Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти

на тему

“ Дослідження та програмна реалізація веб-застосунків засобами фреймворків JS як інструментів розробки”

Виконав здобувач вищої освіти
ІІ курсу, групи КН22М-2
ОПП «Комп'ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Кудря В.Д
«____» _____ 2023р.

Керівник проекту
кандидат технічних наук, доцент
_____ Босько В.В.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *магістр*

Галузь знань *12 "Комп'ютерні науки"*

Спеціальність *122 "Комп'ютерні науки "*

Освітньо-професійна (освітньо-наукова) програма *" Комп'ютерні науки "*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
"____" _____20___року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Кудрі Владиславу Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація веб-застосунків засобами фреймворків JS як інструментів розробки*

2. Керівник роботи *Босько Віктор Васильович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 33-13 від 04.08.23

3. Строк подання роботи до захисту *08.01.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне дослідження та програмна реалізація застосування JS фреймворків*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Діаграма процесів процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	09.11.2023 р.	17.11.2023 р.
Охорона праці	Оришака О.В., к.т.н., доцент	03.11.2023 р.	21.11.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	12.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	18.10.2023 р.	
3.	Розробка моделі компонента	23.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	32.10.2023 р.	
6.	Програмування алгоритмів	11.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	16.11.2023 р.	
9.	Оформлення ПЗ	18.11.2023 р.	
10.	Попередній захист роботи	04.12.2023 р	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Кудря В.Д. Дослідження та програмна реалізація веб-застосунків засобами фреймворків JS як інструментів розробки. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації веб-сайту компанії засобами фреймворку AngularJS. Також у роботі представлено загальну та порівняльну характеристику трьох найпоширеніших JavaScript фреймворків: Angular, React.js та Vue.js. Проаналізовано їх переваги та недоліки, доцільність використання для багатосторінкових та односторінкових застосунків.

Метою розробки є дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS.

Об'єктом дослідження є процес реалізації веб-сайту засобами фреймворку AngularJS.

Предметом дослідження є методи реалізації веб-сайту засобами фрейморку AngularJS.

Методи дослідження базуються на методах теорії кодування, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація веб-сайту засобами фреймворку AngularJS.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися у браузері Chrome, Firefox, Safari але завантажується в ОС Linux.

Програму розроблено в середовищі JavaScript.

Ключові слова: комп'ютерна інженерія, веб-сайт, AngularJS.

ABSTRACT

Kudria V.D. Research and software implementation of web applications using JS frameworks as development tools. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023

In this master's thesis, software has been developed that is designed to implement the company's website using the AngularJS framework.

The purpose of the development is to research and software implementation of the company's website using the AngularJS framework.

The object of research is the process of website implementation by means of the AngularJS framework.

The subject of the study is the methods of website implementation by means of the AngularJS framework.

Research methods are based on methods of coding theory, methods of mathematical statistics, methods of software development.

The result is a software implementation of the website using the AngularJS framework.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used in the browser Chrome, Firefox, Safari but is loaded in Linux.

The program is developed in the JavaScript environment.

Keywords: computer engineering, website, AngularJS

№ рядка	Формат	Позначка	Найменування	Кіл. арк.	№ екз.	Додаток
			Текстові документи			
1	А4	ВКРМ-123. 23.0072.00.00.ПЗ	Пояснювальна записка	104	—	
2	А4	ВКРМ-123. 23.0072.00.00.ТЗ	Технічне завдання	6	—	Додаток А
3	А4	ВКРМ -123. 23.0072.00.00.РП	Робоча програма	28	—	Додаток Б
			Графічні документи			
4	А4	ВКРМ -123. 23.0072.00.00.НН	Наукова новизна	1	—	
5	А4	ВКРМ -123. 23.0072.00.00.Е1	Структурна схема	1	—	
6	А4	ВКРМ -123. 23.0072.00.00.Е2	Функціональна схема	1	—	
7	А4	ВКРМ -123.23.0072.00.00.Д1	Діаграма взаємодії процесів	1	—	
8	А4	ВКРМ -123.23.0072.00.00.Д2	Блок-схема основної програми	1	—	
9	А4	ВКРМ -123.23.0072.00.01.Д2	Блок-схема алгоритму роботи підпрограми	1	—	
10	А4	ВКРМ -123.23.0072.00.00.ДЗ	Показники економічної ефективності	1	—	

Вим	*Арк.*	*№ докум.*	*Підпис*	*Дата*	ВКРМ-122.23.0072.00.00.ВП
Розробив		*Кудря В.Д.*			
Перевірив		*Босько В.В.*			*Дослідження та програмна реалізація веб-застосунків засобами фреймворків JS як інструментів розробки*
Н. контр.		*Коваленко А.С*			
Затв.		*Смірнов О.А.*			

Літ.	*Аркуш*	*Аркушів*
М		*1*

ЦНТУ КІ22М-2

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ.....3
ВСТУП.....4
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 7
1.1 Призначення системи..... 7
1.2 Область застосування..... 8
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 13
2.1 Огляд існуючих систем.....13
2.2 Обґрунтування вибору методів розробки 27
2.3 Розгорнута постановка завдання 30
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ.....32
3.1 Опис функціонування системи. 32
3.2 Розробка структурної схеми 37
3.3 Розробка функціональної схеми.....39
3.4 Розробка діаграми процесів.....41
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ..45
4.1 Розробка блок-схем та опис алгоритмів функціонування системи 45
4.2 Захист розробленого програмного забезпечення.....60
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ.....62
6 НАУКОВА НОВИЗНА 67
7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ.....68
7.1 Техніко економічне обґрунтування теми магістерської роботи 68
7.2 Розрахунок трудомісткості розробки програмної продукції.....70
7.3 Визначення чисельності виконавців і планового фонду зарплати. 72

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника.....76
7. 5 Визначення собівартості розробки та ціни програмної продукції. 80
7.6 Визначення об'єму капітальних вкладень та експлуатаційних витрат у споживача програмної продукції. 84
7.7 Визначення експлуатаційних витрат.....84
7.8 Визначення економічної ефективності програмної продукції.....86
7.9 Висновоки. 87
8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ.....88
8.1 Аналіз умов праці програміста. 88
8.2 Заходи профілактики при роботі з комп'ютерною технікою. 92
8.3 Розрахунок та проектування інженерно-технічного заходу захисту від шкідливого (небезпечного) виробничого фактору (освітленість приміщення).....94
8.4 Висновки. 97
9 ОСНОВНІ ВИСНОВКИ.....98
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....100

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

JSON - JavaScript Object Notation

MIME - Multipurpose Internet Mail Extensions

REST - Representational State Transfer

MVC - Model-View-Controller

CSS - Cascading Style Sheets

HTML - Hypertext Markup Language

ОС - операційна система

XML - Extensible Markup Language

CMS - Content Management System

ORM - Object-relational mapping

Sass - Syntactically Awesome Stylesheets

DOM - Document Object Model

EJS - Embedded JavaScript templating

ВСТУП

Актуальність теми. Неспадна популярність мови JavaScript породжує значну кількість нових інструментів для швидкої та зручної розробки, ускладнюючи процес вибору. На сьогоднішній день відсутній один універсальний фреймворк серед розробників, оскільки існує кілька різних за принципом роботи. Проведення порівняльного аналізу дозволяє оцінити головні переваги та недоліки, а також доцільність використання для різних завдань.

Минуло 20 років з моменту виникнення AngularJS, і ця інноваційна технологія в світі веб-розробки, представлена Google в 2010 році, встановила нові стандарти для розробки веб-сайтів і впровадила низку потужних технологій. За цей час команда Google виправила недоліки AngularJS та перебудувала фреймворк з нуля, випустивши в 2016 році абсолютно новий Angular 2, що об'єднав у собі найкращі рішення AngularJS. Таким чином, всі версії 1.xx отримали назву AngularJS, а наступні версії отримали назву Angular.

Репозиторії Angular і AngularJS не мають схожих назв або не є просто новими версіями фреймворку. Ці технології відрізняються своєю суттю. AngularJS базується на JavaScript, тоді як Angular написаний на TypeScript. Крім того, варто відзначити, що архітектурна база тісно пов'язана з даними. З іншого боку, Angular поліпшив продуктивність, сумісність з браузерами і підтримку розробки мобільних додатків у кількох оновленнях.

Ключовим моментом став анонс Google про закінчення розвитку AngularJS версії 1.7 без планів щодо подальших випусків. Таким чином, з 1 липня 2018 року AngularJS увійшов у тривалий трирічний період обслуговування, який через пандемію було продовжено ще на шість місяців.

Мета й завдання дослідження. Метою даної роботи є проведення дослідження та реалізація веб-сайту компанії за допомогою фреймворку AngularJS. Для досягнення цієї мети визначено програму дослідження, яка включає наступні завдання:

- огляд існуючих систем для розробки веб-сайту з використанням фреймворку AngularJS: Розгляд і аналіз існуючих рішень для розробки веб-сайтів на базі фреймворку AngularJS для здобуття інсайтів та визначення найкращих практик;
- дослідження системи та бібліотек для розробки веб-сайту засобами фреймворку AngularJS: Глибоке вивчення функціоналу та можливостей фреймворку AngularJS, а також вивчення відповідних бібліотек та інструментів;
- визначення можливостей та функціоналу AngularJS відповідно до сучасних вимог у веб-розробці: Аналіз здатностей та функціональності AngularJS в контексті сучасних вимог та стандартів веб-розробки;
- програмна реалізація веб-сайту з використанням фреймворку AngularJS: Практична розробка веб-сайту, використовуючи можливості та інструменти, які надає фреймворк AngularJS;
- порівняння фреймворку AngularJS з Angular: Аналіз відмінностей і подібностей між AngularJS та Angular для з'ясування переваг та недоліків кожного з них.

Об'єктом дослідження є процес розробки веб-сайту з використанням фреймворку AngularJS.

Предметом дослідження - методи розробки веб-сайту з використанням цього фреймворку. Застосовуються методи теорії кодування, методи побудови архітектури програмного забезпечення та методи розробки програмного забезпечення для здійснення дослідження.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

Вивчення та порівняння двох фреймворків дозволили визначити їх переваги та недоліки. Хоча AngularJS значно зменшує час розробки додатків, важливо враховувати, що його підтримка припиняється в грудні 2024 року. У той час як Angular може бути більш майбутньо орієнтованим вибором.

Використання NodeJS на сервері для односторінкових додатків: Вказівка на активне використання NodeJS на сервері підкреслює важливість використання цього інструменту в разі розробки односторінкових додатків, де NodeJS може забезпечити ефективну серверну сторону.

Зростання популярності MongoDB для зберігання даних користувачів: Вказівка на тенденцію використання MongoDB, нереляційної бази даних, для зберігання даних користувачів вказує на зміну у виборі баз даних, де віддається перевага гнучкості та масштабованості MongoDB над класичними реляційними базами даних.

Практична цінність результатів. Розроблені веб-сайти не лише відображають можливості фреймворків, але й служать конкретними прикладами швидкості та функціоналу, що важливо для практичного використання отриманих результатів.

Достовірність наукових результатів. Достовірність результатів підтверджена теоретичними концепціями та даними, отриманими під час тестування розроблених систем, що підкреслює надійність отриманих висновків.

На основі цих висновків можна зробити висновок, що хоча існують нові, більш сучасні фреймворки для розробки веб-сайтів, перехід на них може бути обґрунтованим лише у випадку, якщо це вимагають конкретні поточні або майбутні вимоги проекту.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1. Призначення системи

Світ JavaScript надає широкий спектр інструментів для розробки веб-додатків, включаючи різноманітні бібліотеки та фреймворки. Зараз існує близько тридцяти фреймворків, спрямованих на веб-розробку, що стають важливою частиною інтерфейсного програмування та надають зручні інструменти для створення великих та інтерактивних програм. Зростання популярності цих фреймворків вражає, і все більше компаній вибирає їх для своїх проектів. Отже, важливо мати знання про найпопулярніші фреймворки, оскільки це стає обов'язковою складовою для багатьох вакансій.

Попит на розробників інтерфейсів залишається високим, а мова програмування JavaScript утримується на першому місці серед запитань на платформі Stackoverflow протягом восьми років. Це привертає увагу новачків у IT-галузі, і вибір найбільш підходящого інструменту для роботи стає актуальним питанням. Розвиток нових фреймворків, спрямованих на впровадження шаблонів архітектури MVC (MVVM, MVP та інші), внесли свої унікальні підходи та структуру в процес розробки. Це спричинило питання про використання вже відомих шаблонів програмування в світі JavaScript та розділення логіки додатків та відображення контенту.

Використання фреймворків має значні переваги порівняно із власноруч розробленими методами. Це полегшує читання, підтримку та тестування коду, спрощує процес навчання нових розробників та звільняє від необхідності вирішувати однотипні завдання у кожному проекті. Однією з ключових переваг є відсутність необхідності вдумуватися в процес отримання даних від клієнта та їх передачі на сервер, оскільки це вже реалізовано авторами фреймворку. Розробник може фокусуватися на функціоналі додатка.

Ще однією вагомою перевагою є стандартизація кодування, яка визначається правилами фреймворку та дозволяє легше зрозуміти його структуру, особливо для новачків.

1.2. Область застосування

AngularJS знайшов широке застосування у сфері створення односторінкових додатків. У свій час його переваги полягали у високій структурованості додатків, спрощенні процесів тестування та розробки. На момент виходу на ринок, AngularJS володів активною підтримкою від Google, двостороннім зв'язуванням даних, можливістю взаємодії з іншими бібліотеками та декларативним програмуванням, що полегшувало підтримку та розуміння коду.

У сучасних умовах веб-сайти все частіше перетворюються на складні додатки з багатофункціональним інтерфейсом, який дозволяє взаємодіяти з користувачем. Відмінною особливістю AngularJS є можливість створення зручного інтерфейсу та реалізації нових, унікальних функцій, що робить його популярним серед розробників веб-сайтів. Спільно з тим, сучасні тенденції вимагають від розробників використання більш сучасних технологій.

Сьогодні для створення веб-сторінок широко використовуються дві технології: MPA (Multi Page Application - додатки з кількома сторінками) та SPA (Single Page Application - односторінкові додатки). Зокрема, AngularJS виявляється корисним при розробці SPA, де він може забезпечити ефективне взаємодіюче середовище та сприяти створенню динамічних та ефективних веб-додатків.

Метод розробки багатосторінкових додатків (MPA) є традиційним підходом у веб-розробці. У процесі розробки створюється кілька сторінок, які використовують шаблони для зберігання статичних даних, таких як заголовки та зображення, а також посилання на інші сторінки із подібним вмістом. Кожен

перехід на нову сторінку породжує запит на сервер для отримання даних та їх відображення, при цьому сервер надсилає інформацію, яка вже була на попередній сторінці. Це може призводити до зайвого використання ресурсів та уповільнення роботи сервера і браузера.

Традиційні технології, такі як CSS для стилізації та HTML для розмітки сторінок, використовуються для створення багатосторінкових сайтів. Проте, при потребі в розробці веб-сайту із складним функціоналом, таким як динамічне оновлення даних, можна залишатися в рамках традиційних методів, але це обмежить можливості та вимагатиме використання більш сучасних технологій, наприклад, JavaScript та AJAX.

Із завдяки розвитку веб-технологій сьогодні можливо створювати веб-сайти зі складними анімаціями, графікою та динамічним оновленням інформації на сторінці без її перезавантаження. Хоча ці технології мають свої недоліки, такі як складність адаптації для мобільних версій сайтів та ускладнення розробки нових функцій, серед їх переваг слід відзначити легкість оптимізації для пошукових систем та можливість додавання спеціальних метатегів.

Однією з переваг MPA є менша кількість технологій, що сприяє зменшенню вартості розробки таких додатків.

SPA (односторінкова програма) представляє сучасну технологію, яка ставить за мету надати програмі більше схожості з настільною. Основна відмінність від MPA полягає в тому, що при переході на нову сторінку браузер відправляє запит на сервер лише для отримання необхідної інформації, а оновлення відбувається лише для тих даних, які були отримані від сервера. Це спрощує використання та створює зовнішній вигляд більш ефективним. SPA став популярним серед великих технологічних компаній, таких як Google у Gmail та Facebook у своїх соціальних мережах, і часто використовується для розробки за допомогою JavaScript або TypeScript.

Хоча jQuery може бути використаний для розробки менших програм, він не є оптимальним для великих проектів через відсутність структури та

ускладнення читання коду. У разі складних проектів розробники звертаються до фреймворків, таких як AngularJS, Angular, React, Vue, які встановлюють архітектурну структуру та спонукають дотримуватися певної організації коду. Ці фреймворки також надають функціонал для взаємодії з сервером, реалізуючи динамічне оновлення даних та стилізацію підключень. Вони також підходять для створення мобільних додатків, використовуючи один і той же код для клієнтської та серверної частин.

Однією з недоліків SPA є обмежена SEO оптимізація, оскільки пошукові системи не можуть індексувати вміст сторінок. Хоча існують рішення для цієї проблеми, вони вимагають додаткового часу. Крім того, на пристроях з обмеженими можливостями завантаження сторінка може бути ускладненою, і для великих проектів рекомендується розділити їх на окремі компоненти JavaScript для швидшого завантаження. Однак переваги SPA включають високу швидкість оновлення даних, велику кількість готових бібліотек та компонентів для ефективного використання, а також можливість створення мобільних додатків, використовуючи загальний код для клієнтської та серверної частин.

Кожен із зазначених методів розробки програми має свої переваги та недоліки відповідно до вимог проекту. SPA (односторінкова програма) має кілька ключових переваг, таких як висока швидкість роботи додатків та можливість розробки мобільних додатків. Однак вона також стикається з труднощами у взаємодії з пошуковими системами. Цей архітектурний підхід найбільш підходить для соціальних мереж, де ключовими є висока швидкість оновлення даних та продуктивність браузера, а необхідність індексації в пошукових системах є менш суттєвою.

Навпаки, MPA (багатосторінковий додаток) виявляється ефективним для бізнес-сайтів та інтернет-магазинів, але може викликати труднощі при розробці мобільних додатків через свою структуру. Враховуючи це, вибір між SPA і MPA повинен бути здійснений на основі типу проекту, вимог замовника та потреб

користувачів, з метою розробки системи, яка відповідає всім необхідним функціональним вимогам і може ефективно масштабуватися.

Що стосується AngularJS використовується як фреймворк для односторінкових додатків (SPA) і базується на патерні проектування MVC, що стало стандартом для багатьох фреймворків у світі JavaScript. Цей патерн, хоч і вже добре відомий та розповсюджений, надає структуру для розробки програм та забезпечує йому доступ до світу JavaScript.

AngularJS дозволяє використовувати HTML як мову шаблонів та розширювати її синтаксис для чіткого та лаконічного вираження компонентів програм. Використання зв'язування даних та ін'єкції залежностей допомагає скорочувати кількість коду та поліпшує його читабельність.

AngularJS має низький поріг входження для новачків, але при цьому документація фреймворку може залишати певні питання. Вона структурована і містить приклади коду, але деякі аспекти можуть виглядати недостатньо деталізовано. Розробникам іноді доводиться самостійно розібратися з певними нюансами, вивчаючи вихідний код або звертаючись за консультацією до колег.

Незважаючи на ці виклики, AngularJS залишається активно використовуваним фреймворком. Він був одним із перших, що представив концепцію SPA, і його можливості вплинули на багато інших фреймворків. Хоча для нових проектів рекомендується використовувати більш сучасні рішення, AngularJS залишається важливим історичним кроком у розвитку SPA.

Цей рейтинг, який був опублікований на веб-ресурсі State of JS 2020, визначає найбільш використовувані фреймворки на основі задоволеності користувачів, рівня зацікавленості в інструменті, відсотка використання та загальної обізнаності. За останніми двома критеріями протягом останніх чотирьох років фреймворки Angular, React і Vue.js зберігають лідерство (рисунок. 1.1), що свідчить про їх надійність, стійкий інтерес серед компаній та гарантію наявності обширної документації та готових рішень.

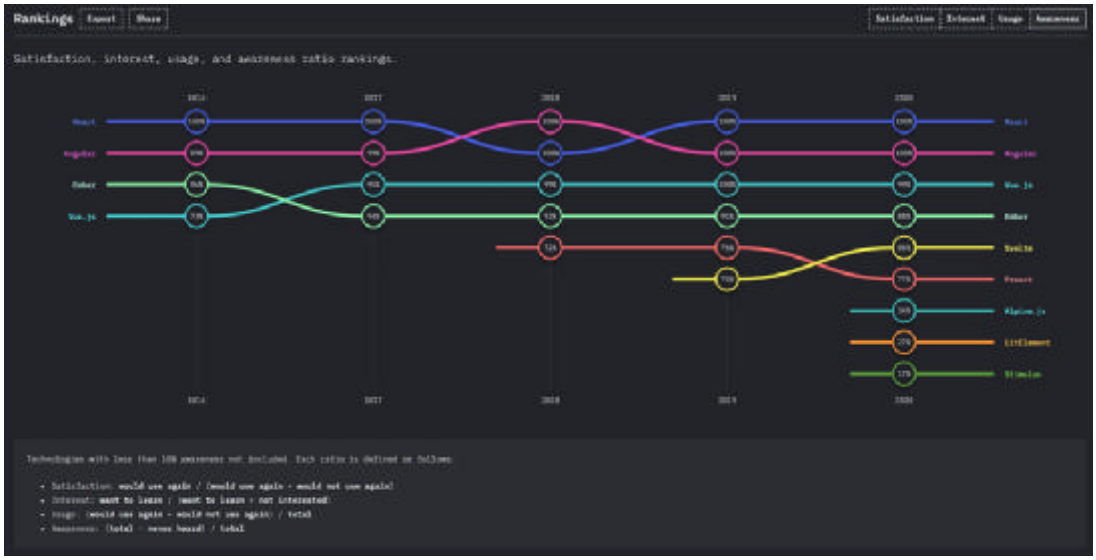

Рисунок 1.1 - Рейтинг State of JS 2023

У рейтингу 2020 Developer Survey [2], представленому компанією Stackoverflow (рисунок 1.2), зафіксовано зростаючий інтерес до обраних фреймворків. Спостереження за цим рейтингом показує, що бібліотека jQuery, хоча ще й є лідером серед запитів, поступово втрачає свої позиції на користь Angular та React. Зазначено, що понад 35% респондентів використовують один із цих технологічних інструментів.

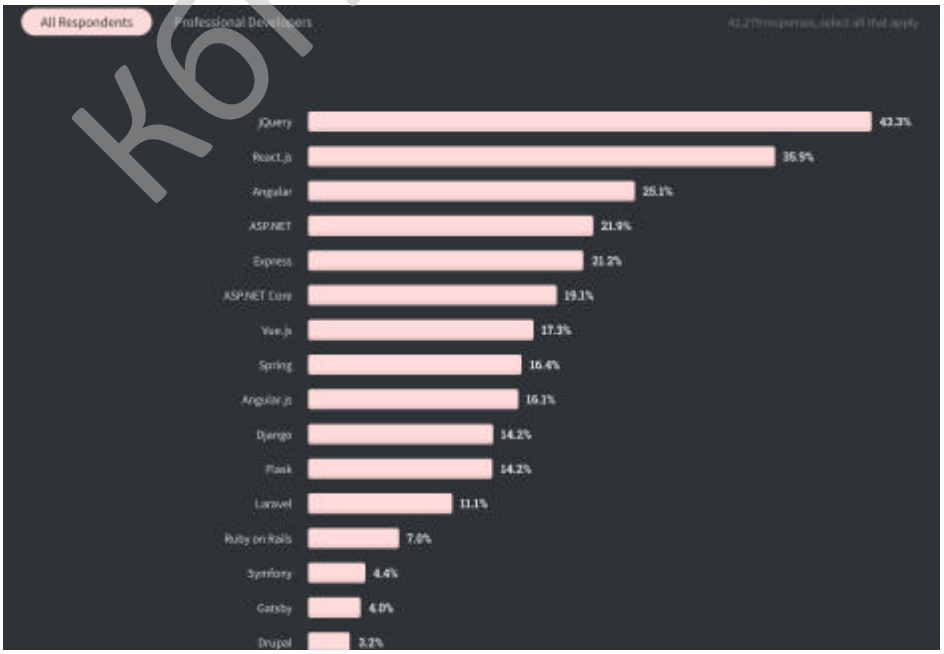

Рисунок 1.2 - Рейтинг 2023 Developer Survey

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1. Огляд існуючих систем

Світ JavaScript представляє собою динамічне середовище, де клієнт, вводячи веб-адресу в браузер, ініціює запит на сервер. Відповідь, яку отримує клієнт, часто містить HTML-документ зі стилізацією та посиланнями на JavaScript-файли. Це дозволяє виконувати код на стороні клієнта, змінюючи вміст сторінки без перезавантаження.

JavaScript визначається своєю можливістю працювати на стороні клієнта, що створює інтерактивність та поліпшує користувацький досвід. Наприклад, використання модальних вікон та динамічних елементів робить взаємодію з сайтом приємною та зручною.

"Ванільний" JavaScript — це базова мова програмування без додаткових інструментів. Хоча JavaScript ES5 підтримується більшістю браузерів, використання лише "ванільного" коду може бути заплутаним та складним. Тому виникли бібліотеки, такі як jQuery та LoDash, що спрощують написання програм та оптимізують код.

Бібліотеки — це пакети від сторонніх розробників, які містять готові функції та об'єкти. Вони дозволяють програмістам прискорити розробку, використовуючи оптимізовані методи. Наприклад, jQuery та LoDash є прикладами бібліотек, які сприяють ефективному написанню програм та забезпечують функціональність з меншим обсягом коду.

Важлива різниця між бібліотеками та фреймворками полягає в рівні контролю, який вони надають розробнику над структурою та організацією коду.

Бібліотеки зазвичай спрямовані на вирішення конкретних завдань, таких як робота з DOM (наприклад, jQuery) або робота з колекціями даних (наприклад, LoDash).

Вибір розробника. Розробник визначає структуру та організацію програми, використовуючи функціональність бібліотеки при необхідності.

Фреймворки. Більше контролю: Фреймворки визначають загальну структуру додатка та контролюють, як розробляється весь код. Вони надають заготовлені шаблони та правила, за якими слід будувати додаток.

Структурованість. Фреймворки визначають патерни проектування, що спрощує розробку і забезпечує консистентність між різними частинами додатка.

Автоматизація. Багато рутинних задач, таких як маршрутизація, керування станом і взаємодія з сервером, автоматизовані в рамках фреймворку.

Бібліотеки допомагають з конкретними завданнями, розробники контролюють структуру і організацію коду, фреймворки забезпечують більше стандартизації та автоматизації, контролюючи більше аспектів розробки за замовчуванням.

Конструктори проектів - Babel і webpack

Babel - це транскомпілятор, який дозволяє використовувати нові функції мови JavaScript, такі як ES6 і вище, на старших версіях (наприклад, ES5), що робить код більш сучасним та сумісним з різними браузерами.

Webpack - це інструмент для збірки та пакування ресурсів, таких як JavaScript, стилі, зображення і інше. Він дозволяє оптимізувати, мінімізувати і керувати завантаженням ресурсів у веб-додатку.

Версії JavaScript - ECMAScript 2015+ (ES6): Введення нових можливостей у JavaScript, таких як стрілкові функції, класи, деструктуризація та інші покращення синтаксису.

TypeScript: Він розширює JavaScript, додаючи статичну типізацію, інтерфейси, перечислення та інші функції. TypeScript покращує якість коду та допомагає у визначенні та уникненні помилок під час розробки.

Використання таких інструментів дозволяє створювати більш сучасні, ефективні та сумісні з різними середовищами додатки. Ці інструменти також

підтримують розвиток прикладних програм та полегшують управління залежностями та збіркою проектів.

Давайте розглянемо кілька концепцій та інструментів, які грають важливу роль у розробці JavaScript-додатків.

Babel є транскомпілятором для JavaScript. Він дозволяє використовувати нові функції мови, які можуть ще не підтримуватися всіма браузерами. Babel перетворює код, написаний з використанням нових можливостей (наприклад, ECMAScript 2015+), в код, який може бути виконаний на більшій кількості браузерів.

Webpack — це інструмент для збірки і пакування ресурсів веб-додатка, таких як JavaScript-файли, таблиці стилів, зображення тощо. Він дозволяє створювати оптимізовані та ефективні версії файлів для розгортання вироблених додатків.

TypeScript — це розширення JavaScript, яке додає статичну типізацію до мови. Це дозволяє розробникам визначати типи даних для змінних, параметрів та інших елементів коду. Відсутність динамічної типізації може допомогти у виявленні та усуненні помилок під час розробки.

ECMAScript — це стандарт, на якому базується JavaScript. Різні версії, такі як ES6 (ECMAScript 2015), ES7 і т. д., вводять нові функції мови, які розширюють можливості розробників. TypeScript може використовувати більшість нововведень ECMAScript.

Ці інструменти та концепції сприяють полегшенню розробки, забезпечують вищу продуктивність та допомагають створювати більш масштабовані та ефективні JavaScript-додатки.

Важливі аспекти використання ECMAScript та TypeScript у світі JavaScript. ES5 - це стандарт, який використовується більшістю браузерів, коли пишете код в стилі ECMAScript 2015+ (ES6 і вище), Babel транскомпілює його у код, зрозумілий для більшості сучасних браузерів, основаних на стандарті ES5.

Нововведення ES6+. Використання нововведень ES6+ дозволяє розробникам писати більш сучасний та зрозумілий код, використовуючи класи, стрілкові функції, деструктуризацію та інші покращення.

TypeScript додає статичну типізацію, що дозволяє розробникам визначати типи даних для змінних, функцій, об'єктів та інших елементів коду під час розробки. Це може допомогти виявляти та усувати помилки під час компіляції.

Розширення JS: TypeScript — це розширення JavaScript, що означає, що весь синтаксис JavaScript доповнюється можливостями TypeScript, такими як інтерфейси, перечислення, узагальнення та інші.

Webpack та Babel

Babel транслює код, написаний з використанням нововведень ECMAScript, у сумісний з ES5. Webpack використовується для збірки та пакування різних ресурсів додатка.

Webpack може виконувати оптимізацію та мінімізацію коду, забезпечуючи ефективніший виріб для розгортання.

В сучасному веб-розвитку використання фреймворків JavaScript є ключовим елементом для створення потужних та ефективних веб-додатків. Підкреслимо деякі основні переваги використання фреймворків у веб-розробці:

- швидкість розробки. Фреймворки надають готові рішення, що дозволяє розробникам прискорити процес розробки. Вони включають готові компоненти, модулі та інструменти, що спрощує створення великих та складних додатків;
- структурованість. Фреймворки визначають структуру додатка, що сприяє організації коду. Це полегшує розуміння, розширення та підтримку додатка, особливо великих проектів;
- масштабованість: Завдяки вбудованій структурі та конвенціям фреймворків, додатки стають більш масштабованими. Це означає, що їх можна легко розширювати та адаптувати для росту в майбутньому;

- односторінкові додатки (SPA). Фреймворки ідеально підходять для створення SPA, які забезпечують більш плавний та динамічний користувацький досвід без необхідності перезавантаження сторінок.
- Безпека. фреймворки зазвичай включають в себе вбудовані заходи безпеки та допомагають управляти аспектами, такими як захист від атак, обробка введення користувача та інші аспекти безпеки.
- Спільнота та документація. Популярні фреймворки мають великі та активні спільноти розробників, які надають підтримку, розробляють доповіді та діляться знаннями. Також, добра документація фреймворку полегшує вивчення та використання.
- Інструментарій. Багато фреймворків інтегруються з іншими інструментами розробки, такими як пакетні менеджери, системи контролю версій, засоби тестування та інші, що робить розробку більш зручною.

Загалом, використання фреймворків у веб-розробці допомагає ефективно вирішувати виклики, пов'язані зі створенням сучасних, масштабованих та безпечних веб-додатків.

Огляд фреймворків

Перед тим як розпочати розробку додатків за використання фреймворків, важливо ознайомитися з основними підходами та передумовами.

Є два основних підходи до використання фреймворків - багатосторінкові програми (MPA) та односторінкові програми (SPA). Обидва підходи мають свої переваги та особливості, і вибір між ними залежить від конкретних вимог та характеристик.

Багатосторінкова програма (MPA):

- кожна URL-адреса відповідає окремій HTML-сторінці;
- сервер повертає окремі сторінки для кожного запиту;
- потрібно додавати імпорт фреймворку до кожної сторінки;
- компоненти фреймворку можуть підсилювати окремі сторінки.

Односторінкова програма (SPA)

- сервер повертає лише одну сторінку для всіх URL-адрес;
- сторінка містить програму JavaScript, яка обробляє весь інтерфейс;
- зміни внесені за допомогою JavaScript, без перезавантаження сторінки;
- дії відбуваються швидше, оскільки не потрібно завантажувати нові сторінки.

Подивимось в порівнянні.

MPA

Використовується, коли кожна сторінка має свої унікальні характеристики.

SPA

Ефективний для додатків, де потрібна висока швидкість та динаміка, і де більше акції відбуваються на одній сторінці. В порівнянні як працюють кожна з сторінок.

MPA: Facebook, де кожна сторінка (новини, профіль, повідомлення) має свою унікальну URL-адресу.

SPA: Gmail, де всі дії здійснюються на одній сторінці без перезавантаження.

Обираючи між цими підходами, важливо враховувати специфіку вашого проекту, його потреби та очікувану взаємодію з користувачем.

Однією з ключових переваг багатосторінкових програм (MPA) є їхні переваги з точки зору пошукової оптимізації (SEO). Дійсно, для веб-сайтів, де важлива індексація окремих сторінок пошуковими системами, MPA може бути вибором, що надає переваги в SEO.

Пошукові системи, такі як Google, індексують вміст веб-сайтів, а SEO-оптимізація допомагає зробити вміст більш доступним та зрозумілим для пошукових алгоритмів. У багатосторінкових програмах, де кожна сторінка має

унікальну URL-адресу та зазвичай генерується сервером для кожного запиту, кожна сторінка може бути індексована окремо.

На прикладі Facebook, важливо, щоб пошукові системи (індексатори) могли знаходити та індексувати різні частини сайту, такі як сторінка новин, профіль користувача чи повідомлення. Це поліпшує видимість веб-сайту у пошукових результатах та дозволяє користувачам знаходити більше відомостей через пошукові запити.

У той час як SPA також можуть використовувати різні стратегії для SEO (наприклад, використання серверного рендерингу або спеціальних практик для покращення індексації), MPA надає більше "стандартного" підходу, де кожна сторінка має свою власну адресу і може бути проіндексована окремо.

Роботам пошукової системи Google важче обробляти сторінки, які містять значну кількість JavaScript-коду, і, зрозуміло, вони не можуть передбачити, як буде динамічно змінюватися кожна частина односторінкової програми. Таким чином, концепція багатосторінкового застосунку (MPA) виявляється більш ефективною, особливо тоді, коли користувач шукає не лише головну сторінку. Ще однією перевагою є те, що цей підхід є традиційнішим, і для нього існують безліч шаблонів, посібників, готових дизайнів та рішень для вирішення проблем веб-безпеки. Також існує значна ймовірність того, що навіть у випадку вимкнення JavaScript користувач все одно отримає значний обсяг інформації на сторінці, що не може бути сказано про програму, яка повністю залежить від коду. Серед недоліків MPA можна відзначити затримку, оскільки перезавантаження сторінки після кожного кліку рідко коли полегшує використання сайту, особливо в мобільних версіях.

Односторінковий додаток видається особливо привабливим завдяки високій швидкості реагування на дії користувача. Очевидно, що відсутність необхідності чекати на відповідь після кожного кліку забезпечує практично миттєву взаємодію, що може залишити чудове враження на користувачів. Також слід відзначити кращий розподіл бізнес-логіки та логіки презентації, оскільки

використання односторінкової програми примушує розробників чітко визначати фронтенд та бекенд дії.

Важливим плюсом є здатність підтримувати веб-ресурс при тимчасовій відсутності Інтернет-підключення, оскільки сторінка завантажується повністю, і лише ті частини, що відповідають за запити до сервера, залишаються неактивними.

Однак основним недоліком є втрата переваги альтернативного підходу - пошукової оптимізації. Якщо це важливо для програми, використання односторінкового підходу може бути більш складним, хоча не неможливим. Зараз існують нові інструменти, які створюють можливість розв'язання цієї проблеми, використовуючи попередньо відрендерені частини програми на сервері, щоб Google індексував те, що бачить користувач. Однак такі технології тільки починають розвиватися, тому оптимізація для цього підходу може бути складнішою.

Також важливо врахувати можливі проблеми безпеки через меншу кількість інформації про цей шаблон, а також факт, що програма не буде працювати без увімкненого JavaScript, хоча відсоток таких користувачів становить менше 2%.

Перед тим як розпочати розробку, важливо визначити передумови кожного з обраних фреймворків з точки зору JavaScript та мови, яку слід використовувати. Vue.js ефективно працює як з ES5, так і з ES6. Завдяки додатковим інструментам у робочому просторі, ви легко можете використовувати всі функції підтримки, такі як розділення коду, мініфікація чи розширені методи. Хоча використання TypeScript теж можливе, воно не офіційно підтримується.

У випадку React.js, він також працює з ES5, але це не є найкращим підходом для розробки програми. Документація React.js менш обширна порівняно з ES6, і принцип "Все є JavaScript" передбачає використання останніх

версій цієї мови. Хоча TypeScript також можна використовувати, він також неофіційно підтримується.

Angular відрізняється тим, що теоретично можна використовувати ES5 або ES6, але він працює краще з TypeScript. Angular використовує багато функцій, таких як класи, типи, інтерфейси та декоратори. Інструмент Angular CLI полегшує налаштування робочого процесу.

При виборі фреймворку також важливо врахувати, хто є розробником обраного інструменту, оскільки це частково вказує на перспективи їхнього розвитку. Angular розроблений внутрішньою командою Google, React створений командою Facebook, а Vue.js – це ініціатива групи розробників, які об'єднались за бажанням приєднатися до ініціативи Евана Ю.

Angular є масштабним фреймворком з багатьма вбудованими функціями, такими як маршрутизація, перевірка форм та обробка HTTP-запитів. Багато користувачів називають його не лише фреймворком, але і платформою через широкий функціонал. Angular має власну екосистему з інтерфейсом командного рядка, прогресивною підтримкою програм і іншими інструментами, розробленими офіційною командою. Такий підхід звільняє розробника від необхідності шукати сторонні пакети, оскільки більшість функціоналу вже вбудовано.

З іншого боку, React приймає мінімалістичний підхід і фокусується на створенні інтерфейсу користувача. React позиціонується як бібліотека, а не повноцінний фреймворк. У нього мінімальна кількість вбудованих функцій, і для реалізації деяких функціональностей, таких як маршрутизація чи обробка HTTP-запитів, розробникам доводиться використовувати додаткові пакети.

Отже, вибір між Angular та React залежить від ваших потреб та стилю розробки. Angular надає велику функціональність "з коробки", тоді як React дає більше свободи в обиранні необхідних інструментів.

Vue є своєрідним компромісом між Angular та React, пропонуючи ряд властивостей, що роблять його середнім варіантом між цими двома фреймворками.

З одного боку, Vue надає вбудовані можливості, які можна знайти у Angular, але не настільки обширні. Наприклад, фреймворк зосереджується на ключових аспектах написання коду, таких як маршрутизація та управління станом програми. В той час як деякі функції, такі як перевірка форм, залишаються на розсуд розробника, дозволяючи вибирати доступні рішення.

З іншого боку, Vue залишає простір для вибору в інших аспектах, подібних React. Це означає, що фреймворк більше орієнтований на написання коду, не завантажений великою кількістю вбудованих функцій. Він також надає потужний інтерфейс командного рядка та кілька вбудованих функцій для зручності розробки.

Можна сказати, що Vue є щось більше, ніж React, але менше, ніж Angular. Цей підхід робить його гнучким і підходить для різних типів розробників та завдань.

Огляд SPA фреймворків

React є бібліотекою JavaScript з відкритим вихідним кодом, розробленою командою Facebook у 2011 році для створення односторінкових програм на стороні клієнта. Цей інструмент зараз активно використовується у продуктах, таких як Instagram, і продовжує розвиватися та підтримуватися Facebook і Instagram.

Основне призначення React - забезпечити високу швидкість, простоту використання та зручний інтерфейс для розробки додатків, які працюють як у браузері, так і на мобільних платформах. Хоча React є бібліотекою функцій, а не повноцінним фреймворком, його можна використовувати для створення мобільних додатків.

Сильні сторони React включають високу швидкість, легкість використання, кросплатформенність і широкою спільноту розробників. Однак

він вимагає додаткових сторонніх бібліотек для вирішення певних завдань, що може ускладнити процес розробки.

Слабкими місцями React є потреба в сторонніх бібліотеках JavaScript, відсутність єдиної стандартизації при написанні коду в HTML і CSS, а також часті оновлення, які можуть застаріти деякі функції. Для покращення SEO-оптимізації та рендерингу на стороні сервера, можна використовувати фреймворк NextJS, який базується на React та використовує технологію SSR.

Vue - створено для розробки інтерфейсу користувача. Перевага Vue полягає в тому, що він дозволяє поступово вводити нові функції, на відміну від Angular або React. Це означає, що ви можете впроваджувати цю структуру поступово, починаючи з конкретних сторінок, що значно спрощує розробку. Vue.js розроблено Іваном Ю. Vue широко використовується серед китайських компаній, наприклад, Alibaba, Baidu, Xiaomi тощо. Нещодавно система керування репозиторієм GitLab також перейшла на Vue.js.

Vue в основному спрямований на обробку завдань на рівні перегляду, що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. Цей фреймворк вдало поєднує переваги Angular і React, пропонуючи швидкість, легкість використання та підтримку таких технологій, як TypeScript і JSX.

Однак Vue залишається відданим стандартам написання коду на HTML і CSS, що робить процес розробки та підтримки проекту більш простим і зрозумілим для розробників, які вже знають основи клієнтських технологій.

До сильних сторін Vue слід віднести швидкість, легкість використання та здатність до інтеграції з іншими інструментами. Зазвичай, Vue є привабливим вибором для тих, хто шукає баланс між функціональністю і простотою.

Однією зі слабких сторін Vue може бути не дуже велика спільнота порівняно з Angular або React. Тим не менше, популярність Vue зростає, і цей фреймворк отримує все більше уваги та підтримки з боку розробників.

Angular - це кросплатформенний фреймворк, який використовує шаблони та імплементує концепцію архітектурного шаблону MVC (Модель-Вид-

Контролер). Він сприяє створенню програм, які мають слабкий зв'язок між презентацією, даними та логікою компонентів, що сприяє полегшенню розробки та розвантаженню деяких служб сервера, переносячи їх на сторону клієнта.

Angular активно використовується в крупних компаніях, де розробляється значна кількість коду. Використання TypeScript у проекті дозволяє зробити код більш чистим і легким для розуміння, а також зменшити кількість помилок.

До переваг Angular слід віднести відмінну документацію, підтримку від Google та наявність розширеного набору інструментів розробки, таких як Material UI, CLI та інші.

Однак однією зі слабких сторін Angular є високий поріг входу для розробників, оскільки для роботи з фреймворком потрібно мати знання TypeScript. Це може ускладнити розвиток проектів, особливо при їх передачі від однієї команди до іншої. Ще однією проблемою є частий випуск нових версій Angular, що може ускладнювати підтримку проектів у довгостроковій перспективі.

AngularJS - це фреймворк для розробки додатків JavaScript з відкритим вихідним кодом, створений компанією Google. Слід зауважити, що його наступні версії відомі як Angular 2 та інші. AngularJS став першим фреймворком, що дозволив розробникам створювати інтерактивні веб-сайти та односторінкові програми, які мають відмінну реактивність та інтуїтивний інтерфейс користувача.

Випущений у 2010 році, AngularJS отримав широку популярність та підтримку завдяки можливості перетворювати статичні сторінки HTML в інтерактивні. Однак з часом з'явилися інші фреймворки, такі як ReactJS і VueJS, які вирізняли недоліки AngularJS. У відповідь на цю конкуренцію Google вирішив здійснити повний перезавантаження і перейшов від Angular 1 до Angular 2, використовуючи TypeScript як нову мову програмування.

Перехід від JavaScript до TypeScript був здійснений для уникнення помилок, характерних для JavaScript, і впровадження статичних типів, що було

важливим для багатьох розробників веб-додатків.

З Angular 2 багато чого змінилося. Компанія змінила парадигму, яку використовував AngularJS, не лише мову, але й архітектуру та підхід до зв’язування даних. Однак і AngularJS, і Angular все ще використовуються розробниками та веб-розробниками відповідно до їхніх вимог.

Основні фактори, які відрізняються між Angular і AngularJS

Angular 2 вніс значні зміни порівняно з AngularJS, охоплюючи не лише мову програмування, а й архітектурні та концептуальні аспекти. Основні відмінності між Angular і AngularJS включають:

Мова програмування

AngularJS Використовує JavaScript та ECMAScript 5;

Angular Використовує TypeScript, який є строго типізованим суперсетом JavaScript.

TypeScript дозволяє розробникам використовувати статичні типи та інші функції, щоб полегшити розробку та зменшити кількість помилок.

Архітектурний паттерн

AngularJS Використовує контролери та $scope для зв'язку даних між представленням та моделлю.

Angular Використовує компонентний підхід, де додаток розбивається на компоненти, які мають свої власні представлення, логіку та стилі. Зв'язок даних відбувається через властивості та події.

Зв’язування даних

AngularJS Використовує двостороннє зв'язування даних з допомогою директив ng-model;

Angular Застосовує одностороннє зв'язування даних, де дані течуть з моделі до представлення або від представлення до моделі.

Система модулів:

- AngularJS Має менш потужну систему модулів;
- Angular Використовує більш сучасну та гнучку систему модулів, що

дозволяє ефективно організовувати код.

Швидкість та продуктивність

AngularJS Може виявити проблеми з продуктивністю при роботі з великою кількістю даних через його двостороннє зв'язування;

Angular Вдосконалено швидкість та продуктивність завдяки застосуванню Virtual DOM та іншим оптимізаціям.

Обидва фреймворки використовуються в різних проектах, залежно від потреб розробників та характеристик проекту.

AngularJS і Angular використовують різні підходи до архітектури та зв'язування даних. AngularJS базується на архітектурі Model View Controller (MVC), де розробник розміщує бізнес-логіку в моделі, представленні та контролері, і AngularJS обробляє обчислення для отримання результату. Основні конструкції будуються за допомогою компонентів та директив в AngularJS, де компоненти є модернізованими директивами і надають структуру додаткам, полегшуючи створення та підтримку великих програм.

AngularJS використовує JavaScript, тоді як Angular від Angular 2 і далі використовує TypeScript, розширення JavaScript, що додає статичні типи до розробки. Використання статичних типів дозволяє уникнути помилок та покращує продуктивність, що є важливим для великих і складних програм.

Щодо ін'єкції залежностей (DI), обидва фреймворки використовують DI, але Angular та AngularJS реалізують його в різний спосіб. AngularJS впроваджує DI в функції зв'язку, функції контролера та визначення директив, тоді як Angular використовує ієрархічну систему введення залежностей з деклараціями, функціями конструктора та постачальниками.

У Angular 2 і вище існує власний інтерфейс командного рядка (CLI), призначений для генерації компонентів, служб і т.д. Це спрощує розробку та компіляцію додатків. У AngularJS немає власного інтерфейсу командного рядка.

За використанням технології прив'язки даних Angular виявляється більш інтуїтивно зрозумілим порівняно з AngularJS. Angular використовує () для

прив'язки подій та для прив'язки властивостей, що спрощує розробку.

Angular є набагато швидшим порівняно з AngularJS завдяки однонаправленому потоку даних, що забезпечується потоковою архітектурою, що робить програми більш ефективними.

2.2 Обгрунтування вибору методів розробки

AngularJS був розроблений з метою спрощення складних процесів створення та управління JavaScript-програмами. Оснований на структурі MVC, цей фреймворк стає особливо корисним для розробки односторінкових веб-сайтів. Використовуючи простий JavaScript і HTML, AngularJS автоматизує операції з DOM і запитами AJAX, звільняючи розробників від необхідності написання цього коду вручну. Легко і швидко інтегрується в будь-яку HTML-сторінку за допомогою модульних блоків коду JS, які можна легко комбінувати та тестувати.

AngularJS забезпечує ефективний контроль над клієнтськими даними. Розробнику не потрібно переймати вибірку елементів з DOM та їхнє заповнення значеннями, оскільки зв'язування даних дозволяє оновлювати дані в частині JavaScript та відстежувати зміни в частині HTML. Це працює в обидві сторони: зміни в JavaScript автоматично відображаються в HTML і навпаки. Крім того, фреймворк активно використовує інжектори, включаючи попередньо визначені класи для виконання AJAX-запитів та управління маршрутами.

AngularJS виділяється серед конкурентів з кількох причин.

Спрощена двостороння прив'язка даних. AngularJS дозволяє легко прив'язувати дані до HTML за допомогою виразів, а також розширювати функціонал HTML за допомогою директив. Маніпуляції з DOM та код для прив'язки даних знаходяться в простих елементах, які можна швидко та легко вставляти в HTML-шаблони.

Універсальність. AngularJS спроектований як універсальний фреймворк,

що дозволяє створювати веб-додатки різних типів. Це особливо корисно при розробці односторінкових веб-додатків. Участь у пакеті MEAN: AngularJS входить до складу пакета MEAN, що також включає MongoDB, Express.js і Node.js.

Таким чином, якщо клієнтська та серверна частини проекту виконуються за допомогою JS, то весь стек технологій MEAN може бути використаний. Також AngularJS добре взаємодіє з ASP.NET і C#. Фокус на функціональності. AngularJS побудований за принципом "функціональність на перше", що робить його ідеальним для розробки зверху вниз. Модульна концепція AngularJS дозволяє легше поділити роботу між різними командами у великих проектах. Це призводить до компактних і легко редагованих AngularJS-додатків.

При виборі AngularJS для проекту важливо враховувати кілька ключових аспектів, що допоможуть ефективно спроектувати логіку додатку та вирішувати завдання:

Структура інструментів. AngularJS нав'язує свою структуру розробникам, що може бути корисним для початківців, але менш гнучким для тих, хто шукає більше гнучкі рішення.

Зайвість для деяких проектів. Для деяких проектів AngularJS може бути зайвим, особливо для легших фреймворків, що призначені для статичних сайтів. Може виникнути проблема обробки маніпуляцій з DOM при великій кількості даних.

Обмеження продуктивності. AngularJS може мати обмеження продуктивності при використанні високонавантажених галерей фото чи в деяких GUI редакторах.

Швидкість додатку. Щоб поліпшити швидкість додатків на AngularJS, слід звертати увагу на дайджест цикл та кількість відстежень.

Використання одноразових прив'язок і мінімізація відстежень можуть поліпшити продуктивність.

Доступ до DOM може бути дорогим, тому рекомендується зберігати

маленький розмір дерев DOM та уникати змін без реальної необхідності.

Області видимості і збірка сміття. Важливо слідкувати за областями видимості та не надавати змінним занадто велику область видимості, щоб уникнути затримок від збірки сміття JS.

Ці поради допоможуть оптимізувати та покращити швидкість додатків на AngularJS, роблячи їх більш продуктивними та ефективними.

AngularJS і Node.js виявляються відмінними технологіями, які можна успішно комбінувати для створення ефективних веб-додатків.

AngularJS і Node.js. AngularJS добре взаємодіє з Node.js, яке є асинхронним середовищем JavaScript.

Node.js використовує асинхронну модель подій для створення масштабованих мережевих програм, що робить його ідеальним для взаємодії з AngularJS.

Модель Node.js. У порівнянні з традиційною моделлю, яка використовує паралельні потоки ОС, Node.js використовує асинхронний підхід і переходить у режим сну, коли з'єднань немає.

Цей підхід дозволяє створювати ефективні, необтяжливі системи, оскільки немає блокувань процесів.

Angular і Node.js. Нові версії Angular використовуються частіше через свою стабільність, багато модулів, CLI та інші переваги.

Вони використовують TypeScript, що поліпшує структуру коду та полегшує тестування.

Переваги Angular:

-використання бібліотеки RxJS для реактивного програмування та observables для асинхронного коду.

-використання Dependency Injection для розділення структури коду.

-наявність Angular CLI для зручної генерації та підтримки додатків.

Недоліки Angular:

-значна кількість коду та складний поріг входу в фреймворк.

-об'єднання AngularJS і Node.js в додатку може стати вдалим вибором для розробки високопродуктивних та ефективних веб-додатків.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання на магістерську роботу передбачено впровадження програмного забезпечення, яке демонструватиме роботу сучасних веб-фреймворків.

У процесі створення магістерської роботи необхідно виконати наступний обсяг робіт:

а) провести аналіз існуючих аналогічних систем з метою виявлення їх позитивних і негативних якостей. Результати аналізу будуть враховані при подальшій розробці;

б) вибрати та обґрунтувати спосіб побудови системи контролю за роботою технологічного обладнання виробництва в автоматизованому режимі. Розробити функціональні та структурні схеми системи;

в) розробити системне програмне забезпечення, яке дозволить реалізувати поставлене технічним завданням завдання. Поудувати блок-схеми програмних алгоритмів і підпрограм;

г) організувати інтерфейс користувача з метою формування та відображення повідомлень про некоректні дії користувача та нестандартні ситуації;

г) надати рекомендації щодо організаційно-методичних заходів, які забезпечать введення системи в промислову експлуатацію та її подальше успішне функціонування;

г) виконати розрахунки для визначення економічної ефективності розробленої системи;

є) розробити заходи з охорони праці під час впровадження та експлуатації системи, а також розробити заходи цивільного захисту;

з) сформувати висновки про обсяги виконаної роботи та отримані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Веб-фреймворки Angular і AngularJS використовують шаблон проектування Model-View-Controller (MVC), що розбиває програму на компоненти для поліпшення розробки та підтримки. Цей шаблон також є основним у багатьох серверних фреймворках, таких як Spring MVC, NestJS, SailsJS, ASP.NET MVC Framework та інші.

Основні компоненти шаблону MVC включають:

- Model (Модель). Відповідає за зберігання даних користувача. Надає необхідний інтерфейс для доступу до цих даних;
- View (Представлення). Відповідає за відображення даних користувачу та забезпечує форму та стиль представлення даних;
- Controller (Контролер). Містить бізнес-логіку додатка. Реагує на події в браузері, викликані користувачем. Повідомляє модель про оновлення даних.

Робота шаблону MVC включає такі етапи:

- взаємодія з користувачем. Контролер відстежує дії користувача, такі як введення даних чи натискання кнопок;
- обробка подій. Контролер викликає функції або генерує запит, що взаємодіє з компонентом Model для отримання або оновлення даних;
- оновлення View. За допомогою компонента View дані передаються користувачеві в потрібному вигляді та з формою, зазначеною в представленні.

Ця архітектура дозволяє робити компоненти незалежними один від одного, а взаємодія з контролером спричиняє зміни в базі даних і оновлення даних для користувача. Шаблон MVC допомагає розділити логіку програми на окремі компоненти, зменшуючи вплив змін або додавання нових елементів на інші частини системи.

Існують різні модифікації шаблону MVC, серед найпоширеніших з яких

виділяють пасивну модель та активну модель. У пасивній моделі контролер відстежує зміни в моделі і відповідає за їх відображення користувачеві при оновленні даних. У випадку активної моделі сама модель повідомляє шаблон про оновлення своїх даних за допомогою системи сповіщень, надаючи більшу незалежність між моделями, контролерами та шаблонами.

З використанням сучасних технологій, таких як SPA (односторінкові додатки) та PVA (прогресивні веб-додатки), взаємодія з користувачем може стати більш інформативною та зручною. В цьому контексті фреймворки, такі як Angular, AngularJS, React та інші, відіграють ключову роль.

AngularJS вимагає від розробників впровадження нативного підходу MVC у своїх веб-додатках, надаючи необхідні сервіси та технології для реалізації залежностей. Це сприяє створенню додатків із чіткою структурою, високоякісною підтримкою та легкою перевіркою на наявність помилок.

Основні технологічні можливості AngularJS включають в себе сервіси, модулі, фільтри, директиви та області застосування. Директиви, зокрема, використовуються для розширення можливостей HTML та змінюють DOM під час компіляції, що дозволяє додавати нову поведінку або змінювати існуючу.

Також важливо відзначити ієрархічну структуру в AngularJS, представлену областями, які використовуються для передачі даних між контролером і видом. Ці області схожі на DOM, успадковуючи властивості своїх батьківських доменів, і можуть бути використані для відстеження та оновлення даних за допомогою директиви $watch.

Вибір фреймворку або бібліотеки для розробки веб-додатків, таких як Angular, React або Vue, визначається великою кількістю факторів, і не існує універсально правильної відповіді. Ваш вибір буде залежати від конкретних потреб вашого проекту, ваших умінь і досвіду, а також особистих вподобань.

Angular

- використовуйте Angular, якщо вам потрібен фреймворк, який надає повний стек рішень для розробки веб-додатків;

- добре підходить для великих і складних проектів, де потрібна велика структура і висока масштабованість.
- якщо ви вже володієте досвідом роботи з Java або C#, Angular може бути більш зрозумілим, оскільки він використовує мову TypeScript.

React

- використовуйте React, якщо вам потрібна бібліотека для побудови інтерфейсу користувача, і ви хочете мати більше гнучкості в виборі інших частин технологічного стеку;
- добре підходить для односторінкових додатків, де акцент робиться на відображенні даних і взаємодії з користувачем;
- має велику і активну спільноту, що сприяє наявності багатьох сторонніх бібліотек і рішень.

Vue

- використовуйте Vue, якщо ви хочете легко внедрити бібліотеку в існуючий проект або почати новий проект без значного навчання;
- добре підходить для невеликих і середніх проектів, де простота і швидкість розробки є ключовими факторами;
- має дружній API та докладну документацію, що робить його доступним для розробників на різних рівнях досвіду.

Важливо також враховувати власний досвід та командні вподобання. Якщо у вас або вашої команди вже є досвід роботи з конкретним інструментом, це може бути важливим фактором вибору.

Справжньо, AngularJS використовує концепцію служб та фільтрів для виконання конкретних завдань у веб-додатках. Давайте розглянемо кожну з цих концепцій більш детально:

Служби (Services):

- призначення: Служби в AngularJS використовуються для виконання конкретних завдань або надання певних функціональностей. Наприклад, служба $http використовується для надсилання HTTP-запитів;

- реалізація: Для створення служби в AngularJS використовується шаблон singleton. Служби можна викликати з будь-якої точки програми, оскільки існує лише один екземпляр служби для всього проекту;
- використання: Служби можна впроваджувати в контролери, директиви та інші складові AngularJS для спільного використання логіки та даних.

Фільтри (Filters)

- призначення: Фільтри в AngularJS використовуються для фільтрації та форматування даних перед їх відображенням користувачеві. Вони також можуть застосовуватися до колекцій, масивів та інших типів даних;
- ралізація: Фільтри визначаються як функції, які можна легко викликати в шаблоні для обробки даних перед їх відображенням;
- використання: Фільтри можна використовувати разом з директивами та виразами у шаблонах для досягнення певного вигляду або формату відображення даних.

Важливо відзначити, що AngularJS використовує шаблон MVC, але його структура може бути трошки відмінною. Модульність є ключовою особливістю Angular, і розбиття додатка на окремі модулі, такі як компоненти, служби та директиви, сприяє кращій організації та управлінню кодом.

Angular 2.0 та пізніші версії представляють суттєве вдосконалення порівняно з AngularJS та додають багато нових можливостей та оптимізацій. Ось кілька ключових різниць та переваг, які важливо враховувати при порівнянні Angular та AngularJS.

Компонентна архітектура:

- AngularJS: Використовує контролери та $scope для роботи з представленням та логікою.
- Angular: Використовує компоненти як основні будівельні блоки додатків. Компоненти мають власний стан, логіку та представлення.

Мова програмування:

- AngularJS: Основна мова - JavaScript (або ECMAScript 5).
- Angular: Рекомендується використовувати TypeScript, який додає статичну типізацію та інші функції.

Декларативне програмування:

- AngularJS: Використовує імперативний підхід до програмування.
- Angular: Зосереджено на декларативному програмуванні та використовує шаблони та директиви.

Компіляція

- AngularJS: Код виконується безпосередньо у браузері.
- Angular: Використовує компілятор Angular Ivy, що оптимізує та компілює код для ефективнішої роботи у браузері.

Мобільна розробка

- AngularJS: Не має спеціальних засобів для мобільної розробки.
- Angular: Має функціонал для розробки мобільних додатків за допомогою Angular Mobile Toolkit та Ionic.

Розширення

- AngularJS: Має менше можливостей для розширення.
- Angular: Забезпечує більше можливостей для розширення та використовує Angular CLI для автоматизації задач розробки.

Швидкодія

- AngularJS: Може виникати проблеми з продуктивністю при обробці великих обсягів даних.
- Angular: Завдяки вдосконаленим алгоритмам та компіляції Ivy має кращу продуктивність, особливо великих проектах.

Тестування

- AngularJS: Вимагає специфічних підходів до тестування.
- Angular: Має удосконалену систему для тестування, що полегшує створення та виконання тестів.

Отже, вибір між AngularJS та Angular залежить від конкретних потреб проекту, вимог до продуктивності, масштабування та рівня сучасних можливостей, які вам необхідні. Angular є більш сучасним та ефективним фреймворком, але використання AngularJS може бути виправданим для деяких легасі проектів або задач.

3.2 Розробка структурної схеми

В межах програми та взаємодію між різними компонентами. Структурні схеми, такі як діаграми блоків чи ієрархічні діаграми, можуть значно полегшити розуміння та комунікацію між членами розробчого колективу.

Структурні схеми можуть мати різні рівні деталізації, що дозволяє розглядати програму як на високому, так і на деталізованому рівні. Основні компоненти програми, такі як підсистеми, бази даних, бібліотеки, можуть бути представлені як блоки на схемі, а їхні взаємозв'язки показані стрілками або лініями зв'язку.

Під час розробки структурної схеми важливо враховувати розділення програми на логічні блоки, що робить її структуру зрозумілою та піддається управлінню. Взаємодія та обмін інформацією між цими блоками також може бути відображена на схемі, надаючи загальний огляд архітектури програми.

Видача розшифровки функцій та пояснення того, за що відповідають модулі, допомагає розробникам та іншим учасникам проекту краще розуміти програмну систему та полегшує її подальше управління та розвиток.

У цій структурній схемі зовнішні специфікації, зв'язки між модулями та інші ключові елементи надають важливу інформацію, необхідну для розуміння функціональної структури програми.

На структурній схемі (рисунок 3.1) вказано зовнішні специфікації програми, які дають розуміння функціональної частини програми, за що вони відповідають, як взаємодіють із вхідними та вихідними даними. Особливо важливо розуміти тип структури даних.

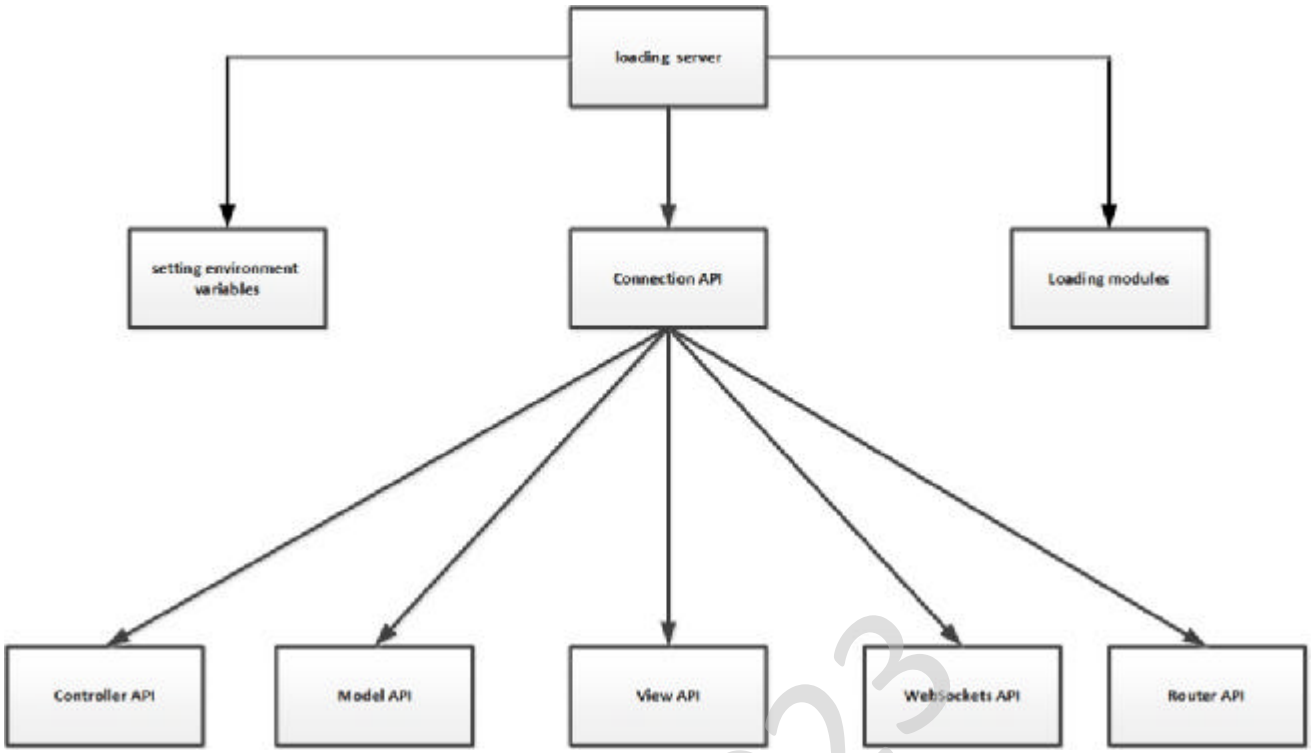

Рисунок 3.1 - Структурна схема

Підхід до поділу програми на модулі за принципом "від загального до конкретного", де окремі модулі виступають як сервіси, є розумним. Це дозволяє створити ієрархію, яка є легко зрозумілою та підтримуваною. Принципи створення модульних програм і використання сервісів для реалізації окремих завдань сприяють якості коду і його перевірці.

Структурна схема, яка відображає основні та вторинні блоки програми та їх взаємодію, дійсно полегшує розуміння роботи програми і її підтримку в майбутньому. Це важливий етап в розробці, який може значно полегшити співпрацю команди розробників та удосконалити обслуговування програми в подальшому.

3.3 Розробка функціональної схеми

Функціональна схема, або схема даних, є важливим інструментом у розробці програмного забезпечення. Вона надає візуальне відображення взаємодії компонентів програми та показує, як дані передаються та обробляються.

Основна ідея функціональної схеми полягає в тому, щоб відобразити функції, які виконуються в системі, та їх взаємодію. Вона містить інформацію про дані, які обробляються кожним компонентом, і вказує на вхідні та вихідні дані.

Стандартні позначення використовуються для розробки функціональних схем. Це може включати блоки для компонентів, стрілки для позначення потоку даних, та інші символи для визначення конкретних операцій чи функцій.

Функціональна схема допомагає відстежувати вимоги та функції програми, визначати взаємодію між компонентами та контролювати обмін даними. Вона є важливим інструментом для розробників та команд розробки, дозволяючи їм краще розуміти логіку системи та приймати участь у розвитку та підтримці програмного забезпечення.

Функціональні схеми грають ключову роль в розумінні та візуалізації взаємодії компонентів програми. При їх розгляді можна вивчати, як взаємодіють між собою клієнтська та серверна частини програми, а також які бібліотеки та технології використовуються для зберігання та передачі даних.

Такі схеми допомагають не лише зрозуміти, як працює алгоритм, а й визначити його складність. Їх використання особливо актуальне при аналізі та вдосконаленні алгоритмів, де кожен блок представляє певну функцію чи дію.

Функціональні схеми становлять ефективний спосіб візуалізації та аналізу роботи програми, допомагаючи розробникам та інженерам краще розуміти всі аспекти їхнього програмного забезпечення. Використання стандартних позначень та національних стандартів забезпечує уніфікований підхід до розробки та сприяє ефективній комунікації в команді розробників.

Функціональні схеми, будь то принципові схеми чи змішані структури, є ефективним інструментом для візуалізації та розуміння призначення пристроїв чи програмних систем.

На схемі (рисунок 3.2) вказано:

- Умовне позначення у прямокутнику, якщо це функціональна частина

Умовне графічне позначення, яке відображає позицію та дію функціональної частини.

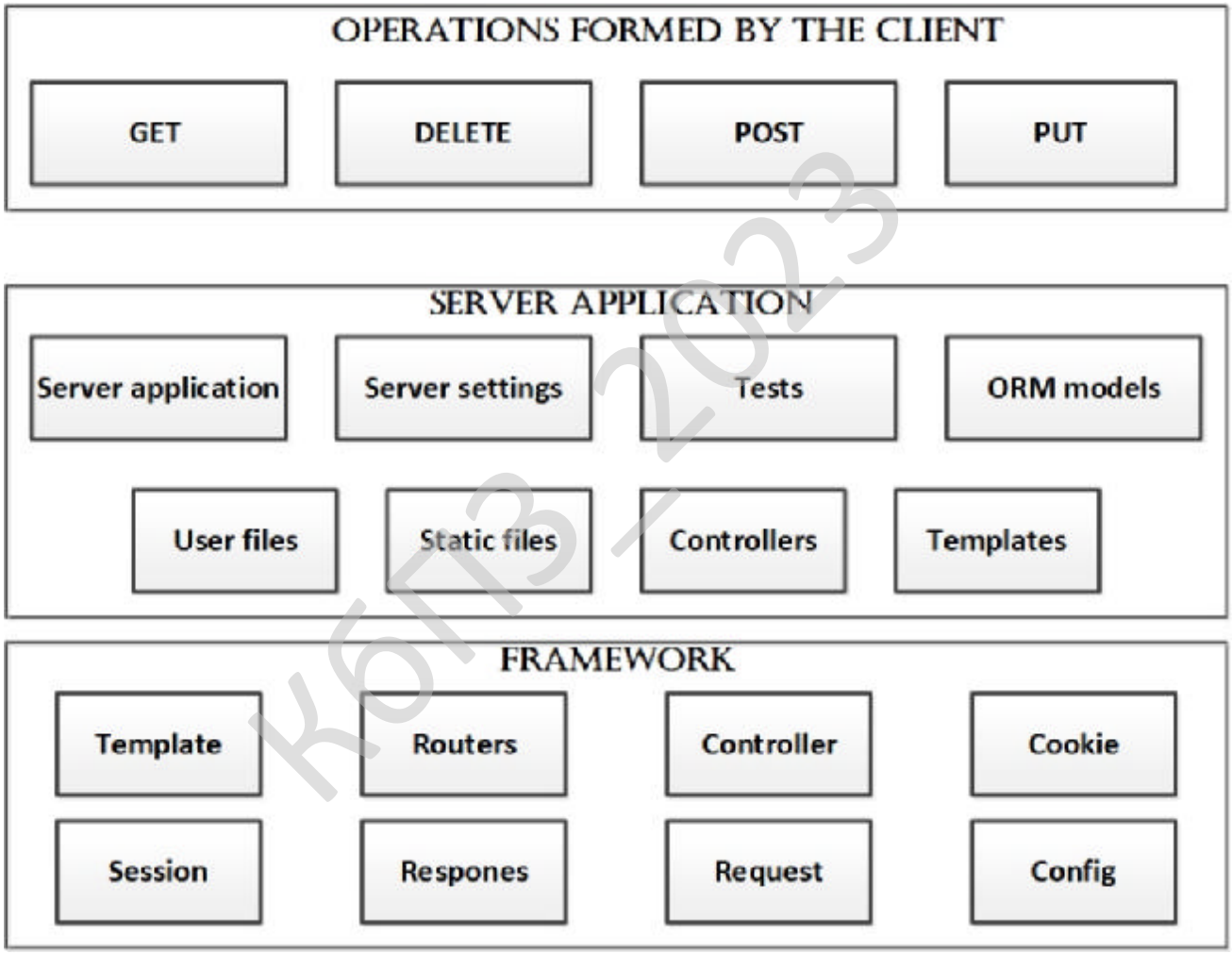

Рисунок 3.2 - Функціональна схема системи

У випадках, коли використовується принципова схема, функціональні частини разом із позначеннями розташування компонентів мають бути подібними до схеми. У таких випадках список елементів не використовується

через використання цих принципових схем. Коли функціональна схема розробляється без використання схеми, для представлення функціональних частин використовуються загальні правила.

3.4 Розробка діаграми процесів

Діаграма процесу графічно ілюструє взаємодію різних етапів або операцій в програмній системі. Найчастіше такі діаграми використовуються для візуалізації послідовності виконання задач, обробки даних та потоків управління в програмі. Давайте розглянемо основні елементи та принципи побудови діаграми процесу.

Елементи діаграми процесу:

- процес (операція). Кожен процес або операція відображає окремий етап виконання програми. Це може бути конкретна функція, задача чи операція;
- стрілки (переходи). Стрілки показують порядок виконання операцій і напрямок потоку управління в програмі. Вони вказують, яка операція виконується після іншої;
- рішення (рішення або умова. Це блок, що визначає рішення або умову, яка визначає, який шлях вибрати в процесі виконання програми. Зазвичай позначається ромбом;
- початок і завершення. Стандартні символи для вказання початку та завершення діаграми. Початок може бути позначений кругом, а завершення - овалом.

Принципи побудови

Послідовність операцій

- діаграма повинна чітко показувати послідовність виконання операцій. Кожен блок повинен слідувати за попереднім логічно та зрозуміло;
- умови та розгалуження. Використовуйте блоки рішень для відображення умов та розгалужень в програмі. Це може включати конструкції типу "if-else" або інші рішення;

- лаконічність. Діаграма повинна бути лаконічною та зрозумілою. Уникайте перевантаження деталями;

- діаграми процесу є потужним інструментом для візуалізації та розуміння роботи програми, сприяють уточненню та оптимізації логіки програмного забезпечення;

Основні компоненти та концепції, які зазвичай представлені на таких діаграмах:

- об’єкти (процеси). Ваші об’єкти на діаграмі відображають процеси чи фази програми. Наприклад, "Користувач робить запит," "Запит потрапляє на сервер," "Контролер обробляє дані," "Отримання даних з бази даних," і "Відповідь користувачеві.";

- стрілки (переходи). Стрілки вказують напрямок потоку управління між процесами. Вони показують, як управління переходить від одного об’єкта до іншого. Наприклад, "Користувач робить запит" переходить до "Запит потрапляє на сервер.";

- фази програми Об'єкти, які представляють різні фази програми, найчастіше розташовані в порядку виконання. Це дозволяє відслідковувати послідовність дій в програмі;

- структурні діаграми. Як ви вірно вказали, існують також структурні діаграми, які описують статичність даних та зв'язки між об'єктами. Ці діаграми часто використовуються для відображення структури системи, включаючи класи, об'єкти, або інші складові.

- схеми процесів та структури даних. Використання обох видів діаграм (процесів і структури даних) надає повніший огляд програми, оскільки вони охоплюють як динаміку, так і статику.

Щоб використовувати такі діаграми, важливо тримати їх чіткими, лаконічними та зрозумілими для аудиторії, для якої вони призначені. Ці діаграми допомагають розробникам і іншим зацікавленим сторонам краще зрозуміти логіку та взаємодію компонентів програми.

Важливо мати чітке розуміння послідовності дій та логіки програми для досягнення потрібного результату. Особливо важливо враховувати динаміку взаємодії між об'єктами та процесами, щоб уникнути конфліктів та забезпечити правильне виконання програми.

Метод покрокового алгоритму для побудови схем — це чудовий вибір, оскільки він дозволяє структурувати та розкривати послідовність дій в програмі. За допомогою такого методу можна легко передавати інформацію про логіку та взаємодію між компонентами.

Також важливо зауважити, що ви врахували різні аспекти, такі як взаємодія з базою даних, обробка запитів користувачів, розподіл ролей між серверною та клієнтською частинами, що є ключовими компонентами багатьох програм.

Ваша здатність вибирати оптимальні дизайнерські рішення і вирішувати завдання проекту говорить про професіоналізм у розробці програмного забезпечення.

Детальне розглядання та аналіз динаміки програми допомагають не тільки зрозуміти її роботу, але і забезпечити її ефективність та надійність.

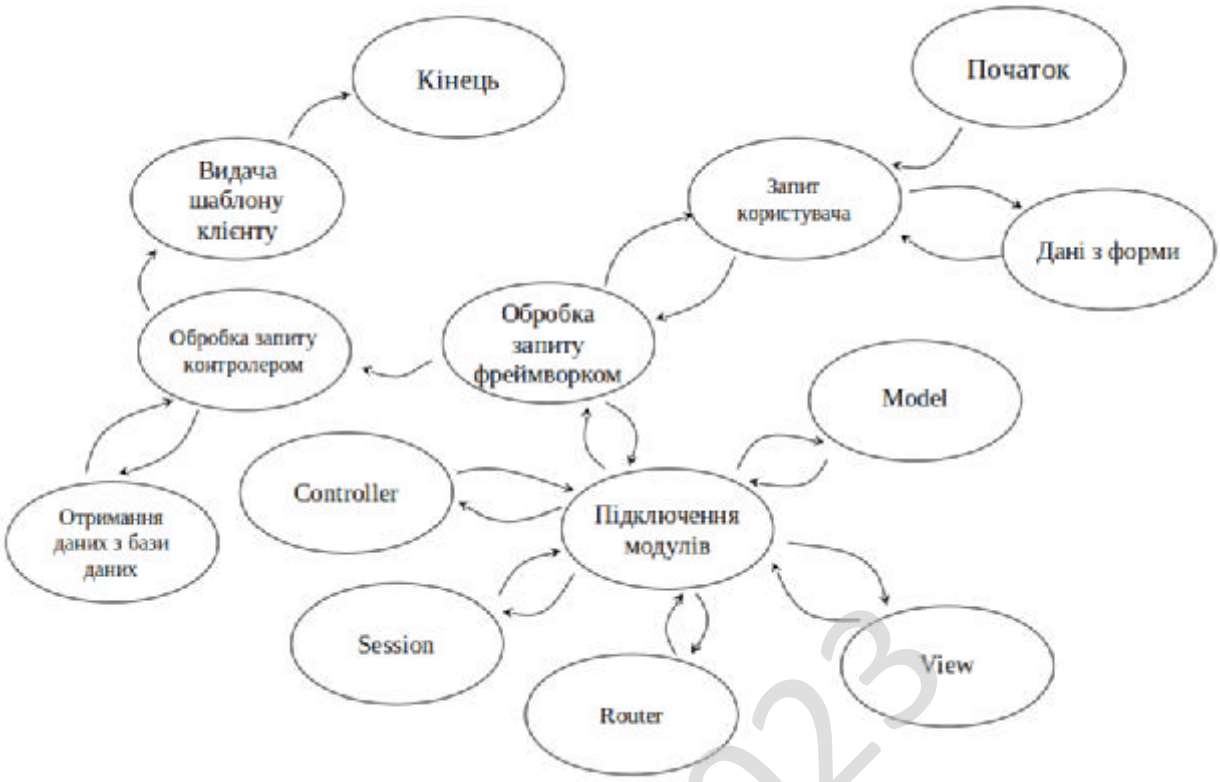

Рисунок 3.3 – Діаграма процесів системи

Розроблена діаграма дає представлення роботи додатку, як клієнтської так і серверної частини. Демонструє як взаємодіють між собою окремі частини додатку, та який результат отримує користувач при роботі з програмним забезпеченням.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Блок-схема надає можливість за допомогою графічних блоків та елементів, представити роботу додатку у зручній для сприйняття формі. За допомогою графічних блоків можна представити потоки, операції, дані, як вони обробляються та модифікуються. Дані які використовуються для вводу або виводу інформації прийнято позначати паралелограмом, дані які обробляються та модифікуються — прямокутником, графічний блок для прийняття рішення за допомогою заданої умови - ромбом, початок та кінець алгоритму — еліпсом. Всі ці графічні позначення використовуються згідно стандарту ДСТУ ISO 5807:2016. Використовуючи схему зручно простежити як виконується програма, з якими даними повинна працювати, та як вони послідовно змінюють свій стан. Це допомогає зрозуміти головне призначення програми, та її функціонал.

Але при створенні блок-схеми для додатка на AngularJS та NodeJS важливо брати до уваги його асинхронну модель, яка схоже працює як в браузері і NodeJS. Ця модель заснована на обробці подій в неблокуючому режимі та використанні обробників зворотних викликів. Це забезпечує виконання окремих фунуцій тільки у випадку виклику спеціальних сигналів, які виконуються викликом зворотних функцій або викликом слухачів подій, наприклад можна звернутися до бази даних, щоб отримати результат запиту, коли відповідь буде готова, тоді буде викликана функція зворотного виклику з отриманим результатом. Такі функції допомагають отримати результат, обробити його не витрачаючи зайві ресурси. Слухачі подій виконують таку саму функцію, як і зворотні функції, але є більш читабельні. Прикладом такої події може бути

натискання клавіші в браузері, на сервері це може бути HTTP запит. Для створення власних прослуховувачів подій, добре підходить EventEmitter — це клас, який можна використовувати при наслідуванні, та для обробки власних подій.

При використанні асинхронних подій, слід уважно стежити за послідовністю виконання операцій в алгоритмі, а також за даними і їх змінними в циклах або в операторах вибору, оскільки це може спричинити проблеми в роботі логіки програми. Під час виконання асинхронного коду, деякі частини коду можуть виконуватися незалежно від іншої частини програми, або виконуватися тільки до або після певної частини коду. Для цього існує концепція (flow control), яка слідкує за асинхронними операціями, розбиває їх на черги, об'єднує в групи, та слідкує за правилами оптимізації такого коду.

Асинхронні операції можуть виконуватися послідовно або паралельно, також важливо чи потрібні отриманні дані для передачі в наступні функції або чи важливий порядок виконання асинхронних функцій. Такі випадки можуть спричинити велику кількість залежного і важко зрозумілого коду, але на сьогодні існують різні технології для їх обробки. Наприклад технологія async-await, проміси. Нові ж версії Angular використовують бібліотеку RxJS, яка реалізує концепції реактивного програмування і використовуючи тип Observable спрощує використання асинхронного коду. Observables має велику функціональність і здатен оброблювати велику кількість потоків даних: відповіді сервера, примітивні типи, синхронний і асинхронний код.

Node це платформа, яка реалізує асинхронну подієво-керовану модель робот з даними, це схоже на роботу JavaScript в браузері. В обох цих середовищах використовується цикл подій, який не блокує виконання коду під час операцій вводу-виводу. У середовищі NodeJS варто пам'ятати про клас EventEmitter, який можна отримати зі стандартного модуля events, він використовується в багатьох стандартних модулях, наприклад для обробки HTTP запитів, або для роботи з потоками. У створених модулях, за допомогою нього

зручно обробляти подію «error», та додавати нові події за допомогою «newListener», та видаляти за допомогою «removeListener». А за допомогою on і emit можна створювати власні події, де функція on прив'язує функцію, а emit запускає подію. Цей клас активно використовувався при розробці чата на веб-сокетах.

На блок-схемі (рисунок 4.1) можна простежити послідовність виконання дій розробленого клієнтського додатка, як в сукупності завантажується в браузері, як він приймає дії користувача, обробляє їх та надсилає на сервер.

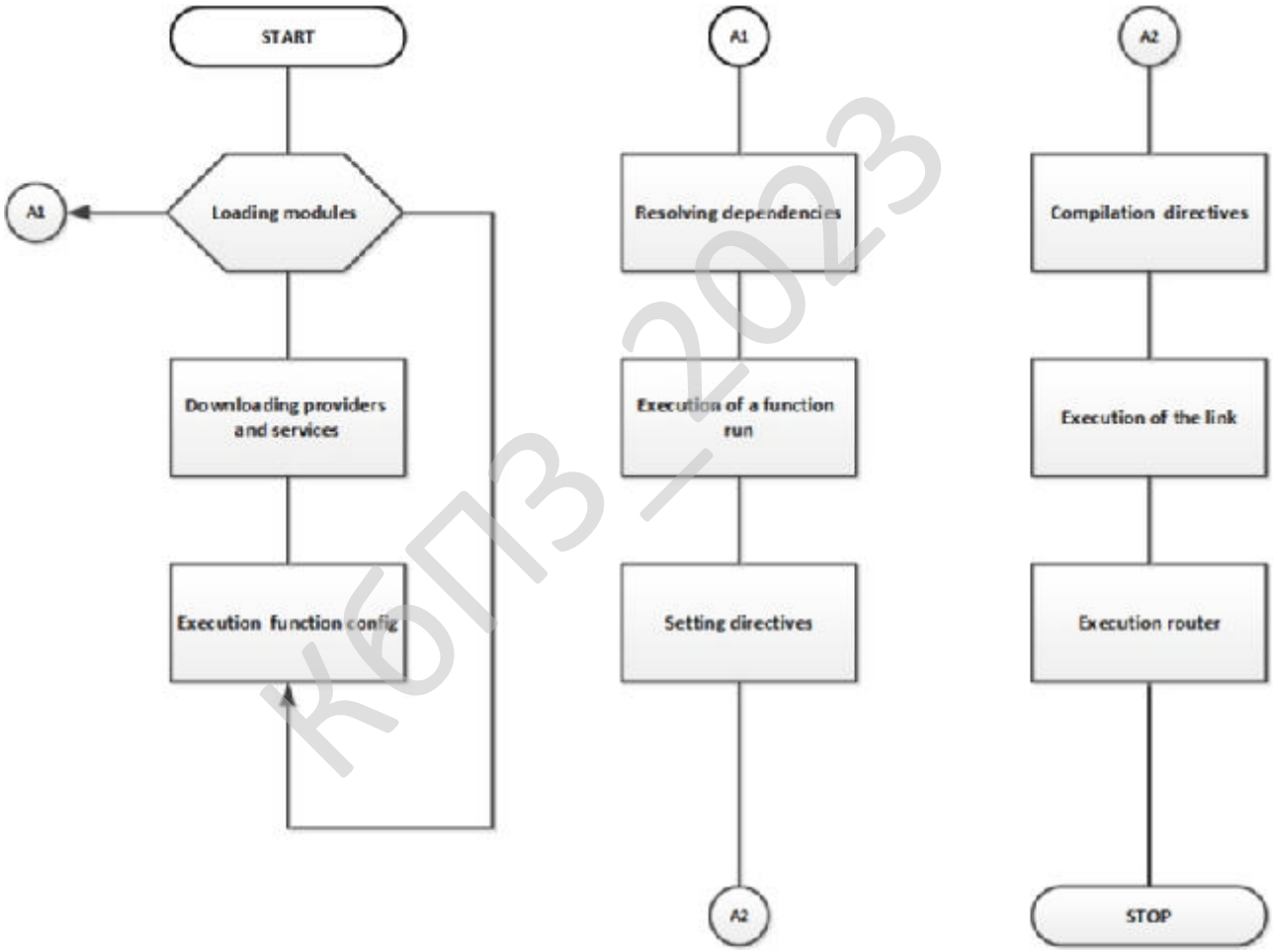

Рисунок 4.1 - Блок-схема підпрограми

Оскільки код є досить великим і має велику кількість не пов'язаних між собою сервісів та запитів, він поділений на окремі компоненти, в різних папках. Це допоможе зробити код більш зрозумілим, та зручним для сприйняття логіки

програми. У більшості випадків модуль представлений у вигляді одного контролера, в якому представлена вся бізнес-логіка модуля. Крім контролера, також є сервіс, в якому знаходяться допоміжні функції, та репозиторій який містить функції, які роблять запити до бази даних.

На серверній стороні головним модулем є файл main.ts, який ініціює весь проект, завантажує документацію, валідацію та починає прослуховувати запити. Файл app.module.ts імпортує інші розроблені модулі, та виконує всі необхідні налаштування, на серверній частині додатку. NestJS вимагає від користувача дотримуватися архітектурного стилю MVC, при розробці логіки програми. Читаючи файл конфігурації користувача, він налаштовує сервер відповідно, до вказаних вимог, або обирає налаштування за замовчуванням з файла .env. Використовуючи вище згадані парадигми, до головного модулю, підключаються не тільки розроблені сервіси, а й база даних MongoDB та Redis, які відповідають за дані користувача та за горизонтальне масштабування. Такі вимоги фреймворку повинні допомогти користувачу програмного забезпечення на стороні сервера побудувати зручну для використання та підтримки архітектуру програми.

У головному файлі програми, який є основою всього програмного забезпечення, знаходиться декоратор @Module(), де відбувається підключення всіх налаштувань та створених компонентів, які допоможуть побудувати архітектуру MVC. В методі bootstrap читається файл налаштування сервера, імпортуються та підключаються необхідні модулі, та завантажується сервер на обраному домені та порту.

Однією з перших дій, які виконує програма це читання файлу користувача .env, де повинні бути вказані основні властивості налаштування веб-серверу, якщо файл відсутній, або більшість інформації відсутня програма бере необхідні дані з файлу constants.js, де вказані всі властивості за замовчуванням. Для читання файлу і отримання необхідної інформації, використовується бібліотека `@nestjs/config`, яка використовує модуль `dotenv`. Після встановлення модулю потрібно в файлі app.module.ts імпортувати ConfigModule в

корінь AppModule, після чого можна використовувати ConfigServise для отримання глобальних налаштувань в інших модулях. В цьому файлі відбувається підключення бібліотеки mongoose для роботи з базами даних, обробка запитів користувача контролером, перегляд встановленої маршрутизації.

Наступними діями є створення необхідних компонентів, які будуть містити набір API для взаємодії з додатком, за логікою MVC. В першу чергу це створення компоненту авторизації, якій містить контролер, сервіс, репозиторій та модуль. За допомогою декоратора @Controller(), який імпортується з стандартного NestJS фреймворку @nestjs/common. В контролері відбувається вся бізнес логіка запиту, де користувач може зареєструватися, перевірити свої дані, та отримати JWT токен для подальших запитів. У цілях безпеки перед збереженням паролю користувача він спершу хешується за допомогою бібліотеки bcrypt та функцій hash, а за допомогою функції compare перевіряє валідність паролю. За допомогою JwtService з бібліотеки @nestjs/jwt виписуються access та refresh токен, які використовуються для перевірки юзера. При отриманні запиту AuthGuard з бібліотеки @nestjs/passport перевіряє валідність токена. А за допомогою бібліотеки class-validator перевіряються вхідні дані запиту при реєстрації та створюється документація у сваггері. Бібліотеки які використовувалися при розробці серверної частини додатку:

- @nestjs/cli — інструмент для ініціалізації та підтримки додатака. Він надає набір команд для швидкого створення контролерів, сервісів, репозиторіїв, модулів а також файлів для тестування;

- @nestjs/common — надає набір декораторів для створення контролерів, модулів, провайдерів, документації, для отримання тіла запиту, валідації, обгортки навколо Request та Response та статуси для повернення відповіді;

- @nestjs/config — використовує бібліотеку dotenv для отримання доступу до налаштувань;

- @nestjs/core — за допомогою цього модулю можна обрати на основі fastify чи express буде побудований додаток;

- @nestjs/jwt — модуль для генерації та перевірки токенів авторизації;

- @nestjs/mongoose — модуль для підключення бази даних MongoDB;

- @nestjs/passport — це проміжне програмне забезпечення, для перевірки аутентифікації;

- @nestjs/swagger — модуль надає декоратори для створення документації за допомогою сваггера;

- @nestjs-modules/mailer — використовується для надсилання e-mail повідомлень;

- bcrypt — бібліотека для шифрування даних;

- class-validator — надає набір декораторів для перевірки валідації;

- express — фреймворк для написання додатків на NodeJS;

- handlebars — інструмент для створення шаблонів.;

- ioredis - клієнт для роботи з Redis;

- supertest — бібліотека для тестування запитів.

При кожному HTTP запиті, на сервері, викликається відповідний контролер, в якості аргументу може приймати декоратори Request і Response, але краще використовувати декоратори для отримання тіла запиту чи параметра. Щоб вернути відповідь досить просто вернути результат з функції. Така структура дозволяє працювати з асинхронною логікою, але якщо не буде відповіді запит залишиться відкритим. Оскільки код є досить великим і має велику кількість не пов'язаних між собою сервісів та запитів, він поділений на окремі компоненти, в різних папках. Це допоможе зробити код більш зрозумілим, та зручним для сприйняття логіки програми. У більшості випадків модуль представлений у вигляді одного контролера, в якому представлена вся бізнес-логіка модуля. Крім контролера, також є сервіс, в якому знаходяться допоміжні функції, та репозиторій який містить функції, які роблять запити до бази даних.

На серверній стороні головним модулем є файл main.ts, який ініціює весь проект, завантажує документацію, валідацію та починає прослуховувати запити. Файл app.module.ts імпортує інші розроблені модулі, та виконує всі необхідні

налаштування, на серверній частині додатку. NestJS вимагає від користувача дотримуватися архітектурного стилю MVC, при розробці логіки програми. Читаючи файл конфігурації користувача, він налаштовує сервер відповідно, до вказаних вимог, або обирає налаштування за замовчуванням з файла .env. Використовуючи вище згадані парадигми, до головного модулю, підключаються не тільки розроблені сервіси, а й база даних MongoDB та Redis, які відповідають за дані користувача та за горизонтальне масштабування. Такі вимоги фреймворку повинні допомогти користувачу програмного забезпечення на стороні сервера побудувати зручну для використання та підтримки архітектуру програми.

У головному файлі програми, який є основою всього програмного забезпечення, знаходиться декоратор @Module(), де відбувається підключення всіх налаштувань та створених компонентів, які допоможуть побудувати архітектуру MVC. В методі bootstrap читається файл налаштування сервера, імпортуються та підключаються необхідні модулі, та завантажується сервер на обраному домені та порту.

Однією з перших дій, які виконує програма це читання файлу користувача .env, де повинні бути вказані основні властивості налаштування веб-серверу, якщо файл відсутній, або більшість інформації відсутня програма бере необхідні дані з файлу constants.js, де вказані всі властивості за замовчуванням. Для читання файлу і отримання необхідної інформації, використовується бібліотека @nestjs/config, яка використовує модуль dotenv. Після встановлення модулю потрібно в файлі app.module.ts імпортувати ConfigModule в корінь AppModule, після чого можна використовувати ConfigServise для отримання глобальних налаштувань в інших модулях. В цьому файлі відбувається підключення бібліотеки mongoose для роботи з базами даних, обробка запитів користувача контролером, перегляд встановленої маршрутизації.

Наступними діями є створення необхідних компонентів, які будуть містити набір API для взаємодії з додатком, за логікою MVC. В першу чергу це

створення компоненту авторизації, якій містить контролер, сервіс, репозиторій та модуль. За допомогою декоратора @Contr

Окрім стандартних та сторонніх бібліотек було розроблено декілька окремих сервісів, які оголошують окремі класи і надають додаткові можливості розробнику серверного додатку. Кожен файл містить свій об'єкт з набором властивостей та методів, який відповідає за окремий компонент програмного забезпечення, його функціонал та логіку. Модулі відповідають за різні компоненти програми, які можна використати, щоб зробити логіку програми більш незалежною.

Для структурування серверної частини були взяті принципи архітектурного типового рішення MVC. Розглянемо основні компонентні для налаштування серверної частини, які були розроблені:

- маршрутизатор (Route) - відповідно до маршруту обирає який контролер буде обробляти запит користувача;

- контролер (Controller) - відповідає за обробку запитів. Містить методи для реалізації функціональних вимог до веб додатку. Використовує модель для отримання даних предметної області та передачі їх до шаблонів;

- модель (Model) - відповідає за бізнес-логіку безпосередньо пов'язану з предметною областю. Є моделлю суті. Надає інтерфейс для роботи з сутностями. Інкапсулює обробку даних відповідної їй суті. Mongoose - представляє "обгортку" для моделей, надає функціональність для роботи з моделями;

- шаблон (Template) - представлення інтерфейсу користувача у вигляді HTML розмітки;

- вид (View) - представлення, код який необхідно обробити сервером додатка;

- API - стандартизований інтерфейс для роботи з даними предметної області. Інкапсуляція бізнес-логіки за певним маршрутом. Повертає всі дані в форматі JSON. Запит до API визначається параметром "api" в рядку URL.

Відповідно до REST, на прикладі роботи, семантика запитів була побудована в такий спосіб.

На стороні клієнта AngularJS активно використовує атрибути та директиви для побудови додатку. Наприклад головний атрибут ng-app знаходиться в корінному тегі html і використовується для позначення елемента HTML, який AngularJS буде розцінювати як головний (кореневий) елемент програми. За допомогою такого тегу можна повідомити AngularJS де він буде використовуватися, та де знаходиться початок програми, та де йому потрібно завантажувати подвійне фігурне зв'язування

Також в додатках створений за допомогою AngularJS часто можна побачити подвійне фігурне зв'язування за допомогою подвійних фігурних скобок. Так в AngularJS працює шаблонування і так він вставляє необхідні дані у DOM, а також оновлює ці дані при їх модифікації.

Завантаження програми за допомогою директиви дуже зручне, але при використанні великого об'єму коду і винесенні його в скрипт, варто використовувати ручний спосіб завантаження програми. При цьому відбувається:

- створюється новий інжектор, який вводить нові залежності;

- за допомогою інжектора створюється контекст у вигляді корневої області;

- AngularJS компілює DOM, починаючи з корневого елемента, обробляючи всі директиви та прив'язки на своєму шляху.

Використовуючи модель MVC, після запуску, додаток очікує нові події від користувача, наприклад натискання клавіши, щоб оновити модель та відобразити нові зміни користувачу. Дані моделі передаються всередину UserListController контролера. Контролер є просто функцією конструктора, який приймає параметр $scope.

Після реєстрації в головному модулі userApp модуля UserListControlleri, він буде визначений під час завантаження програми. Оскільки контролер виступає у ролі контекста, він буде встановлювати звязок між моделлю та

поданням. Це відбувається за допомогою директиви HgController, яка розташована в body і через імя контролера підключає дані на $scope. Ця область завантажується при створенні програми і доступна для всіх прив'язок, розташованих у тезі body.

В AngularJS область сприймається як основа, яка зв'язує шаблони, модель та контролер. Це допомагає тримати модель та представлення даних окремо, але синхронізованими між собою. Тому будь-які оновлення в моделі, будуть зразу відображені користувачу.

Якщо контролер керує логікою і контектом для даних шаблону, то шаблон виступає в якості форми, яка буде представлена користувачу. Через таку поширену методику, AngularJS пропонує обєднувати їх в компоненти, а також створить область ізоляції для кожного екземпляра компонента, щоб вони були незалежними між собою. Для створення компонента використовується метод .component(), який приймає імя компонента та його налаштування (об'єкт визначення).

З шаблону до контролера, можна звернутися через псевдоним. Також можна використати директиву $ctrl для псевдоніму контролера, або перекрити доступ, якщо буде потрібно.

Завдяки модульній архітектурі в AngularJS, код можна використовувати повторно, особливо корисно коли функції декларують свій власний модуль, а всі пов'язані суб'єкти повинні бути в об'явленому модулі. Це дасть змогу переносити код не тільки в проекті, а й між додатками.

Наприклад всі компоненти містять шаблони, які будуть виводитися користувачу у вигляді HTML. Використовуючи CDO (Component Definition Object) можна вказати шаблон для компонента у вигляді рядка, це не є правильним підходом, особливо у великих шаблонах, тому зазвичай HTML-код виносять в окремий файл, що робить код в компонентах чистішим.

Хоча це і корисна практика використовувати CDO, але краще використовувати зовнішні шаблони в компонентах. У цьому допоможе властивість templateUrl в якій вказується URL, щоб визначити зовнішній шаблон.

Директива ngModel допомагає робити пошук в списку за заданими критеріями. Це можливо за допомогою технології зв'язування даних. При завантажені сторінки, поля прив'язуються до моделі даних, вказаної за допомогою з ngModeli та синхронізує ці дані. Наприклад, коли користувач введе у поле пошуку нові дані, вони зразу відсортуються у списку, через те що зміни в моделі даних зразу призводить до оновлення DOM, і відображення поточної інформації.

Ще хорошим прикладом двостороньої привязки даних є випадаюче меню, де можна відсортувати список на найновіші. Це відбувається у контролері за допомогою властивості orderPropв 'date', при завантаженні додатку. Тобто спочатку оновлюється модель після чого зміни відображаються на шаблоні. Якщо ж вибрати в випадаючому меню "алфавіт" то прив'язка спрацьовує в зворотному напрямку, спочатку оновляться дані інтерфейсу, а після дані моделі.

AngularJS також має набір вбудованих служб, наприклад $http використовується для надсилання HTTP запитів. Служби керуються за допомогою DI підсистемою в AngularJS. Введення залежностей допомагає задати структуру додатку та послабити зв'язки між сутностями. В результаті додаток набагато простіше тестувати.

За допомогою метода $http робиться HTTP запит на веб-сервер, для отримання даних з бази, та преобразуючи їх в формат JSON. Служба $http повертає дані асинхронно та присвоює ці дані контролеру, після чого лані з JSON можна отримати в функціях зворотнього виклику. Цей метод приймає декілька параметрів, статус запиту і адресу запиту, після чого за допомогою функції then можна перевірити відповідь на помилку та отримати дані. Крі методу get є і інші методи: post, delete, put, head, jsonp за допомогою яких можна реалізувати повноцінний REST API.

Інжектор залежностей в AngularJS слідкує за залежностями, та надає ряд послуг контролеру, які можуть мати служби, в першу чергу він звертає увагу на назви аргументів. Якщо ж треба створити власні послуги, в цьому допоможе метод scope та методи, які мають префікс $ перед іменеи. Префікс $ означає що це AngularJS-послуги.

Щоб реалізувати маршрутизацію, треба використовувати модуль ngRoute, який не є стандартним в AngularJS. Для його використання треба встановити додатковий модуль з npm: angular-route. Після чого можна оголосить нові маршрути за допомогою $routeProvider , який є постачальником послуги $route. Ця послуга має гарний функціонал, який дозволяє з'єднувати контролери, переглядати шаблони та поточну URL-адресу в браузері, або використовуувати історію веб-переглядача та закладки.

Як можна побачити всі вищезгадані фреймворки: Angular, AngularJS, NestJS використовують паттерн проектування MVC. Його поширеність полягає в зручному розподілі логіки і даних. Спочатку користувач робить запит на сервер, де контролер запрошує дані додатку у моделі і передає їх дані шаблону, який виконує остаточне форматування перед тим як надати дані користувачу. Шаблон часто реалізують за допомогою шаблонізатора, на серверній стороні це виконує модуль handlebars, він приймає дані повернуті моделлю і вказує який шаблон для відображення треба використовувати.

Шаблони зазвичай містять фрагменти HTML, CSS і іноді логіку JavaScript коду, для впровадження динамічної поведінки (наприклад віджетів) або для зміни інтерфейсу. Але оскільки шаблони більше зв'язані з представленням даних, розробники серверної та клієнтської частини повинні рівномірно розподілити логіку.

При кожному запиті користувача, викликається відповідний контролер. Спочатку за допомогою бібліотеки class-validator перевіряються вхідні дані, якщо на контролері є інтерцептор він буде викликаний наступний, після чого буде перевірка гарди. Якщо під час виконання контролера була викликана помилка,

вона буде записана у файлі з логами з використанням модуля winston. Дані повернені з функції будуть надісланий клієнту.

Блок-схема демонструє роботу програми сервера та клієнта. На схемі можна побачити послідовність дій, які виконують клієнт щоб зробити запит на сервер та отримати необхідні дані з бази даних. Для цього спершу клієнту потрібно пройти авторизацію, щоб підтвердити свої дані. Дивлячись на схему можна прослідкувати які операції виконуються, як дані взаємодіють, та як програма змінює свій стан.

Схема (рисунок 4.2) містить головні функції програми, та побудована так, щоб при її перегляді було зрозуміло функціонал та призначення системи. Вона є досить інформативна, щоб зрозуміти як побудована програма.

Розроблений веб-додаток складається з наступних компонентів:

- app — головний компонент, який ініціалізує проект;

- auth — компонент який містить логіку авторизації та автентифікації;

- public_chat — компонент який відповідає за чат, та кативно використовує сокети;

- users - компонент для роботи з даними юзера;

- mail — компонент для відсилання e-mail повідомлень.

У розроблених прикладах за допомогою фреймворків AngularJS та NestJS було створено функціонал для реєстрації, авторизації, а також надавало інформацію про користувачів за допомогою операції CRUD (операцій читання, створення, оновлення та видалення), а за допомогою клієнтських фреймворків Angular та AngularJS ці дані надавались користувачу у зручній для сприйняття формі.

В додатку було реалізовано концепцію REST-API, існують також інші способи взаємодії клієнта та сервера, але у даному випадку REST підійшов найкраще.

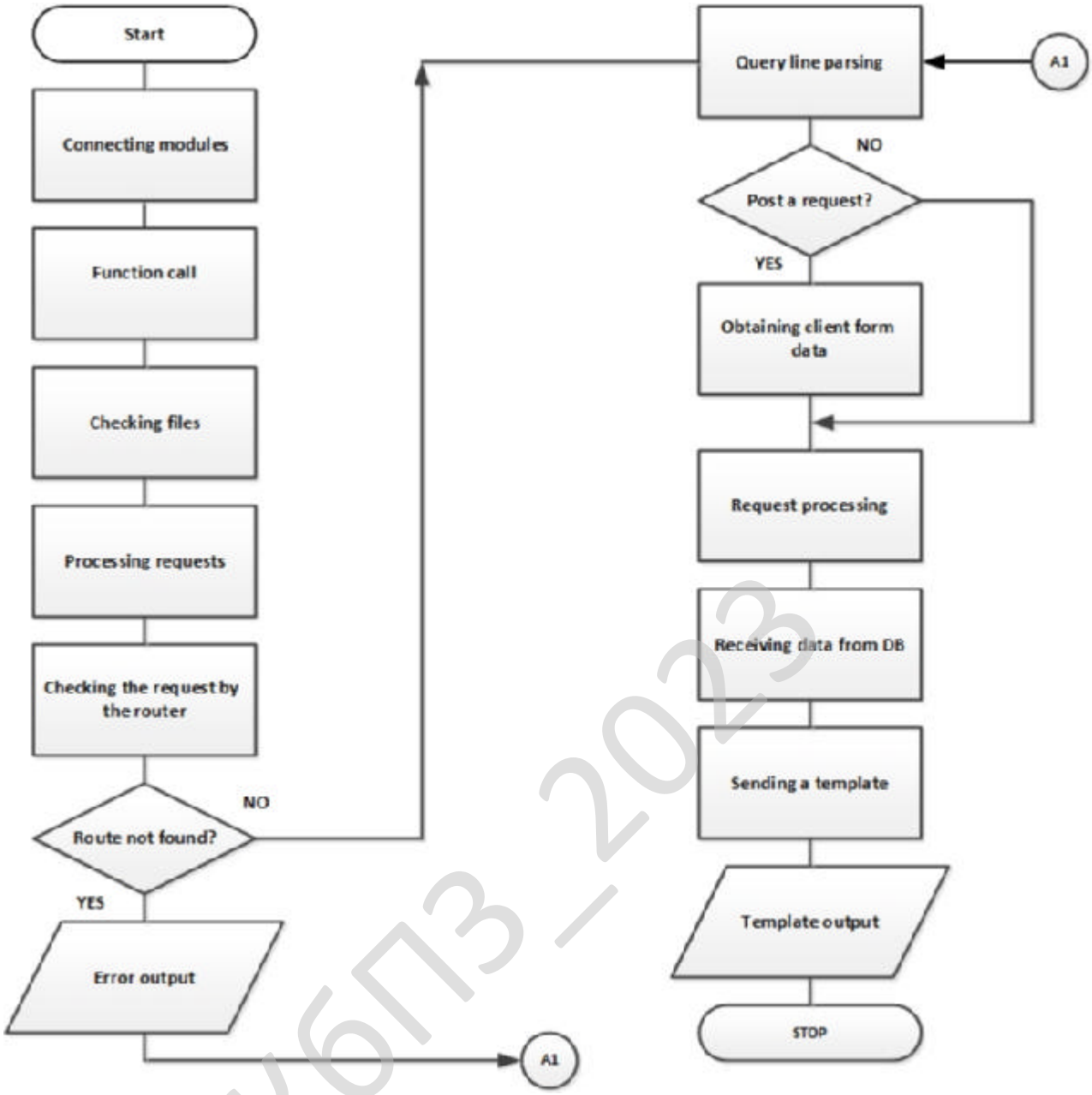

Рисунок 4.2 - Блок-схема роботи системи

Всі запити виконувалися за допомогою HTTP-методів: GET, POST, PUT і DELETE, які використовувалися для створення, оновлення, читання та видалення даних з сервера.

Кожний метод відповідав за власну операцію:

- POST — створював та додавав нові дані в базу;
- GET — отримував список заданих полів з сервера;
- DELETE — видаляв дані з бази;

- PUT — оновлював дані в базі.

Angular новішої версії має трохи іншу структура. Головним файлом в нього є main.ts, а в папці src знаходяться всі компоненти, стилі та модулі. Angular надає ряд декораторів: @Component, @NgModule, за допомогою яких створюються контролери, сервіси та модулі. Вони мають зручний інтерфейс, але будуть складними для розробників які бачуть їх вперше. Прив'язка даних в Angular має декілька форм. Наприклад її можна реалізувати, за допомогою фігурних дужок або прив'язки DOM елемента до компоненту. Для двосторонньої прив'язки використовуються [(ngModel)], цей вираз допомагає прив'язати DOM елемент до значення, яке використовується на компоненті.

Також варто сказати про технології, які використовувалися під час розробки додатку.

Postman — інструмент призначений для тестування розробленого API. Він надає можливість зберігати запити, створювати колекції, документацію, а також слідкувати за показниками запиту. За допомогою цього інструмента можна значно прискорити розробку та тестування додатків.

MongoDB Compass - представляє графічну оболонку для роботи з базою даних MongoDB. Він був розроблений MongoDB Inc і надає велику кількість функціоналу для адміністрування бази даних. За допомогою нього можна: додавати, переглядати, оновлювати дані в базі, додавати або видаляти індекси, виконувати агрегації. Також він показує швидкість запиту, його навантаження, та кількість займаємо пам’яті. Але треба бути обережним при його використанні щлб не пошкодити дані в базі.

Для заповнення бази, даними для тестування або налаштування бази в продакшині потрібно створювати міграції. Краще за всіх з цим справляється пакет migrate-mongo. Для нього потрібна мінімальна кількість коду, а це означає що можна витратити більше часу на впровадження необхідної логіки для міграцій. Також він надає набір команд для роботи з міграціями, наприклад

migrate_status виводить список всіх сценаріїв а також їх статус. Команди migrate_down та migrate_up запускають файли міграції.

4.2 Захист розробленого програмного забезпечення

В першу чергу варто перевірити розроблений додаток автоматизованими засобами, вони зможуть перевірити додаток на найрозповсюдженіші помилки. Вони перевіряють не тільки код написаний розробником, а й бібліотеки та модулі які використовує розробник, а вони складають найбільшу частку зломів. Згідно з останніми перевірками, понад 50% npm модулів, не оновлюються більше декількох років.

Є декілька способів перевірити модулі перед їх використанням, в першу чергу варто переглянути код, та перевірити чи повністю модуль покриває потреби розробника. Він повинен перевірити які ще бібліотеки використовує цей модуль і коли вони останній раз оновлювалися, чим менше він має залежностей, тим краще.

Великі компанії мають спеціальних працівників, які проводять ревізії пакетів перед їх використанням, та складають спеціальні білі списки дозволених модулів.

Сама npm занепокоєна тим, як часто у бібліотеки почали впроваджувати майнерів та віруси для збору приватних даних. Тому вони випустили спеціальний модуль npm audit. Він сканує встановленні модулі в проект і порівнює їх з чорним списком модулів, які містять уразливості. Сьогодні навіть команда npm install підкаже, чи мають встановленні модулі вразливості. А нова команда npm audit fix, пропонує замінити вразливі пакети, або оновити до більш захищених версій, якщо такі існують. Але так можна знайти лише ті модулі, про які вже повідомили користувачі та розробники.

Також для перевірки безпеки додатку, важливим є написання тестів. Зазвичай розробники на AngularJS використовують концепцію Jasmine's Behavior-

Driven Development (BDD), при написанні тестів. Jasmine − це вільний фреймворк для тестування коду написаного за допомогою JavaScript, його можна запустити на будь-якій платформі. Його перевагою є зручний інтерфейс, та зрозуміле налаштування. Для тестування одиничних тестів часто використовують бібліотеку Karma, головна мета якого наблизити тестування додатку до його налаштувань в продакшен.

Також важливо використовувати протокол HTTPS, він є набагато надійнішим HTTP. HyperText Transfer Protocol Secure (HTTPS) — забезпечує шифрування даних і дозволяє надійно захищати дані користувачів при передачі в Інтернеті. Сьогодні більшість сайтів використовують саме цей протокол, оскільки він є обов'язковим при передачі приватних даних на сервер, особливо коли передаються паролі або дані кредитних карт. Також варто бути обережним с cookie файлами, які передаються під час авторизації. Зловмисник може перехопити їх та підробити запит на сервер, щоб цього уникнути потрібно використовувати HTTPS протокол.

Також слід брати до уваги поширені CORS та CSRF атаки. CSRF- атака це коли на сторінки, робиться непомітна форма, щоб відправляє запит з приватними даним користувача. Якщо на сайті авторизація відбувається тільки за допомогою кукі, то така атака може бути успішна. Щоб цього уникнути потрібно використовувати спеціальні токени. Для підпису XMLHttpRequest токен також зберігається в кукі. Тоді JavaScript може прочитати його з домену і додати в заголовок, а сервер - перевірити, що в заголовку міститься коректний токен.

При крос-домених запитів, браузер додає заголовок Origin, який зберігає домен, з якого відбуваються запити. Сервер у цьому випадку повинен перевірити домен і відповісти спеціальним заголовком. Якщо сервер дозволяє доступ, він повинен відправити заголовок Access-Control-Allow-Origin, що зберігає домен запиту. Якщо Access-Control-Allow-Origin відсутній, то сервер завершує запит з помилкою. При таких запитах не передаються куки і заголовки HTTP-авторизації.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблений додаток використовує AngularJS та NestJS для демонстрації можливостей сучасних веб-технологій. Також другий розроблений додаток був розроблений для порівняння технологій AngularJS та його новішої версії Angular. В обох випадках у якості серверної частини використовувався платформа NodeJS та фреймворк NestJS.

Оба фреймворки AngularJS та Angular використовуються для створення SPA(Single Page) додатків, які все більше поширюються на просторах Інтернету. В розроблених додатках було продемонстровано стандартний функціонал, який використовується на більшості сайтів: REST API, авторизація та чат на сокетах.

Метою створення додатків було продемонструвати, які сьогодні існують фреймворки для створення веб-додатків, їх можливості, функціонал. Також були створенні шаблони, які можна використовувати для швидкого розгортання нового проекту. Порівнюючи одні з найвідоміших фреймворків для створення клієнтської частини та її взаємодії з сервером, можна побачити такі результати. Отримані результати можна побачити на рисунку 5.1.

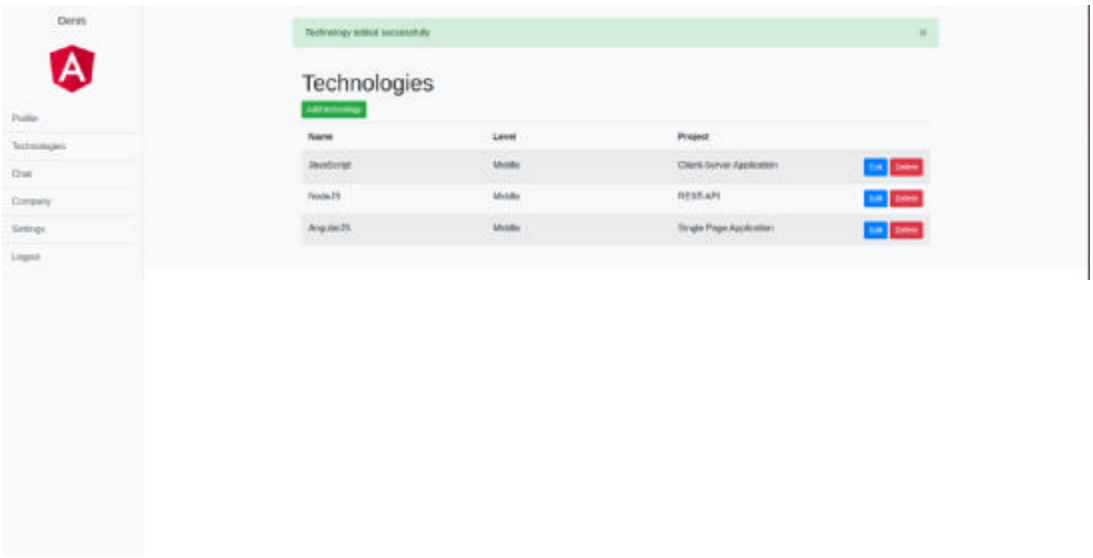

Рисунок 5.1 – Веб-сторінка реалізована за допомогою AngularJS

За час розробки, можна зробити такі висновки:

- на освоєння Angular було витрачено більше часу, оскільки він використовує TypeScript і вимагає більше часу на його освоєння;

- Angular має зручний набір бібліотек для розробки додатку;

- AngularJS став відомим за його декларативний стиль коду, модульність, простоту тестування, завчасно готові рішення які спрощували розробку. Сучасні фреймворки багато успадкували від нього, але після припинення підтримки краще використовувати більш сучасні фреймворки;

- якщо додаток вже розроблений за допомогою AngularJS і потрібно розробити нову модифікацію і AngularJS дозволяє це зробити, краще продовжити розробку на AngularJS, якщо проект новий краще використовувати Angular;

- сучасні SPA фреймворки мають набагато більший функціонал, завдяки якому можна створювати додатки навіть на мобільні пристрої;

- швидкість роботи Angular швидша, але швидкість розробки швидша на AngularJS.

Розроблений додаток використовує REST API для взаємодії клієнта з сервером, особливо було зручно використовувати RxJS для отримання та обробки даних. Оскільки його підтримують як Angular на клієнті, так і NestJS на сервері.

Дані передавались у форматі JSON, такий формат добре підходить для додатків, які повинні швидко обмінюватися даними. REST API-інтерфейс забезпечують зручну взаємодію з користувачем, легко масштабується.

Завдяки використанню шаблону MVC, додаток має чітку структуру, зручну для підтримки та подальшого розширення, де бізнес-логіка відділена від інтерфейсу користувача. В результаті, додаток легше масштабувати, тестувати та підтримувати. Була розроблена інфраструктура з гнучкою логікою, простою маршрутизацією, інтуїтивно зрозумілі представлення даних.

На рисунку можна побачити доступний інтерфейс та функціонал:

- головну сторінку та форму реєстрації;
- доступну документації;
- чат створений на сокетах;
- можливість зберігати медіа-файли;
- профільну сторінку юзера.

Розроблена програма має простий, зрозуміло інтуїтивний інтерфейс. Якщо користувач вже користувався браузером, він без проблем зможе зрозуміти де який функціонал знаходиться.

Фреймворки дуже допомагають при проектуванні архітектури ПЗ, оскільки в концепціях фреймворків закладені кращі практики розробки, тому просто дотримуючись цих правил можна уникнути багатьох проблем і помилок при проектуванні додатку. Як правило фреймворки представляють собою набір класів та функцій, які активно взаємодіють між собою і підтримують головну методологію фреймворку, а також надають свій функціонал для розширення можливостей додатку використовуючи свої концепції.

Також фреймворки впливають на швидкість розробки, та слідкують щоб розробник не зробив помилок при проектуванні додатку. Оскільки вибір фреймворку такий важливий, треба звертати увагу на його підтримку, а також на кількість розробників які його використовують. Щоб при виявленні якоїсь помилки, можна було її швидко усунути та повідомити ком'юніті.

Крім фреймворків також можна використати CMS, але вони як правило працюють набагато повільніше, їх важче масштабувати, та додавати новий функціонал, вони витримують менші навантаження. Також у фреймворків рівень безпеки як правило кращий, але розробка на CMS на багато швидша, як і рівень для їх використання, оскільки не потрібно вивчати концепції.

Для демонстрації роботи додатку використовувався ngrok — інструмент, який дозволяє розвернути додаток в Інтернеті, але з певними обмеженнями. Щоб використовувати його повний функціонал потрібно купувати підписку. Сам додаток завантажувався через Docker, використовуючи Dockerfile були встановленні всі налаштування для сервера, був використаний образ node:14.17.1-slim, який оснований на Debian, за допомогою команди COPY були скопійовані всі файли проекту, а за допомогою команди RUN встановлено всі необхідні залежності. Потім встановлювався порт, використовуючи команду EXPOSE, та запускався додаток через команду CMD. База даних та Redis теж підіймались в Докері, вони використовували локальні ресурси підіймались, але сайти mongodb atlas та redis надають обмежені ресурс, для зберігання даних. Для зберігання медіа файлів використовувався minio. При впровадженні програмного забезпечення потрібно урахувати наступні дії:

- налаштувати доступи;
- підключитись до MongoDB;
- підключитись до Redis;
- підключитись AWS, або сервера який буде зберігати медіа файли;
- підключитись до SendGrid для надсилання повідомлень.

Також перед впровадженням системи слід запустити тести, для перевірки працездатності всієї програми. Тестування включає не тільки пошук помилок, але й випробування доступного навантаження на сервер та швидкість запитів. За допомогою результатів тестів, можна отримати:

- максимальне навантаження;
- перевірка працездатності запитів;

- швидкість обробки запиту;

- виконання поставлених вимог;

- результати усіх вихідних даних;

- час виконання функцій;

З розроблене програмного забезпечення можна отримати необхідні дані для прийняття рішень про вибір технології відповідно до заданих вимог розробника. А також використати як шаблон для створення нових додатків, оскільки розроблене ПЗ використовує більшість стандартного функціоналу, який використовується в більшості веб-сайтів.

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено продемонструвати можливості сучасних веб-технологій на стороні клієнта та сервера.

Метою розробки є програмна реалізація та порівняння існуючих фреймворків, з метою отримання результатів працездатності та визначення можливостей технологій, для розуміння їх вибору в залежності від вимог розробник.

Об'єктом дослідження є процес визначення можливостей фреймворку AngularJS в сучасному світі веб-розробки.

Предметом дослідження є методи для визначення можливостей фреймворку AngularJS в 2023 році.

Методи дослідження базуються на методах теорії кодування, паттернах проектування, методологіях розробки програмного забезпечення та технологіях взаємодії клієнт-серверних додатках.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– порівнянно та отримано результати роботи сучасних веб-фреймворків, представлена інформація в яких випадках краще використовувати Angular та AngularJS;

– розроблено шаблони для швидкого ініціювання проектів з технологіями: Angular, AngularJS, NestJS, MongoDB, Redis, Docker, SocketIO, MinIO або AWS;

– Розроблено веб-сайт, де користувач може зберігати та публікувати дані про свої знання, та взаємодіяти з іншими користувачами.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми дипломного проекту

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці).

В магістерській роботі проведене дослідження та виконана програмна реалізація системи передачі мультимедійних даних за технологією IMS.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 - Початкові данні

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	57
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження таблиці 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	_	2
29. Досвід роботи з програмними інструментами розробки (1-6)	_	3
30. Розробка ПО для декількох серверів одночасно (1-6)	_	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	_	2
32. Вартість ПЗ у розробника (НМА), грн	–	60000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B \tag{7.1}$$

де А - коефіцієнт Боема, А=2,45;

Size - загальний об'єм відлагодженого програмного коду, тис. рядків;

В - показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \tag{7.2}$$

де Wi - сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1,01+0,001(2,43+3,64+3,38+3,95+2,73) = 1,026$$

$$T_{ном} = 2,45 \cdot 2,7^{1,026} = 6,78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \tag{7.3}$$

де $ПВ$j - добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6,78 \cdot (0,88 \cdot 0,93 \cdot 0,88 \cdot 0,91 \cdot 0,95 \cdot 1 \cdot 1 \cdot 0,87 \cdot 1,22 \cdot 1,16 \cdot 1,1 \cdot 1,1 \cdot$$

$$\cdot 1,12 \cdot 1,1 \cdot 1 \cdot 1,1 \cdot 1,1) = 9,37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0,3 C T_{уточн}^{0,33+0,2(B-1,01)} S, \tag{7.4}$$

де C - визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S - коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0,3 \cdot 2,66 \cdot 9,37^{0,33+0,2(1,026-1,01)} \cdot 100 = 168 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 - Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	168	Ф 7.1-7.4
Впровадження	13	Д13
Всього	209	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де $F_{рq}$ - плановий фонд робочого часу одного спеціаліста, днів,

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{209 \cdot 1}{48 - 5} = 4{,}8 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 - Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	10	900	15
Монітор	60	10	600	10
Клавіатура	30	10	300	5
Маніпулятор «мишка»	30	10	300	5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	4	120	2
Кабельні господарства ЛВС на 1 м. п.	2,5	400	1000	16,67
Копіювальний апарат	140	2	280	4,67
Усього за рік:			$З_ч$	61,67

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{61,67 \cdot 3}{1,2} = 154,2 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=154,2/(60·8)=0,3 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів–електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 - Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2022, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,8	0,2
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,2	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,4	
Всього		1,6	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,5	0,2
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,3	
	Розміщення графіки і контенту на Інтернет сторінках	0,3	
Всього		1,6	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,2
	Верстка друкованих видань	0,2	
	Додрукова підготовка макетів	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		1,6	

Складемо штатний розклад виконавців у таблицю 7.5.

Таблиця 7.5 - Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (IT-менеджер)	1	14128,5	28257
Продакт-менеджер	0,5	12000	12000
Інженер-програміст	4,8	13500	129600
Інженер-електронщик	0,3	9000	5400
Інженер-системотехнік	0,2	9000	3600
Адміністратор мережі	0,2	11000	4400
Системний програміст	0,2	9000	3600
Дизайнер WEB	0,2	9000	3600
Інженер-верстальник	0,2	9000	3600
Бухгалтер-економіст	0,2	10000	4000
Всього за період розробки	$R_{сп}$=7,8	-	$Ф_{роб}$=198057

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{198057}{7,8 \cdot 48} = 529 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць.

$S_у$ – питома площа на одне робоче місце, $м^2$,

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пектораль» ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 37 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно $8 м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались за комерційною пропозицією фірми Brain за 11.11.23 – джерело http://brain.com.ua/

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11771
Системний блок		7771
Процесор	AMD Ryzen 5 2400G (YD2400C5M4MFB) AM4, 4 ядра, 8 потоків, 3.6 GHz, 3.9 GHz, TDP - 65 Вт, 14nm, L1: 384KB, L2: 2MB, L3: 4MB, Radeon Vega 11, Zen, Tray	-
Системна плата	GIGABYTE A520M H сокет - AM4, 5100 MHz, LAN - 1 Гбіт/с, DVI, HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	-
Жорсткий диск	SSD M.2 2280 240GB Apacer (AP240GAST280-1) 240 GB, TLC, M.2, SATA III (6Gb/s)	-
Оперативна пам’ять	DDR4 8GB 2400 MHz Patriot	-
Система охолодження	ID-Cooling DK-15 PWM INTEL, AMD	-
Корпус	Logicpower 8702 - 550w 12cm	-
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	-
інше	Клавіатура, мишка	-
Монітор	Монітор BenQ GL2450HM Black	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Безперебійник живлення	Powercom BNT-600AP USB	1400

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 - Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	8	11771	9416,8	103584,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	125216,3

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	125216	-	-
Всього по групі	125216	50	62608
Група 5, 6			
4. Вимірювальні пристрої	5190	-	-
5. Транспортні засоби	143000	-	
6. Господарський інвентар	28000	-	-
Всього по групі	176190	20	35238
7. Нематеріальні активи	60000	10	6000
Разом	$К_р$= 1769406		$А_р$= 174246

Примітка: вартість автомобіля Sens (Standard+) взята по даним з автосалону «Кіровоград-Авто», джерело http://kirovograd-avto.ukravto.ua /catalog/tm-19/model-81/description, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_e}, \quad (7.11)$$

де N_e – Кількість екземплярів програм, шт.

$$З_о = 529 \cdot 209/57 = 1940 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 1940 \cdot 10 \cdot 0{,}01 = 194 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(1940+194) = 470 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом *Нг*=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 1940 \cdot 15 \cdot 0{,}01 = 291 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картриджів, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно виданих норм приймаємо 1/6 пачку паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=240 грн., визначаємо вартість паперу за період розробки $N_м$=2 міс:

$$З_{М1} = Ц_п \cdot N. \quad (7.16)$$

$$З_{М1} = 240 \cdot 1/6 \cdot 2 = 80 \text{ грн.}$$

Згідно виданих норм до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків в кількості 20 примірників:

$$З_{М2} = \sum Ц_д., \quad (7.17)$$

де $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 23,3 грн/шт., DVD-R LG 4,7Gb, 16x speed Cake box – 23,3 грн/шт.

$$З_{М2} = 23{,}15 \cdot 20 = 466 \text{ грн.}$$

Згідно виданих норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{МЗ} = \sum Ц_{з}., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570=1702 \text{ грн.}$$

$$З_{М} = (80 + 466 + 1702)/57=39 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де $Н_п$ - норматив витрат на освоєння нових мов програмування, %

$$О_п = 1940 \cdot 15 \cdot 0{,}01=291 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 57 прим.)

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 174246 \cdot 2/(57 \cdot 12)=509 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$С_п = 1940+194+470+291+39+291+509=3734 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (Рп) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \quad (7.22)$$

де P_c – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 50 \cdot 3734 = 1867 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	1940
2. Додаткова зарплата виконавців	$З_д$	194
3. Відрахування на соціальні потреби	$С_{оц}$	470
4. Загальногосподарські витрати	$Г_{осп}$	291
5. Витрати на матеріали	$З_М$	39
6. Освоєння нових операційних систем, мов програмування	$О_п$	291
7. Амортизація основних фондів	$А_м$	509
8. Повна собівартість програмного забезпечення	$С_п$	3734
9. Плановий прибуток	$П_р$	1867
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	5601
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1120,2
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	6721,2

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 - Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	6721
Всього капітальних витрат	–	6721

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 - Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позна-чення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	26840	16104
2. Витрати на електроенергію	$З_{ел}$	152	92
3. Витрати на амортизацію	$З_{ам}$	0	3361
Всього витрат за рік	$І$	26992	19557

Витрати на технічне обслуговування:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \qquad (7.23)$$

де $Т_р$ – кількість годин обслуговування системи за рік, *год.*,

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 200 годин на рік до 120 годин на рік, тому витрати на технічне обслуговування зменшилися з

$$З_{р\ баз} = 200 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 26840\ грн.$$

до

$$З_{р\ нов} = 120 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 16104\ грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$).

$$З_{ел}=П_{ел} \cdot Т_р \cdot Ц_{ел}. \qquad (7.24)$$

Зел баз=0,2·200·3,8=152 грн

Зел нов=0,2·120·3,8=92 грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 - Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	6721	–	3360,5
Всього відрахувань	-	–	6721	–	3360,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (5601 - 3734) \cdot 57 - (0,05 \cdot 1408000 + 0,5 \cdot 125216 +$$
$$+0,2 \cdot 176190 + 0,1 \cdot 60000) \cdot 2/12 = 77378 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1769406}{(5601 - 3734) \cdot 57 \cdot 12 / 2} = 3 \text{ роки}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н (K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (26992\text{-}19557)\text{-}0,5 \cdot 6721 = 4075 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{6721}{26992 - 19557} = 0,9 \text{ років}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 - Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	57
2. Повна собівартість розробленої програми	Грн.	3774
3. Ціна розробленої програми	Грн.	5601
4. Плановий прибуток від реалізації розробленої програми	Грн.	1827
5. Рентабельність програмної продукції	%	50
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1769406
7. Загальний прибуток від реалізації програмної продукції	Грн.	104139
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	77378
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	3
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	6721
11.Величина економічного ефекту у користувача програмної продукції	Грн.	4075
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,9

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Аналіз умов праці програміста

Комп'ютерні мережі вриваються у професійну діяльність. Вони породили істотно нові технології обробки інформації – мережеві технології. Мережеві технології дозволяють спільно використовувати ресурси – накопичувачі великої ємності, друкуючі пристрої, доступ в Internet, бази і банки даних. Найбільш сучасні і перспективні підходи до мереж пов'язані з використанням колективного розподілу праці при спільній роботі з інформацією – розробці різних документів і проектів, управлінні установою або підприємством, різними програмами обліку і т.д. За всім цим знаходиться працівник, робота якого тісно пов’язана з комп’ютерною технікою.

Забезпечення безпечних і здорових умов праці в значній мірі залежить від правильної оцінки небезпечних, шкідливих виробничих факторів. Однакові по складності зміни в організмі людини можуть бути викликані різними причинами. Це можуть бути фактори виробничого середовища, надмірне фізичне і розумове навантаження, нервово-емоційна напруга, а також різне сполучення цих причин. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

Напруженість праці користувача ПЕОМ

Робота програміста пов’язана з значним зоровим навантаження, що вимагає забезпечення належного освітлення. В даному приміщенні рівень природного освітлення є достатнім, а рівень штучного – понижений. Інженер – програміст працює з ЕОМ та іншим офісним обладнанням, що є джерелом небезпеки ураження електричним струмом. Трудова діяльність програміста пов’язана з постійним перебуванням в приміщенні, тому для комфортних умов праці

Згідно нормативним документам «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу». Наказ Міністерства охорони здоров'я України 08 квітня 2014 року N 248 можна виділити такі шкідливі виробничі чинники, що діють на працівника даної комп'ютерної лабораторії:

1. Недостатній рівень штучного освітлення.

2. Мікроклімат робочої зони: температура, відносна вологості, швидкість руху повітря.

3. Підвищений рівень шуму на робочому місці.

4 .Небезпечна напруга в електричному ланцюзі.

Далі проведемо аналіз перших трьох шкідливих та небезпечних виробничих чинників, що діють в комп'ютерній лабораторії на програміста. Виконаємо якісний та кількісний аналіз цих чинників. Також, розробимо заходи з охорони праці, для цих трьох шкідливих виробничих чинників, які забезпечують покращення умов праці для програміста в комп'ютерній лабораторії.

Рівень штучного освітлення

Основним документом, який регламентує норми освітленості є ДБН В 2.5.28 – 2006 «Природне та штучне освітлення».

В комп'ютерній лабораторії розташовані шість світильників по дві люмінесцентні лампи ЛБ20 в кожному. Джерелом живлення світильників є електрична мережа у 220 В.

Фактична величина освітленості даного робочого приміщення становить всього Е=210-220 Лк.

Категорія виконуваних робіт програміста відноситься до робіт високої точності з присвоєнням розряду III в. Тому нормативне значення загального освітлення робочого приміщення повинно бути Е =300-500 Лк.

Отже, необхідно вжити заходів для збільшення освітленості приміщення. Освітлення на робочому місці програміста повинно бути таким, щоб працівник

міг без напруги зору виконувати свою роботу. Розрахунок освітленості робочого місця зводиться до вибору системи освітлення, визначенню необхідного числа світильників, їхнього типу і розміщення.

Відповідно до вибраного розрядом зорових робіт допустиме значення освітленості робочої поверхні приймається E = 400 лк.

Для покращення освітлення комп'ютерній лабораторії будуть використовуватися світлодіодні лампи, а саме LITWELL LED-T8S-120 світловий потік яких Фл=1500лм.

Розрахунок для покращеного рівня штучного освітлення буде описаний в пункті 8.4.

Мікроклімат робочої зони: температура, відносна вологості, швидкість руху повітря

Праця програміста за важкістю відноситься до легкої фізичної роботи категорії Іа.

Основним документом, який регламентує норми мікроклімату робочої зони є ДСН 3.3. 6.042 -99 «Державні санітарні норми мікроклімату виробничих приміщень».

Комп'ютери і офісна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вогкості в приміщенні. В приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. В санітарних нормах встановлені величини параметрів мікроклімату, що створюють комфортні умови. (див. табл. 8.1). Значення параметрів оптимальних параметрів мікроклімату згідно з ДСН 3.3. 6.042 – 99 для приміщень, та фактичних параметрів представленні в таблиці 8.1.

Для забезпечення комфортних умов використовуються як організаційні методи (раціональна організація проведення робіт залежно від пори року і доби, чергування праці і відпочинку), так і технічні засоби (вентиляція, кондиціонування повітря, опалювальна система).

Таблиця 8.1 – Значення параметрів мікроклімату

Період року	Параметр	Оптимальний	Фактичний
Теплий	Температура	23 – 25 ^{0} С	26^0 С
	Вологість	40 – 60%	50%
	Швидкість руху повітря	< 0,1м/с	
Холодний	Температура	22 – 24^0 С	19 ^{0} С
	Вологість	40 – 60%	35%
	Швидкість руху	< 0,1м/с	

Значення фактичної вологості повітря в приміщенні в холодний період -35%, не потрапляє в діапазон допустимих значень. Отже, в холодну пору року в приміщенні необхідно використовувати зволожувачі повітря, а також для підвищення температури потрібно встановите додаткове опалення. В теплу пору року для пониження температури потрібно встановити кондиціонер.

Рівень шуму на робочому місці

Підвищений рівень шуму в комп'ютерній лабораторії спричинений чотирма ПК, двома багатофункціональними пристроями, а також гудінням пускового реле світильників. Фактичне значення рівня шуму становить 88- 92 дБ, коли допустимий рівень звуку становить ≤ ГДР, а саме 50 дБ. Методи вимірювання шуму та допустимі рівні звукового тиску у октавних смугах частот, еквівалентні рівні звуку на робочому місці регламентовані [15].

Шум погіршує умови праці здійснюючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, пониження апетиту, болі у вухах і т.і. Такі порушення в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових ситуацій. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється стомленість у зв'язку з

підвищеними енергетичними витратами і нервово – психічною напругою, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці.

Для пониження рівня шуму необхідна додаткова звукоізоляція. У якості звукоізолюючих матеріалів, які застосовують у конструкціях перекриттів для зниження передачі структурного (ударного) звуку переважно використовують мати та плити із скляного та мінерального волокна, м'які плити з деревних стружок, картон, гуму, утеплений лінолеум, а також заміна вікон на звукоізолюючі.

8.2 Заходи профілактики при роботі з комп’ютерною технікою

Умови праці працівників в цілому відповідають існуючим санітарно-гігієнічним нормам. Для того щоб особи, які працюють з ВДТ, меншою мірою втомлювались і зберігали високий рівень працездатності, потрібно раціонально організовувати їхні робочі місця.

З точки зору забезпечення електробезпеки до цих заходів можна віднести: устаткування розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв; періодична перевірка всіх приладів і пристроїв; щорічна здача іспитів з охорони праці.

З точки зору забезпечення оптимальних умов мікроклімату і освітленості до цих заходів можна віднести: організацію природної вентиляції, за допомогою дефлектора, для забезпечення необхідного повітрообміну в приміщенні; для забезпечення необхідних умов зорової роботи, що відповідають нормативним, оформлення паспорта на приміщення, з занесенням в нього вимірювань освітленості, проведених відділом охорони праці.

Таким чином, умови праці працівників в цілому відповідають існуючим санітарно-гігієнічним нормам. Але у зв’язку з тим, що більшу частину часу працівник займає сидячу позу і мало рухається, то для збереження здоров’я

працівників запобігання професійним захворюванням і підтримки працездатності слід дотримуватися вимог СанПіН 3.3.2.007-98 щодо режиму праці та відпочинку. Для цього призначаються регламентовані перерви для відпочинку.

Протягом робочого дня мають передбачатися:

– перерви для відпочинку і вживання їжі (обідні перерви);

– перерви для відпочинку і особистих потреб (згідно з трудовими нормами);

– додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності.

В окремих випадках, при постійних скаргах на зорове стомлення тих хто працює перед відеотерміналом, при дотриманні санітарно-гігієнічних вимог до режиму праці та відпочинку, а також вимог щодо застосування індивідуальних засобів локального захисту очей, допускається індивідуальний підхід до обмеження тривалості робіт перед відеотерміналом, зміни змісту роботи, чергування з іншими видами діяльності, не пов'язаними з відеотерміналом.

При виконанні робіт, що належать до різних видів трудової діяльності за основну роботу з ПК вважають роботу, що займає не менше 50% часу впродовж робочого дня. При 8-годинному робочому дні в залежності від характеру праці встановлюються режими праці та У випадках коли виробничі обставини не дозволяють застосовувати регламентовані перерви тривалість безперервної роботи за ВДТ не повинна перевищувати 4 години.

З метою зниження нервово-емоційного напруження, стомлення зорового аналізатора, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втоми ДСанПіН 3.3.2.007-98 рекомендується деякі перерви використовувати для психофізіологічного розвантаження.

Для покращення пожежної безпеки ситуації пропоную заходи, що направлені на покращення ситуації в галузі безпеки в надзвичайних ситуаціях такі:

– видання розпорядження про призначення осіб, що відповідальні за пожежну безпеку приміщення;

– щорічне проведення повторних протипожежних інструктажів та занять за програмою пожежно-технічного мінімуму з особами, що відповідальні за пожежну безпеку;

– утримання в справному стані засоби протипожежного захисту і зв'язку, обладнання та інвентар, не допускати їх використання не за призначенням;

– своєчасне інформування пожежної охорони про несправність пожежної техніки, систем протипожежного захисту, водопостачання тощо.

Необхідне також приведення систем автоматичної пожежної сигналізації та пожежогасіння у відповідності до вимог ДБН В.2.5-56:2010 та ВБН В.2.2.-00032106-1-95 що передбачає використання вогнестійких кабелів в системах живлення та забезпечення автоматичного запуску системи оповіщення та управління евакуацією людей у випадку пожежі.

В цілому приміщення по категорії вибухо- і пожежонебезпечності та ступеню вогнестійкості відповідають нормам, але особливу увагу потрібно звернути на утримання в справному стані засобів протипожежного захисту та своєчасне інформування пожежної охорони про несправність пожежної техніки, впровадження систем протипожежного захисту.

8.3 Розрахунок та проектування інженерно-технічного заходу захисту від шкідливого (небезпечного) виробничого фактору (освітленість приміщення)

Пропонуються наступні покращення умов праці на даному робочому місці: покращення умов штучного освітлення (ДБН В.2.5-28-2006 – "Природне та штучне освітлення)".

Розрахунок штучного освітлення проведемо для кімнати площею 18 м^2, ширина якої складає 3 м, довжина – 6 м, висота – 3,0 м. Скористаємося методом використання світлового потоку.

Для визначення потрібної кількості світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = \frac{E \cdot K \cdot S \cdot Z}{n}, \quad (8.1)$$

де

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення (у нашому випадку S=18м^2);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1,1...1,2, в нашому випадку Z =1,1);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

n – коефіцієнт використання світлового потоку, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп, і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{СТ.}$) і стелі ($\rho_{СТЕЛІ}$)), значення коефіцієнтів дорівнюють ρ = 40% і ρстелі=60%.

Обчислимо індекс приміщення за формулою:

$$I = \frac{S}{h(A+B)}, \quad (8.2)$$

де

S – площа приміщення, S = 18 м^2;

h – розрахункова висота підвісу, h = 3,0 м;

А – ширина приміщення, А = 3 м;

В – довжина приміщення, В = 6 м.

Підставивши значення отримаємо:

$$I = \frac{18}{3 \cdot (3+6)} = 0{,}66$$

Знаючи індекс приміщення I, за таблицею знаходимо п =0,25. Підставимо всі значення у формулу для визначення світлового потоку F:

$$F = \frac{300 \cdot 1{,}5 \cdot 18 \cdot 1{,}1}{0{,}25} = 35640\,\text{Лм}$$

Для освітлення використані люмінесцентні лампи типу ЛБ 40-1, світловий потік яких F = 4320 Лм.

Розрахуємо необхідну кількість ламп у світильниках за формулою:

$$N = \frac{F}{Fл}, \tag{8.3}$$

де

N – визначуване число ламп;

F – світловий потік, F = 35640 Лм;

Fл – світловий потік лампи, Fл = 4320 Лм.

$$N = \frac{35640}{4320} = 8{,}25$$

В приміщенні використовуються світильники типу ОД. Кожен світильник комплектується двома лампами. Тобто необхідно використовувати 4 світильника із 8 працюючими лампами в них. На момент атестації робочого місця оператора працювало 6 ламп, тому рівень штучного освітлення не задовольняв санітарним нормам.

Для покращення умов праці рекомендуємо збільшити рівень загальної освітленості приміщення шляхом встановлення 2 додаткові лампи.

8.4 Висновки

Професія програміста характеризується наявністю багатьох шкідливих та небезпечних факторів, як то: недостатній або надмірний рівень освітленості, умови мікроклімату, рівень шуму, випромінювання, особливості фізичного навантаження організму тощо.

Була проведена робота по дослідженню шкідливих та небезпечних умов праці на робочому місці програміста та дослідженню ергономічних умов праці на цьому ж робочому місці.

Розглянувши тему освітлення робочого місця можна зробити висновок, що працездатність кожного співробітника залежить не тільки від правильно організованого трудового процесу і від внутрішніх відносин в колективі, але і від того, який організований в службових приміщеннях в цілому і робоче місце даного співробітника, зокрема його раціональне освітлення.

Запропоновані технічні рішення щодо забезпечення раціонального освітлення приміщень з ЕОМ та проведено розрахунок необхідної кількості світильників в даному приміщенні, щоб забезпечити достатній рівень освітленості на робочому місці програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для порівняння та виявлення кращих технологій відновідно до поставлених сучасних вимог у галуз веб-розробки, засобами фреймворка AngularJS.

У магістерській роботі представлені дані з яких можна зробити висновки, які технології сьогодні використовуються, у яких сферах, та для його функціоналу призначені сучасні фреймворки. Які можливості та для яких проектів краще використовувати Angular та AngularJS.

Для отримання результатів були проведені наступні дослідження:

– створено односторінковий додаток засобами AngularJS та NodeJS;

– розробленно та протестовано функціонал REST API та веб-сокети за допомогою AngularJS;

– розроблено додаток засобами Angular нової версії для порівняння можливостей та функціоналу AngularJS;

– створенно сервер засобами NestJS, з підтримкою бази даних MongoDB, Redis;

– створено шаблони, які можна надалі використовувати при створенні нових додатків.

Розроблені під час виконання магістерської роботи додатки дозволяють чітко зрозуміти для яких проектів краще підходять Angular, AngularJS, NestJS, MongoDB які їх переваги та недоліки. Як вони взаємодіють між собою, та скільки займає час розробки.

Розроблені додатки підтримують функціонал, який використовується на більшості сайтів в Інтернеті, серед основних можливостей це додавання медіа файлів, можливість надсилання e-mail повідомлень, використання сокетів для спілкування в чаті, авторизація, динамічний інтерфейс.

За допомогою фреймворків Angular була продемонстрована технологія односторінкових додатків, яка надала можливість розробити швидкий та зручний для сприйняття інтерфейс.

При розробці використовувалися мови JavaScript, та TypeScript що дало можливість порівняти на скільки сильно відрізняються фреймворки Angular та AngularJS та їх основні концепції. З цього можна зробити висновки скільки займає час розробки, надійність та можливість подальшого підтримання продукту.

Програма призначена для використання у браузерах для перегляду сайта, та на Linux для завантаження сервера. В роботі були надані всі необхідні дані для розуміння роботи системи та її запуску.

В результаті розроблені додатки відповідають поставленим вимогам технічного завдання, та мають можливість на подальше вдосконалення

Створені додатки можуть бути впровадженні у виробництво.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання.

Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі *4075* грн. Враховуючи вартість розробки та необхідне обладнення, строк окуплення становить 0,9 років.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мова програмування JavaScript [Електроннй ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/

2. Introduction to Node.js [Електронний ресурс] – Режим доступу до ресурсу:https://nodejs.dev/

3. Серверне програмування [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Server-side

4. Node.js v14.0.0 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/api/

5. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих // “Node.js в деї”.

6. Янг А., Мек Б., Кантелон М. // “Node.js в действии. 2-е издание”.

7. John Resig, Bear Bibeault, Josip Maras // “Secrets of theJavaScript Ninja”.

8. Express [Електронний ресурс] – Режим доступу до ресурсу: http://expressjs.com/

9. Фреймворк AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AngularJS

10. AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://angularjs.org/

11. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/

12. React.js [Електронний ресурс] – Режим доступу до ресурсу: https://https://uk.reactjs.org/

13. AngularJS MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

14. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

15. Angular 2 [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

16. Односторінковий застосунок [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Односторінковий_застосунок

17. Что такое MVC [Електронний ресурс] – Режим доступу до ресурсу: https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami

18. Build Node.js Apps [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial

19. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

20. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/

21.Тренды веб-разработки [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/451572/

22.Веб приложение [Електронний ресурс] – Режим доступу до ресурсу: https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/

23.Мова розмітки гіпертексту [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTML

24. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/HTML

25. Каскадні таблиці стилів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CSS

26. Стек MEAN [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MEAN_(веброзробка)

27. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MongoDB

28. Веб-технології для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web

29. CORS, XSS [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/maleta/cors-xss-and-csrf-with-examples-in-10-minutes-35k3

30. AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX

31. Fetch API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API

32. AngularJS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/angular/default.asp

33. jQuery [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JQuery

34. Основи Web-технологій [Електронний ресурс] – Режим доступу до ресурсу: https://pidruchniki.com/1243020547796/informatika/web-tehnologiyi_pidpriyemstvah

35. npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/

36. Install MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mongodb.com/manual/administration/install-on-linux/

37. Как установить Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/node-js-ubuntu-18-04-ru

38. Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Linux

39. npm-audit [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/cli/audit

40. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

41. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

42. MongoDB Compass [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass

43. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/

44. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

45. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA

46. Охорона праці [Електронний ресурс]: реферат – Режим доступу до ресурсу: https://revolution.allbest.ru/life/00468031_0.html

47. Природне і штучне освітлення ДБН В.2.5-28:2018: державні будівельні норми України [Електронний ресурс] / Ю. Громадський, С. Облакевич, М.Громадський, Г. Фаренюк, Є. Фаренюк, О. Підгорний, О. Сергейчук, Є.Рейцен, В. Єгорченков, Л. Коваль, Д. Радомцев, В. Злоба, Н. Кучеренко, Г.Кожушко, О. Гончар, О. Козенко, Б. Шабашкевіч, Ю. Добровольський, В.Акіменко, С. Гозак, А. Яригін, В. Назаренко, В. Мартиросова, В. Сорокін, Є.Пугачов - Київ 2018 - Режим доступу до ресурсу: https://ledeffect.com.ua/images/__branding/dbn2018.pdf

48. Розрахунок світлодіодного освітлення кімнати [Електронний ресурс] – Режим доступу до ресурсу: https://luxled.biz.ua/rozrahynok-svitlodoidnogo-osvitlennja-kimnatu-v-kvarturi-abo-bydunky

49. Охорона праці, охорона праці та безпека в надзвичайних ситуаціях : метод. вказ. до викон. розділів у дипломних роботах / [укл. В.М. Челябієва, О.Л. Гуменюк] - Чернігів ЧДТУ 2013 - [Електронний ресурс] – Режим доступу до ресурсу: http://ir.stu.cn.ua/bitstream/handle/123456789/12461/Охорона праці та безпека. в надзв. ситуац;метод.вказ..pdf?sequence=1&isAllowed=y

50. Освітленість робочих місць [Електронний ресурс] – Режим доступу до ресурсу:https://ua-referat.com/Освітленість_робочих_місць_сучасні_підходи_до_вимірів_і_оцінки

51. Температурний режим праці [Електронний ресурс] – Режим доступу до ресурсу: http://poltava.medprof.org.ua/poltava/zakhist-trudovikh-ta-socialno-ekonomichnikh-prav-pracivnikiv-galuzi/pravova-dopomoga/temperaturnii-rezhim-praci-jakim-vin-maje-buti/

52. Шум. Методи захисту від його дії : метод. вказ. до лабораторної роботи / [укл. В. І. Шмирко, С. М. Журавель] — Запоріжжя: ЗНТУ, 2014. - 14 с. [Електронний ресурс] – Режим доступу до ресурсу: https://zp.edu.ua/sites/default/files/konf/oop_shum-2014.pdf

53. Системи протипожежного захисту ДБН В.2.5-56:2014 / Б. Платкевич, В. Носач, В. Федюк, В. Мусійчук, В. Євстіфєєв, Г. Дубінський, В. Сокол, А.Бушиленко, В. Дунюшкін, Р. Уханський, С. Пономарьов, В. Приймаченко, А.Приймаченко, С. Пітайчук, Н. Морозова, І. Колосов, О. Лагода, П. Мізін, В.Савченко, М. Федорович, П. Шаповалов, Л. Фесенко — Київ 2015 — [Електронний ресурс] – Режим доступу до ресурсу: http://kbu.org.ua/assets/app/documents/dbn2/98.1. ДБН В.2.5-56~2014. Системи протипожежного захисту.pdf

54. Охорона праці. Ч. 2. Занулення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ IBM−сумісного типу / [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака, А. Е. Солових, С. Е. Катеринич] ; Центральноукраїн. нац. техн. ун-т. - 2−ге вид., перероб. та доп. - Кропивницький : ЦНТУ, 2019. - 27 с.[Електронний ресурс] – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/8769

55. The Top JavaScript Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/

56. JavaScript Frameworks 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/migrate-mongo

57. Web Technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/web-technology/

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Економічні вимоги.....5
8 Вимоги щодо охорони праці.....5
9 Перелік документів, що розробляються.....6
10 Етапи розробки.....6
11 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи захисту в інформаційній системі.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 33-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою магістерської дипломної роботи є дослідження та програмна реалізація веб-сайту компанії засобами фреймворків JS

4 Джерела розробки

Джерелом цієї магістерської дипломної роботи є розробки, які ведуться спільнотою розробників фреймворків і стосовна до теми технічна література.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– створення і налаштування серверних додатків;

– веб-сайт створений на AngularJS;

– простий, інтуїтивно зрозумілий інтерфейс з користувачем.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення повинно дозволити достатньо легко зберігати, редагувати, переглядати дані у браузері.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10/11 та Linux.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

JavaScript з використанням платформи NodeJS та фреймворку AngularJS

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці та техніки безпеки в магістерській роботі повинен бути розглянутий аналіз умов праці програміста та розрахунок освітлення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 1 аркуш.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 104 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі бакалаврської дипломної роботи. Постановка задачі на виконання бакалаврської дипломної роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень бакалаврської дипломної роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

11.1 Подання бакалаврської дипломної роботи на попередній захист 10.12.2023 р.

1.2 Подання магістерської роботи на захист 20.12.2023 р.

Додаток Б

(обов'язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Босько В.В.

Дослідження та програмна реалізація веб-застосунків засобами фреймворків JS як інструментів розробки

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 28

Літера: РП

Кропивницький – 2023 року

Файл основного файлу програми для завантаження AngularJS

```
<!DOCTYPE html>
<html ng-app="app">
<head>
    <meta charset="utf-8" />
    <title>AngularJS</title>
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" />
    <link href="app-content/app.css" rel="stylesheet" />
</head>
<body>
    <div class="jumbotron2">
        <div class="container">
            <div class="col-sm-8 col-sm-offset-2">
                <div ng-class="{ 'alert': flash, 'alert-success':
flash.type === 'success', 'alert-danger': flash.type === 'error' }" ng-
if="flash" ng-bind="flash.message"></div>
                <div ng-view></div>
            </div>
        </div>
    </div>

    <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular-
route.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular-
cookies.min.js"></script>

    <script src="app.js"></script>
    <script src="app-services/authentication.service.js"></script>
    <script src="app-services/flash.service.js"></script>

    <script src="app-services/user.service.local-storage.js"></script>

    <script src="home/home.controller.js"></script>
    <script src="login/login.controller.js"></script>
    <script src="register/register.controller.js"></script>
    <script src="sidebar/sidebar.controller.js"></script>
    <script src="chat/chat.controller.js"></script>
    <script src="technologies/technologies.controller.js"></script>
    <script src="company/company.controller.js"></script>
    <script src="settings/settings.controller.js"></script>
</body>
</html>
```

Файл який відповідає за маршрутизацію додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app', ['ngRoute', 'ngCookies'])
        .config(config)
        .run(run);

    config.$inject = ['$routeProvider', '$locationProvider'];
    function config($routeProvider, $locationProvider) {
        $routeProvider
            .when('/', {
                controller: 'HomeController',
                templateUrl: 'home/home.view.html',
                controllerAs: 'vm'
            })
            .when('/login', {
                controller: 'LoginController',
                templateUrl: 'login/login.view.html',
                controllerAs: 'vm'
            })
            .when('/register', {
                controller: 'RegisterController',
                templateUrl: 'register/register.view.html',
                controllerAs: 'vm'
            })
            .when('/sidebar', {
                controller: 'RegisterController',
                templateUrl: 'sidebar/sidebar.view.html',
                controllerAs: 'vm'
            })
            .when('/technologies', {
                controller: 'TechnologiesController',
                templateUrl: 'technologies/technologies.view.html',
                controllerAs: 'vm'
            })
            .when('/settings', {
                controller: 'SettingsController',
                templateUrl: 'settings/settings.view.html',
                controllerAs: 'vm'
            })
            .when('/company', {
                controller: 'CompanyController',
                templateUrl: 'company/company.view.html',
                controllerAs: 'vm'
            })
            .when('/chat', {
                controller: 'ChatController',
                templateUrl: 'chat/chat.view.html',
                controllerAs: 'vm'
            })
            .otherwise({ redirectTo: '/login' });
    }

    run.$inject = ['$rootScope', '$location', '$cookies', '$http'];
    function run($rootScope, $location, $cookies, $http) {
        $rootScope.globals = $cookies.getObject('globals') || {};
        if ($rootScope.globals.currentUser) {
            $http.defaults.headers.common['Authorization'] = 'Basic ' +
$rootScope.globals.currentUser.authdata;
        }
    }
```

```
})();
```

Файл який відповідає логіку реєстрації додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('RegisterController', RegisterController);

    RegisterController.$inject = ['UserService', '$location',
'$rootScope', 'FlashService'];
    function RegisterController(UserService, $location, $rootScope,
FlashService) {
        var vm = this;

        vm.register = register;

        function register() {
            vm.dataLoading = true;
            UserService.Create(vm.user)
                .then(function (response) {
                    if (response.success) {
                        FlashService.Success('Registration successful',
true);
                        $location.path('/login');
                    } else {
                        FlashService.Error(response.message);
                        vm.dataLoading = false;
                    }
                });
        }
    }

})();
```

Файл який відповідає за представлення реєстрації додатка на AngularJS

```
      <div class="col-md-6 col-md-offset-3">
          <h2>Register</h2>
          <form name="form" ng-submit="vm.register()" role="form">
              <div class="form-group" ng-class="{ 'has-error':
form.firstName.$dirty && form.firstName.$error.required }">
                  <label for="username">First name</label>
                  <input type="text" name="firstName" id="firstName"
class="form-control" ng-model="vm.user.firstName" required />
                  <span ng-show="form.firstName.$dirty &&
form.firstName.$error.required" class="help-block">First name is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.lastName.$dirty && form.lastName.$error.required }">
                  <label for="username">Last name</label>
                  <input type="text" name="lastName" id="Text1" class="form-
control" ng-model="vm.user.lastName" required />
                  <span ng-show="form.lastName.$dirty &&
form.lastName.$error.required" class="help-block">Last name is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.username.$dirty && form.username.$error.required }">
                  <label for="username">Username</label>
                  <input type="text" name="username" id="username" class="form-
control" ng-model="vm.user.username" required />
                  <span ng-show="form.username.$dirty &&
form.username.$error.required" class="help-block">Username is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.password.$dirty && form.password.$error.required }">
                  <label for="password">Password</label>
                  <input type="password" name="password" id="password"
class="form-control" ng-model="vm.user.password" required />
                  <span ng-show="form.password.$dirty &&
form.password.$error.required" class="help-block">Password is required</span>
              </div>
              <div class="form-actions">
                  <button type="submit" ng-disabled="form.$invalid ||
vm.dataLoading" class="btn btn-primary">Register</button>
                  <img ng-if="vm.dataLoading"
src="
05FVFNDQVBFMi4wAwEAAAAh/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAE
AAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzkYDAdKa+dIAAAh+
QQJCgAAACwAAAAAEAAQAAADNAi63P5OjCEgG4QMu7DmikRxQlFUYDEZIGBMRVsaqHwctXXf7WEYB4Ag1
xjihkMZsiUkKhIAIfkECQoAAAAsAAAAABAAEAAAAzYIujIjK8pByJDMlFYvBoVjHA70GU7xSUJhmKtwH
PAKzLO9HMaoKwJZ7Rf8AYPDDzKpZBqfvwQAIfkECQoAAAAsAAAAABAAEAAAAzMIumIlK8oyhpHsnFZfh
YumCYUhDAQxRIdhHBGqRoKw0R8DYlJd8z0fMDgsGo/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIu
nInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAA
AAAEAAQAAADMwi6IMKQORfjdOe82p4wGccc4CEuQradylesojEMBgsUc2G7sDX3lQGBMLAJibufbSlKA
AAh+QQJCgAAACwAAAAAEAAQAAADMgi63P7wCRHZnFVdmgHu2nFwlWCI3WGc3TSWhUFGxTAUkGCbtgENB
MJAEJsxgMLWzpEAACH5BAkKAAAALAAAAAAQABAAAAMyCLrc/jDKSatlQtScKdceCAjDII7HcQ4EMTCpy
rCuUBjCYRgHVtqlAiB1YhiCnlsRkAAAOwAAAAAAAAAAA==" />
                  <a href="#!/login" class="btn btn-link">Cancel</a>
              </div>
          </form>
      </div>
```

Файл який відповідає за представлення логіна додатка на AngularJS

```
      <div class="col-md-6 col-md-offset-3">
          <h2>Login</h2>
          <form name="form" ng-submit="vm.login()" role="form">
              <div class="form-group" ng-class="{ 'has-error':
form.username.$dirty && form.username.$error.required }">
                  <label for="username">Username</label>
                  <input type="text" name="username" id="username" class="form-
control" ng-model="vm.username" required />
                  <span ng-show="form.username.$dirty &&
form.username.$error.required" class="help-block">Username is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.password.$dirty && form.password.$error.required }">
                  <label for="password">Password</label>
                  <input type="password" name="password" id="password"
class="form-control" ng-model="vm.password" required />
                  <span ng-show="form.password.$dirty &&
form.password.$error.required" class="help-block">Password is required</span>
              </div>
              <div class="form-actions">
                  <button type="submit" ng-disabled="form.$invalid ||
vm.dataLoading" class="btn btn-primary">Login</button>
                  <img ng-if="vm.dataLoading"
src="
05FVFNDQVBFMi4wAwEAAAAh/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAE
AAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzkYDAdKa+dIAAAh+
QQJCgAAACwAAAAAEAAQAAADNAi63P5OjCEgG4QMu7DmikRxQlFUYDEZIGBMRVsaqHwctXXf7WEYB4Ag1
xjihkMZsiUkKhIAIfkECQoAAAAsAAAAABAAEAAAAzYIujIjK8pByJDMlFYvBoVjHA70GU7xSUJhmKtwH
PAKzLO9HMaoKwJZ7Rf8AYPDDzKpZBqfvwQAIfkECQoAAAAsAAAAABAAEAAAAzMIumIlK8oyhpHsnFZfh
YumCYUhDAQxRIdhHBGqRoKw0R8DYlJd8z0fMDgsGo/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIu
nInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAA
AAAEAAQAAADMwi6IMKQORfjdOe82p4wGccc4CEuQradylesojEMBgsUc2G7sDX3lQGBMLAJibufbSlKA
AAh+QQJCgAAACwAAAAAEAAQAAADMgi63P7wCRHZnFVdmgHu2nFwlWCI3WGc3TSWhUFGxTAUkGCbtgENB
MJAEJsxgMLWzpEAACH5BAkKAAAALAAAAAAQABAAAAMyCLrc/jDKSatlQtScKdceCAjDII7HcQ4EMTCpy
rCuUBjCYRgHVtqlAiB1YhiCnlsRkAAAOwAAAAAAAAAAA==" />
                  <a href="#!/register" class="btn btn-link">Register</a>
              </div>
          </form>
      </div>

      <div ng-controller="SidebarController">
          <div ng-class="'test'">
              {{phone.name}}
          </div>
      </div>
```

Файл який відповідає за логіку логіна додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('LoginController', LoginController);

    LoginController.$inject = ['$location', 'AuthenticationService',
'FlashService'];
    function LoginController($location, AuthenticationService,
FlashService) {
        var vm = this;

        vm.login = login;

        (function initController() {
            // reset login status
            AuthenticationService.ClearCredentials();
        })();

        function login() {
            vm.dataLoading = true;
            AuthenticationService.Login(vm.username, vm.password, function
(response) {
                if (response.success) {
                    AuthenticationService.SetCredentials(vm.username,
vm.password);
                    $location.path('/');
                } else {
                    FlashService.Error(response.message);
                    vm.dataLoading = false;
                }
            });
        };
    }

})();
```

Файл для завантаження сервера main.ts

```
import 'module-alias/register';

import { NestFactory } from '@nestjs/core';
import { ValidationPipe, ValidationError } from '@nestjs/common';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
import ValidationExceptions from './exceptions/validation.exceptions';

import AppModule from './routes/app/app.module';

import AllExceptionsFilter from './filters/all-exceptions.filter';

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

  app.useGlobalPipes(new ValidationPipe({
    exceptionFactory: (errors: ValidationError[]) => new
ValidationExceptions(errors),
  }));
  app.useGlobalFilters(new AllExceptionsFilter());

  const port = process.env.SERVER_PORT || 3000;

  const options = new DocumentBuilder()
    .setTitle('Api v1')
    .setDescription('The boilerplate API for nestjs devs')
    .setVersion('1.0')
    .addBearerAuth({ in: 'header', type: 'http' })
    .build();
  const document = SwaggerModule.createDocument(app, options);

  SwaggerModule.setup('api', app, document);

  await app.listen(port, async () => {
    console.log(`The server is running on ${port} port:
http://localhost:${port}/api`);
  });
}
bootstrap();
```

Файл контролер для реєстрації та длгіну на сервері

```
import {
  Body,
  Controller,
  HttpCode,
  Get,
  Post,
  Delete,
  Param,
  Request,
  UnauthorizedException,
  UseGuards,
  NotFoundException,
  ForbiddenException,
  HttpStatus,
  UseInterceptors,
} from '@nestjs/common';
import {
  ApiTags,
  ApiBody,
  ApiOkResponse,
  ApiInternalServerErrorResponse,
  ApiUnauthorizedResponse,
  ApiBearerAuth,
  ApiNotFoundResponse,
  ApiBadRequestResponse,
  ApiConflictResponse,
  ApiNoContentResponse,
  ApiExtraModels,
  getSchemaPath,
} from '@nestjs/swagger';
import { JwtService } from '@nestjs/jwt';
import { Request as ExpressRequest } from 'express';
import { MailerService } from '@nestjs-modules/mailer';

import UsersService from '@v1/users/users.service';
import JwtAccessGuard from '@guards/jwt-access.guard';
import RolesGuard from '@guards/roles.guard';
import { User } from '@v1/users/schemas/users.schema';
import WrapResponseInterceptor from '@interceptors/wrap-
response.interceptor';
import AuthBearer from '@decorators/auth-bearer.decorator';
import { Roles, RolesEnum } from '@decorators/roles.decorator';
import authConstants from '@v1/auth/auth-constants';
import { DecodedUser } from './interfaces/decoded-user.interface';
import LocalAuthGuard from './guards/local-auth.guard';
import AuthService from './auth.service';
import RefreshTokenDto from './dto/refresh-token.dto';
import SignInDto from './dto/sign-in.dto';
import SignUpDto from './dto/sign-up.dto';
import JwtTokensDto from './dto/jwt-tokens.dto';
import UsersEntity from '@v1/users/entity/user.entity';

@ApiTags('Auth')
@UseInterceptors(WrapResponseInterceptor)
@ApiExtraModels(JwtTokensDto)
@Controller()
export default class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly jwtService: JwtService,
    private readonly usersService: UsersService,
```

```
  private readonly mailerService: MailerService,
) {}

@ApiBody({ type: SignInDto })
@ApiOkResponse({
  schema: {
    type: 'object',
    properties: {
      data: {
        $ref: getSchemaPath(JwtTokensDto),
      },
    },
  },
  description: 'Returns jwt tokens',
})
@ApiBadRequestResponse({
  schema: {
    type: 'object',
    example: {
      message: [
        {
          target: {
            email: 'string',
            password: 'string',
          },
          value: 'string',
          property: 'string',
          children: [],
          constraints: {},
        },
      ],
      error: 'Bad Request',
    },
  },
  description: '400. ValidationException',
})
@ApiInternalServerErrorResponse({
  schema: {
    type: 'object',
    example: {
      message: 'string',
      details: {},
    },
  },
  description: '500. InternalServerError',
})
@ApiBearerAuth()
@HttpCode(HttpStatus.OK)
@UseGuards(LocalAuthGuard)
@Post('sign-in')
async signIn(@Request() req: ExpressRequest): Promise<JwtTokensDto> {
  const user = req.user as User;

  return this.authService.login(user);
}

@ApiBody({ type: SignUpDto })
@ApiOkResponse({
  description: '201, Success',
})
@ApiBadRequestResponse({
  schema: {
    type: 'object',
    example: {
```

```
            message: [
              {
                target: {
                  email: 'string',
                  password: 'string',
                },
                value: 'string',
                property: 'string',
                children: [],
                constraints: {},
              },
            ],
            error: 'Bad Request',
          },
        },
        description: '400. ValidationException',
      })
      @ApiConflictResponse({
        schema: {
          type: 'object',
          example: {
            message: 'string',
          },
        },
        description: '409. ConflictResponse',
      })
      @ApiInternalServerErrorResponse({
        schema: {
          type: 'object',
          example: {
            message: 'string',
            details: {},
          },
        },
        description: '500. InternalServerError',
      })
      @HttpCode(HttpStatus.CREATED)
      @Post('sign-up')
      async signUp(@Body() user: SignUpDto): Promise<any> {
        const { _id, email } = await this.usersService.create(user) as
UsersEntity;

        const token = this.authService.createVerifyToken(_id);

        await this.mailerService.sendMail({
          to: email,
          from: process.env.MAILER_FROM_EMAIL,
          subject: authConstants.mailer.verifyEmail.subject,
          template: `${process.cwd()}/src/templates/verify-password`,
          context: {
            token,
            email,
            host: process.env.SERVER_HOST,
          },
        });

        return { message: 'Success! please verify your email' };
      }

      @ApiOkResponse({
        schema: {
          type: 'object',
          properties: {
            data: {
```

```
              $ref: getSchemaPath(JwtTokensDto),
            },
          },
        },
        description: '200, returns new jwt tokens',
      })
      @ApiUnauthorizedResponse({
        schema: {
          type: 'object',
          example: {
            message: 'string',
          },
        },
        description: '401. Token has been expired',
      })
      @ApiInternalServerErrorResponse({
        schema: {
          type: 'object',
          example: {
            message: 'string',
            details: {},
          },
        },
        description: '500. InternalServerError ',
      })
      @ApiBearerAuth()
      @Post('refresh-token')
      async refreshToken(
        @Body() refreshTokenDto: RefreshTokenDto,
      ): Promise<JwtTokensDto | never>  {
        const decodedUser = this.jwtService.decode(
          refreshTokenDto.refreshToken,
        ) as DecodedUser;

        if (!decodedUser) {
          throw new ForbiddenException('Incorrect token');
        }

        const oldRefreshToken:
          | string
          | null = await
this.authService.getRefreshTokenByEmail(decodedUser.email);

        // if the old refresh token is not equal to request refresh token then
this user is unauthorized
        if (!oldRefreshToken || oldRefreshToken !==
refreshTokenDto.refreshToken) {
          throw new UnauthorizedException(
            'Authentication credentials were missing or incorrect',
          );
        }

        const payload = {
          _id: decodedUser._id,
          email: decodedUser.email,
          role: decodedUser.role,
        };

        return this.authService.login(payload);
      }

      @ApiNoContentResponse({
        description: 'No content. 204',
      })
```

```
        @ApiNotFoundResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
              error: 'Not Found',
            },
          },
          description: 'User was not found',
        })
        @HttpCode(HttpStatus.NO_CONTENT)
        @Get('verify/:token')
        async verifyUser(@Param('token') token: string): Promise<User | null> {
          const { id } = await this.authService.verifyEmailVerToken(
            token,
            authConstants.jwt.secrets.accessToken,
          );
          const foundUser = await this.usersService.getUnverifiedUserById(id) as
UsersEntity;

          if (!foundUser) {
            throw new NotFoundException('The user does not exist');
          }

          return this.usersService.update(foundUser._id, { verified: true });
        }

        @ApiNoContentResponse({
          description: 'no content',
        })
        @ApiUnauthorizedResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
            },
          },
          description: 'Token has been expired',
        })
        @ApiInternalServerErrorResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
              details: {},
            },
          },
          description: 'InternalServerError',
        })
        @ApiBearerAuth()
        @UseGuards(JwtAccessGuard)
        @Delete('logout/:token')
        @HttpCode(HttpStatus.NO_CONTENT)
        async logout(@Param('token') token: string): Promise<{} | never> {
          const decodedUser: DecodedUser | null = await
this.authService.verifyToken(
            token,
            authConstants.jwt.secrets.accessToken,
          );

          if (!decodedUser) {
            throw new ForbiddenException('Incorrect token');
          }
```

```
      const deletedUsersCount = await this.authService.deleteTokenByEmail(
        decodedUser.email,
      );

      if (deletedUsersCount === 0) {
        throw new NotFoundException();
      }
      return {};
    }

    @ApiNoContentResponse({
      description: 'no content',
    })
    @ApiInternalServerErrorResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
          details: {},
        },
      },
      description: '500. InternalServerError',
    })
    @ApiBearerAuth()
    @Delete('logout-all')
    @UseGuards(RolesGuard)
    @Roles(RolesEnum.admin)
    @HttpCode(HttpStatus.NO_CONTENT)
    async logoutAll(): Promise<{}> {
      return this.authService.deleteAllTokens();
    }

    @ApiOkResponse({
      type: User,
      description: '200, returns a decoded user from access token',
    })
    @ApiUnauthorizedResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
        },
      },
      description: '403, says you Unauthorized',
    })
    @ApiInternalServerErrorResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
          details: {},
        },
      },
      description: '500. InternalServerError',
    })
    @ApiBearerAuth()
    @UseGuards(JwtAccessGuard)
    @Get('token')
    async getUserByAccessToken(
      @AuthBearer() token: string,
    ): Promise<DecodedUser | never> {
      const decodedUser: DecodedUser | null = await
this.authService.verifyToken(
        token,
```

```
      authConstants.jwt.secrets.accessToken,
    );

    if (!decodedUser) {
      throw new ForbiddenException('Incorrect token');
    }

    const { exp, iat, ...user } = decodedUser;

    return user;
  }
}
```

Файл сервіс для реєстрації та лгіну на сервері

```
import * as bcrypt from 'bcrypt';

import { Injectable, NotFoundException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Types } from 'mongoose';

import UsersRepository from '@v1/users/users.repository';
import { UserInterface } from '@v1/users/interfaces/user.interface';
import { DecodedUser } from './interfaces/decoded-user.interface';
import JwtTokensDto from './dto/jwt-tokens.dto';
import { LoginPayload } from './interfaces/login-payload.interface';

import authConstants from './auth-constants';
import AuthRepository from './auth.repository';
import UsersEntity from '@v1/users/entity/user.entity';

@Injectable()
export default class AuthService {
  constructor(
    private readonly jwtService: JwtService,
    private readonly usersRepository: UsersRepository,
    private readonly authRepository: AuthRepository,
  ) {}

  public async validateUser(
    email: string,
    password: string,
  ): Promise<null | UserInterface> {
    const user = await this.usersRepository.getVerifiedUserByEmail(email)
as UsersEntity;

    if (!user) {
      throw new NotFoundException('The item does not exist');
    }

    const passwordCompared = await bcrypt.compare(password,
user.password);

    if (passwordCompared) {
      return {
        _id: user._id,
    });

    await this.authRepository.addRefreshToken(
      payload.email as string,
      refreshToken,
    );

    return {
      accessToken,
      refreshToken,
    };
  }

  public getRefreshTokenByEmail(email: string): Promise<string | null> {
    return this.authRepository.getToken(email);
  }

  public deleteTokenByEmail(email: string): Promise<number> {
    return this.authRepository.removeToken(email);
  }
```

```
  public deleteAllTokens(): Promise<string> {
    return this.authRepository.removeAllTokens();
  }

  public createVerifyToken(id: Types.ObjectId): string {
    return this.jwtService.sign(
      { id },
      {
        expiresIn: authConstants.jwt.expirationTime.accessToken,
        secret: authConstants.jwt.secrets.accessToken,
      },
    );
  }

  public verifyEmailVerToken(token: string, secret: string) {
    return this.jwtService.verifyAsync(token, { secret });
  }

  public async verifyToken(
    token: string,
    secret: string,
  ): Promise<DecodedUser | null> {
    try {
      const user = (await this.jwtService.verifyAsync(token, {
        secret,
      })) as DecodedUser | null;

      return user;
    } catch (error) {
      return null;
    }
  }
}     email: user.email,
        role: user.role,
      };
    }

    return null;
  }

  public async login(data: LoginPayload): Promise<JwtTokensDto> {
    const payload: LoginPayload = {
      _id: data._id,
      email: data.email,
      role: data.role,
    };

    const accessToken = this.jwtService.sign(payload, {
      expiresIn: authConstants.jwt.expirationTime.accessToken,
      secret: authConstants.jwt.secrets.accessToken,
    });
    const refreshToken = this.jwtService.sign(payload, {
      expiresIn: authConstants.jwt.expirationTime.refreshToken,
      secret: authConstants.jwt.secrets.refreshToken,
    });

    await this.authRepository.addRefreshToken(
      payload.email as string,
      refreshToken,
    );

    return {
      accessToken,
```

```
      refreshToken,
    };
  }

  public getRefreshTokenByEmail(email: string): Promise<string | null> {
    return this.authRepository.getToken(email);
  }

  public deleteTokenByEmail(email: string): Promise<number> {
    return this.authRepository.removeToken(email);
  }

  public deleteAllTokens(): Promise<string> {
    return this.authRepository.removeAllTokens();
  }

  public createVerifyToken(id: Types.ObjectId): string {
    return this.jwtService.sign(
      { id },
      {
        expiresIn: authConstants.jwt.expirationTime.accessToken,
        secret: authConstants.jwt.secrets.accessToken,
      },
    );
  }

  public verifyEmailVerToken(token: string, secret: string) {
    return this.jwtService.verifyAsync(token, { secret });
  }

  public async verifyToken(
    token: string,
    secret: string,
  ): Promise<DecodedUser | null> {
    try {
      const user = (await this.jwtService.verifyAsync(token, {
        secret,
      })) as DecodedUser | null;

      return user;
    } catch (error) {
      return null;
    }
  }
}
```

Файл репозиторій для реєстрації та лгіну на сервері

```
import * as Redis from 'ioredis';
import { Injectable } from '@nestjs/common';

import { RedisService } from 'nestjs-redis';
import authConstants from './auth-constants';

@Injectable()
export default class AuthRepository {
  private readonly redisClient: Redis.Redis;

  constructor(private readonly redisService: RedisService) {
    this.redisClient = redisService.getClient();
  }

  public async addRefreshToken(userEmail: string, refreshToken: string):
Promise<void> {
    await this.redisClient.set(
      userEmail,
      refreshToken,
      'EX',
      authConstants.redis.expirationTime.jwt.refreshToken,
    );
  }

  public getToken(key: string): Promise<string | null> {
    return this.redisClient.get(key);
  }

  public removeToken(key: string): Promise<number> {
    return this.redisClient.del(key);
  }

  public removeAllTokens(): Promise<string> {
    return this.redisClient.flushall();
  }
}
```

Файл модуль для маршрутизації в Angular

```
      import {NO_ERRORS_SCHEMA, NgModule} from '@angular/core';
      import { Routes, RouterModule } from '@angular/router';

      import { HomeComponent } from './home';
      import { AuthGuard } from './_helpers';

      const accountModule = () => import('./account/account.module').then(x =>
x.AccountModule);
      const usersModule = () => import('./users/users.module').then(x =>
x.UsersModule);
      const profileModule = () => import('./profile/profile.module').then(x =>
x.ProfileModule);
      const documentModule = () => import('./document/document.model').then(x =>
x.DocumentModule);
      const documentChat = () => import('./chat/chat.model').then(x =>
x.ChatModule);
      const documentCompany = () => import('./company/company.model').then(x =>
x.CompanyModule);
      const documentTechnologies = () =>
import('./technologies/technologies.model').then(x => x.TechnologiesModule);
      const documentSettings = () => import('./settings/settings.model').then(x
=> x.SettingsModule);

      const routes: Routes = [
          { path: '', component: HomeComponent, canActivate: [AuthGuard] },
          { path: 'users', loadChildren: usersModule, canActivate:
[AuthGuard] },
          { path: 'account', loadChildren: accountModule },
          { path: 'profile', loadChildren: profileModule },
          { path: 'document', loadChildren: documentModule },
          { path: 'chat', loadChildren: documentChat },
          { path: 'company', loadChildren: documentCompany },
          { path: 'technologies', loadChildren: documentTechnologies },
          { path: 'settings', loadChildren: documentSettings },

          { path: '**', redirectTo: '' }
      ];

      @NgModule({
          imports: [RouterModule.forRoot(routes)],
          exports: [RouterModule],
          schemas: [NO_ERRORS_SCHEMA]
      })
      export class AppRoutingModule { }
```

Файл який відповідає за логіку логіна додатка на Angular

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'login.component.html' })
export class LoginComponent implements OnInit {
    form: FormGroup;
    loading = false;
    submitted = false;
    returnUrl: string;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            username: ['', Validators.required],
            password: ['', Validators.required]
        });

        this.returnUrl = this.route.snapshot.queryParams['returnUrl'] ||
'/';
    }

    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        this.alertService.clear();

        if (this.form.invalid) {
            return;
        }

        this.loading = true;
        this.accountService.login(this.f.username.value,
this.f.password.value)
            .pipe(first())
            .subscribe(
                data => {
                    this.router.navigate([this.returnUrl]);
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                });
    }
}
```

Файл який відповідає за представлення логіна додатка на Angular

```
      <div class="card">
          <h4 class="card-header">Login</h4>
          <div class="card-body">
              <form [formGroup]="form" (ngSubmit)="onSubmit()">
                  <div class="form-group">
                      <label for="username">Username</label>
                      <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
                      <div *ngIf="submitted && f.username.errors"
class="invalid-feedback">
                          <div *ngIf="f.username.errors.required">Username is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="password">Password</label>
                      <input type="password" formControlName="password"
class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.password.errors }" />
                      <div *ngIf="submitted && f.password.errors"
class="invalid-feedback">
                          <div *ngIf="f.password.errors.required">Password is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <button [disabled]="loading" class="btn btn-primary">
                          <span *ngIf="loading" class="spinner-border spinner-
border-sm mr-1"></span>
                          Login
                      </button>
                      <a routerLink="../register" class="btn btn-
link">Register</a>
                  </div>
              </form>
          </div>
      </div>
```

Файл який відповідає за логіку реєстрації додатка на Angular

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'register.component.html' })
export class RegisterComponent implements OnInit {
    form: FormGroup;
    loading = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            firstName: ['', Validators.required],
            lastName: ['', Validators.required],
            username: ['', Validators.required],
            password: ['', [Validators.required, Validators.minLength(6)]]
        });
    }

    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        this.alertService.clear();

        if (this.form.invalid) {
            return;
        }

        this.loading = true;
        this.accountService.register(this.form.value)
            .pipe(first())
            .subscribe(
                data => {
                    this.alertService.success('Registration successful',
{ keepAfterRouteChange: true });
                    this.router.navigate(['../login'], { relativeTo:
this.route });
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                });
    }
}
```

Файл який відповідає за представлення реєстрації додатка на Angular

```
      <div class="card">
          <h4 class="card-header">Register</h4>
          <div class="card-body">
              <form [formGroup]="form" (ngSubmit)="onSubmit()">
                  <div class="form-group">
                      <label for="firstName">First Name</label>
                      <input type="text" formControlName="firstName"
class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.firstName.errors }" />
                      <div *ngIf="submitted && f.firstName.errors"
class="invalid-feedback">
                          <div *ngIf="f.firstName.errors.required">First Name is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="lastName">Last Name</label>
                      <input type="text" formControlName="lastName" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.lastName.errors }" />
                      <div *ngIf="submitted && f.lastName.errors"
class="invalid-feedback">
                          <div *ngIf="f.lastName.errors.required">Last Name is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="username">Username</label>
                      <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
                      <div *ngIf="submitted && f.username.errors"
class="invalid-feedback">
                          <div *ngIf="f.username.errors.required">Username is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="password">Password</label>
                      <input type="password" formControlName="password"
class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.password.errors }" />
                      <div *ngIf="submitted && f.password.errors"
class="invalid-feedback">
                          <div *ngIf="f.password.errors.required">Password is
required</div>
                          <div *ngIf="f.password.errors.minlength">Password must
be at least 6 characters</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <button [disabled]="loading" class="btn btn-primary">
                          <span *ngIf="loading" class="spinner-border spinner-
border-sm mr-1"></span>
                          Register
                      </button>
                      <a routerLink="../login" class="btn btn-link">Cancel</a>
                  </div>
              </form>
          </div>
      </div>
```

Файл сервіс який відповідає за юзера додатка на Angular

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { environment } from '@environments/environment';
import { User } from '@app/_models';

@Injectable({ providedIn: 'root' })
export class AccountService {
    private userSubject: BehaviorSubject<User>;
    public user: Observable<User>;

    constructor(
        private router: Router,
        private http: HttpClient
    ) {
        this.userSubject = new
BehaviorSubject<User>(JSON.parse(localStorage.getItem('user')));
        this.user = this.userSubject.asObservable();
    }

    public get userValue(): User {
        return this.userSubject.value;
    }

    login(username, password) {
        return
this.http.post<User>(`${environment.apiUrl}/users/authenticate`, { username,
password })
            .pipe(map(user => {
                localStorage.setItem('user', JSON.stringify(user));
                this.userSubject.next(user);
                return user;
            }));
    }

    logout() {
        localStorage.removeItem('user');
        this.userSubject.next(null);
        this.router.navigate(['/account/login']);
    }

    register(user: User) {
        return this.http.post(`${environment.apiUrl}/users/register`,
user);
    }

    getAll() {
        return this.http.get<User[]>(`${environment.apiUrl}/users`);
    }

    getById(id: string) {
        return this.http.get<User>(`${environment.apiUrl}/users/${id}`);
    }

    update(id, params) {
        return this.http.put(`${environment.apiUrl}/users/${id}`, params)
            .pipe(map(x => {
                if (id == this.userValue.id) {
```

```
                    const user = { ...this.userValue, ...params };
                    localStorage.setItem('user', JSON.stringify(user));

                    this.userSubject.next(user);
                }
                return x;
            }));
    }

    delete(id: string) {
        return this.http.delete(`${environment.apiUrl}/users/${id}`)
            .pipe(map(x => {
                if (id == this.userValue.id) {
                    this.logout();
                }
                return x;
            }));
    }
}
```

Файл схеми юзера на сервері

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

import { RolesEnum } from '@decorators/roles.decorator';

@Schema()
export class User {
  @Prop({
    required: true,
    unique: true,
    type: String,
  })
  email: string = '';

  @Prop({
    required: true,
    type: String,
  })
  password: string = '';

  @Prop({
    required: true,
    type: Boolean,
  })
  verified: boolean = false;

  @Prop({
    type: RolesEnum,
    required: false,
    default: RolesEnum.user,
  })
  role: RolesEnum = RolesEnum.user;
}

export type UserDocument = User & Document;

export const UserSchema =
SchemaFactory.createForClass(User).set('versionKey', false);
```