертексту).

CSS – Cascading Style Sheets (мова стилю сторінок).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

API (Application Programming Interface) – набір підпрограм для протоколів, щоб взаємодіяти користивацькому інтрефейсу та серверу, створювати програмне забезпечення.

UX (User experience) – досвід користувача.

UI (User interface) – користувацький інтерфейс.

CLR – Common Language Runtime – віртуальна машина, на якій виконуються всі мови платформи .NET Framework.

Хост – будь-який комп'ютерний пристрій, що має доступ до IP мережі.

Інтерсептор – класс, що перехоплює запити на клієнтській частині перед їх відправкою для модифікації.

ВСТУП

З давніх часів у людей була проблема із захистом свого майна та обмеження доступу до нього. Ще до нашої ери було розроблено перші замки. З'явились люди, що почали виробляти засоби захисту майна, але і з'явились ті, що спеціалізувались на зломі. Завдяки цій боротьбі механізми удосконалювались, винаходили нові засоби захисту та злому.

В нашу епоху цифрових технологій нікого не здивувати простим замком. Крім звичайних механічних з'явились кодові замки, біометричні, електронні. Всі вони мають свої плюси і мінуси, але виконують одну і ту саму ціль.

Так само як фізичні замки є й цифрові. Важко уявити додаток або сайт, що не запропонує зареєструватись. Для доступу до функцій додатку як специфічний користувач і використовують так звані процеси авторизації та автентифікації. Вони представляють собою сукупність методів для надання доступу до ресурсів користувачу (ресурси як замок, а логін та пароль – ключ) та розпізнання користувача, його ролей.

Як і в прикладі з замками люди створили і створюють зовсім різні типи автентифікації. Від найпростіших, таких як Basic Authentication, де в параметрах запиту передається логін та пароль, до найбільш складних, таких як Strong Authentication, що потребують додаткового підтвердження з зареєстрованного смартфона та відслідковують підозрілу активність.

Кожен ресурс хоче мати найкращі механізми захисту даних, але як в прикладі з замком часто недоцільно використовувати автентифікацію, що потребує багато дій від користувача. Так, наприклад, веб-ресурс для нагадування людині, коли потрібно пити ліки, малоймовірно буде цілью хакерських атак. Так само додаток, що має всі дані людини, її кредитні картки і тд, має захистити себе та своїх користувачів якнайкраще.

Автентифікація – це акт підтвердження того, що користувачі є тими, ким себе представляють. Це перший крок у будь-якому процесі безпеки. Надання

дозволу на завантаження певного файлу на сервер або надання окремим користувачам адміністративного доступу до програми є хорошими прикладами автентифікації.

Авторизація в системі безпеки – це процес надання користувачеві дозволу на доступ до певного ресурсу або функції. Цей термін часто використовується як взаємозамінний з контролем доступу або привілеями клієнта. У безпечних середовищах авторизація повинна завжди слідувати за автентифікацією. Користувачі повинні спочатку довести, що їхні особисті дані справжні, перш ніж адміністратори організації нададуть їм доступ до запитуваних ресурсів.

Цей проект спрямований показати різницю між різними типами автентифікації та авторизації, їх сукупностями, затратами ресурсів на їх підтримання, рівень безпеки та зручність для користувачів.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

Є різноманітна кількість схем авторизації та автентифікації. При розробці програмного продукту компанія вирішує яку саме схему впровадити в залежності від потреб та бажань розробників. Деякі використовують існуючі рішення, але є випадки, що потребують розроблення власних рішень. Буде розглянуто декілька популярних способів авторизації, їх плюси та мінуси.

1.1 Обґрунтування вибору критеріїв оцінювання існуючих рішень

Так як авторизація використовується для захисту даних та запобіганню втрати даних користувачів, найважливішими критеріями для оцінювання будуть ті, що впливають саме на ці параметри.

При написанні сучасного програмного забезпечення методи авторизації, які можна легко впровадити в свою систему та можливість розширення їх функціоналу грають велику роль на ринку. Саме тому в багатьох випадках головними критеріями оцінювання будуть ті, що впливають на доступність впровадження та керування такими схемами.

Найменш важливими критеріями будуть ті, що дають можливість клієнтській частині зчитувати верифіковані дані без додаткових запитів, але в деяких випадках цей критерій грає велику роль.

Критерії, що впливають на захист даних:

- рівень захисту від злому;
- наявність захисту від крадіжки авторизаційних даних користувача.

Критерії, що впливають на зручність використання схем авторизації:

- універсальність;
- зручність використання незалежно від вибору мови програмування та стеку технологій на проекті;
- низька ресурсозатратність;

- можливість використання окремого серверу для авторизації;
- швидкість написання коду для роботи зі схемою.

1.2 Огляд існуючих програмних рішень

Basic Authentication

Реалізація Basic Authentication (BA) - це найпростіший спосіб для забезпечення автентифікації, оскільки для цього не потрібні файли cookie, ідентифікатори сеансів або сторінки входу; замість цього BA використовує стандартні поля в заголовку HTTP.

Механізм BA не забезпечує захист конфіденційності переданих облікових даних. Вони просто кодуються Base64 під час передачі, а не зашифровуються чи хешуються будь-яким способом. Тому цю схему краще використовувати разом із HTTPS для забезпечення конфіденційності користувацьких даних.

Оскільки поле Authorization має бути надіслане в заголовку кожного HTTP-запиту, веб-браузер повинен кешувати авторизаційні дані протягом певного періоду часу, щоб уникнути постійного запитування користувача щодо свого логіна та пароля. Політика кешування відрізняється між браузерами, тому цей спосіб може працювати по різному в залежності від браузера.

HTTP не має методу для виходу клієнта. Є різні способи для отримання потрібного результату, але кожен з них не є правильним, хоч і всі вони працюють.

API Keys

Ключі API широко використовуються в галузі та стали певним стандартом, однак цей метод не слід вважати хорошою схемою автентифікації.

Ключі API були створені як виправлення для ранніх проблем HTTP Basic Authentication та інших подібних схем. У цьому методі кожному користувачеві, який вперше користується, присвоюється унікальне згенероване значення, що означає, що користувач відомий в системі. Коли користувач намагається повторно увійти в систему, його унікальний ключ (іноді генерується з апаратної

конфігурації та даних IP-адреси, а інколи випадково генерується сервером, що знає користувача) використовується для верифікації цього користувача.

Використовувати API ключі можна де завгодно (в тілі запиту, в авторизаційному або звичайному заголовку, в URL рядку), але найкращим місцем буде саме авторизаційний заголовок. Так він напряму виконує свою ціль. Найгіршим вибором стане URL рядок, так як отримати доступ до ключа зможе будь-яка людина, що побачить адресний рядок в браузері.

Безумовно, є кілька поважних причин використання API ключів. Використання одного ідентифікатора є простим, а для деяких випадків – найкращим рішенням. Наприклад, якщо API обмежений функціоналом, де зчитування є єдиною можливою дією, API ключ буде адекватним рішенням. Без необхідності редагувати, модифікувати чи видаляти безпека не є значною проблемою.

Проблема, однак, полягає в тому, що кожен, хто робить запит на послугу, передає свій ключ, і теоретично цей ключ можна взяти так само просто, як будь-яку мережеву передачу при наявності доступу до мережі у хакерів.

OAuth 2.0

OAuth 2.0 – найкращий вибір для ідентифікації особистих облікових записів користувачів та надання належних дозволів. У цьому методі користувач входить в систему. Потім ця система вимагатиме автентифікацію, як правило, у формі токена. Далі користувач перенаправить цей запит на сервер автентифікації, який або відхилить, або дозволить цю автентифікацію. Звідси токен надається користувачеві, а потім запитувачу. Потім такий токен може бути перевірений у будь-який час, незалежно від користувача, запитувачем для перевірки і може використовуватися з часом із суворо обмеженим часом дії.

Це принципово набагато безпечніший та потужнішийметод, ніж інші підходи, головним чином тому, що він дозволяє встановлювати області, які можуть забезпечити доступ до різних частин API, а оскільки токен відкликається

через певний час – це значно ускладнює його повторне використання зловмисниками.

Потоки OAuth 2.0 – це сценарії, які виконує клієнт для отримання токена доступу від сервера авторизації.

OAuth 2.0 забезпечує кілька популярних потоків, придатних для різних типів клієнтів API:

- **код авторизації** – найпоширеніший потік, в основному використовується для серверних і мобільних веб-додатків. Цей потік схожий на те, як користувачі реєструються у веб-додатку, використовуючи свій обліковий запис Facebook або Google;
- **неявний** потік вимагає від клієнта безпосереднього отримання токена доступу. Це корисно у випадках, коли облікові дані користувача не можуть зберігатися в коді клієнта, оскільки до них може легко отримати доступ третя сторона. Він підходить для настільних, мобільних та веб-додатків, які не містять жодного серверного компонента;
- **пароль власника ресурсу** вимагає вхід за допомогою імені користувача та пароля. Оскільки в цьому випадку облікові дані будуть частиною запиту, цей потік підходить лише для надійних клієнтів (наприклад, офіційні програми, випущені постачальником API);
- **клієнтські дані** – спосіб, призначений для автентифікації між серверами, цей потік описує підхід, коли клієнтська програма діє від свого імені, а не від імені будь-якого окремого користувача. У більшості сценаріїв цей потік забезпечує засоби, що дозволяють користувачам вказувати свої облікові дані в клієнтській програмі, щоб він міг отримати доступ до ресурсів, які контролюються клієнтом.

OpenID Connect

OpenID Connect – це простий рівень ідентифікації поверх протоколу OAuth 2.0, який дозволяє клієнтам перевіряти ідентичність кінцевого користувача на основі автентифікації, що виконується сервером авторизації, а також отримувати основну інформацію профілю кінцевого користувача.

OpenID Connect дозволяє цілому ряду клієнтів, включаючи веб-клієнтів, мобільних клієнтів та клієнтів JavaScript, запитувати та отримувати інформацію про автентифіковані сеанси та кінцевих користувачів. Набір специфікацій є розширюваним та підтримує такі функції, як шифрування ідентифікаційних даних, виявлення постачальників OpenID та управління сеансами.

OpenID Connect визначає потік входу, що дозволяє клієнтській програмі автентифікувати користувача та отримувати інформацію (або "клейми") про цього користувача, наприклад ім'я користувача, електронну адресу тощо. Інформація про ідентифікацію користувача кодується у захищеному JSON токені (JWT), який називається токеном ідентифікатора.

OpenID Connect визначає механізм виявлення, який називається OpenID Connect Discovery, де сервер OpenID публікує свої метадані за відомою URL-адресою, як правило, https://my-app.com/openid-config.

Ця URL-адреса повертає JSON-список кінцевих точок, підтримуваних областей і клеймів, публічних ключів OpenID / Oauth, що використовуються для підписання токенів. Клієнти можуть використовувати цю інформацію для побудови запиту на сервер OpenID. Назви полів та значення визначені у специфікації OpenID Connect Discovery Specification.

1.3 Порівняльна таблиця існуючих рішень

На основі проведеного аналізу за відібраними критеріями можна оцінити існуючі рішення, скориставшись порівняльною таблицею з оцінкою по десятибальній шкалі, що наведена нижче (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця існуючих рішень

Критерії оцінки	Існуючі рішення			
	Basic Authentication	API Keys	OAuth 2.0	OpenID Connect
Безпека при використанні	2	7	9	9
Складність злому для хакерських атак	2	8	10	10
Зручність/простота впровадження в свій веб-сервіс	10	7	9	7
Зручність використання користувачам API веб-сервісу	9	10	9	9
Підтримка та зручність використання на великих проектах	3	9	10	10
Ресурсозатратність та швидкість (більше – краще)	9	9	8	6

1.4 Висновки

При виборі методів авторизації для свого веб-додатку потрібно відштовхуватись від кількості користувачів сервісу, наявністю даних користувачів, що може зацікавити хакерів, схемою доступів до ресурсів та іншим. Так як безпека повинна бути на першому місці, то не варто використовувати складні схеми без належних знань. Це може створити певні «діри» в безпеці авторизації та надати доступ хакерам до закритих ресурсів серверу.

На даний момент явним переможцем чотирьох методів є OAuth 2.0, є деякі випадки, коли API ключі або Basic Authentication можуть бути доречними. OpenID Connect стає все більш популярним, головним чином тому, що він базується на вже популярній OAuth 2.0 та є універсальним рішенням практично для будь-якої системи. Він є більш ресурсоємним ніж його конкуренти тому, що потребує окремий сервер із базою даних для авторизації.

OAuth 2.0 забезпечує масу переваг, від простоти використання до об'єднаного системного модуля і, найголовніше, пропонує масштабованість безпеки. Також цю схему можна впроваджувати при будь-яких архітектурних рішеннях у веб-системі. Саме ці фактори у методах автентифікації є дуже цінними. Саме вони забезпечують зменшену вартість впровадження в довгостроковій перспективі.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Для розробки веб-системи, що реалізує всі схеми авторизації, описані в першому розділі, необхідні:

- серверна частина системи;
- клієнтська частина системи;
- бази даних.

Серверна частина потрібна для авторизації користувачів та роботою з базою даних. Клієнтська частина виступає як GUI для запитів на серверну частину та не має свого складного функціоналу. В даному проекті замість написання клієнтської частини можна було б використати програму Postman для надсилання HTTP запитів, але було прийняте рішення наглядно показати весь шлях авторизації – від вводу своєї пошти та паролю до отримання доступу до ресурсів. База даних в цьому проекті, як і клієнтська частина, має малий функціонал, а для зв’язку з нею використовується ORM на серверній частині, тому для простоти було прийняте рішення не використовувати «важкі» бази даних.

2.1 Обґрунтування вибору мови програмування та фреймворку для розробки серверної частини веб-системи

Для розробки надійної серверної частини було розглянуто три сучасних технології разом із мовами програмування, на яких буде написаний основний код програми. Веб-система має оброблювати велику кількість запитів, але вибір технологій для проекту не має великого значення, так як метою є дослідження методів автентифікації та авторизації, а не технологій для реалізації.

1) **JavaScript**, **Node.js, Express.js**;
2) **Python**, **Django**;
3) **C#, ASP.NET**.

Вибір технологій був обумовлений можливістю швидкого та якісного

написання веб-системи.

На рисунку 2.1 зображено популярність фреймворків для серверної частини.

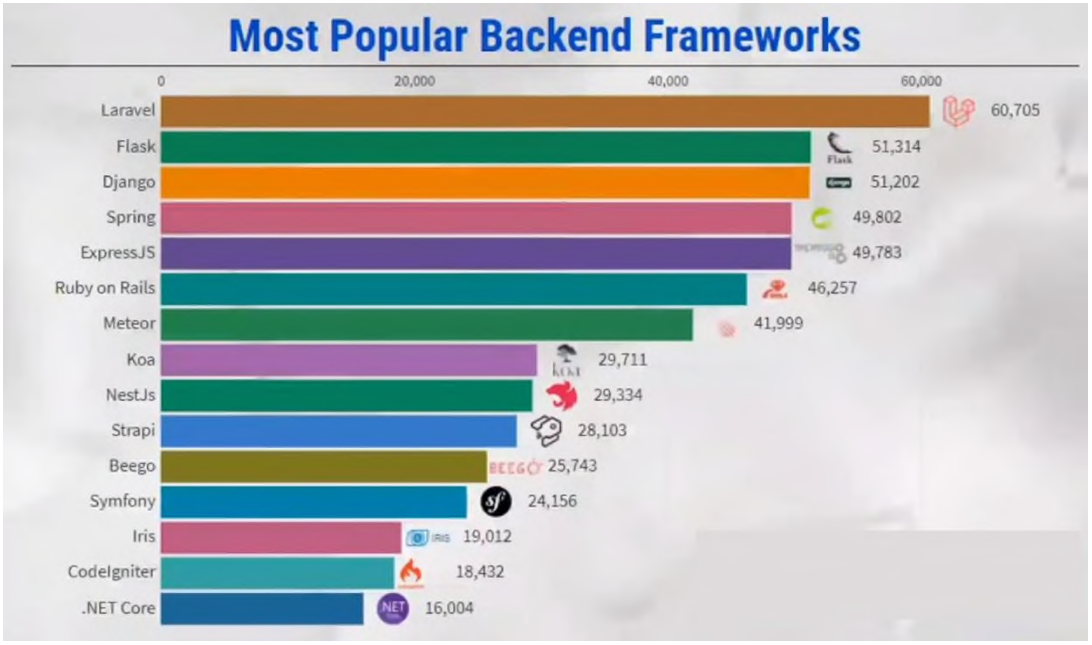

Рис 2.1. Популярність фреймворків для розробки серверної частини

2.1.1 JavaScript, Node.js, Express.js

JavaScript (часто скорочується до JS) – це легка, інтерпретована, об'єктно-орієнтована мова з першокласними функціями і найвідоміша як скриптова мова веб-сторінок, але вона використовується і в багатьох середовищах, що не належать до браузера. Це скриптова мова на основі прототипів, мультипарадигмова, яка є динамічною та підтримує об’єктно-орієнтований, імперативний та функціональний стилі програмування.

JavaScript працює на клієнтській частині Інтернету, що може бути використано для програмування поведінки веб-сторінок при виникненні певних подій. JavaScript – це проста у вивченні, а також потужна мова, широко використовувана для контролю поведінки веб-сторінок.

Незважаючи на те, що JavaScript використовують по більшій частині для написання клієнтської частини, він став популярним при розробці серверної

частини через його потужність та простоту, а також через велику базу коду, що дає можливість використовувати велику кількість бібліотек замість написання та тестування власного кода. Найпростішу серверну частину на JavaScript можна написати за 5 хвилин, використавши всього 10 рядків коду.

Node.js – це міжплатформне середовище виконання з відкритим вихідним кодом, що використовує JavaScript. Це популярний інструмент практично для будь-якого типу проекту.

Node.js запускає движок JavaScript V8, ядро Google Chrome, поза браузером. Це дозволяє Node.js бути дуже продуктивним та потужним.

Додаток, написаний на Node.js, працює в одному процесі, не створюючи новий потік для кожного запиту. Node.js надає набір асинхронних примітивів вводу-виводу у своїй стандартній бібліотеці, які перешкоджають блокуванню коду JavaScript, і загалом бібліотеки в Node.js записуються з використанням неблокуючих парадигм, що робить поведінку блокування винятком, а не нормою.

Коли Node.js виконує операцію введення-виведення, наприклад, зчитування з мережі, доступ до бази даних або файлової системи, замість того, щоб блокувати потік і витрачати цикли процесора на очікування, Node.js відновить операції при поверненні відповіді. Це дозволяє Node.js обробляти тисячі одночасних з'єднань з одним сервером, не вкладаючи тягаря управління паралельністю потоків, що може бути значним джерелом помилок.

Node.js має унікальну перевагу, оскільки мільйони фронтенд розробників, які пишуть на JavaScript для клієнтської частини, тепер можуть писати код на стороні сервера на додаток до коду на клієнтській стороні без необхідності вивчати зовсім іншу мову та інші технології.

До того ж для Node.js існує реєстр коду npm. З ним не потрібно думати про те, як інтегрувати бібліотеки до себе в проект, а достатньо написати команду і почати використання «чужого коду».

npm – найбільший у світі реєстр програмного забезпечення. Розробники програм з відкритим кодом з усіх континентів використовують npm для спільного

використання та запозичення пакетів, а багато організацій використовують npm для управління приватною розробкою.

Express.js є найпопулярнішим веб-середовищем Node і є базовою бібліотекою для ряду інших популярних веб-платформ Node. Він забезпечує механізми для:

- написання обробників запитів з різними HTTP методами за різними URL шляхами;
- інтеграції з механізмами візуалізації "view", щоб генерувати відповіді, вставляючи дані в шаблони;
- встановлення загального налаштування веб-додатків, такі як порт для підключення та розташування шаблонів, які використовуються для надання відповіді;
- додаткової обробки запитів "Middleware" в будь-яку точку конвеєру виконання запиту.

Хоча сам Express є досить мінімалістичним, розробники створили сумісні пакети проміжного програмного забезпечення для вирішення практично будь-якої проблеми веб-розробки. Існують бібліотеки для роботи з файлами cookie, сеансами, логінами користувачів, параметрами URL, даними POST, заголовками безпеки та багатьма іншими.

Однією з причин, чому інженери вибирають Node.js, є швидкість. Express – це тонкий шар, укомплектований функціями для веб-додатків, включаючи основну маршрутизацію, проміжне програмне забезпечення, сервер двигуна шаблонів та статичні файли та інше. Це не погіршує потужну продуктивність вводу – виводу JS. Express має потужний маршрутизований API, який дозволяє будувати API REST та маршрути для складних веб-додатків. Node.js у поєднанні з цим фреймворком в свою чергу є прекрасним вибором для додатків у реальному часі, а також для проєктів, які потребують обробки великої кількості одночасних з'єднань.

Переваги технологій JavaScript, Node.js, Express.js достатньо очевидні.

Швидкість, потужність та простота. Але є й мінуси. Із них виділимо такі:

- проблеми з сумісністю з’являються через використання великої кількості сторонніх пакетів, що можуть потребувати різні версії середовища;
- може бути порушена безпека веб-додатку через те, що використовуваний сторонній пакет може містити вірус або проводити складні обчислювання, наприклад для майнингу криптовалюти, що може сильно збільшити затрати при використанні в якості серверу хмарні технології;
- з цими технологіями складно розробляти великі, добре розширювані проекти в команді та реалізовувати новітні архітектурні підходи до розробки серверної частини веб-додатків.

2.1.2 Python+Django

Протягом багатьох років мова Python стала відомою як динамічна, гнучка та високоздатна мова програмування, яку багато програмістів вибирають серед традиційних варіантів, таких як C ++ та Java. Python також набув величезної популярності серед багатьох веб-розробників. Хоча спільнота веб-розробників розділилась на за та проти ефективності Python для створення багатофункціональних веб-сайтів, при правильному архітектурному підході Python може показувати гарні результати.

Python – це потужна мова програмування, яка забезпечує більшість якісних можливостей, потрібних для веб-сайтів та сучасних додатків. Ось деякі найважливіші аспекти, які роблять Python настільки потужною мовою програмування.

Найбільша привабливість мови програмування Python полягає в тому, що вона надзвичайно проста у використанні для веб-проектів.

Python вивчати так само просто, як і англійську мову, що використовується у повсякденному житті. Простий синтаксис забезпечує дуже низький рівень

вивчення цієї мови.

Що стосується представлення даних на веб-сайті або у додатоку, Python є суперефективною опцією для веб-розробників. Це може легко дозволити створювати зручні для розуміння звіти та візуальне представлення даних.

За допомогою простого та зрозумілого синтаксису Python пропонує чудову читабельність розробникам веб-додатків та допомагає легко зрозуміти код. Це забезпечує більш доступне спілкування розробників у проекті.

Асинхронні шаблони кодування допомагають вирішити багато проблем, з якими час від часу стикаються веб-розробники. Хороша новина полягає в тому, що Python підтримує асинхронний код. Можливість кожного процесу працювати окремо допомагає швидше вирішувати багато проблем.

Незважаючи на всі вищезазначені переваги, Python також має деякі серйозні обмеження.

Python є інтерпретованою мовою програмування, тому він повільніший, ніж інші мови програмування. Також через це помилки виникають в рантаймі, а не на етапі компіляції.

Глобальний інтерпретатор блокування (GIL) Python не дозволяє виконувати більше одного потоку в даний момент часу. Це створює значні обмеження для мови та не дає створювати високонавантажені багатопоточні додатки.

Хоча простота мови програмування Python здається перевагою, вона також є одним із ключових недоліків мови. Програмістам, звиклим до простого синтаксису, часто буває важко перейти на мови зі складним синтаксисом, такі як Java або C#. Ось чому, завдяки великим бібліотекам та динамічним моделям з пізнім прив’язуванням, стає складним перехід на нову мову з Python.

Також через те, що Python є ООП мовою, але також є динамічним, до нього важко прив’язати деякі шаблони проектування для ООП мов. За рахунок цього можуть виникати багато проблем при розробці великих проектів.

Загалом великі плюси мови програмування Python перемагають недоліки мови. Через неперевершену гнучкість, простоту використання та модульність

мови, вона продовжує залишатися улюбленим варіантом для веб-розробників у всьому спектрі.

Django – це безкоштовний веб-фреймворк для Python, який стимулює швидкий розвиток та чистий, прагматичний дизайн, що відповідає архітектурному зразку «Model-Template-Views» (MTV). Він підтримується Фондом програмного забезпечення Django (DSF).

Основна мета Django – полегшити створення складних веб-сайтів, керованих базами даних. Структура підкреслює багаторазовість та модульність компонентів, менше коду, слабку зв'язність, швидкий розвиток та принцип "не повторюватися". Python використовується всюди, навіть для налаштувань, файлів та моделей даних, тому для всіх налаштувань буде один синтаксис.

Django допомагає розробникам, прискорюючи процес розробки. Він включає власний рівень зіставлення об'єктів (ORM) для обробки доступу до бази даних, сеансів, маршрутизації та багатомовної підтримки. Він також дбає про безпеку під час обробки запитів. Django також надає додатковий адміністративний інтерфейс створення, читання, оновлення та видалення, який генерується динамічно за допомогою самоаналізу та налаштовується за допомогою адміністративних моделей, що сильно прискорює роботу над панелями адміністратора.

Django має інструменти запобігання поширеним хакерським атакам, таким як підробка міжсайтових запитів (CSRF) та SQL ін’єкції.

Але Django має декілька серйозних проблем, серед яких основною є швидкість.

Погано розроблена архітектура в поєднанні з Python, яка не є найшвидшою мовою, може призвести до повільних веб-сайтів. Тож потрібно постійно думати про оптимізацію своїх процесів, щоб веб-додаток працював швидше. Django пропонує власні тести для перевірки швидкості роботи та виявлення проблемних місць. Можна застосувати кешування та купу різних оптимізацій, тому для досвідчених розробників оптимізація не є великою проблемою.

Забезпечення добре оптимізованої та масштабованої архітектури з самого початку має позбавити багатьох проблем зі швидкістю в майбутньому. Django працює у величезних веб-додатках – проблеми зі швидкістю – це не проблема самого Django, а питання належної конфігурації та дизайну архітектури.

2.1.3 C#, ASP.NET

C# – це мова програмування загального призначення з багатьма парадигмами, що охоплює статичну типізацію, сильну типізацію, лексичне масштабування, імперативну, декларативну, функціональну, загальну, об'єктно-орієнтовану та компонентно-орієнтовану дисципліни програмування.

C# був розроблений Microsoft близько 2000 року в рамках його ініціативи .NET. Це була одна з мов програмування, розроблена для Спільної мовної інфраструктури (CLI).

Стандарт Ecma містить цілі проектування для C#:

- мова задумана як проста, сучасна, об'єктно-орієнтована мова програмування загального призначення;
- мова та її реалізації повинні забезпечувати підтримку таких принципів програмної інженерії, як сильна перевірка типу, перевірка меж масиву, виявлення спроб використання неініціалізованих змінних та автоматичний збір сміття. Важлива надійність програмного забезпечення, довговічність та продуктивність;
- мова призначена для використання при розробці програмних компонентів, придатних для розгортання в розподілених середовищах;
- переносимість дуже важлива для вихідного коду та програмістів, особливо тих, хто вже знайомий з C та C++;
- підтримка інтернаціоналізації дуже важлива;
- C# призначений для придатності для написання додатків, починаючи

від дуже великих, які використовують складні операційні системи, і закінчуючи дуже малими, що мають певні виділені функції;

- Незважаючи на те, що програми C # призначені для економічного використання пам’яті та вимог до потужності обробки, мова не була призначена для прямої конкуренції за продуктивністю та розміром з мовою C або мовою асемблера.

.NET – це програмне забезпечення, розроблене корпорацією Майкрософт. Він включає велику бібліотеку класів під назвою Framework Class Library (FCL) і забезпечує взаємодію мови (кожна мова може використовувати код, написаний іншими мовами) для декількох мов програмування. Програми, написані для .NET, виконуються в програмному середовищі (на відміну від апаратного середовища) з назвою Common Language Runtime (CLR). CLR – це програмна віртуальна машина, яка надає такі послуги, як безпека, управління пам’яттю та обробка винятків. Таким чином, комп'ютерний код, написаний за допомогою .NET, називається "керованим кодом". FCL і CLR разом складають .NET.

ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

ASP.NET Core може працювати поверх крос-платформного середовища .NET Core, який може бути розгорнутий на основних популярних операційних системах: Windows, Mac OS, Linux. Таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі превалює, тепер ми вже не обмежені тільки цією операційною системою. Тобто ми можемо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатків можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатку

можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору незалежних компонентів. Ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

2.1.4 Порівняльний аналіз мов програмування та фреймворків

На основі приведеного вище матеріалу була зроблена порівняльна таблиця трьох технологій за обраними критеріями (табл. 2.1).

Хоч при порівнянні видно, що ASP.NET має значні переваги, його найбільшим мінусом є високий поріг входження через складність розробки. Для цього проекту він підходить найкраще, тому що підтримує додавання кастомних схем автентифікації. Достатньо розширити базові класи автентифікації і сервер почне при кожному запиті проводити перевірку за цими схемами. В області даного проекту такий підхід найкращий, так як він значно зменшує затрати на розробку і робить код більш «чистим».

Таблиця 2.1

Порівняльна таблиця технологій

Критерії оцінки	Мови та технології		
	JavaScript, Node.js, Express.js	Python, Django	C#, ASP.NET
Висока швидкість обробки запитів	+	+/-	+
Надійний доступ до даних	+	+	+
Підтримка великих систем у майбутньому	+/-	+/-	+
Стабільність при великому навантаженні	+	+	+
Кросплатформена підтримка	+	+/-	+
Весь потрібній функціонал «з коробки»	-	+	+
Не потребує багато досвіду для використання	+	+	-

2.2 Обґрунтування вибору мов програмування та технологій для GUI веб-системи

На даний момент існує велика кількість фреймворків для розробки клієнтської частини. Досить складно вибрати певний, так як у всіх є свої унікальні особливості.

Нижче наведено рисунок 2.2, який демонструє популярність сучасних

бібліотек та фреймворків за останні 4 роки.

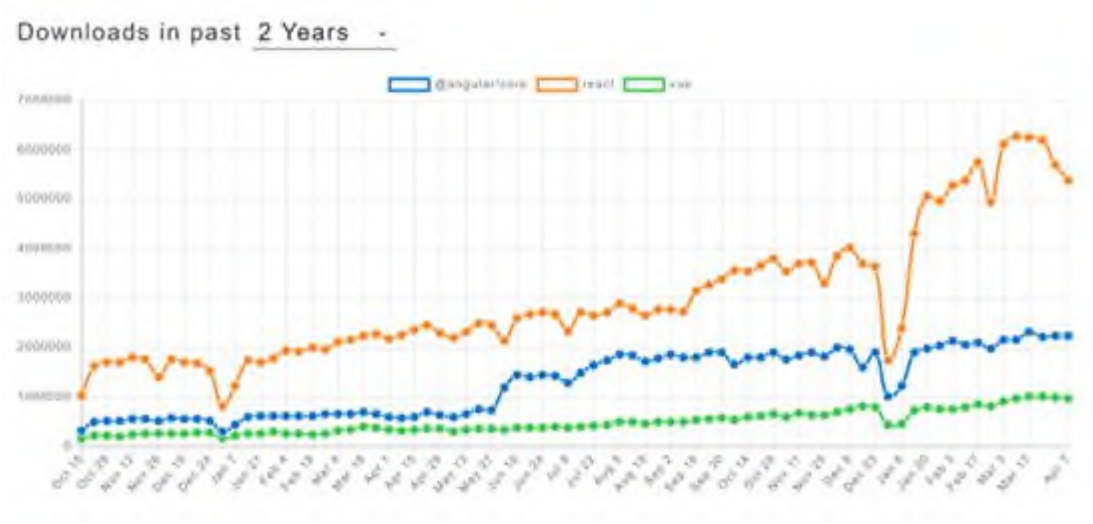

Рис 2.2. Популярність фреймворків для розробки веб-додатку користувацької частини

Нижче буде представлено 3 найпопулярніші веб-фреймворки на даний момент. Вони сильно відрізняються один від одного та мають велику кількість фанатів, тому обрати між ними буде важко. Цими фреймворками є React.js, Angular, Vue.js.

При виборі клієнтського фреймворка потрібно враховувати швидкість, розмір, можливість масштабування технології. Також важливим фактором є підтримка та наявність бібліотек для даного фреймворка.

2.2.1 React

React – це інтерфейсна бібліотека JavaScript із відкритим кодом для побудови користувацьких інтерфейсів або компонентів інтерфейсу. Він підтримується компанією Facebook та спільнотою окремих розробників та компаній. React можна використовувати як основу при розробці SPA або мобільних додатків. Однак React займається лише керуванням станом і наданням цього стану в DOM, тому для створення програм React зазвичай потрібно використовувати додаткові бібліотеки для маршрутизації, а також певну функціональність на стороні клієнта.

Код складається з сутностей, які називаються компонентами. Компоненти можуть бути відтворені на певному елементі в DOM за допомогою бібліотеки React DOM.

Два основних способи оголошення компонентів у React – це функціональні компоненти та компоненти на основі класу:

- функціональні компоненти оголошуються за допомогою функції, яка потім повертає деяку кількість JSX сутностей;
- компоненти на основі класів оголошуються за допомогою класів ES6.

Ще однією помітною особливістю є використання віртуальної об'єктної моделі документа або віртуальної DOM. React створює кеш-структуру даних в пам’яті, обчислює отриману різницю, а потім ефективно оновлює відображуваний DOM браузера. Цей процес називається погодженням. Це дозволяє програмісту писати код так, ніби вся сторінка відображається при кожній зміні, тоді як бібліотеки React відображають лише підкомпоненти, які насправді змінюються. Цей вибірковий рендеринг забезпечує значне підвищення продуктивності. Це економить зусилля з перерахунку стилю CSS, макета для сторінки та рендерингу для всієї сторінки.

React не намагається надати повну "бібліотеку програм". Він розроблений спеціально для побудови користувацьких інтерфейсів, тому не включає багато інструментів, які деякі розробники можуть вважати необхідними для створення додатка. Це дозволяє вибрати, яку бібліотеку розробник хоче використовувати для виконанная таких завданнь, як забезпечення доступу до мережі або локального зберігання даних. У міру удосконалення бібліотеки з’являються загальні шаблони використання фреймворка.

Для забезпечення керування станом додатку React використовує архітектуру Flux.

Для підтримки концепції React про односпрямований потік даних архітектура Flux була розроблена як альтернатива популярній архітектурі Model-View-Controller. Flux виконує команди, які надсилаються через центральний

диспетчер до станового сховища, а зміни в сховищі передаються назад до View. За своєю концепцією Flux був замінений такими бібліотеками, як Redux та MobX, але суть залишилась такою ж. Flux можна вважати шаблоном моделі спостерігача.

2.2.2 Angular

Angular – це веб-фреймворк з відкритим кодом на основі TypeScript, очолювана Angular Team в Google та спільнотою приватних осіб та корпорацій.

Основними особливостями Angular є:

- Angular не має поняття "область" або контролери; натомість він використовує ієрархію компонентів як свою основну архітектурну характеристику;
- модульність – основна функціональність перенесена на модулі Angular рекомендує використовувати мову TypeScript від Microsoft, яка включає такі функції:
  - статичну типізацію, що включає узагальнення;
  - анотації.
- TypeScript є поліпшенням ECMAScript 6 (ES6) і сумісний з ECMAScript 5 (тобто: JavaScript);
- використання динамічного навантаження;
- підтримка асинхронної компіляції шаблонів;
- Ітеративні зворотні виклики, надані RxJS. RxJS обмежує видимість стану та налагодження, але ці проблеми можна вирішити за допомогою реактивних доповнень, таких як ngrx або ngxs.

Компоненти – це будівельні блоки, які складають додаток. Компонент включає клас TypeScript із декоратором @Component(), шаблоном HTML та стилями. Декоратор @Component() складається з таких частин:

- селектор CSS, який визначає, як компонент використовується в

шаблоні;

- елементи HTML у шаблоні, які відповідають цьому селектору, стають екземплярами компонента;
- шаблон HTML, який вказує Angular як відображати компонент;
- необов'язковий набір стилів CSS, які визначають зовнішній вигляд елементів HTML шаблону.

На відміну від React Angular надає розробникам весь потрібний функціонал «з коробки» та диктує правила використання фреймворка, але не забороняє використовувати зручніші технології. Так при виборі Angular зі своїм стеком технологій потрібно розуміти, що велика частина вбудованого функціоналу фреймворка буде простоювати, що тільки збільшить кінцевий розмір збірки.

Для забезпечення керування станом Angular використовує сервісну архітектуру та менеджер станів RxJs. Також ця бібліотека використовується у всіх асинхронних операціях Angular. При правильному використанні RxJs стає потужним інструментом, що забезпечує високу швидкість роботи додатку, зрозумілість коду та легкість в масштабуванні.

2.2.3 Vue

Vue – це прогресивний фреймворк для JavaScript, який використовується для побудови веб-інтерфейсів та SPA додатків.

Назва фреймворку – Vue – в англійській мові таке саме фонетичне, як і view, і воно відповідає традиційній архітектурі Model-View-Controller (MVC). Простіше кажучи, view – це інтерфейс користувача програми / веб-сайту, а основна бібліотека Vue.js за замовчуванням фокусується на шарі представлення. Але, MVC не означає, що Vue.js не можна використовувати з іншим архітектурним підходом, таким як архітектура на основі компонентів (CBA), що використовується в React.

Як і будь-яка технологія, яка набирає популярність, Vue.js викликає суперечки у програмістській спільноті. І на це є причини.

Перш за все Vue дуже легкий – завантажений zip-файл із фреймворком важить всього 18 КБ. Це не тільки швидка завантаження та встановлення бібліотеки, це також позитивно впливає на SEO та UX.

Для швидкості Vue.js використовує віртуальний DOM: це як копія оригінального DOM, який з'ясовує, які елементи оновлювати, не перетворюючи цілий DOM. Цей підхід робить візуалізацію сторінок досить швидким та покращує продуктивність програми. Цей підхід також використовує React.

Ефективність є одним із ключових факторів, який може зумовити вибір Vue. Наприклад, під час тестування компонентів DOM, пов'язаних із оновленими даними, Vue.js є більш продуктивним, ніж Angular та React. Звичайно, він не моє лідируючої позиції, де Vanilla.js займає перше місце. Але враховуючи весь функціонал Vue.js невеликий відрив в швидкості не грає велику роль.

Кожна частина програми або веб-сторінки у Vue є компонентом. Компоненти представляють інкапсульовані елементи інтерфейсу. У Vue.js компоненти можна писати у форматі HTML, CSS та JavaScript, не розділяючи їх на окремі файли як, наприклад, в Angular.

Розбиття коду програми насправді є архітектурним підходом, який називається Component Based Architecture (CBA), і він також використовується в Angular та React. Такий архітектурний підхід має багато переваг:

- багаторазове використання компонентів. Інкапсульовані компоненти – це, в основному, фрагменти коду, які можна використовувати повторно як шаблони для подібних системних елементів;
- читаємість коду. Оскільки всі компоненти зберігаються в окремих файлах (а кожен компонент – це лише один файл), код легше читати та розуміти, що полегшує обслуговування та виправлення;
- зручність юніт-тестування. Юніт-тестування – це перевірка якості, спрямована на перевірку того, як найменші частини програми

працюють самі по собі. Наявність компонентів значно спрощує це завдання.

2.2.4 Висновки та обгрунтування вибору

Кожна з цих бібліотек має свої переваги та недоліки. Залежно від проекту та індивідуальних вимог, один із них буде кращим, ніж інші. Завжди важливо проводити власні дослідження перед тим, як прийняти рішення, особливо при роботі у бізнесі, а не в особистому проекті.

Angular набагато складніший при вивченні та має ряд недоліків для людей з невеликим досвідом в даній сфері. При поганих архітектурних рішеннях та недостатній уважності додаток може працювати повільно та спричиняти так звані «витоки пам’яті», що ще більше погіршить ситуацію.

При цьому Angular має як вбудований менеджер стану, роботу та валідацію форм, HTTP-клієнт і ще велику кількість вбудованого функціоналу, який відсутній у суперників без дозавантаження сторонніх бібліотек.

Основні характеристики кожного із фреймворків показано нижче (табл. 2.2).

Проаналізувавши всі дані, було прийняте рішення використовувати Angular як фреймворк для написання клієнтської частини проекту.

Так як мета даного проекту показати різні популярні методи автентифікації, то клієнтська частина не грає великої ролі. При розробці API програмісти в першу чергу думають про те, щоб їх функціоналом можна було користуватись не залежно від пристрою та браузера. Саме через це на вибір клієнтської частини в даному проекті впливає суб’єктивна думка та наявність функціоналу «з коробки».

При виборі React або Vue довелось би багато часу налаштовувати бібліотеки, альтернативи яких вже працюють в Angular з перших секунд. Це є недоцільним вважаючи малі розміри клієнтської частини.

Таблиця 2.2

Порівняльна таблиця фреймворків

Особливості	Фреймворки		
	React	Angular	Vue
Розмір, кб	97.5	143	58.8
Мова програмування	Javascript	Typescript	Javascript
Вбудовані компоненті користувацького інтерфейсу	React UI tools	Material Design	Компоненті бібліотеки
Архітектура	На основі компонентів	На основі компонентів	На основі компонентів
Сладність вивчення	Низька	Висока	Середня
DOM синтакс	Virtual DOM	Real DOM	Virtual DOM
Масштабованість	Модульна структура	Підхід на основі компонентів	Синтакс на основі шаблонів

2.3 Обґрунтування вибору СУБД

Для більшості проектів найважливішим є правильний вибір бази даних, так як від швидкості отримання та обробки даних найбільше залежить швидкість роботи додатку.

Потрібно враховувати багато факторів, серед яких можна виділити навантаженість, розмір та владеність даних для зберігання, можливості адміністрування та багато інших. Якою б сучасною та бездоганною технологія не

була, знайдуться багато проектів на яких вона буде показувати себе гірше ніж простіша та старіша технологія.

Бази даних також мають різні принципи та формати зберігання даних, тому і потрібно відштовхуватись від того які дані ми будемо використовувати.

Далі буде розглянуто 3 бази даних з зовсім різними принципами технологіями роботи та їх представники. Це будуть SQL та NoSQL(документо-орієнтована, графова, «ключ-значення»).

Далі на рисунку 2.3 зображено популярність різних СУБД.

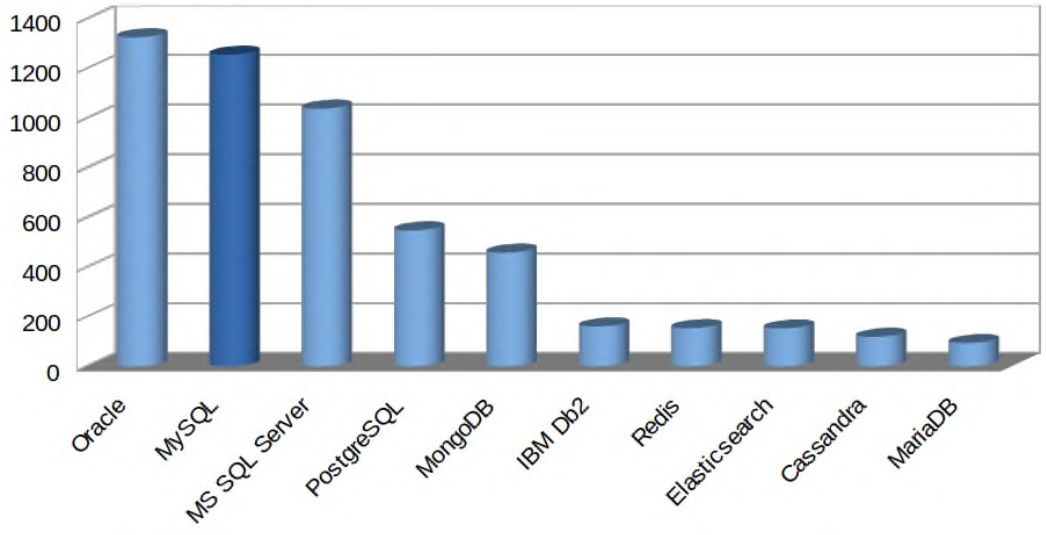

Рис. 2.3 Популярність СУБД

2.3.1 SQL

SQL – це доменно-специфікована мова, що використовується в програмуванні і призначена для керування даними, що зберігаються в реляційній системі керування базами даних (RDBMS), або для обробки потоків в реляційній системі керування потоками даних (RDSMS). Це особливо корисно при обробці структурованих даних, тобто даних, що включають відносини між сутностями та змінними.

SQL пропонує дві основні переваги перед старими API для читання-запису, такими як ISAM або VSAM. По-перше, він представив концепцію доступу до багатьох записів за допомогою однієї команди. По-друге, це позбавляє від

необхідності вказувати, як отримати запис, наприклад з індексом або без нього.

Заснований на реляційній алгебрі та кортежному реляційному численні, SQL складається з багатьох типів тверджень, які можуть бути неофіційно класифіковані як підмови, як правило:

- мова запитів даних (DQL);
- мова визначення даних (DDL);
- мова керування даними (DCL);
- мова обробки даних (DML).

Область SQL включає запит даних, маніпулювання даними (вставка, оновлення та видалення), визначення даних (створення схеми та модифікація) та контроль доступу до даних. Хоча SQL по суті є декларативною мовою, він також включає процедурні елементи.

SQL має багато переваг, що робить його дуже популярним. Це надійна та ефективна мова, яка використовується для спілкування з базою даних. Деякі переваги SQL такі:

- Швидка обробка запитів. Великий обсяг даних отримується швидко та ефективно. Такі операції, як вставка, видалення, маніпулювання даними, також виконуються майже моментально.
- Не потребує багато навичок для розробки. Для отримання даних не потрібна велика кількість рядків коду. Використовуються всі основні ключові слова, такі як SELECT, INSERT INTO, UPDATE тощо, а також синтаксичні правила прості в SQL, що робить його зручною для користувача мовою.
- Стандартизована мова. Завдяки документації та тривалому встановленню протягом багатьох років, SQL забезпечує єдину платформу для всіх користувачів.
- Портативність. SQL може використовуватися в програмах на ПК, серверах, ноутбуках, незалежно від операційної системи. Крім того,

його можна вбудовувати в інші програми відповідно до потреби.

Хоча SQL має багато переваг, все ж є кілька недоліків:

- SQL має складний інтерфейс, через який декому стає незручно під час роботи з базою даних;
- деякі версії програмного забезпечення є дорогими, а отже, багато людей не можуть їх собі дозволити;
- через приховані бізнес-правила, повний контроль над базою даних не надається.

Серед найпопулярніших провайдерів SQL є такі бази даних як MS SQL Server, PostgreSQL, MySQL. Вони маєть певні відмінності, але все ж використовують мову SQL, тому для тих, хто вже знайомий із цією технологією не буде проблемою використовувати незнайомий SQL провайдер.

Для даного проекту серед SQL баз даних я вибрав би SQLite через простоту використання та налаштування. Також SQLite не потребує додаткового сервера, а знаходиться в файловому сховищі серверної частини.

За рахунок того, що проект не використовує велику кількість даних та не потребує складних операцій з ними найкращим вибором буде база даних, що буде найпростіша у використанні.

Також ORM для вибраної технології для серверної частини працює з SQLite. Це означає, що при розробці не буде різниці, яка SQL база буде обрана.

2.3.2 Документо-орієнтована NoSQL

Документо-орієнтована база даних або сховище документів – система зберігання даних, призначена для зберігання, отримання та управління інформацією, орієнтованою на документи, також відома як напівструктуровані дані.

Ці СУБД є однією з основних категорій баз даних NoSQL, і популярність терміна "документо-орієнтована база даних" зросла із застосуванням самого терміна NoSQL.

Бази даних XML – це підклас баз даних, орієнтованих на документи, які оптимізовані для роботи з документами XML. Бази даних графів подібні, але додають ще один шар, взаємозв'язок, що дозволяє їм зв'язувати документи для швидкого обходу.

Бази даних, орієнтовані на документи, за своєю суттю є підкласом сховища ключ-значення, ще однією концепцією бази даних NoSQL. Різниця полягає в способі обробки даних; у сховищі ключ-значення дані вважаються за своєю суттю непрозорими для бази даних, тоді як система, орієнтована на документ, покладається на внутрішню структуру в документі для вилучення метаданих, які механізм бази даних використовує для подальшої оптимізації. Незважаючи на те, що різниця часто незначна через інструменти в системах, концептуально сховище документів розроблено, щоб запропонувати більш багатий досвід використання сучасних методів програмування.

Документо-орієнтовані бази даних сильно контрастують із традиційною реляційною базою даних (RDB). Реляційні бази даних зазвичай зберігають дані в окремих таблицях, визначених програмістом, і один об’єкт може бути розподілений по декількох таблицях. Бази даних документів зберігають всю інформацію для даного об’єкта в одному екземплярі в базі даних, і кожен збережений об’єкт може відрізнятися від іншого. Це позбавляє від необхідності об'єктно-реляційного відображення під час завантаження даних у базу даних.

Найвідомішими прикладами такої структури є MongoDB, Azure Cosmos DB, Apache CouchDB.

Вибирати між ними потрібно, в першу чергу, відштовхуючись від технологій, що використовуються для розробки серверної частини. Так, наприклад, використовуючи .NET від Microsoft і хостячи додаток в хмарі Azure від Microsoft, краще вибрати Azure Cosmos DB. За рахунок того, що все розроблене однією компанією та буде знаходитись в одній хмарі, працювати такий додаток буде швидше.

Також можна відштовхуватись від зручності користування та

адміністрування базами. Кожна з них має свій зручний інтерфейс та своє API, тому спробувавши кожну, знайдеться найзручніша.

2.3.3 «Ключ-значення» NoSQL

«Ключ-значення» ,аза даних або сховище «ключ-значення» – це парадигма зберігання даних, призначена для зберігання, отримання та управління асоціативними масивами, а також структура даних, більш відома сьогодні як словник або хеш-таблиця. Словники містять колекцію об’єктів або записів, які, в свою чергу, мають у собі безліч різних полів, кожен з яких містить дані. Ці записи зберігаються та отримуються за допомогою ключа, який однозначно ідентифікує запис, і використовується для пошуку даних у базі даних.

Бази даних ключових значень працюють зовсім не так, як більш відомі реляційні бази даних (RDB). RDB заздалегідь визначають структуру даних у базі даних як серію таблиць, що містять поля з чітко визначеними типами даних. Виставлення типів даних програмі баз даних дозволяє їй застосувати ряд оптимізацій. На відміну від них, системи ключ-значення трактують дані як єдину непрозору колекцію, яка може мати різні поля для кожного запису. Це забезпечує значну гнучкість і більш точно відповідає сучасним концепціям.

Оскільки необов’язкові значення не представлені вхідними параметрами, як у більшості RDB, бази даних «ключ-значення» часто використовують набагато менше пам'яті для зберігання тієї самої кількості даних, що може призвести до значного збільшення продуктивності при певних робочих навантаженнях.

Продуктивність, відсутність стандартизації та інші проблеми протягом багатьох років обмежували системи «ключ-значення» нішевим використанням, але швидкий перехід до хмарних обчислень після 2010 року призвів до ренесансу як частини більш широкого руху NoSQL. Деякі бази даних графів, також є внутрішньою базою даних з ключовим значенням, додаючи концепцію взаємозв'язків між записами як тип даних першого класу.

До найпопулярніших «ключ-значення» СУБД відносять Amazon

DynamoDB, Oracle Berkeley DB, Redis.

Вибір технології бази даних «ключ-значення» для використання може бути непростим завданням. Якщо основна задача полягає в зберіганні невеликих бітів даних протягом коротких періодів часу і пріоритетом є швидкість над глибиною функцій, просте сховище ключових значень, таке як Redis, стане чудовим інструментом.

Якщо цілі є більш амбітними, варто вибрати документо-орієнтовану базу даних, наприклад MongoDB, яка зберігає дані у названих колекціях і в якій більшість речей можна використовувати як ключі, за допомогою яких їх можна шукати. MongoDB також підтримує розширене індексування та інші потужні способи доступу та оновлення документів, а також структури, починаючи від дуже простих словників і закінчуючи складними вкладеними об'єктами.

Як і в ситуації з документо-орієнтованою базою даних, вибір залежить від власних вподобань, але варто заявити, що такий тип СУБД краще використовувати в якості кеша, а не основної бази.

Обирати такий тип СУБД можна в комбінації з реляційною базою для забезпечення великої швидкості обробки запитів.

2.3.4 Висновки з вибору СУБД

Все більше і більше сучасних проектів, що використовують хмарні сховища, переходять на NoSQL СУБД. Це логічно при великій кількості даних, так як це зменшує затрати пам’яті та збільшує продуктивність додатку.

В розрізі даного проекту краще підходить SQL база даних, так як дані, що використовуються є доволі простими та мають малу вкладеність. Також всі поля об’єктів є обов’язковими (логін, пароль, пошта і тд). За рахунок цього ми не отримаємо того, що хочуть від NoSQL баз даних.

В майбутньому при великій кількості запитів на сервер буде доцільно додати кешування останніх користувачів у «ключ-значення» базу данихЮ що виступатиме як кеш. Це дасть великий поштовх у швидкості роботи додатку.

При виборі SQL СУБД краще використовувати найпростішу для налаштування та використання через простоту даних та відсутністю виконання складних операцій, а також з відсутністю великих навантажень.

Враховуючи всі ці фактори було обрано SQLite базу даних, що буде розміщено у файловому сховищі серверної частини.

2.5 Висновок

У цьому розділі представлені деякі технології для розробки веб-додатків. При порівнянні критеріїв для frontend, backend, та database частин були відібрані найбільш доцільні сучасні рішення.

Основні критерії вибору – це найдійність, висока швидкість обробки запитів, час на налаштування та розробку додатку.

Було вибрано ASP.NET з C# для серверної частини, Angular з Typescript для клієнтської та SQLite як СУБД.

3. ОПИС АРХІТЕКТУРИ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Серверна частина

Серверна частина була написана за допомогою мови програмування C# та фрейморку ASP.NET через їх потужність та багатофункціональність. Також важливим елементом у даній технології є простота в додаванні та обробці власних схем авторизації.

Сервер виконує функції валідації та надання доступу до ресурсів бази даних, так як не можна давати користувачам доступ до даних без потрібних перевірок. Це спричинить витік інформації і можливість здобуття персональних даних.

Проект складається з контролерів, сервісів, репозиторіїв, контексту бази даних, компоненту ініціалізації Program та компоненту налаштування сервера Startup.

3.1.1 Контролер

Контролери відповідають за відповіді на запити, подані з API. Кожен запит йде на певний контролер. Наприклад, якщо ввести таку URL-адресу в адресний рядок свого браузера:

https://localhost:5000/User

У цьому випадку викликається контролер з іменем UserController. UserController відповідає за генерування відповіді на запит браузера. Наприклад, контролер може повернути певні дані назад у браузер або перенаправити користувача на інший контролер.

Контролер включає в себе дії контролера. Дія – це метод контролера, який викликається при введенні певної URL-адреси в адресний рядок браузера. Наприклад, при запиті на таку URL-адресу:

https://localhost:5000/User/Index/3

У цьому випадку метод Index() викликається для класу UserController. Метод Index() є прикладом дії контролера.

Дія контролера має бути загальнодоступним методом класу контролера. Будь-який загальнодоступний метод, який додається до класу контролера, автоматично відображається як дія контролера. Треба бути обережним при написанні контролера, щоб випадково не зробити певний метод публічним, тим само дати можливість зловмисникам отримати дані, які вони не мають знати.

Існує кілька додаткових вимог, яким повинна відповідати дія контролера. Метод, що використовується як дія контролера, не може бути перевантажений. Крім того, дія контролера не може бути статичним методом. При цьому можна використовувати практично будь-який метод як дію контролера.

3.1.2 Репозиторій

Створюючи програму ASP.NET, не слід розміщувати логіку бази даних всередині дій контролера. Поєднання бази даних та логіки контролера з часом ускладнює обслуговування додатка. Краще розмістити всю логіку бази даних в окремому рівні сховища – так званому репозиторії.

Створення рівня сховища дозволяє підтримувати чітке розділення логіки роботи сервера. Контролери відповідають за логіку управління потоком додатків, а репозиторій відповідає за логіку доступу до даних.

Репозиторій здійснює посередницьку діяльність між шарами відображення домену та даних, діючи як колекція об’єктів домену в пам'яті. Клієнтські об'єкти декларативно створюють специфікації запитів і надсилають їх до репозиторія. Об'єкти можна додавати та видаляти із репозиторія, як і з простої колекції об'єктів, а інкапсульований ним код буде виконувати відповідні операції.

Концептуально репозиторій інкапсулює набір об'єктів, що зберігаються в сховищі даних, і операції, що виконуються над ними, забезпечуючи більш об'єктно-орієнтоване уявлення. Репозиторій також має мету досягнення чистого розділення та односторонньої залежності між шарами домену та відображення даних.

3.1.3 Сервіс

Отже, логіка управління потоком додатків належить до контролера, а логіка доступу до даних – до сховища. У такому випадку, логіку перевірки можна розмістити на сервісному рівні.

Сервісний рівень – це додатковий рівень у програмі ASP.NET, який опосередковує зв'язок між контролером та репозиторієм. Сервісний рівень містить бізнес-логіку. Зокрема, він містить логіку перевірки.

Сервіси залежать від репозиторіїв, і вони можуть запитувати декілька класів репозиторіїв та об’єднувати їх дані для формування нових, більш складних бізнес-об’єктів. Крім того, вони вводять рівень абстракції між веб-програмою та репозиторіями, щоб вони могли змінюватися більш самостійно.

Сервіс не повинен знати, як репозиторій оброблює дані, так само як репозиторій не повинен валідувати введені дані.

3.1.3 Контекст бази даних

Екземпляр контексту бази даних являє собою комбінацію шаблонів одиниці роботи та репозиторія, таким чином, що його можна використовувати для запиту в базу даних та групувати зміни, які потім будуть записані назад до сховища як одиниці.

За надання контексту відповідає фреймворк Entity Framework Core та класс DbContext, який потрібно успадкувати та налаштувати.

Entity Framework Core – це ORM (Object-Relational Mapper), що є легким, розширюваним програмним забезпеченням з відкритим кодом. Як і .NET Core, EF Core також є крос-платформним. Він працює у Windows, Mac OS та Linux. EF Core – офіційна платформа доступу до даних від Microsoft.

Одним з найважливіших класів в Entity Framework Core є клас DbContext. Це клас використовується в коді програми для взаємодії з базою даних. Саме цей клас керує підключенням до бази даних і використовується для отримання та збереження даних у базі даних.

DbContext у EF Core дозволяє нам виконувати такі завдання:

- керування підключенням до бази даних;

- налаштування моделі та відносин;
- створення запитів до бази даних;
- збереження даних у базі даних;
- налаштування відстеження змін;
- кешування;
- управління транзакціями.

3.1.4 Компонент ініціалізіції Program

Веб-додаток ASP.NET Core насправді є консольним проектом, який починає виконуватися з точки входу public static void Main() у класі Program, де ми можемо створити хост для веб-програми.

Метод Main() викликає вираз методу BuildWebHost() для побудови веб-хосту з попередньо налаштованими значеннями за замовчуванням.

WebHost – це статичний клас, який можна використовувати для створення екземпляра IWebHost та IWebHostBuilder із заздалегідь налаштованими значеннями за замовчуванням. Метод CreateDefaultBuilder() створює новий налаштований екземпляр WebHostBuilder. Внутрішньо він налаштовує конфігурації Kestrel, IISIntegration та інші. Далі йде метод CreateDefaultBuilder().

Метод CreateDefaultBuilder створює екземпляр WebHostBuilder і встановлює Kestrel, кореневий каталог вмісту та інтеграцію IIS.

Kestrel – це міжплатформенний веб-сервер з відкритим кодом для ASP.NET Core. Він призначений для використання за проксі-сервером, оскільки він ще не має достатньо функціоналу, щоб бути виставленим як повнофункціональний веб-сервер.

3.1.5 Компонент ініціалізіції Startup

Додаток ASP.NET Core повинен містити клас автозавантаження Startup. Як випливає з назви, він виконується спочатку при запуску програми.

Клас запуску можна налаштувати за допомогою методу UseStartup<T>() під час налаштування хосту в методі Main() класу Program.

Назва "Startup" відповідає умовам ASP.NET Core. Однак ми можемо дати будь-яке ім'я класу Startup, просто вказавши його як загальний параметр у методі UseStartup<T>(). Наприклад, щоб назвати клас автозавантаження MyStartup, потрібно використати його як .UseStartup<MyStartup>().

Клас Startup повинен містити метод налаштування Configure і за бажанням може містити метод ConfigureService.

Шаблон ін’єкції залежностей широко використовується в архітектурі ASP.NET Core. Він включає вбудований IoC-контейнер для забезпечення залежних об'єктів за допомогою конструкторів.

Метод ConfigureServices – це місце для реєстрації залежних класів за допомогою вбудованого контейнера IoC. Після реєстрації залежного класу його можна використовувати в будь-якому місці програми. Просто потрібно включити його в параметр конструктора класу, де він буде використаний. Контейнер IoC ін’єктить його автоматично.

Метод Configure – це місце, де можна налаштувати конвеєр запитів для свого додатка за допомогою екземпляра IApplicationBuilder, який надається вбудованим контейнером IoC.

ASP.NET Core представляє компоненти проміжного програмного забезпечення для визначення конвеєру запитів, який буде виконуватися при кожному запиті. Включаючи лише ті компоненти проміжного програмного забезпечення, які потрібні програмі, підвищується продуктивність додатку.

3.2 Клієнтська частина

Клієнтська частина була розроблена на мові Typescript за допомогою фреймворку Angular. Ці технології були вибрані як найзручніші для даного проекту.

Клієнтська частина виконує функції візуалізації даних та виконання запитів на сервер. Також важливою частиною веб-додатку є зберігання певних

авторизаційних даних для їх відправки на сервер з кожнім запитом. Для цього використовуються так звані інтерсептори.

Проект складається з модулів, компонентів, сервісів, інтерсепторів, шаблонів та стилей.

3.2.1 Модуль

Angular додатки є модульними. Angular має власну систему модульності, яка називається NgModules. NgModules – це контейнери для цілісного блоку коду, присвяченого домену програми, робочому процесу або тісно пов'язаному набору можливостей. Вони можуть містити компоненти, постачальників послуг та інші файли коду, обсяг яких визначається вміщуючим NgModule. Вони можуть імпортувати функціонал, який експортується з інших NgModules, та експортувати вибрану функціональність для використання іншими NgModules.

Кожна програма Angular має принаймні один клас NgModule, кореневий модуль, який умовно називається AppModule і знаходиться у файлі з іменем app.module.ts. Додаток запускається, завантажуючи корінь NgModule.

Хоча невелика програма може мати лише один NgModule, більшість програм мають набагато більше модулів функцій. Кореневий NgModule для програми названий так, оскільки він може включати дочірні NgModules в ієрархію будь-якої глибини.

NgModules забезпечує контекст компіляції для своїх компонентів. Кореневий NgModule завжди має кореневий компонент, який створюється під час завантаження, але будь-який NgModule може включати будь-яку кількість додаткових компонентів, які можна завантажити через маршрутизатор або створити через шаблон. Компоненти, що належать NgModule, мають спільний контекст компіляції.

3.2.2 Компонент

Компонент керує областю екрану, що називається View. Визначається логіка застосування компонента – що він робить для підтримки подання всередині класу. Клас взаємодіє з View через API властивостей і методів.

Компоненти є основними блоками для Angular додатку. Кожен компонент складається з:

- шаблону HTML, який оголошує, що відображається на сторінці;
- клас Typescript, який визначає поведінку компанента;
- селектора CSS, який визначає, як компонент використовується в шаблоні;
- необов’язкових стилей CSS, застосованих до шаблону.

Компоненти використовують сервіси, які надають певні функціональні можливості, не пов'язані безпосередньо з поданнями. Постачальники послуг можна ін’єктити в компоненти як залежності, роблячи код модульним, багаторазовим та ефективним.

Angular створює, оновлює та знищує компоненти під час переміщення користувача через додаток. Програма може вживати заходи у будь-який момент цього життєвого циклу за допомогою додаткових hooків життєвого циклу, таких як ngOnInit().

3.2.3 Сервіс

Сервіс – це широка категорія, що охоплює будь-яке значення, функцію чи можливість, які потрібні додатку. Сервіс – це, як правило, клас із вузьким, чітко визначеним призначенням. Він повинен робити щось конкретне і робити як потрібно.

Angular відрізняє компоненти від сервісів для збільшення модульності та повторного використання. Виділяючи функціональність, пов’язану з переглядом компонента, від інших видів обробки, можна зробити класи своїх компонентів більш специфічними та ефективними.

В ідеалі завдання компонента полягає в тому, щоб забезпечити взаємодію з користувачем і не більше того. Компонент повинен представляти властивості та методи прив'язки даних, щоб бути посередником між поданням (відтвореним шаблоном) та логікою програми (яка часто включає певне поняття моделі).

Компонент може делегувати певні завдання службам, наприклад, отримувати дані з сервера, перевіряти введення даних користувача або реєструватись безпосередньо на консолі. Визначаючи такі завдання обробки в ін’єкційному сервісному класі, завдання стають доступними для будь-якого компонента. Також можна зробити свою програму більш пристосованою, ввівши різних постачальників одного і того ж сервісу, як це доречно в різних обставинах.

Angular не забезпечує виконання цих принципів. Angular допомагає слідувати цим принципам, полегшуючи врахування логіки додатка до служб і надання цих послуг доступним для компонентів за допомогою ін’єкції залежностей.

При розробці проекту слідувались всі правила щодо створення та підтримки модулів, компонентів та сервісів. Хоч додаток і не містить багато коду та сервіси не перевикористовуються з різних компонентів, правильні практики забезпечать велику степінь розширюваності додатку. Так, якщо буде потреба в великих доробках, не потрібно буде чіпати існуючий код, а лише дописати потрібний.

3.2.4 Інтерсептор

Шаблон проекту перехоплюючого фільтра використовується, коли ми хочемо виконати певну попередню або подальшу обробку із запитом або відповіддю програми. Фільтри визначаються та застосовуються до запиту перед передачею запиту до фактичної цільової програми. Фільтри можуть виконувати аутентифікацію, авторизацію, реєстрацію або відстеження запиту, а потім передавати запити відповідним обробникам. Нижче наведено сутності цього типу шаблону дизайну:

- **фільтр**, який виконує певне завдання до або після виконання запиту обробником запиту;
- **ланцюжок фільтрів** містить кілька фільтрів і допомагає виконувати їх у визначеному порядку на цілі;

- **об’єктом цілі** є обробник запиту;
- **менеджер фільтрів** керує фільтрами та ланцюжком фільтрів;
- **клієнт** – це об’єкт, який надсилає запит об’єкту цілі.

Інтерсептори перехоплюють запити на сервер перед їх відправкою. Їх дуже зручно використовувати для авторизації. Написавши такий інтерсептор разом із авторизаційним сервісом можна не використовувати авторизацію на кожному запиті адже інтерсептор бере це на себе.

3.3 База даних

Для веб-додатку була використана база даних MS SQL Server. Так як для проекту найкраще підходить SQL СУБД, то було обрано саме SQL Server через те, що технології серверної частини підтримуються компанією Microsoft так само як і ця база даних. Це означає, що зручніше за все буде використовувати їх в парі. Також існує зручне програмне забезпечення для роботи саме з цією СУБД – MS SQL Server Management Studio, яке дозволяє швидко та зручно керувати базою та даними.

Далі буде описано структуру бази даних, що включає в себе такі таблиці:

- User – таблиця, що зберігає авторизаційні дані користувача;
- UserInfo – таблиця, що зберігає дані про користувача;
- Role – таблиця, що містить ролі та їх коди;
- UserRole – таблиця, що містить дані про роль користувача, тобто які права він має;
- AccessCode – таблиця для зберігання кодів, висланих користувачам після входу з нової сесії;
- RefreshToken – таблиця, що використовується тільки для OAuth 2.0 авторизації;
- UserLogInfo – таблиця, що містить дані про те, коли користувач зайшов / вийшов із системи;
-

3.3.1 User

Таблиця User містить інформацію, яку користувач надає при реєстрації та яка є обов’язковою для авторизації. В даному випадку це логін, електронна пошта та захешований пароль. Використовується ця таблиця для перевірки даних користувача при вході в систему та не містить непотрібної для ідентифікіції інформації.

Структура цієї таблиці описана нижче (табл. 3.1)

Таблиця 3.1

Структура таблиці User

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор користувача
Username	Унікальне ім’я в системі, яке придумує користувач
PasswordHash	Захешований пароль, тобто змінений пароль для забезпечення безпеки даних користувача
Base64Salt	Так звана «сіль», що використовується для хешування паролю
Email	Електронна адреса користувача
IsAccountConfirmed	Булеве поле, що показує чи користувач підтвердив свій аккаунт

Причина, через яку ця таблиця не містить інших користувацьких даних, таких як ім’я або номер телефону, проста. Таблиця User не відповідає за зберігання всіх даних користувача, а тільки обов’язкових. Так, наприклад, якщо в майбутньому потрібно буде додати нове поле даних, не потрібно буде модифікувати існуючу таблицю і думати, що робити із зареєстрованими користувачами, що не вводили ці дані.

Взагалі краще створювати для різних видів даних різні таблиці. Наприклад, такі поля як місто, вулиця, квартира краще помістити в окрему таблицю, як і ім’я, прізвище, по-батькові.

Краще притримуватись позиції «одна сутність – одна ціль» та не змішувати все разом, навіть якщо зв’язок завжди один до одного.

3.3.2 UserInfo

Таблиця UserInfo містить інформацію про користувача, але яка не є обов’язковою та не використовується системою для певних процесів. До такої інформації входить ім’я та прізвище користувача, номер телефону. Також ця таблиця має зв’язок один до одного із таблицею User по полю UserId.

Структура цієї таблиці описана нижче (табл. 3.2)

Таблиця 3.2

Структура таблиці UserInfo

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор даних кориистувача
PhoneNumber	Номер телефону користувача
FirstName	Ім’я користувача
LastName	Прізвище користувача
UserId	Унікальний ідентифікатор користувача з таблиці User, має зв’язок один до одного

Було вирішено записувати всю необов’язкову інформацію користувача в цю таблицю. На великих проектах краще розділяти домен сутностей. Ім’я та прізвище в одну таблицю, номер телефону в іншу.

В майбутньому може виникнути необхідність записувати декілька номерів телефону для користувача, що спричинить певні проблеми з міграцією даних. При створенні окремої таблиці для номерів телефонів зв’язок може бути як один до одного, так і багато до одного. Це означає, що при необхідного додати

можливість запису декількох телефонів для користувача не потрібно буде змінювати структуру база даних.

3.3.3 Role

Таблиця Role містить інформацію про ролі в системі. Такими ролями може бути «Користувач», «Адміністратор», «Надкористувач».

Ролі потрібні для задання рівня доступу користувача у системі та рівень доступу до даних.

Структура цієї таблиці описана нижче (табл. 3.3)

Таблиця 3.3

Структура таблиці Role

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор ролі
Code	Не повторюваний номер для ролі (може бути лише числом, що є степенем двійки)
Name	Назва ролі
Description	Опис ролі

У кожного користувача може бути декілька ролей. Для цього краще всього використовувати так звані бітові прапори.

Роль кожного користувача можна записати як число. При наявності декількох ролей можна використовувати побітові операції для надання прав та їх визначення. Для цього кожна роль повинна мати своє число, що є степенем двійки. Нприклад адміністратор має число $2^0 = 1$, користувач має $2^1 = 2$.

Для надання певних ролей потрібно використати бітовий оператор АБО (|). При цьому в бінарному форматі це виглядає так 001 | 100 = 101. Так як у кожної ролі є тільки одна одиничка в бінарному вигляді, то кожна комбінація ролей є унікальної і можна точно визначити прилеглість користувача до ролі.

Для перевірки прав достатньо взяти число ролі користувача, наприклад, 5 і використати бітовий оператор І (&). Так виходить, що $5_{10} = 101_2$. Тепер можна

порівнювати двійкове значення із запитуваною роллю. Якщо виходить 0, то користувач не має такої ролі, якщо не 0 – має.

Для прикладу можна визначити належність користувача до ролі з числом 2. $2_{10} = 010_2$. $101 \ \& \ 010 = 000_2 = 0_2$. 0 означає, що у користувача немає такої ролі.

3.3.4 UserRole

Таблиця UserRole містить інформацію про ролі користувачів в системі. Таблиця потрібна для чіткого розділення домену. Ця сутність керує даними про наявність певного користувача до певної ролі.

Хоч її можна замінити полем з номером ролі у таблиці User, краще створити окрему таблицю для дотримання моделі позиції «одна сутність – одна ціль». Так UserRole контролює ролі, а User авторизаційні дані.

Структура цієї таблиці описана нижче (табл. 3.4)

Таблиця 3.4

Структура таблиці UserRole

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор запису
UserId	Унікальний ідентифікатор користувача в таблиці User
RoleCode	Номер ролей

У розділі 3.3.3 було описано механізм взаємодії ролей. В даній таблиці поле RoleCode містить номер, за яким ведеться верифікація належності користувача до ролі. Таблиці UserRole та Role не пов’язані між собою.

Для швидшої роботи програми краще кешувати дані з таблиць User та UserRole, адже ці дані потрібні при кожному запиті на сервер. В такому випадку без кешування кожен раз буде проходити процес зчитування з бази даних, що сильно збільшить навантаження на базу та зросте час виконання кожного запиту.

Як вже було описано раніше, використання «ключ-значення» баз даних ідеально підходить для такої задачі, але при невеликій кількості користувачів

краще використовувати кешування в оперативній пам’яті. Для нинішніх технологій ця задача не є ресурсозатратною.

Щоб була можливість розширення в майбутньому при збільшенні кількості користувачів, краще розробити інтерфейс, а реалізацію залишити на потім. Так на початку проекту буде використано реалізацію з кешуванням в оперативній пам’яті, а в майбутньому реалізацію з «ключ-значення» СУБД, наприклад, Redis. Так достатньо буде написати реалізацію та не змінювати весь код.

3.3.5 AccessCode

Таблиця AccessCode містить коди, що відравляються на телефон після реєстрації для підтвердження аккаунта.

Структура цієї таблиці описана нижче (табл. 3.5)

Таблиця 3.5

Структура таблиці AccessCode

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор запису
UserId	Унікальний ідентифікатор користувача в таблиці User
Code	Код для підтвердження
TimeSent	Час відправти повідомлення

Після входу з нової сесії (найчастіше це означає, що вхід був виконаний з пристрою, з кого ще не було виконано вхід до цього аккаунту) генерується код, що складається с чотирьох випадкових цифр. Його потрібно ввести для підтвердження особистості.

На сервері задається час життя такого кода. Після виходу цього часу код перестає бути дійсним.

При успішному вводі кода запис видаляється та здійснюється вхід. Якщо ж код був введений неправильно, повертається помилка, та потрібно здійснити запит на новий код підтвердження.

Таку схему можна покращити, використовуючи технологію, що називається CAPTCHA. Вона дозволяє запобігти машинному вводу даних, що може бути видом хакерської атаки, при якій комп’ютер намагатиметься підібрати цей код доти, доки не вгадає його.

Також можна блокувати на якийсь час аккаунт після певої кількості невірних введень такого коду.

3.3.6 RefreshToken

Таблиця RefreshToken містить токен для оновлення токена доступу, що закінчився. Вона використовується тільки при авторизації за допомогою OAuth 2.0.

Структура цієї таблиці описана нижче (табл. 3.6)

Таблиця 3.6

Структура таблиці RefreshToken

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор запису
AccessToken	Токен доступу
RefreshToken	Токен оновлення токену доступа

Раніше було описано принцип роботи авторизації OAuth 2.0, що виконується за допомогою двух токенів. Для оновлення токена, що використовується для авторизації потрібен токен оновлення.

При запиті на оновлення токена після його валідації вібдувається видалення запису з бази, заміна двух токенів, запис їх в базу та відправка відповіді на клієнтську частину.

3.3.7 UserLogInfo

Таблиця UserLogInfo включає в себе дані про те, коли користувач зайшов або вийшов із системи.

Структура цієї таблиці описана нижче (табл. 3.7)

Таблиця 3.7

Структура таблиці UserLogInfo

Назва поля	**Опис поля**
Id	Унікальний ідентифікатор запису
UserId	Унікальний ідентифікатор користувача в таблиці User
LogType	Тип лога
LogTime	Час запису лога

Таблиця UserLogInfo має чисто утилітарний характер та не використовується напряму при авторизації, але є прикладом гарної практики. Краще робити записи всього, що відбувається в системі.

Таблиця містить інформації про те який користувач (UserId) коли (LogTime) та яку операцію (LogType) виконав. Так можна відслідкувати всі дії користувача та надати потрібну допомогу, знаючи всі умови, в яких був користувач.

ВИСНОВКИ

У цьому дипломному проекті було створено програмний комплекс для порівняння різних типів автентифікації та авторизації. Метою розробки такої системи є дослідження та імплементація різних сучасних схем авторизації, що зустрічаються як в невеликих персональних проектах, так і у гігантів IT індустрії.

Розроблений програмний комплекс не має цілі подальшої інтеграції в існуючі системи, а є набором прикладів використання існуючих рішень.

Цей проект може допомогти вирішити яку схему авторизації краще використати при певних умовах, що включають в себе такі ключові характеристики:

- розмір проекта;
- кількість активних користувачів;
- потужність сервера;
- досвід розробників.

Так, наприклад, при написанні малого проекту розробником без великого досвіду в цій сфері, краще використати найпростішу схему авторизації Basic Authentication. Хоч вона і є небажаним рішенням практично в будь-якому випадку, її легко зрозуміти та імплементувати, що зменшує шанси створити діру в безпеці.

Для великих проектів з великою кількістю активних користувачів найкращим варіантом буде OAuth 2.0 або OpenId Connect. Хоч вони і потребують більшу потужність від сервера та бажано окремий контейнер для хостинга, але є добре масшабованими.

Також при великій кількості користувачів краще використовувати окрему «ключ-значення» базу даних для кешування даних, що зменшить навантаження на основну базу та сприятиме збільшенню продуктивності веб-додатку.

Було розглянуто різні технології як для серверної, так і для клієнтської частини. Висновок щодо вибору певної технології складно дати, так як всі сучасні мови програмування підтримують можливість інтеграції схем авторизації, що були описані вище. Тому вибір технологій не повинен бути пирв’язаним до вибору автентифікації та авторизації в проекті.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Авторизація [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Authorization.
2. HTTP Авторизація [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization.
3. Авторизація [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Authorization.
4. Введення до ASP.NET [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0.
5. Документація Node.js [Електронний ресурс]. – Режим доступу до ресурсу: https://nodejs.org/en/docs/.
6. Документація Django [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.2/.
7. Документація Express.js [Електронний ресурс]. – Режим доступу до ресурсу: https://expressjs.com/en/4x/api.html.
8. Специфікація OAuth 2.0 [Електронний ресурс]. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc6749.
9. Basic Authentication [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Basic_access_authentication.
10. Ключі API [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Application_programming_interface_key.
11. Специфікація OpenId Connect [Електронний ресурс]. – Режим доступу до ресурсу: https://openid.net/developers/specs/.
12. Документація Python [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.python.org/3/reference/index.html.
13. Документація C# [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/.

14. Документація Javascript [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
15. Документація Typescript [Електронний ресурс]. – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/.
16. Документація React [Електронний ресурс]. – Режим доступу до ресурсу: https://reactjs.org/docs/getting-started.html/.
17. Документація Vue [Електронний ресурс]. – Режим доступу до ресурсу: https://vuejs.org/v2/guide/index.html.
18. Документація Angular [Електронний ресурс]. – Режим доступу до ресурсу: https://angular.io/docs.
19. Найпопулярніші серверні фреймворки [Електронний ресурс]. – Режим доступу до ресурсу: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/.
20. Найпопулярніші клієнтські фреймворки [Електронний ресурс]. – Режим доступу до ресурсу: https://www.simform.com/best-frontend-frameworks/.
21. Документація MS Sql Server [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15 .
22. Документація PostgreSQL [Електронний ресурс]. – Режим доступу до ресурсу: https://www.postgresql.org/docs/ .
23. Типи СУБД [Електронний ресурс]. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/ .
24. Найпопулярніші СУБД [Електронний ресурс]. – Режим доступу до ресурсу: https://ormuco.com/blog/most-popular-databases.
25. NoSQL СУБД [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/NoSQL.
26. Хеш [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Hash.
27. Хешування паролів у C# [Електронний ресурс]. – Режим доступу до ресурсу:

https://stackoverflow.com/questions/4181198/how-to-hash-a-password/10402129#10402129 .

28. Документація Redis [Електронний ресурс]. – Режим доступу до ресурсу: https://redis.io/documentation.