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
“____” _____ 2021 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS”

Виконав здобувач вищої освіти
ІІ курсу, групи _____
ОПП «Комп'ютерна інженерія»
спеціальності 123 «Комп'ютерна інженерія»
_____ Фесечко Д.В.
«____» _____ 2021р.

Керівник проекту
кандидат технічних наук, доцент
_____ Босько В.В.
«____» _____ 2021 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *магістр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123 "Комп'ютерна інженерія"*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

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

Фесечка ДенисаВікторовича
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS*

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

затверджені наказом вищого навчального закладу № 42-13 від 02.08.21

3. Строк подання роботи до захисту *20.12.2021 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS*

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	*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	09.11.2021 р.	17.11.2021 р.
Охорона праці	Оришака О.В., к.т.н., доцент	03.11.2021 р.	21.11.2021 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	12.10.2021 р.	
2.	Постановка задачі, оформлення ТЗ	18.10.2021 р.	
3.	Розробка моделі компонента	23.10.2021 р.	
4.	Розробка структур даних	25.10.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	32.10.2021 р.	
6.	Програмування алгоритмів	11.11.2021 р.	
7.	Розрахунок економічної ефективності	13.11.2021 р.	
8.	Розрахунки з охорони праці та техніки безпеки	16.11.2021 р.	
9.	Оформлення ПЗ	18.11.2021 р.	
10.	Попередній захист роботи	04.12.2021 р	

Дата видачі завдання
«___»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«___»_____20 р.
Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Фесечко Д.В. Дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації веб-сайту компанії засобами фреймворку AngularJS.

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. В повній мірі описані всі компоненти розробленого програмного забезпечення.

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

Fesechko D.V. Research and software implementation of the company's website using the AngularJS framework. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

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

		Позначка	Найменування			
			Текстові документи			
1	А4	ВКРМ-123. 21.0024.00.00.ПЗ	Пояснювальна записка	96	—	
2	А4	ВКРМ-123. 21.0024.00.00.ТЗ	Технічне завдання	6	—	Додаток А
3	А4	ВКРМ -123. 21.0024.00.00.РП	Робоча програма	28	—	Додаток Б
			Графічні документи			
4	А4	ВКРМ -123. 21.0024.00.00.НН	Наукова новизна	1	—	
5	А4	ВКРМ -123. 21.0024.00.00.Е1	Структурна схема	1	—	
6	А4	ВКРМ -123. 21.0024.00.00.Е2	Функціональна схема	1	—	
7	А4	ВКРМ -123.21.0024.00.00.Д1	Діаграма процесів	1	—	
8	А4	ВКРМ -123.21.0024.00.00.Д2	Блок-схема алгоритму роботи основної програми	1	—	
9	А4	ВКРМ -123.21.0024.00.01.Д2	Блок-схема алгоритму роботи підпрограми	1	—	
10	А4	ВКРМ -123.21.0024.00.00.ДЗ	Техніко-економічні показники	1	—	

					ВКРМ-123.21.0024.ВП			
Вим	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розробив		*Фесечко Д.В.*						
Перевірив		*Босько В.В.*			*Дослідження та програмна реалізація*	*Літ.*	*Аркуш*	*Аркушів*
						М		*1*

Н. контр.	*Гермак В.С.*			*веб-сайту компанії засобами фреймворку AngularJS*	*ЦНТУ КІ-20м*
Затв.	*Смірнов О.А.*				

ЗМІСТ

					ВКРМ-123.21.0024.00.00.ПЗ	
Вим.	Арк.	№ докум.	Підпис	Дат.		
Розроб.		Фесечко Л.В.			*Дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS*	Літ. М / Арк. 1 / Аркушів 96
Перевір.						
Н. Контр.		Гермак В.С.				ЦНТУ КІ-20м
Затверд.		Смірнов О.А.				

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. Вже пройшло 11 років з моменту виходу AngularJS. Це була новаторська технологія у світі веб-розробки, випущена компанією Google у 2010 році. Вона впровадила нові стандарти у розробку веб-сайтів, та випустила ряд потужних інноваційних технологій. Ця платформа стала кроком уперед у створенні прогресивних веб-додатків. Але з часом команда Google, побачила недоліки в AngularJS, які неможливо було вирішити шляхом еволюції технологій та оновлень, і використала отриманий досвід, щоб переробити фреймворк з нуля.

Це стало переломним моментом у розвитку фреймворку. Враховуючи всі недоліки, команда Google презентувала абсолютно новий фреймворк у 2016 році, назвав його Angular 2, який увібрав в себе всі найкращі досягнення AngularJS. Тому всі версії 1.xx називаються AngularJS, а 2 та пізніші версії — Angular.

Різниця Angular і AngularJS не стосується назви або просто нових версій фреймворку. Технології відрізняються за своєю суттю. Перш за все, AngularJS заснований на JavaScript, тоді як Angular написаний на TypeScript. По-друге, також змінилася базова архітектура та підходи до прив'язки даних. Нарешті, Angular покращив продуктивність, та сумісність браузерів і впровадив підтримку розробки мобільних додатків у кількох наступних випусках.

Ключовим моментом стало оголошення Google про остаточну версію AngularJS версії 1.7 без намірів подальших випусків. Так, 1 липня 2018 року AngularJS вступила у трирічний довгостроковий період підтримки, який пізніше був продовжений на шість місяців через пандемію.

Отже, головним питанням є те, чого очікувати від припинення підтримки AngularJS після 31 грудня 2021 року. За останні роки AngularJS не отримав багато нових впроваджень, але більшість помилок були виявлені та виправлені. Проекти написанні на ньому досі працюють, тому не має необхідності

переходити на нові технології, якщо цього не вимагають поточні чи майбутні вимоги.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для розробки веб-сайту засобами фреймворку AngularJS.

– Дослідження системи та бібліотек для розробки веб-сайту засобами фреймворку AngularJS.

– Визначення можливостей та функціоналу AngularJS відповідно до сучасних вимог у веб-розробці.

– Програмна реалізація веб-сайту засобами фреймворку AngularJS.

– Порівняння фреймворку AngularJS з Angular.

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

Предметом дослідження є методи з розробки веб-сайту засобами фреймворку AngularJS.

Методи дослідження базуються на методах теорії кодування, методах побудови архітектури програмного забезпечення, методах розробки програмного забезпечення.

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

– Розроблено та порівняно веб-сайти за допомогою фреймворків AngularJS та Angular, зроблено висновки про їх переваги та недоліки.

– AngularJS значно скорочує час розробки додатків, але його підтримка завершується в грудні 2021 року.

– NodeJS активно використовується на сервері, при розробці односторінкових додатків.

– MongoDB все частіше використовується для зберігання даних користувачів, замість класичних реляційних баз даних.

Практична цінність отриманих результатів полягає в тому, що розроблені веб-сайти відображають можливості фреймворків, швидкість розробки веб-сайтів з їх допомогою, та їх функціонал.

Достовірність наукових результатів підтверджена теоретичними викладеннями та отриманими даними під час тестування розроблених систем.

Таким чином, виходячи з вищеперерахованого, дослідження та програмної реалізації веб-сайту засобами фреймворку AngularJS, можна зробити висновок що на сьогодні існують нові більш сучасні та розвинуті фреймворки для створення веб-сайтів, але для розроблених веб-сайтів немає необхідності переходити на нові технології, якщо цього не вимагають поточні або майбутні вимоги.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1. Призначення системи

Довгий час на просторах веб-технологій використовувався тільки Javascript, HTML та CSS для створення веб-сайті. З часом почали розповсюджуватися бібліотеки, такі як JQuery для спрощення розробки, але розміри проектів весь час збільшувалися і потрібна була чітка архітектура, щоб сайт можна було підтримувати та розвивати.

В цей час почали розповсюджуватися нові фреймворки які реалізують відомі патерни MVC (MVVM, MVP і так далі). Фреймворки почали впроваджувати структуру у розробку та альтернативні погляди на вже існуючі речі. Все більше стали підійматися питання застосування патернів, що добре зарекомендували себе у світі програмування, а JavaScript розробники почали розділяти логіку додатку, та подання контенту.

Фреймворки надають багато переваг в порівнянні з винаходом власних методів розробки. Стало набагато легше читати, підтримувати та тестувати код. Новим розробникам стало простіше вивчати новий код. Одне з найбільш істотних переваг, позбавлення від рутинного коду, який тягнеться від проекту до проекту. Фреймворк надає розробникам каркас майбутньої програми для розв'язання задач, що зустрічаються в більшості проектів. Наприклад, програмісту не потрібно думати, як прийняти дані від клієнта і передати їх на сервер, оскільки все необхідне реалізовано авторами фреймворку. Замість цього розробнику пропонується зосередитися на функціоналі власного додатка.

Іншим важливим плюсом всіх фреймворків є стандартизація кодування. Якщо розробник вирішується застосовувати готовий каркас у своєму проекті, то він повинен бути готовим слідувати його правилам. Це означає, що йому потрібно ознайомитися з правилами, і згодом код може доопрацьовуватися

іншими розробниками. Новому розробнику буде простіше розібратися в добре документованій структурі (особливо якщо він вже мав досвід роботи з цим рішенням), ніж зрозуміти кимось розроблений підхід проектування.

1.2. Область застосування

Головна область застосування AngularJS створення односторінкових додатків. Його головними перевагами на старті було те що додатки на AngularJS були більш структурованими, що полегшувало процес тестування та розробки. Під час виходу його сильними сторонами була активна підтримка Google, двостороннє зв'язування даних, взаємодія з іншими бібліотеками, та декларативне програмування, яке полегшувало підтримку та читання коду.

Оскільки сьогодні веб-сайти стають все більш схожими на десктопні додатки, які мають більше можливостей для взаємодії з користувачем, а не тільки являються статичними сторінками, як це було раніше. Розробники веб-сайтів мають можливість розробити більш зручний інтерфейс для користувачів, впровадити нові унікальні можливості, яких не має на інших сайтах. Тому для побудови клієнтської частину веб-сайту потрібні більш сучасні технології. На сьогодні є дві технології, які використовуються для побудови веб-сайтів: MPA (Multi Page Application) - багатосторінковий додаток, та SPA (Single Page Application) - односторінковий додаток.

Багатосторінковий додаток (MPA) можна назвати більш традиційним способом розробки веб-сайтів. При розробці створюється декілька сторінок, які використовують шаблони, для зберігання статичних даних (заголовків, зображень) та посилань для переходу на інші сторінки зі схожим вмістом. Коли відбувається перехід на нову сторінку, браузер виконує запит на сервер, для отримання даних і відображення їх користувачу. Але під час цього процесу сервер відправляє навіть та дані, які були на попередній сторінці. В результаті, ресурси витрачаються на отримання тієї самої інформації, що негативно впливає

на швидкість роботи сервера та браузера. Для побудови багатосторінкових сайтів зазвичай використовуються старі технології, наприклад CSS для встановлення стилів та HTML для розмітки сторінки, які використовувались ще при створенні перших веб-сайтів, та активно використовуються сьогодні.

Але якщо розробка веб-сайту вимагає складний функціонал, наприклад динамічному оновленні даних, тоді все ще можна використовувати традиційні способи розробки, але вони будуть накладати певні обмеження і вимагатимуть використання більш сучасних технологій, таких як JavaScript та AJAX. Завдяки яким веб технології почали активно розвиватися. Сьогодні, завдяки цим технологіям можливо створювати складні анімації, відображати графіки, динамічно відображати оновлену інформацію на сторінки без її перезавантаження. Серед недоліків цієї технологій, варто відзначити складність розробки для адаптації веб-сторінок під мобільні версії сайту, через що час розробки збільшується в рази, а також складність у додавані нових функціональних модифікацій. До переваг можна віднести легкість оптимізації сторінок для пошукових систем, та можливість додавання спеціальних метатегів. Оскільки для розробки MPA додатків потрібна менша кількість технологій, вартість таких додатків значно дешевша.

Односторінкова програма (SPA) більш сучасна технологія, мета якої зробити додаток більш схожим на десктопну програму. Його головна відмінність від MPA, полягає у тому, що при переході на іншу сторінку, коли браузер робить запит на сервер, він отримує тільки необхідну інформацію, а в браузері оновлюється тільки ті дані які були отримані з сервера. В результаті, сервер надсилає тільки нові дані і немає потреби у перезавантаженні всієї сторінки. Це виглядає набагато краще, та зручніше у використанні для користувачів. SPA зараз отримав велике поширення серед великих технологічних компаній, його активно використовує Google у своїх сервісах Gmail, та Facebook у свої соцмережах. Зазвичай для розробки SPA використовується JavaScript або TypeScript.

Хоча невеликі додатки можуть бути р озроблені за допомогою JQuery, для великих додатків він не підходить. Оскільки він не задає структуру для проекту і робить код складним для розуміння. Тому для складних проектів краще використовувати фреймворки: AngularJS, Angular, React, Vue. Вони задають архітектуру проекту, та змушують розробника дотримуватися певної структури. Для цього кожен з них надає певний функціонал для роботи з сервером, впровадження динамічного оновлення даних, підключення стилів. Також за допомогою їх можна створювати повноцінні мобільні додатки. Головним недоліком SPA є погана SEO оптимізація, оскільки пошукові системи не можуть просканувати вміст сторінок. Хоча існує ряд рішень для вирішення цієї проблеми, але вони вимагають більше часу. Також слід звернути увагу, що для мобільних пристроїв з слабкими характеристиками, такі сторінки буде важче завантажувати, а якщо проект дуже великий, потрібно розділити його на окремі JavaScript компоненти, щоб вони швидше завантажувались. Головними перевагами таких додатків є висока швидкість оновлення даних, оскільки браузер оновлює тільки необхідні дані, велика кількість бібліотек та готових рішень у вигляді розроблених компонентів, які є дуже зручними у використанні, можливість створювати мобільні додатки, використовуючи код з клієнтської та серверної частини, а також можливість кешування даних та відображення цих даних без підключення до Інтернету.

У кожному з наведених способів розробки додатків є свої переваги та недоліки відповідно до вимог проекту. Головними перевагами SPA є швидкість роботи додатків, а також можливість розробляти мобільні додатки. Але має складності у роботі з пошуковими системами. Такий архітектурний підхід найкраще підходить для розробки соціальних мереж, або додатків де потрібна велика швидкість при оновленні даних, та не має значення робота пошукових систем. MPA краще використовувати для бізнес-сайтів та інтернет-магазинів, але слід звернути увагу на складність розробки мобільного додатку.

Сьогодні односторінкові додатки стають все біль поширенішими, завдяки

їхнім перевагам у створенні масштабованих та динамічних систем. Але слід звертати увагу на тип проекту, вимоги замовника та користувачів, щоб розробити систему яка буде підтримувати весь необхідний функціонал та бути масштабованою.

Що стосується AngularJS у якості SPA фреймворку, він використовує патерн проектування MVC, який добре зарекомендував себе і тепер став доступний для JavaScript світу. Він лежить в більшості сучасних фреймворків. Звичайно, особливої новизни і революції в цій події немає. Розробник з досвідом, може без жодних фреймворків спроектувати додаток, дотримуючись цього патерну. Однак фреймворк зобов'язує розробника дотримуватися правильної архітектури. AngularJS - це структурна основа для динамічних веб-додатків. Це дозволяє використовувати HTML як мову шаблонів та розширювати синтаксис HTML, щоб чітко та лаконічно виражати компоненти програми. Зв'язування даних AngularJS та ін'єкція залежностей зменшують кількість коду і роблять його більш читабельним.

З одного боку, AngularJS має досить низький поріг входження в порівнянні з багатьма подібними рішеннями. З іншого - документація носить злегка суперечливий характер. Вона добре структурована, є приклади коду, але деякі речі висвітлені вкрай слабо. З ними доведеться розбиратися самостійно, вивчаючи вихідний код або запитуючи коментарі від колег по цеху.

Всі ці можливості фреймворку призвели до того, що його досі активно використовують. Він був одним з перших SPA фреймворків, тому нові фреймворки багато що з нього перейняли і хоча для нових проектів краще використовувати більш сучасні рішення. AngularJS зробив великий внесок у розвиток односторінкових додатків.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1. Огляд існуючих систем

Стандарти веб-розробки постійно зростають разом зі складністю сучасних технологій. Все більше часу потрібно витрачати для створення правильної архітектури проекту, яка допоможе масштабувати додаток у майбутньому, треба слідкувати за безпекою веб-додатків, та швидкістю відповіді на запити користувачів.

Javascript фреймворки є однією з найбільш розповсюджених платформ для створення сучасних односторінкових веб-додатків (SPA), соціальних мереж, комерційних продуктів, нових платформ і багато чого іншого.

Огляд SPA фреймворків:

React - це JavaScript бібліотека з відкритим вихідним кодом, яка була створена для розробки односторінкових додатків, на стороні клієнта. Розроблений React був у 2011 командою Facebook, з 2012 почав використовуватись на сторінці Instagram. Він досі розробляється і підтримується Facebook і Instagram. React дуже корисний в тому що дозволяє розробляти додатки не тільки для браузера, але також для мобільних додатків. Головне завдання React - надати високу швидкість, простоту користування і зручний інтерфейс, працездатність додатків на різних платформах. Але React - це не повноцінний фреймворк, а бібліотека функцій. Щоб використовувати його як фреймворк, потрібно додатково підключати сторонні бібліотеки, хоча вони невеликі, але треба витратити час на їх налаштування.

Серед переваг React слід віднести швидкість його роботи, легковажність, здатність працювати у різних середовищах і велике ком'юніті. Слабкою стороною є той факт, що для повноцінної роботи потрібні сторонні Javascript бібліотеки, що ускладнює процес розробки. Другий мінус бібліотеки - це відсутність звичних стандартів при написанні коду на HTML і CSS, які є,

наприклад, у Angular і Vue.js. Хоча React є бібліотекою, він має різноманітний функціонал. В останній час Facebook оновлює React настільки часто, що велика кількість його функціоналу вже застаріли. Найбільш підходящими варіантами для вибору цієї технології є її швидкість розробки невеликих додатків, створення SPA для кросплатформених рішень, а також розширення вже існуючого функціоналу. При розробці на React слід звернути увагу на фреймворк NextJS, який використовує технологію SSR (Server Side Rendering) для рендерингу додатку на стороні сервера і покращення SEO оптимізації.

Vue — був створений для розробки інтерфейсу для користувача. Перевага Vue в тому, що за допомогою нього можна поступово впроваджувати новий функціонал, на відміну від Angular або React. Це означає, що впроваджувати цей фреймворк можна поетапно починаючи з певних сторінок, що значно спрощує розробку. Розробникам Vue.js є Іван Йу. Vue широко використовується серед китайських компаній, наприклад, Alibaba, Baidu, Xiaomi і ін. Недавно система управління репозиторіями GitLab теж перейшла на Vue.js.

Vue в першу чергу вирішує завдання рівня представлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. Vue увібрав в себе кращі сторони Angular і React: швидкість, легковажність, можливість підтримки таких технологій, як TypeScript і JSX. Але, при цьому, Vue залишився вірним стандартам написання коду на HTML і CSS, що полегшує процес розробки і підтримки проекту. Таким чином, для будь-яких розробників, які знають основи клієнтських технологій не буде проблемою розробити або взяти на підтримку проекти на Vue. З слабких сторін можна виділити поки що не дуже велике ком'юніті, тому що фреймворк не підтримує великими ко рпораціями, але популярність фреймворку з кожним роком зростає.

Angular - це багатоплатформний фреймворк, який дотримується шаблону проектування MVC і заохочує слабкий зв'язок між представленням, даними і логікою компонентів (складових частин додатка). Angular переносить на сторону клієнта частину серверних служб, що допомагає зменшити навантаження на

сервер і веб-додаток стає легше. Він активно використовується у великих компаніях, де використовується велика кількість коду.

Завдяки TypeScript код проектів стає чистий, зручний для розуміння розробників і містить менше помилок. Сильні сторони Angular: відмінна документація, підтримка компанією Google, величезний набір інструментів для розробки (Material UI, CLI і т.д.) Однією зі слабких сторін є високий пор ог входження в проекти для розробників, тому що, наприклад, потрібно знати TypeScript. Отже, це ускладнює розробку проектів, особливо якщо проект передається від однієї команди до іншої. Друга проблема Angular - дуже частий реліз нових версій. Цей факт також говорить про те, що проекти на Angular складніше підтримувати.

AngularJS - це програма з відкритим кодом JavaScript від Google для розробки прикладних програм. Наступні його версії більш відомі як Angular 2 і т.д. Будучи частиною екосистеми JavaScript, AngularJS негайно налагодив зв'язок із веб-розробниками, це був перший фреймворк, який дозволяв розробляти інтерактивні веб-сайти. Це призвело до розробки односторінкових додатків, які чудово реагували і мали чудовий користувальницький інтерфейс.

Google випустив AngularJS або Angular 1 у 2010 році. Це отримало негайну популярність та підтримку, оскільки тепер була можливість перетворювати статичні HTML-сторінки на інтерактивні, використовуючи AngularJS. Однак незабаром були випущені інші фреймворки, які почали висвітлювати недоліки AngularJS. Через жорстку конкуренцію з боку ReactJS та VueJS, Google пішов на повне перезавантаження з Angular 1 на Angular 2, використовуючи TypeScript як нову мову. Рішення перейти з JavaScript на TypeScript було прийнято для того, щоб уникнути помилок JavaScript та запровадити невелику кількість статичних типів, особливості, яку вимагали багато існуючих веб-розробників.

Було багато чого змінено, починаючи з Angular 2. Компанія змінила парадигму, яку використовував AngularJS, була змінена не тільки мова, а й

архітектура, та підхід до прив'язки даних. Однак і AngularJS, і Angular продовжують використовувати програмісти та веб-розробники відповідно до їхніх вимог. Основні фактори, які відрізняються в Angular та AngularJS:

- AngularJS використовує архітектуру MVC або Model View Controller. Розробник розміщує бізнес-логіку в моделі, представленні і у контролері, а AngularJS виконує всю необхідну обробку для отримання результату. Основні блоки побудовані за допомогою компонентів та директив в AngularJS. Компоненти - це не що інше, як директиви із заздалегідь заданим шаблоном. Вони надають сучасну структуру додаткам, що полегшує створення та підтримку великих програм.

- AngularJS з самого початку використовував JavaScript, але для Angular 2 та пізніших версій почали використовувати TypeScript. TypeScript - це розширення JavaScript, яке забезпечує статичні типи в процесі розробки. Статичні типи дозволяють уникнути багатьох помилок, при розробці програмного забезпечення, а також покращують продуктивність, на відміну від динамічних типів, які ускладнюють використання AngularJS для великих та складних додатків.

- Ін'єкції залежностей (DI). Як AngularJS, так і Angular використовують залежність від ін'єкцій, але спосіб їх здійснення дуже відрізняється. У AngularJS DI вводиться в різні функції зв'язку, функції контролера та визначення директив. З іншого боку, Angular реалізує ієрархічну систему введення залежностей, використовуючи декларації, функції конструктора та постачальників.

- У Angular 2 існує власний інтерфейс командного рядка або CLI. Його використовують для генерації компонентів, служб тощо і навіть для швидкого та ефективного завершення проектів. Він дозволяє легко генерувати різні версії одного і того ж проекту для різних платформ з динамічною перевіркою типів. Angular CLI спрощує розробку додатків, та їх компіляцію. У AngularJS немає власного CLI.

- Використовуючи технологію прив'язки даних, Angular є більш інтуїтивно зрозумілим, ніж AngularJS. Розробник AngularJS повинен пам’ятати правильну директиву ng для прив’язки властивості чи події. У випадку Angular мова використовує () для прив'язки подій та [] для прив'язки властивостей.

- Angular набагато швидший, ніж AngularJS. Двостороння прив'язка, яка зробила оригінальний Angular JS популярним серед веб-розробників, виявила його складності, оскільки з його допомогою важко розробляти складні програми. Щоб забезпечити та здійснити двостороннє прив'язування, AngularJS продовжує перевіряти кожну масштабовану змінну за її попереднім значенням, використовуючи цикл. Angular має потокову архітектуру, де виявлення змін здійснюється за допомогою однонаправленого потоку даних, що робить програми набагато швидшими.

Кожен з цих фреймворків має свої переваги та недоліки, на які слід звертати уваги при розробці web-додатків, до переваг Angular можна віднести:

- Angular в декілька разів швидший, ніж AngularJS завдяки набагато кращому алгоритму прив’язки даних та архітектурі на основі компонентів.

- Компоненти програми Angular досить незалежні та самодостатні, що робить їх багаторазовими та зручними для тестування. Незалежні компоненти легше замінювати, підтримувати та масштабувати.

- Angular має вбудовані розширення для серверної візуалізації додатків. Це дозволяє розробникам синхронізувати клієнтські та серверні сторони вмісту, що є величезним плюсом для SEO.

- Angular підтримує ліниве завантаження, що робить додатки швидшими, оскільки завантажуються лише ті компоненти, які потрібні.

- Підхід типу Angular сприяє більш чистому коду, кращій навігації у високоякісному продукту.

Також Angular має ряд недоліків:

- Крива навчання Angular складніша, тому що потрібно знати як використовувати статичні типи даних.

- Оскільки Angular 2 був повноцінним перезаписом AngularJS, застарілі проекти, розроблені за допомогою AngularJS, потрібно повністю переписувати.

- Angular також іноді називають багатослівною мовою, тому що компонентами керують в дуже складний спосіб.

- Документація інтерфейсу командного рядка не є повною.

До переваг AngularJS належать:

- Використовуючи JavaScript, набагато простіше та швидше розробляти додатки на AngularJS.

- Двостороння прив'язка даних AngularJS робить швидшою розробку та простішу прив'язку даних без втручання розробників.

- AngularJS підтримує швидше кодування та складання прототипів, надзвичайно скорочуючи час розробки.

- Архітектура MVC та MVVM AngularJS відокремлює дані від дизайну, що полегшує розробку та підтримку складних веб-додатків.

- Чисте та організоване кодування робить коди AngularJS дуже багаторазовими.

AngularJS також має декілька недоліків:

- Якщо система, яка намагається запустити програму AngularJS, має відключений JavaScript, програма не буде працювати на ній.

- Щоб використовувати AngularJS, розробники повинні бути знайомий з архітектурою MVC.

Якщо переглянути на перелік переваг та недоліків Angular та AngularJS, можна побачити що Angular використовується у великих проектах, які вимагають великих навантажень. Якщо додаток є досить простий AngularJS може зробити розробку швидшою і простішою.

Підсумовуючи, можна сказати що якщо проект невеликий і вимагає швидку розробку, варто звернути увагу на Vue, AngularJS та React. Оскільки вони легші в навчанні, а також дозволяють швидко впроваджувати нові модифікації без сильного втручання в архітектуру додатку. Якщо планується

великий проект з довгостроковою підтримкою, краще обрати Angular, через його велику продуктивність, чітку структуру, а також великий досвід на ринку та кращу документацію.

2.2 Обгрунтування вибору методів розробки

AngularJS в першу чергу створений для спрощення складних процесів при створенні та керуванні JS додатків. В його основі лежить структура MVC, що робить цей фреймворк особливо корисним для створення односторінкових сайтів. Бібліотека заснована на звичайному JS і HTML, тому AngularJS автоматично піклується про маніпуляції з DOM і AJAX запити, які в іншому випадку розробникам довелося б писати самим. AngularJS можна швидко додати на будь-яку HTML сторінку за допомогою простого тега, оскільки цей інструмент надає модульні будівельні блоки коду JS, які можна поєднувати і тестувати.

AngularJS надає дуже хороший контроль над даними на стороні клієнта. Не потрібно думати про те, щоб вибрати елементи з DOM та заповнити їх значеннями. Завдяки доступній прив'язці даних є можливість оновити дані в частині JavaScript і побачити зміни в частині HTML. Це справедливо і для зворотних дій. Як тільки користувач щось змінить в частині HTML, він одразу отримає нові значення в частині JavaScript. Також фреймворк має потужну інжекторну залежність. Існують заздалегідь визначені класи для виконання запитів AJAX та керування маршрутами.

Є ряд причин через які AngularJS виділяється серед конкурентів:

- Спрощена двостороння прив'язка даних. AngularJS дозволяє прив'язувати дані до HTML за допомогою виразів, а директиви AngularJS дозволяють розробникам розширювати функціонал HTML і створювати нові конструкції. Маніпуляції з DOM і код прив'язки даних обгорнуті в прості елементи, які можна швидко і просто вставити в HTML шаблони.

- AngularJS спроектований універсальним фреймворком, тому з його допомогою можна створити веб-додаток майже будь-якого типу. Особливо він корисний при створенні односторінкових веб-додатків.

- AngularJS входить в пакет ПЗ MEAN, який також включає MongoDB, Express.js і Node.js. Тому клієнтська та серверна частини проекту виконується за допомогою JS. У якості альтернативи для сервера можна використовувати Ruby on Rails. З AngularJS також добре стикується ASP.NET і C #.

- AngularJS побудований по техніці functionality-first, тому фреймворк найбільше підходить для розробки зверху вниз. Модульна концепція AngularJS дозволяє спростити поділ роботи на різні команди в великих проектах. У таких командах в пріоритеті мінімальна кількість коду, тому додатки AngularJS, як правило, компактні і легкі в редагуванні.

Проте є моменти, які необхідно знати при виборі AngularJS для свого проекту, вони допоможуть правильно спроектувати логіку додатку і розв'язувати поставлені задачі:

- Перше і найголовніше, AngularJS нав'язує свою структуру розробникам. AngularJS спроектований максимально дружньо, тому його інструменти інтуїтивно зрозумілі. Однак розробникам, яким потрібні гнучкі рішення, доведеться шукати обхідні шляхи.

- Для деяких проектів AngularJS надто зайвий. У таких випадках більше підійдуть легкі фреймворки для створення статичних сайтів. AngularJS також не пристосований обробляти маніпуляції з DOM з великою кількістю даних, тому, що покладається на перевірки для управління змінними DOM (будь-яка зміна змінних тягне оновлення DOM). Для більшості сайтів це не буде проблемою, але для GUI редакторів або відеоігор це може викликати проблеми.

- AngularJS також не підтримує високонавантажені галереї фото. Подібні проблеми з продуктивністю можна вирішити через форми з високим рівнем користувальницької взаємодії.

Існує ряд порад, які варто дотримуватися, щоб поліпшити швидкість додатків на AngularJS:

- AngularJS використовує так званий “дайджест цикл”, який займається оновленням DOM. Дайджест цикл додатків AngularJS - хороший індикатор його продуктивності. Його можна представити, як цикл, який перевіряє наявність змін в змінних, та займається їх оновленням. Чим коротше дайджест цикл, тим швидше додаток працює.

- Швидкість додатку залежить від кількості відстежень. Кожен раз при введенні прив'язки даних створюється все більше змінних $$watchers і $scopes, що подовжують дайджест цикл. Занадто багато $$watchers можуть викликати затримки. Тому треба використовувати їх по мінімуму.

- По можливості треба використовувати одноразові прив'язки в старих версіях.

- Доступ до DOM дорого обходитися, тому треба зберігати маленький розмір дерев DOM. Не змінювати DOM без реальної на те необхідності і не встановлювати вбудовані стилі.

- Треба стежити за областями видимості. Не треба давати змінним занадто велику область видимості, щоб збирач сміття JS міг вивільнити пам'ять.

Також варто зауважити, що AngularJS відмінно працює з Node.js. Як асинхронне подієве JavaScript–оточення, Node.js спроектований для побудови масштабованих мережевих додатків. Де для кожного з’єднання викликається функція зворотного виклику, проте коли з’єднань немає Node.js засинає.

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

Якщо говорити про нові версії Angular, вони є більш сучасними і

використовуються частіше. Це спричинено тим, що вони більш стабільні, мають більшу кількість модулів, має систему CLI, зручні в тестуванні і мають кращу структуру коду завдяки використанню TypeScript. Також варто звернути увагу на бібліотеки які використовує Angular, вони значно полегшують розробку. Серед основних переваг Angular у порівнянні з іншими фреймворками:

- Angular використовує бібліотеку RxJS, яка використовує концепції реактивного програмування і observables, для заміни функцій зворотного виклику в написанні асинхронного коду.

- Angular використовує Dependency Injection — патерн проектування, який оголошує сервіси в конструкторі компонента, що допомагає розділити структуру коду.

- Angular CLI — дуже зручний інструмент, для створення та підтримки додатку, який на початку проекту генерує все необхідне.

Серед недоліків Angular можна віднести велику кількість кода, а також складний поріг входу в фреймворк.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на магістерську роботу, реалізації підлягає програмне забезпечення, яке продемонструє роботу сучасних веб фреймворків.

В процесі розробки магістерської роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів

програми та підпрограми;

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

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи з охорони праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Angular та AngularJS це веб-фреймворки, які вимагають від розробника реалізовувати MVC паттерн у своїй программі. Патерн MVC (Model-View-Controller) це один з найрозповсюдженіших шаблонів проектування, який ділить логіку програми на окремі компоненти, що якісно впливає на розробку продукту, та її подальшу підтримку. Особливо зручно коли одночасно з клієнтською частиною, цей шаблон використовується на серверній частині, наприклад він є основним в багатьох серверних фреймворків: Spring MVC, NestJS, SailsJS, ASP.NET MVC Framework та інших.

При використанні шаблону MVC, програма ділиться на окремі компонента: модель, представлення і контролер. Отже у разі внесення змін або додавання нової модифікації у компонент, він повинен мінімально впливати на інші компоненти.

Головні завдання компонентів:

- Model (Модель) — відповідає за зберігання даних користувача, і надання необхідного інтерфейсу до цих данних.

- View (Представлення) — відповідає за данні, які будуть надаватися користувачу, а також за форму у якій вони будуть представленні.

- Controller (Контролер) - містить бізнес логіку додатка, реагуючи на події в браузері, та повідомляючи модель про оновленні данні.

За допомогою такої внутрішньої структури, система ділиться на самостійні частини і розподіляє логіку роботи програми між компонентами, для того щоб введення даних, обробка даних, та вивід цих даних користувачу незалежали один від одного.

Якщо простежити як працює шаблон MVC більш детально, то Controller відстежує дії користувача, наприклад ведення даних, або натискання клавіши.

Після чого викликається запит або функція, яка звертається до компонента Model, щоб отримати необхідні данні, наприклад з бази данних і за допомогою компонента View, ці данні надаються користувачу у потрібній йому формі. Така архітектура робить компоненти незалежними і коли користувач використає контролер, це прийзведе до змін в базі даних, та оновлення цих даних користувачу.

Існують також шаблони, схожі на архітектуру MVC, наприклад MVP (Model-View-Presenter) та архітектура N–Tier. Різниця між MVC та MVP, в тому що дані в MVC, передаються з Model в View, а в MVP проходять через View. Перевагою MVC є чітке розділення логкіи представлення даних і логіки програми.

Найчастіше у реалізації MVC для взаємодії між представленням та моделью, використовують спеціальні протоколи взаємодії, використовуючи апарат події (підписка/оповіщення). Коли внутрішні данні в моделі змінюються, відсилається сигнал всім залежним шаблонам, що дані оновились. Для цього зазвичай використовуються Observer, Strategy або Factory Method, щоб обрати правильний контролер, та взаємодіяти з моделю.

Існують різні модифікації моделі MVC, найрозповсюджені серед них: пасивна та активна модель. Пасивна модель — це коли контролер стежить за змінами в моделі і у разі оновлення даних, відповідає за їх відображення користувачу. Активеа модель — це коли сама модель повідомляє шаблон, що її дані оновилися за допомогою системи повідомлень. Така система є більш незалежна між моделю, контролерам і шаблонами.

За допопомогою сучасних технології, таких як SPA та PWA, розробник може зробити взаємодію з користувачем більш інформативною та зручною. Від цього залежить кількість відвідувачів а також популярність сайту. У цьому можуть допомогти такі фреймворки як Angular, AngularJS, React та інші.

AngularJS вимагає від розробника реалізації орігінального MVC у своїх веб-додатках. Для цього він надає всі необхідні сервіси та технологію

впровадження залежностей. Це допомогає Angular-додаткам мати чітку структуру, мати якісну підтримку, а також в легко тестувати додаток на помилки.

AngularJS має великий технологічний функціонал для створення динамічних сторінок, у цьому йому допомагають: сервіси, модулі, фільтри, директиви, область застосування.

Особливу увагу слід приділити директивам в AngularJS. Оскільки вони використовуються для надання HTML нових можливостей. У процесі компіляції, директиви які знаходяться в HTML модифікують DOM, або дода ють йому нову поведінку. Також є можливість розробки своїх директив, як правило їх прийнято називати за допомогою lowerCamelCase стилю.

В AngularJS також існує ієрархічна структура — scope, представленна у вигляді об'єктів. Вони використовуються для передачи даних між контролером і представленням. Вони схожі на DOM але вони наслідують властивості батьківських scopes. Головним є *$rootScope,* він є батьківським по відношенню до всіх інших об'єктів $scope. За допомогою диретиви *$watch* можна встановити спостереження за виразами в рамках *scope,* в процесі виконання фази зв'язування шаблону. Директива *$Watch* слідкує за даними, та оновлює їх за необхідністю.

Сервіси — призначенні для виконання окремої задачі. Наприклад $http сервіс призначений для надсилання HTTP запитів і є обгорткою над стандартним XMLHttpRequest. Щоб використовувати сервіс необхідно підключити його до контролера, директиви, сервіса і т.д. Для створення сервісу необхідно викоримтовувати фабрику, яка реалізується за допомогою метода factory. В AngularJS сервіси реалізуються за допомогою паттерну singelton, щоб для проекту існував тільки один екземпляр, який можна викликати з кожного місця программи.

Фільтри використовуються для перебору данних перед відображенням їх користувачу, а також для фільтрації колекцій, массив та інших типів даних.

За допомогою того що додатки на AngularJS завантажуються у вигляді декларативного опису, це дає змогу зручно підключати стороні модулі, вказувати

нові налаштування, а також зручно тестувати додаток. За обробку подій, які виконують користувачі відповідає директива *ng-click,* яка схожа на виклик onclick. AngularJS замінює стандартний поток JavaScript на власний цикл обробки подій, за допомогою чого виконується зв'язування даних, обробки виключень, відстеження властивостей та інші операції.

Angular також використовує шаблон MVC, але його структура відрізняється. Сам фреймворк розділений на окремі модулі, кожен з яких реалізує певний функціонал. AppModule є корневим модулем, який має набір елементів:

- component — містить web-сторінки з HTML, CSS та основною логікою.
- service — надає необхідні дані для components.
- directive — преобразує DOM елементи.

Компонент — надає інтерфейс та відповідає за логіку певної частини веб-сторінки. Створюється за допомогою декоратора @Component(). Сервіси же надають всі необхідні данні компоненту, це можуть бути запити до сервера, або виконання певних алгоритмів. Директиви схожі на компоненти, але мають HTML-шаблони, створюються за допомогою декоратора `@Directive()`. `Потім Angular компілює код для браузера за допомогою компілятора Angular Ivy.`

Представленний Angular 2.0 в 2016 виправив більшість недоліків архітектури AngularJS, та впровади багато нових технологій, які ускладнили розробку, але зробили її надійнішою. Одним з нововеденням стала підтрика розробки під: мобільні-додатки, нативні та веб-додатки.

Використовуючи всі вищезгадані технології, за допомогою AngularJS та NestJS було реалізовано веб-сайт, використовуючи REST API та сокети. На розроблених сайтах можна порівняти роботу технологій, їх швидкість роботи, представлений функціонал, та зробити висновки що використовувати для додатків середнього розміру. Яка архітектура та структура краще підійде для розробки додатку, та для яких платформ ці технології краще підійдуть і скільки часу знадобиться на їх розробку.

3.2 Розробка структурної схеми

Структурна схема надає інформацію про взаємодію майбутніх частин программи, за якій функціонал вони будуть відповідати, та як будуть взаємодіяти між собою. Не можна сказати що вони є інформ ативними, оскільки на них не можна відстежити як дані передаються та змінюються. Тому в залежності від технічного завдання, структурні схеми розробляють для окремих частин программи.

Для розробки структурної схеми зазвичай використовують метод покрокової деталізації, щоб вказати всі компоненти з яких складається схема, а також набори підпрограм та підключені бібліотеки.

Структурними компонентами програми можуть називатися підсистеми, бази даних, бібліотеки і т.д. А за допомогою зв'язків можна продемонструв ати як вони взаємодіють між собою, обмінюються інформацією, підклчати нові бібліотеки. При розробці структурної схеми програмного забезпечення були розглянуті основні модулі програми, та їх взаємодія.

Дуже важливо під час розробки, особливо якщо проект великий і має складну архітектуру, поділити программу на структкри, щоб відображити логіку програми, показати місця де підключаються бібліотеки, та за що вони відповідають.

В наведеній схемі, взаємодія модулів, відображена в схемі ієрархії. Де до головного модуля підключаються розроблені модулі, однак схема не відображє порядок функціонування системи. Схема доповнюється розшифровкую функцій які виконують підключення модулів.

У структурній схемі (рис. 3.1) обозначені зовнішні специфікації програми, які дають ро зуміння функціонльній частині програми, за що вони відповідають, та як они взаємодіють з вхідними та вихідними даними. Особливо важливо розуміти тип структури даних.

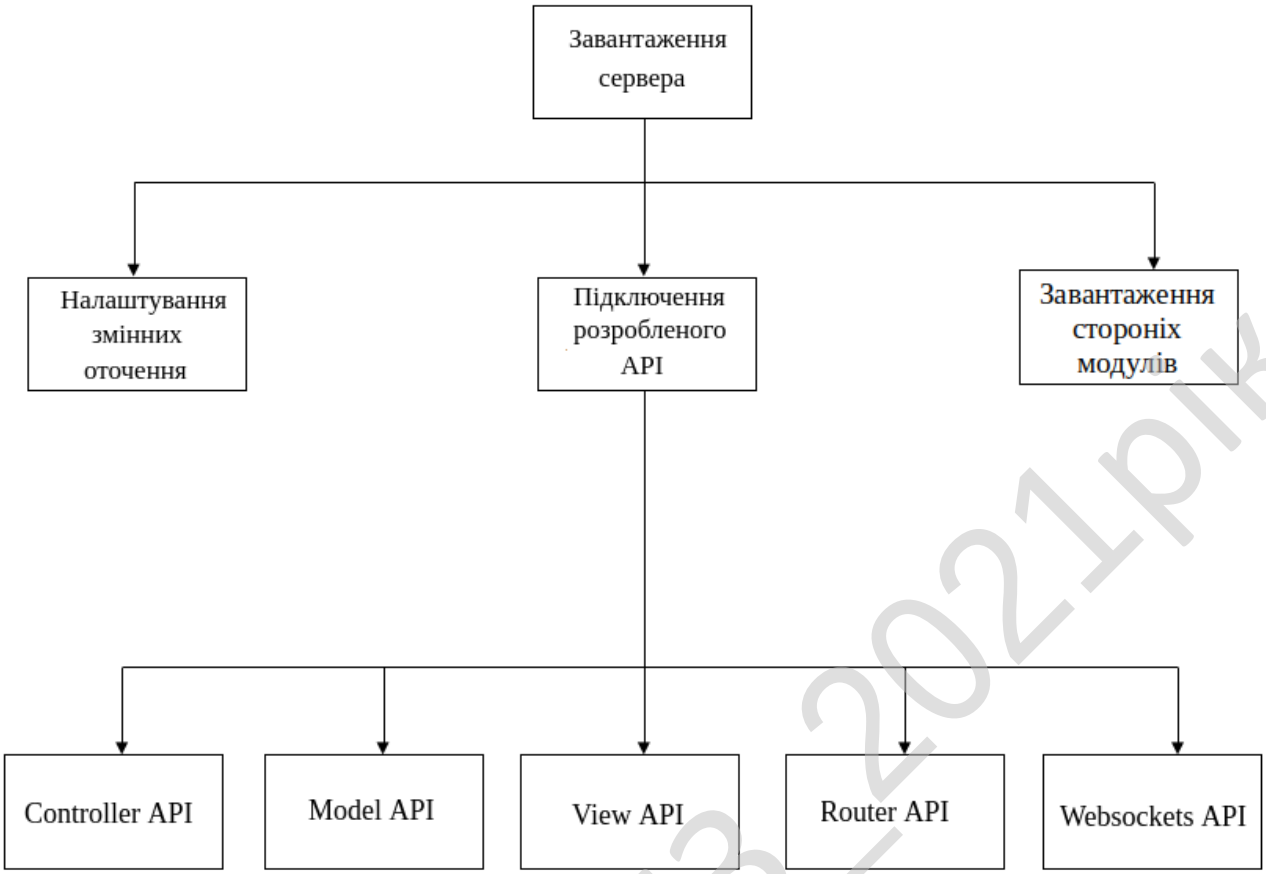

Рисунок 3.1 - Структурна схема

Розподіл програми на окремі модулі відбувався за принципом від загального до конкретного, де окремі модулі виступали у ролі сервісів, які виконвали окремі задачи. Хоча є інші варіанти створення ієрархії, перебираючи декілька варіантів та підтримуючи реалізацію шаблону MVC, такий розподіл надає найзручніший порядок введення частин в експлуатацію, які будуть легко підтримуватися та тестуватися.

Структурна схема відображає головний принцип роботи програми, її основні та другорядні блоки, їх призначення, та їх взаємодію між собою. Це допомогає у розумінні роботи додатку, що полегшує подальшу його підтримку.

3.3 Розробка функціональної схеми

Функціональна схема, або ще називається схема даних, відображає

взаємодію компонентів програми, вказуючи склад даних а також їх призначення. При створенні такої схеми використовуються позначення, прийняті стандартом.

Розроблена функціональноа схема допомагає простежити реалізації існуючих запитів та функцій, відстежити які компоненти та модулі використовуються, як вони взаємодіють між собою, за допомогою вхідних та вихідних даних. Така схема відображає яку роль виконує кожен функціональний елемент у додатку, та з якими компонентами він взамодіє. Для розробки цієї схеми за основу була взята структурна схема.

Переглядаючи функціональну схему можна простежити принцип роботи всього додатку, як між собою взаємодіють клієнтська частина та серверна, а також які бібліотеки та технології використовуються для зберігання даних, передачі.

Цей тип схем зазвичай використовуються щоб візуально продемонструвати роботу алгоритму у спрощенній формі, щоб можна було прослідкувати функції та дії, які виконує алгоритм, які дані використовує і який результат отримує.

Це допомагає оцінити складність алгоритму, зрозуміти принцип його роботи, а також відстежити можливі зміни та налаштування. Побачити як функціональні схеми зв'язані між собою. Для опису компонентів та їх взаємодії використовувався державний стандарт. Де прямокутником відображають окремі функціональні частини, або як вони об'єднуються в функціональні групи, де кожна група має своє позначення та назву.

Є декілька можливостей представити функціональну схему, за допомогою принципової схеми або змішаної стркутури, але в кожному випадку функціональна схема повина відображати призначення пристрою.

На схемі (рисунок 3.2) вказано:

- Умовне позначення у прямокутнику, якщо це функціональна частина

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

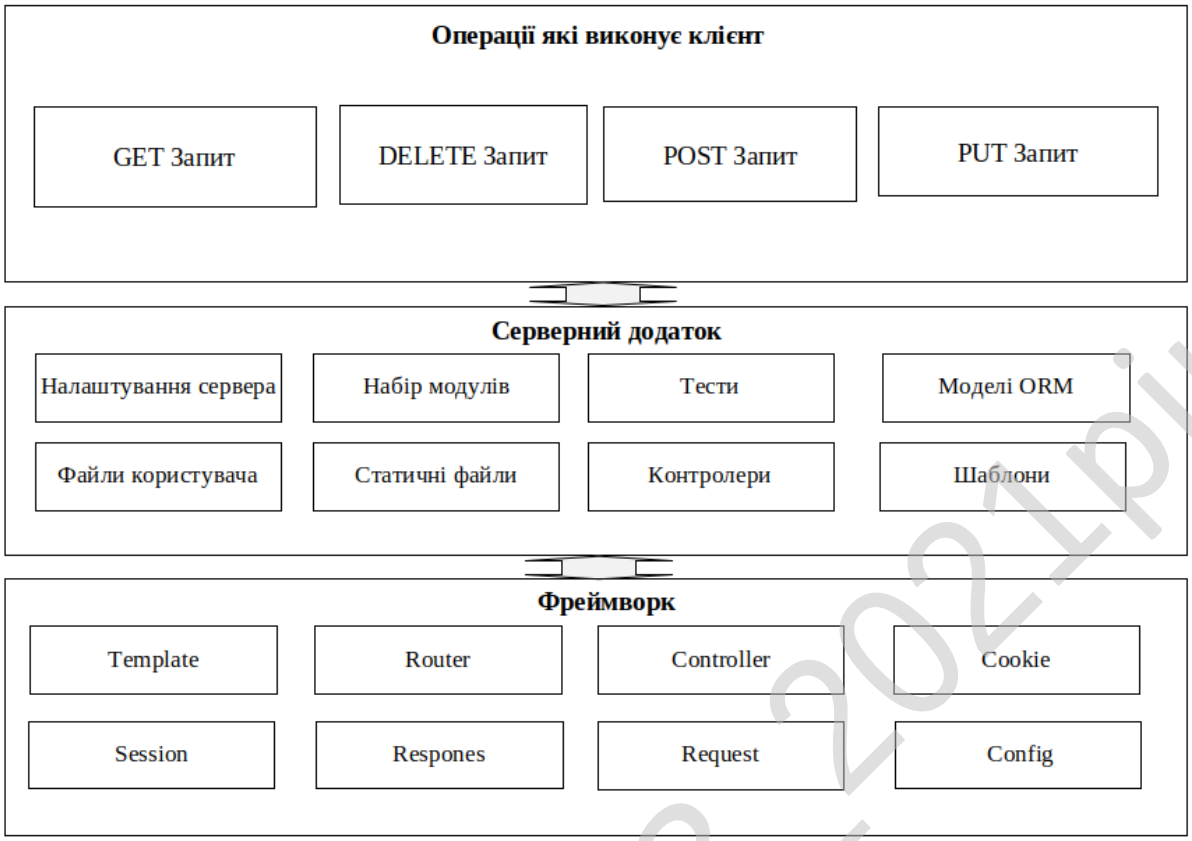

Рисунок 3.2 - Функціональна схема системи

У випадках, коли використовується принципова схема, функціональні частини разом з позиційними позначеннями компонентів мають бути схожими на схемі. У таких випадках перелік елементів не використовується, через використання данних принципової схеми. Коли функціональна схема розробляється без використання принципової, то використовуються загальні правила для відображення функціональних частин.

3.4 Розробка діаграми процесів

При моделюванні поведінки розробленої системи, можна простежити як змінювалася робота додатку, її стан, вхідні та вихідні дані та як все це впливало на розробку додатку. Як працюють алгоритми в додатку та які результати надають в залежності від вхідних даних.

У діаграми процесів, за допомогою графічних об'єктів можна побачити у зручному для сприйнятті форматі, які операції виконуються та як вони між собою пов'язані за допомогою стрілок.

На діаграмі (рисунок 3.3) можна побачити всі етапи роботи програми, від самого початку програми, коли користувач робить запит, як він приходить на сервер, потрапляє в контролер, обробляється, отримує дані з бази даних і повертає їх у вигляді відповіді користувачу. Об'єкти на діаграмі позначають процеси, які виконуються під час роботи програми, стрілки позначають як управління переходить від одного об'єкта до іншого. Є також структурні діограми, вони схожі на діаграми процесів, але відображають статичний харакатер даних, такий опис більше описує взаємозв'язки об'єктів, коли діаграми процесів відображають динаміку роботи програми.

Головною рисою в діаграмі процесів є динаміка, на відмінно від інших, де переважає сценарій. В таких діаграмах сценарій не повинен суперечити послідовності виконання програми та бути безперервним. Проблеми виникають коли об'єкту потрібно бути в декількох місцях одночасно. Таке відбувається коли простежується нечітка логіка в послідовності дій в програмі.

Метою створення діаграм є розуміння послідвності дій процесів в программі, щоб отримати кінцевий результат. Слідкуючи як працюють процеси, між собою, які дані вони отримують, як вони їх обробляють та передають користувачу, можна передбачити як повина працювати программа.

Є різні методи побудови діаграм, які допомагають відстежити логіку та послідовність дії в процесах, але найкраще себе зарекомендував метод погрокового алгоритму побудови діаграм.

З усіх вище перелічених методів, та технічних рішень при проектуванні, та розробки програмного забезпечення, дивлячись на результати, можна зробити висновки, що я обрав оптимальні проектні рішення, які задовольняють головні потреби, які були поставлені переді мною на початку проекту. Оскільки з результатів можна побачити, що программа виконує всі поставленні задачи.

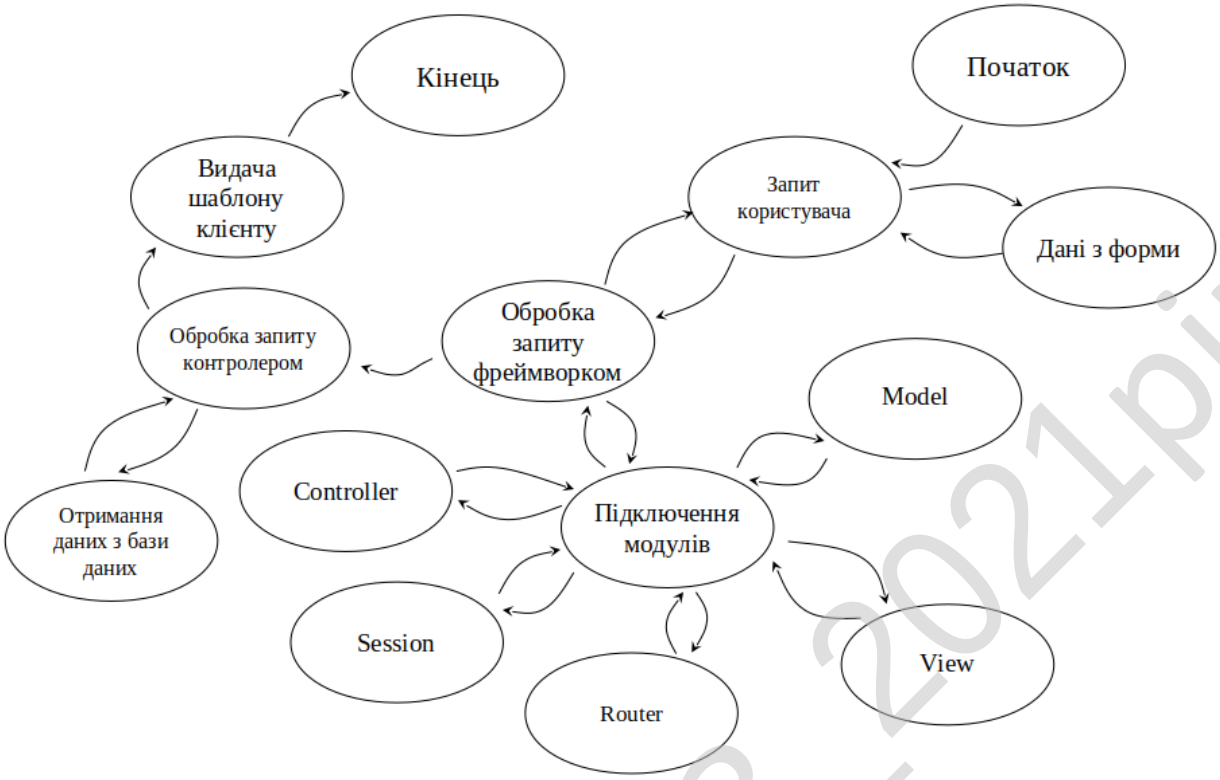

Рисунок 3.3 – Діаграма процесів системи

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 за питу, 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 e mit 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.

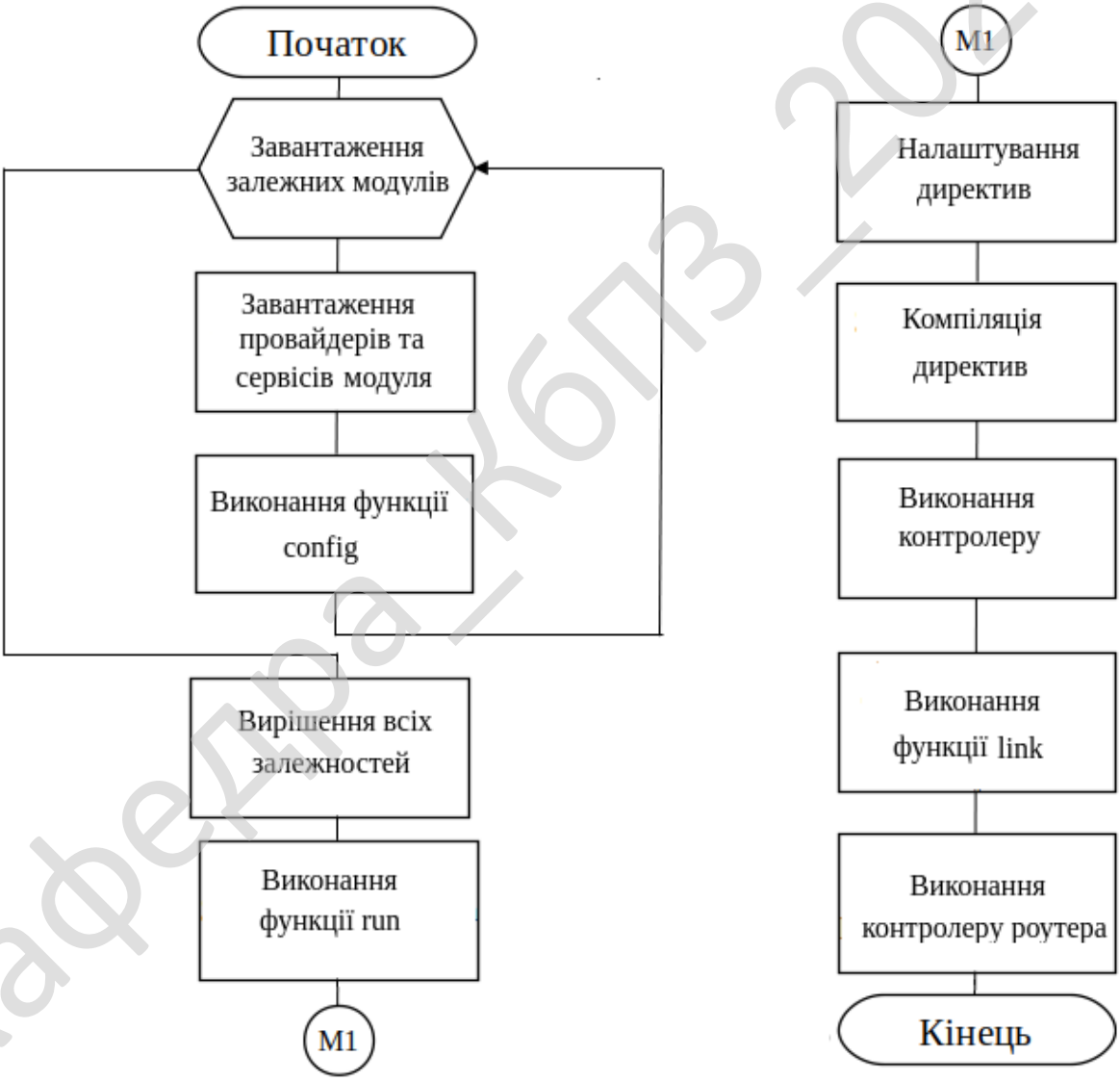

Рисунок 4.1 - Блок-схема головного модулю

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 допоміжн і 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 біл ьшість 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 фун кції, 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 корнівому тегі html і використовується для позначення елемента HTML, який AngularJS буде розцінювати як головний (корневий) 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 створениз 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 обєму 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 відбкваєтья 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 звязує 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 компонена, 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 псевдониму 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 повязані суб'єкти повинні бути в обявленому 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 єправильним підходом, особлио 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 звязування даних. При завантажені сторінки, поля привязуються до моделі даних, вказаної за допомогою з ngModeli та синхронізує ці дані. Наприклад, коли користувач введе у поле пошуку нові дані, вони зрзу відсортируються 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 вибрти в випадаючому меню "алфавіт" то привязка спрацьовує в зворотньому напрямку, спочатку оновляться дані інтерфейсу, а після дані моделі.

AngularJS також має набір вбудованих служб, наприклад $http використовується для надсилання HTTP запитів. Служби керуються задопомогою DI підсистемою в AngularJS. Введення залежностей допомагає задати структуру додатку та послабити звязки між сутностями. В результаі 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 асинхроно та присвоює ці дані контролеру, після чого лані з JSON можна отримати в функціх обратного 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

мнтоди: post, delete, put, head, jsonp за допомогою яких можна реалізуквати повноційний 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 аогументів. 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 оголостьт нові маршрути за допомогою $routeProvider , який є постачальником послуги $route. Ця послуга має гарний функціонал, який дозволяє зєднувати 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 звязані з представленням даних, розробники серверної та клієнтської частини повині рівномірно розподілити логіку.

При кожному запиті користувача, викликається відповідний контролер. Спочатку за допомогою бібліотеки class-validator перевіряються вхідні дані, якщо

на контролері є інтерсептор 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 надіслани 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 автонтифікації.

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

- users - компонент для роботи з даними юзера.

- mail — компонент для відстлання e-mail повідомлень.

У розроблених прикладах за допомогою фреймворків AngularJS та NestJS було створено функціонал для реєстрації, авторизації, а також надавло 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 концепію REST-API, існують також інші способи взаємодії клієнта та сервера, але у даному випадку REST підійшов найкраще.

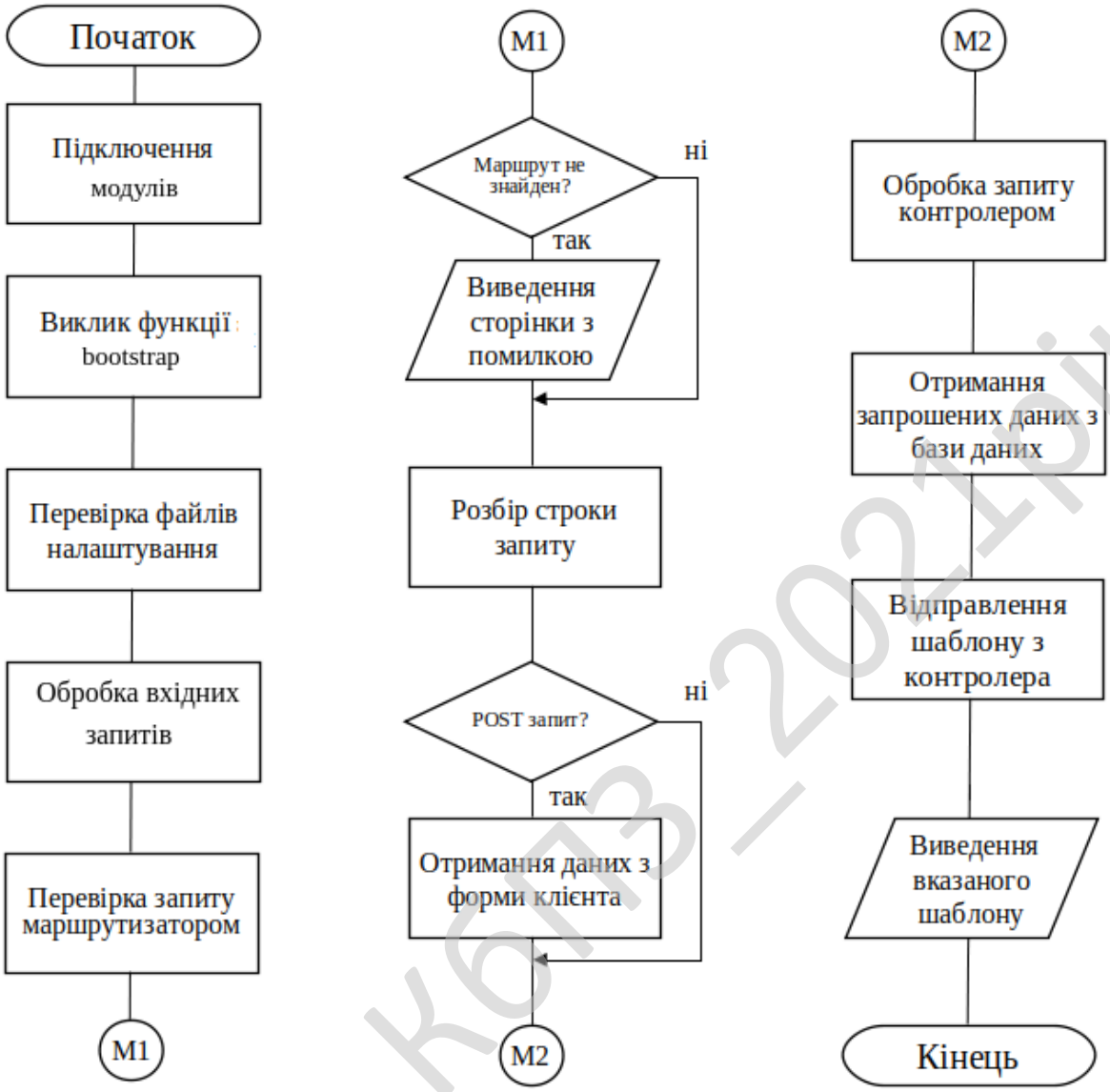

Рисунок 4.2 - Блок-схеми роботи системи

Всі запити виконувалия за допомогою HTTP-методів: GET, POST, PUT і DELETE, які використовувалися для сторення, оновлення, читанная 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 занаходяться 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 зпроси, 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 памяті. 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 нужна мінімальна кількість коду, а це означає що можна витратити більше часу на впровадження необхідної логіки для міграцій. Також він надає набір команд для робрти з міграціями, наприклад migrate_status виводить спсок 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 випа дку 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.

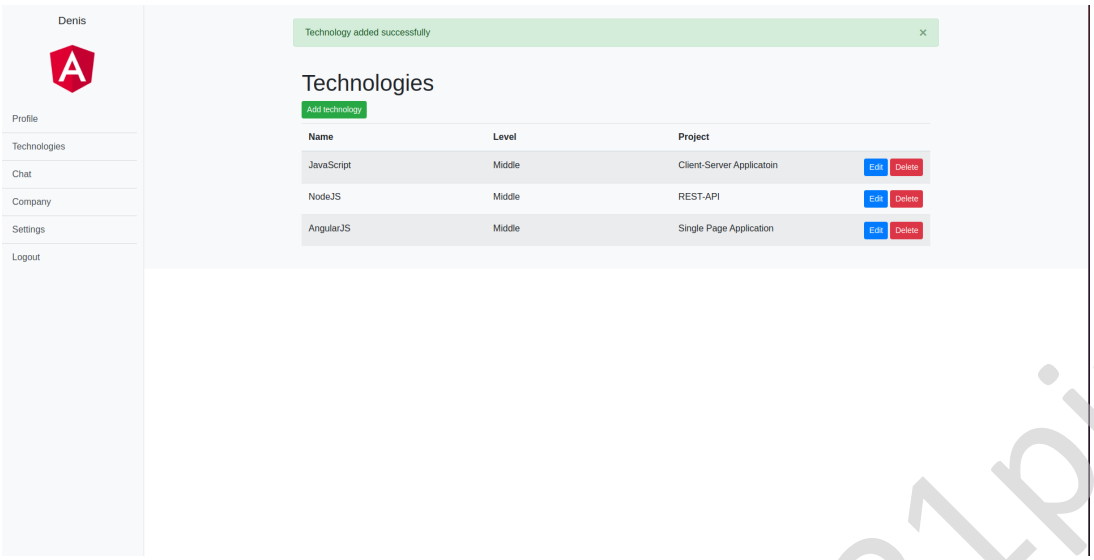

Рисунок 5.1 – Веб-сторінка реалізована за допомогою AngularJS

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 код у, 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 закладані 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 2021 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 магістерської роботи

Після ознайомлення з засобами розробки та необхідними технологіями які використовуались для впровадження та тестування системи, був розроблений план розробки програмного забезпечення. Був визначений необхідний час для розробки та впровадження програми, який склав 22 дні.

В магістерській роботі було проведене дослідження та розроблено два клієнтських додатка та один серверний з використанням бази даних, для вичвлення переваг та недоліків кожної з платформ. Розроблені додатки мають функціонал, які використовують більшість веб-сайтів в Інтернеті, що дає можливість чітко порівняти їх можливості. Серед основного функціоналу наявні:

- Можливість спілкуватися за допомогою чата
- Редагувати профіль користувача, змінювати медіа файли
- Надійна реєстрація та авторизація
- Можливість надсилати e-mail сповіщення
- Динамічне оновлення інформації на стороні клієнта

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	3
2. Кількість екземплярів програм, шт.	Ne	240
3. Запланований термін розробки, днів	Fpq	22

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

27. Досвід роботи з обчислювальною платформою (1-6)	–	4
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	3
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт	–	3
32. Вартість ПЗ у розробника (НМА), грн.	–	36000
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%

Продовження таблиці 7.1

37. Рівень рентабельності програмної продукції, %	Ре	30%
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Трудомісткість розробки програмного забезпечення визначається в людино-днях. Такі норми часу охоплюють всі стадії розробки документації до технічного завдання, технічного проекту та впровадження у виробництво. Стадія технічного завдання (ТЗ) включає в себе збирання необхідної інформаії, розрахування можливих методів вирішення задачи, вибір мови програмування, визначення часу на розробку документації, та затвердження технічгог завдання. Стадія технічного проекту (ТП) включає розробку блок-схеми, визначення технологій, розробку документації, визначення вхідних та вихідних даних а

також структуру програми. Стадія робочого проекту (РП) включає етап розробки програми та її налаштування. Стадія впровадження (ВП) включає перевірку і вихід програмного забезпечення в промислову експлуатацію. Проце створення програмного забезпечення, в першу чергу починається з планування. Правильна оцінка трудовитрат є основою для планування всього подальшого проекту.

Існує багато методик для визначення трудовитрат, але найбільшу отримала методика COCOMO (Constructive Cost Model) в основі якої лежить визначення об'єму робіт по кількості строчок коду. Використаємо її для визнаення трудоміскості розробки ПЗ для тадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \text{ Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, $A = 2{,}45$;

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

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001 \sum Wi, \quad (7.2)$$

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

Таблиця 7.2 – Масштабуючі показники:

Попередній досвід	3	2,43
Гнучкість проекту ПП	4	2,43
Детальність проекту ПП	4	1,69
Рівень спрацьованості колективу	3	2,97
Ступінь вимірності процесів	4	2,97

$$\sum Wi = 2{,}43 + 2{,}43 + 1{,}69 + 2{,}97 + 2{,}97 = 12{,}49,$$

$$B = 1{,}01+0{,}001*12{,}49= 1{,}02249,$$

$$T_{ном} = 2{,}45 \cdot 2{,}5^{\,1{,}02249} = 6{,}2525 \text{ люд-міс.}$$

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

$$T_{уточн} = T_{ном}\, ПV_j, \qquad (7.3)$$

де: $ПV_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в додатку 3.

Таблиця 7.3 – Коефіцієнти трудовитрат:

Необхідна надійність програмного забезпечення	4	1,15
Розмір бази даних (порівняно з розміром програми)	2	0,93
Складність кінцевого програмного продукту	3	1
Необхідний рівень забезпечення повторного використання	3	1
Документованість відповідно до планового життєвого циклу	3	1

Продовження таблиці 7.3

Вимоги до швидкодії ПП	4	1,11
Обмеження на розміри основного сховища даних	1	1
Різноманітність використовуваних обчислювальних платформ	4	1,15
Професійний рівень аналітиків	2	1,22
Професійний рівень програмістів	2	1,16
Постійний склад команди розробників	3	1
Досвід розробки додатків	3	1
Досвід роботи з обчислювальною платформою	4	0,88

Досвід роботи з мовою і інструментами середовища розробки	3	1
Досвід роботи з програмними інструментами розробки	3	1
Розробка ПО для декількох серверів одночасно	2	1,10
Вимоги до дотримання встановленого графіка робіт	3	1

$$Пv_j = 1{,}15*0{,}93*1*1*1*1{,}11*1*1{,}15*1{,}22*1{,}16*1*1*0{,}88*1*1*1{,}10*1 = 1{,}8702$$

$$T_{уточн} = 6{,}2525*1{,}8702 = 11{,}6934 \text{ люд-міс.}$$

Визначаємо підсумкові трудовитрати, люд-дні:

$$T_{рп} = 0{,}3CT^{0{,}33+0{,}2(B-1{,}01)}S \qquad (7.4)$$

де: C – визначений емпірично коефіцієнт;

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 11{,}6934^{0{,}33+0{,}2(1{,}02249-1{,}01)} \cdot 100 = 180{,}75 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.4.

Таблиця 7.4 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава

Технічне завдання	9	Додаток №5
Ескізний проект	10	Додаток №6
Технічний проект	6	Додаток №7
Робочий проект	180,75	Ф 7.1-7.4
Впровадження	18	Додаток №11
Всього	223,75	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Кількість інженерів-програмістів розраховується за формолою:

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

$T_{пз}$ - трудомісткість розробки програми

N - кількість розроблених програм за рік

$F_{рq}$ - запланований термін розробки

$H_{ев}$ - невиходи за поважних причин, днів

$$Ч = (223,75 * 1) / (22- 3) = 11,7763$$

Кількість працівників для проведення технічного обслуговування визначається від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Таблиця 7.5 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на	Кількість облад-	Затрати часу в	Затрати часу в год.

	один. обл.	нання	хв.	
Системний блок ПК	385	3	1155	19
Монітор	160	3	480	8
Клавіатура	140	3	420	7
Маніпулятор «мишка»	30	4	120	2
Принтер матричний	185	1	185	3
Принтер лазерний	355	1	355	6
Принтер струминний	300	1	300	5
Сканер	155	1	155	2.5
Концентратор-маршрутизатор	155	2	310	5
Кабельні господарства ЛВС на 1 м п.	2,5	5	12,5	0.2
Копіювальний апарат	285	1	285	5
Усього за рік:			$З_ч$	62,7

Норми часу на профілактику ЕОМ дорівнюють 10%.

$$Ф^c_{dp} = (З_ч * n_{mic})/1,2 \quad (7.6)$$

$$Ф^c_{dp} = (62,7 * 1)/1,2 = 52,25 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = Ф^c_{dp} / (F_{pq} * Т_{зм}) \quad (7.7)$$

$$Ч_{ел} = 52,25 /(22 \cdot 8) = 0,296875 \text{ ставки.}$$

Чисельність персоналу інженерів-системотехніків, адміністраторів мереж, 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 7.6 – 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 20012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	1,5	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1,5	
Всього		4	
Посада	Вид роботи	Час	К-ть 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	Розробка концепції оформлення та інтерфейсу сайту,	2	0,5

WEB	оптимізація дизайну існуючих, проектує їх структуру та навігацію		
	Створення графічних і стилістичних елементів сайту	1	

Продовження таблиці 7.6

	Оформлення банерів і промо-сторінок	0,5	
	Розміщення графіки і контенту на Інтернет сторінках	0,5	
Всього		4	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.7 - Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	10000	10000
Продакт-менеджер	1	8000	8000
Інженер-програміст	3	8000	24000
Інженер-електронщик	1	8000	8000
Інженер-системотехнік	0,5	8000	4000
Адміністратор мережі	0,5	8000	4000
Системний програміст	1	8000	8000
Дизайнер WEB	0,5	8000	4000
Інженер-верстальник	0,5	8000	4000
Бухгалтер-економіст	1	8000	8000
Всього за період	$R_{сп} = 10$	-	$Ф_{роб} = 82000$

розробки			

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{cd} = Ф_{роб} / (R_{сп} * F_{рq}) \quad (7.8)$$

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

$$З_{cd} = \ 82000/ (10*22) = 372 \ грн$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість складається з ціни, транспортно-заготівельних витрат, вартості будівельно-монтажних та пуско-налагоджувальних робіт, а також витрат на випробовування у виробничих умовах. Норма амортизації визначається в залежності від нормативного строку служби.

Балансова вартість будівель (В буд) визначається з урахуванням кількості робочих місць виконавців (R ñї 1), питомої площі на одне робоче місце (S у = 8...12 м 2) та вартості одного квадратного метра виробничої площі (Ц пл = 1500 грн. і вище).

$$В_{буд} = R_{сп} * S_у * Ц_{пл} \quad (7.9)$$

де: $R_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

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

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

$$В_{буд} = 10 * 8 * 1500 = 75000$$

Вартість передавальних пристроїв (електромережі, водопровід, тепломережі тощо) можна орієнтовно прийняти до 12% вартості будівель і у даному випадку вона складе: 9000 грн.

$$В_{перед} = 0,01 В_{буд} П_{перед} \quad (7.10)$$

$$В_{перед} = 0,01*75000*5 = 3750$$

Балансову вартість господарського інвентарю (меблі, стелажі, крісла тощо) доцільно розрахувати за орієнтовною нормою від 800 грн. до 6000 грн. на одне робоче місце:

$$І_{нв} = R_{сп} * Ц_{м} \quad (7.11)$$

де: $Ц_{м}$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 10 \cdot 4000 = 40000 \text{ грн.}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.8.

Дані по оптовій ціні на обладнання та комплектуючі вибирались фірми Rozetka за 11.11.21– джерело http://rozetka.com.ua/

Вартість вимірювальних пристроїв можна прийняти до 5% від вартості персональних комп'ютерів:

$$В_{вим.пристр} = 0,01В_{пк}П_{вим.пристр}, грн \quad (7.12)$$

$$В_{вим.пристр} = 0,01 * 51150 * 5 = 2557,5$$

де $В_{ПК}$ – вартість персонального комп'ютера;

$П_{вим.пристр}$ – відсоток, який встановлює залежність між вартістю вимірювальних пристроїв та вартістю персонального комп'ютера.

Таблиця 7.8 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп'ютер		15500
Системний блок		

Продовження таблиці 7.8

Процесор	Intel Core i3-10105F 3.7GHz/6MB (BX8070110105F) s1200 BOX	2 599
Системна плата	Asus Prime H410M-K (s1200, Intel H410, PCI-Ex16)	2 118
Відеокарта	Gigabyte PCI-Ex GeForce GT 710 2048MB DDR3 (64bit) (954/1800) (HDMI, DVI, VGA) (GV-N710D3-2GL)	1 949
Жорсткий диск	Toshiba P300 1TB 7200rpm 64MB HDWD110UZSVA 3.5 SATA III	1 049

Оперативна пам’ять	AMD 8 GB DDR4 2400 MHz (R748G2400U2S-U)	792
DVD-привод	Asus DVD±R/RW USB 2.0 SDRW-08D2S-U LITE Black External	861
Корпус	GameMax MT508-NP-2U3	751
Кулер	–	–
Кардрідер внутрішній	Gembird SD/MMC/RS-MMC/MicroSD картами памяти и 2.5" HDD/SSD (FDI2-ALLIN1-03)	468
Інше (Клавіатура, мишка)	A4Tech KM-72620D USB	399

Продовження таблиці 7.8

Монітор	23.8" Acer SA240YBbmipux (UM.QS0EE.B01) USB Type-C 15W	4 499
Принтер лазерний	HP LaserJet 107a (4ZB77A)	4 421
Найменування комплектуючої або обладнання	Тип	Оптова ціна
Принтер струминний	Canon PIXMA Ink Efficiency E414 (1366C009)	1 709
Сканер	IRISCan Express 4 (458510)	4 019

Копіювальний апарат	RICOH MP1600	8 344
Пристрій безперебійного живлення	LogicPower LPM U1400VA-P (LP10394)	3 014

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.9.

Таблиця 7.9 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Середня ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	3	15500	4650	51150
Принтер лаз.	1	4 421	442,1	4863,1

Продовження таблиці 7.9

Принтер струм.	1	1 709	170,9	1879,9
Сканери	1	4 019	401,9	4420,9
Копіюв. апарат	1	8 344	834,4	9178,4
Всього	–	–	–	71492,3

Таблиця 7.10 – 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	*75000*	-	-
2. Передавальні пристрої	*9000*	-	-
Всього по групі	84000	5	4200
Група 4			
3. Обчислювальна техніка	71493	50	35746,5
4. Інше обладненння	2557,5	50	1278.75
Всього по групі	74050,5	50	37025,25
Група 5			
5. Транспортні засоби	714,92	20	142,944
Група 6			
6. Вимірювальні пристрої	2557,5	-	-
7. Господарський інвентар	40000	-	-
Всього по групі	42557,5	25	10639,375
8. Нематеріальні активи	36000	50	18000
Разом	$К_р$ = 237322,92		$А_р$ = 70007,569

Примітка: вартість не враховуємо. Тому вартість транспортних засобів приймаємо рівним нулю.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = (З_{сд} * Т_{пз})/N_е \quad (7.13)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 372 * 223,75/ 240 = 347$$

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

$$З_д = З_о * Н_д * 0,01, \quad (7.14)$$

де: H_q – норматив додаткової зарплати, %.

$$З_д = 347 * 10 * 0,01 = 34,7$$

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

$$С_{оц} = 0,01 * Н_с * (З_о + З_д), \quad (7.15)$$

де: H_c – відрахування на соціальні потреби, %.

$$С_{оц} = 0,01 \cdot 22(346 + 34,7) = 83,754$$

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

$$Г_{осп} = З_о * Н_с * 0,01, \quad (7.16)$$

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 347 * 15 * 0,01 = 52,05$$

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

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.17)$$

де: $З_{М1}$– вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм приймаємо одну пачку паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$ = 100 грн., визначаємо вартість паперу за період розробки $N_м$ = 1 міс:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.18)$$

$$З_{М1} = 100 \cdot 1 = 100 \text{ грн.}$$

Згідно виданих викладачем норм до вартості запам'ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_д , \quad (7.19)$$

де: $Ц_д$ – вартість дисків

Наприклад: CDR TDK 700Mb, 80Min, 52x Cake box – 5 грн/шт., DVD-R LG 4,7Gb, 16x speed Cake box - 7 грн/шт.

$$З_{М2} = 240 \cdot 12 = 2880 \text{ грн.}$$

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

$$З_{МЗ} = \sum Ц_{з}.,$$ (7.20)

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: картридж для CANON LBP-3010 Black Canon 712 – 574 грн.; картридж для EPSON STYLUS PHOTO R390 – 558 грн.; картридж для CANON IR-1022A – LJ Q2612A Cart. HP LJ 1010/1012/1015/3015/3020/3030 (2500 стр.) – 570 грн.

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

$$З_{М} = (100 + 2880 + 1702)/240=19,5 \text{ грн}.$$

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

$$О_п = З_о * Н_п * 0,01$$ (7.21)

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 347 * 15 * 0,01 = 52,05 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 240 прим.):

$$А_м = (А_р * N_{міс})/(N_е * 12)$$ (7.22)

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

$$А_м = 70007,569 * 1/(240*12) = 24,30 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_М + О_п + А_м \quad (7.23)$$

$$С_п = 347+34,7 + 83,754 +52,05 + 19,5 +52,05 + 24,30 =613,354 \text{ грн.}$$

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

Для даного програмного забезпечення рівень рентабельності складає 30%.

$$П_р = 0,01 * Р_п * С_п \quad (7.24)$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0,01 * 30 * 613,354 = 184,0062 \text{ грн.}$$

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

Оптова ціна підприємства складається з повної собівартості і планового прибутку. Визначаємо оптову ціну підприємства:

$$Ц_п = С_п + П_р \quad (7.25)$$

$$Ц_п = 613,354 + 184,0062 = 797,3602$$

Величина податку на додану вартість:

$$ПДВ= 0,01 \cdot 20 * 797,3602 = 159,47204$$

де: Нпдв – ставка податку на додану вартість.

Відпускна ціна програмної продукції:

$$Ц = Цп + ПДВ \quad (7.26)$$

$$Ц = 797,3602 + 159,47204 = 956,83224$$

Таблиця 7.11 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
Основна зарплата виконавців	$З_о$	347
Додаткова зарплата виконавців	$З_д$	34,7
Відрахування на соціальні потреби	$С_{оц}$	83,754
Загальногосподарські витрати	$Г_{осп}$	52,05
Витрати на матеріали	$З_М$	19,5
Освоєння нових операційних систем, мов програмування	$О_п$	52,05
Амортизація основних фондів	$А_м$	24,30
Повна собівартість програмного забезпечення	$С_п$	613,354
Плановий прибуток	$П_р$	184,0062
Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	797,3602
Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	159,47204

Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	956,83224

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 7.12 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	956,83224
Всього капітальних витрат	–	956,83224

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року.

Таблиця 7.13 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	15030	9662
2. Витрати на електроенергію	$З_{ел}$	1188	743
3. Витрати на амортизацію	$З_{ам}$	0	1580

Всього витрат за рік	*І*	16218	11985

Витрати на профілактичні роботи:

$$З_р = Т_р * З_г * (1 + 0,01 * Н_д) * (1 + 0,01 * Н_е) \quad (7.27)$$

де: $Т_р$ – кількість годин обслуговування кожного комп'ютера за рік, год;

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 280 годин на рік до 180 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{рбаз} = 280 * 40 * 1,1 * 1,22 = 15030,$$

$$З_{рнов} = 180 * 40 * 1,1 * 1,2 - 9662.$$

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

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел,} \quad (7.28)$$

$$З_{ел\ баз} = 0,45 \cdot 1200 \cdot 2,2 = 1188 \text{ грн.},$$

$$З_{ел\ нов} = 0,45 \cdot 750 \cdot 2,2 = 743 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.14.

Таблиця 7.14 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн. за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	956,83224	–	239.20806
Всього відрахувань	-	–	956,83224	–	239.20806

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:

$$E_в = (Ц_в — C_п) * N_е - E_р * K_р \quad (7.29)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.

$$E_в = (797,3602 — 613,354) * 240 — 0,15 * 237322,92 * 1/12 = 41194,9515$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = K_р/ ((Ц_в — C_п) * N_е), \quad (7.30)$$

$$T_в = 237322,92/((797,3602 — 613,354) * 240 * 12/1) = 0,447831604$$

Таблиця 7.15 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця	Величина

	виміру	
1. Кількість екземплярів програми	Прим.	240
2. Повна собівартість розробленої програми	Грн.	613,354
3. Ціна розробленої програми	Грн.	797,3602
4. Плановий прибуток від реалізації розробленої програми	Грн.	184,0062
5. Рентабельність програмної продукції	%	30
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	237322,92

Продовження таблиці 7.15

7. Загальний прибуток від реалізації програмної продукції	Грн.	44161
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	41194
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Рік	0,4
10. Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	956,83224
11. Величина економічного ефекту у користувача програмної продукції	Грн.	3993,79
12. Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,22

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) — E_н (K_н - K_б), \quad (7.31)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$К_б$, $К_н$ – об’єм капітальних вкладень за варіантами, що порівнюються.

$$Е_{сп} = (16218\text{-}11985) — 0{,}25 * 956{,}83224 = 3993{,}79194 \text{ грн.}$$

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

$$Т_{сп} = (К_н - К_б)/(І_б - К_н), \quad (7.32)$$

$$Т_{сп} = 956{,}83224 /(16218\text{-}11985) = 0{,}226041162 \text{ роки}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.15.

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

Інтернет відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують Інтернет для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин. Багато людей користуються Інтернетом у професійних цілях, оскільки завдяки Інтернету зявилося багато нових професій. Тому для веб-розробника так важливо розробити зручний інтерфейс для зручного сприйняття інформації, та необхідний функціонал, який буде відповідати необхідним вимогам та навантаженням. Все це вимагає багато багато часу та великого навантаження з боку розробників.

Тому так важливо слідкувати за умовами праці, в яких відбувається робочий процес. Оскільки захворювання можуть бути спричинені надмірним фізичним або розумовим навантаженням, через велику нервово-емоційну напругу, або через виробниче середовище. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

При роботі за компютером, розробник має велике зорове навантаження, тому йому необхідне належне освітлення приміщення. Якщо в приміщенні недостатньо природного освітлення, потрібно використовувати спеціальні світильники. Також оскільки розробник значний час працює з електричними приборами є можливість, бути ураженим електричним струмом, тому потрібно дотримуватись всіх необхідних норм. Серед основних чиників, які впливають на розробників під час трудової діяльності можна виділити[46]:

1. Рівень освітлення в приміщенні.
2. Температура, вологість в приміщенні.
3. Рівень шуму на робочому місці.

4 .Напруга в електричному ланцюзі, електричні показники.

Розберемо кожний з чиників окремо, проаналізуємо які повинні бути стандарти кожного з чиників, відповідно до правил з охорони праці.

Рівень штучного освітлення

Головним документом для встановлення норм необхідних показників освітлення є ДБН В.2.5-28:2018 «Природне та штучне освітлення» [47].

Сьогодні найбільш розповсюдженішими є світлодіодні ламп. В середньому світловіддача від таких ламп знаходиться на рівні 80-120 Лм/Вт [48]. Джерелом живлення прийнято вважати електричну мережу у 220В. А освітленіть робочого приміщення повина бути Е =300-500 Лк, оскільки робота програміста відноситься до робіт середньої точності з присвоєнням розряду зорових робіт IV[49].

Рівень сітла повинен бути достатнім, щоб працівник міг працювати без навантаження на зір. Це залежить від системи освітлення, кількості світильниув, їх типу та розміщення у приміщенні. Допустиме значення освітленості робочої поверхні приймається Е = 400 лк [50].

Для покращення освітлення комп'ютерній лабораторії будуть використовуватися світлодіодні лампи, а саме FL-LED T8-900 світловий потік яких F=1500лм.

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

Головним документом для встановлення норм мікроклімату робочої зони є ДСН 3.3. 6.042 -99 «Державні санітарні норми мікроклімату виробничих приміщень» [51].

Праця програміста за важкістю відноситься до легкої фізичної роботи категорії Іа [49]. Де вказано, що в приміщенні, я кому знаходиться компютерне обладнення, повнині бути встановлені певні норми, оскільки офісна техніка є джерелом тепловиділень, що може спричинити підвищення температури. Серед

потимальних параметрів для роботи встановлена температура 23 – 25^{0}С і вологість на рівні 40 – 60% в залежності ві періоду року.

Для підтримки комфортної температури можна використовувати як організаційні методи, наприклад розпорядок дня, так і технічне обладнення, наприклад кондиціонери, вентіляцію. Як правило в холодний період часу використовуються додаткове опалення для підтримання комфортної температури, а в літку встановлюються кондеціонери.

Рівень шуму на робочому місці

Як правило при використанні великої кількості компютерів в одному приміщенні, через гудіння, рівень шуму має значення більше норми. Допустима норма становить менше 50 дБ [52].

Гучний шум негативно впливає на умови праці та організм людини. Якщо шум триває тривалий час цу може спричинии головні болі, біль у вухах, підвищення стомлюваності, зниження концентрації та уваги. Такі симптоми можуть викликати стресові ситуації у людини. Все це шкодить продуктивності працініка та його стану здоровья.

Щоб встановити необхідний рівень шуму, використовують додаткову звукоізоляцію. Для цього найчатіше використовуються 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

Санітарно-гігієнічні норми є важливим критерієм при роботі в приміщенні. Від них залежить здоровє працівників, їх рівнень працездатності, втомлюваність. Щоб всього цього уникнути потрібно стежити за нормами на робочому місці.

Якщо говорити про електробезпеку в приміщенні, то в приміщенні необхідно устаткування розподільних щитів спеціальними розетками з

заземлюючими контактами, повині бути заземлені всі прилади і пристрої, час від часу повина проводитися перевірка всіх приладів, щорічна здача іспитів з охорони праці.

Для оптимальних показників мікроклімату та освітленості потрібні використовувати дефлектор, для організації вентиляції, та повітрообміну. Та перевірку освітленості в приміщенні згідно відділом охорони праці, щоб відповідати номам для зорової роботи .

Ще однією проблемою з якою часто зустрічаються програмісти є мала рухливість та повний сидячий робочий день. Тому рекомендується час від часу робити невеликі перерви, під час обіднього перериву вживати їжу не на робочому місці. У окремих випадках, коли при дотриманні всіх санітарних норм, працівник все одно себе погано почуває, дозволяється індивідуальний підхід для обмеження роботи з обчислювальними пристроями. Тривалість роботи за компютером не повина безперервно триввати більше 4 годин.

Для зменшення зорового та нервово-емоційного навантаженняє, та поліпшення мозкової діяльності рекомендується робити перерви для психологічного та фізичного розвантаження.

В приміщеннях також поині бути протипожежне обладнення, та інструкція у разі надзвичайних ситуаціх. Повина бути особа, яка відповідає за пожежну безпеку, перевіряє обладнення, та системи протипожежного захисту а також щорічне проведення інструктажів серед працивників.

Автоматична пожежна сигналізація повина відповідати вимогам ДБН ДБН В.2.5-56:2014, яке вимагає використання вогнестійких кабелів та автоматичну роботу системи оповіщення та евакуації людей у випадку надзвичайної ситуації [53].

При перевірці, приміщення повино відповідати всім нормам пожежної безпеки. Це виконується за допомогою перевірки пожежної охорони та техніки, проведенню інструктажу і своєчасне інформування пожежної охорони про несправність пожежної техніки, впровадження систем протипожежного

захисту. Організаційні та технічні заходи, спрямовані на попередження виникнення пожежі, обмеження поширення вогню та успішної евакуації людей.

8.3 Розрахнок занулення глухозаземленої нейтралі

Занулення, як основний засіб захисту, застосовується в електроустановках до 1 кВ з глухозаземленою нейтраллю трансформатора або генератора [54]. Початкові дані для розрахунку занулення глухозаземленої нейтралі трансформатора виробничого приміщення:

Загальна потужність: P = 5 кВт.

Кількість електродвигунів: m = 10

Потужність освітлювальних приладів: Po = 3 кВт.

Довжина магістрального кабеля: LM =70 м.

Довжина розгалудження: l=22м.

Лінійна напруга U=380 В.

Фазна напруга Uф=220 В.

Визначаємо силу номінального струму електроустановки:

$$Iном = Imax = (P * 1000) / (\sqrt{3} * Uл * cosф) \quad (8.1)$$

$$Iном = Imax = (5 * 1000)/(\sqrt{3} * 380 * 0,85) = 8,9 А$$

де: P – номінальна сумарна потужність електроприладів, кВт;

Uл – лінійна напруга, В;

cosф – коефіцієнт потужності, приймається в залежності від типу електрообладнання в межах 0,8..0,87.

Визначаємо силу пускового струму електродвигуна:

$$Iпус = 5 * Iном \quad (8.2)$$

$$Іпус = 5 * 8,9 = 44,5 \text{ А}$$

Визначаємо номінальну силу струму апарата захисту:

$$Ін = Іпус/b \quad (8.3)$$

$$Ін = 44,5 /2,5 = 17,8 \text{ А}$$

b − коефіцієнт пуску електродвигуна − для легких умов пуску − 2,5..3.

Вибираємо запобіжник ПН 2-100 з плавкою вставкою Іном = 50 А. Визначаємо найменше допустиме по умовам спрацьовування захисту значення сили струмукороткого замикання:

$$Іктіп = Ін * К \quad (8.4)$$

$$Іктіп = 50 * 3 = 150 \text{ А}$$

Ін−номінальний струм апарата захисту;

К − коефіцієнт надійності;

Знаходимо переріз провода або кабеля розгалуження з умови допустимого нагрівання:

$$Ідоп = Івст/а \quad (8.5)$$

$$Ідоп = 50 / 3 = 16,6 \text{ А}$$

Вибираємо площу перерізу 10 мм^2 (Sф) при числі проводів і = 4 розташований у повітрі. Визначаємо максимальний робочий струм:

$$Iроб = Ко(Кз*(Р*1000)/(\sqrt{3}*Ул*cosф)*m+Кз*(Ро*1000)/(\sqrt{3}*Ул*cosф)) \quad (8.6)$$

Iроб = 0,75*(0,85*((5*1000)/(1,73*380*0,85)*10+(3*1000)/(1,73*380*0,85))=60,4 А

Ко − коефіцієнт одночасності роботи групи електроприймачів;

Ко = 0.7...0.8; Кз = 0.8...0.9;

Кз – коефіцієнт завантажених електродвигунів;

Ро =3кВт − потужність освітлювальної мережі;

Визначається струм короткочасного перевантаження магістрального кабеля:

$$Iпер = Ко(Кз*(Р*1000)/(\sqrt{3}*Ул*cosф)*n+Кз*(Ро*1000)/(\sqrt{3}*Ул*cosф))+Iпус \quad (8.7)$$

Iпер = 0,75*(0,85 *(5*1000)/(1,73*380*0,85)*9+0,85*(30*1000)/(1,73*380 * 0,85)) + 44,5 = 130,0643 А

Струм спрацювання електромагнітного розчеплювача додатково перевіряємо по максимальному струму перевантаження лінії:

$$Iспр >= 1,25 * Iпер \quad (8.7)$$

Iспр >=1,25 * 130,0643 = 162,5803 А

Приймаємо Iспр = 162,5803 А. Вимикач : А3714Б.

Вибираємо площу перерізу Sф магістрального кабеля (провідника) по Iдоп. Sф = 70mm ^{2} , − кабель АВРГ прокладений в землі, і=3 (число проводів). Вибрану площу перерізу перевіряємо для автоматів з електромагнітним розчеплювачем:

$$Iдоп >= Iспр/4,5 \quad (8.8)$$

$$Iдоп >= 162/4,5 = 36 А$$

Проводимо узгодження з номінальним струмом автомата:

$$Iдоп = Iспр/3 \quad (8.9)$$

$$Iдоп = 162/3 = 54 А$$

Значення 36 і 54 А. менше ніж Imax= 60,4 А., значить площа перерізу кабеля вибрана вірно. Визначаємо потужність трансформатора:

$$Nтр = ((Кп * Рном)/cosф) \quad (8.10)$$

$$Nтр = (0,7 * 53)/0,8 = 46,375 кВ*А$$

Рном − сумарна потужність електроприймачів, кВт;

Cosф − середній коефіцієнт потужності електроприймачів (0,8);

Кп − коефіцієнт попиту (0,7);

Одержане значення потужності трансформатора округляємо до ближчого стандартного значення. Визначаємо опір трансформатора Z_T. Вибираємо трансформатор на 40 кВА (Z_T =0,562 Ом). визначаємо орієнтовно площу перерізу провідника. Для магістрального кабеля:

$$Sn_1 >= 0,5 * Sф \quad (8.11)$$

$$Sn_1 >= 0,5 * Sф = 0,5*70=35 мм^2$$

Визначаємо для розгалуження:

$$Sn_2 >= 0,5*10 = 5 \text{ мм}^2$$

Округляємо ці значення до найближчих більших 35 мм^2 (Sn1), і 6 мм^2 (Sn2). Визначаємо активний і індуктивний опір фазного і нульового захисного провідників на дільницях 1 і 2:

$$Rф = p * (Lм/Sф_1) + p * (L/Sф_2) \quad (8.12)$$

$$Rф = 0,028 * (70/70) + 0,028 * (22/10) = 0,0896 \text{ Ом}$$

$$Rn = p * (Lm/Sn_1) + p * (L/Sn_2) \quad (8.13)$$

$$Rn = 0,028 * (70/35) + 0,028 * (22/6) = 0,1586 \text{ Ом}$$

Для окремо проложених нульових провідників иого приимають рівним 0,6 Ом/км. При прокладці кабелем, або в сталєвих трубах індуктивним опором нехтують.

Знаходимо дійсне значення (модуль) струма однофазного короткого замикання:

$$Iкр = Uф/((Zт/3) + \sqrt{((Rф + Rn)^2 + (Xф + Xn + X'n^2))} \quad (8.14)$$

$$Iкр = 220/((0,562/3) + \sqrt{(0,0896 + 0,1586)^2}) = 418,18 \text{ А}$$

Визначення максимальноі напруги на корпусі обладнання відносно землі при замиканні фази на корпус.

$$Uктах = Iкр * Zн \quad (8.15)$$

$$Uкmax = 418,18 * 0,1586 = 66,32 В$$

Z_H − повний опір нульового провідника.

Умова не виконується. Необхіно збільшити перерізи Sn 1 та Sn 2 до Sф1 та Sф2 і зробити перерахунок:

$$Rn = 0,028 * (70/70) + 0,028 * (22/10) = 0,0896 Ом,$$

$$Iкр = 220/((0,562/3) + \sqrt{(0,0896 + 0,0896)^2}) = 600,21 А,$$

$$Uкmax = 600,21 * 0,0896 = 53,77 В.$$

Умова не виконується, необхідно або замінити запобіжник з плавкою вставкою на автоматичний вимикач із струмовим реле, що дає можливість зменшити час замикання на копус і підвищити допустиму напругу на корпусі або застосувати повторне заземлення нульового захисного провідника. Повторне заземлення нульового захисного провідника:

$$Rn=(Uдоп * Ro)/((Iкр * Zн)−Uдоп) \quad (8.15)$$

$$Rn= 36 * 4/((600,21 * 0,0896) - 36) = 8,09 Ом.$$

8.4 Висновки

Існує багато факторів, які можуть вплинути на роботу розробника, через некомфортні або навіть небезпечні умови праці, які знаходяться в приміщенні. Серед основних причин виділяється: недостатній рівень світла, гучний шум, високий рівень навантаження, умови мікроклімату. Під час дослідження теми, були переглянуті можливі шкідливі та небезпечні ситуації, які можуть виникнути на робочому місці та спооби їх уникнення та ліквідації.

В результаті можна зробити висновки, які умови повині бути для продуктивної роботи працівника, як повино бути організоване приміщення і його робоче місце. За допомогою проведених обчислень, можна становити необхідні для комфортної роботи умови.

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:

– Було створено односторінковий додаток засобами AngularJS та NodeJS.

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

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

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

– Було 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 у сумі

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 *3993* грн. Враховуючи вартість розробки та необхідне обладнення, строк окуплення становить 0,22 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Язык программирования JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/

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

3. Серверное программирование веб-сайтов [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 действии”.

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 ІВМ−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

Це технічне завдання розповсюджується на розробку веб-сайту компанії засобами фреймворку AngularJS.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську дипломну роботу, видане на кафедрі кібербезпеки та програмного забезпечення (наказ № 42-13 від 02.08.2021 року).

3 Мета та призначення розробки

Метою магістерської дипломної роботи є дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS.

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 Вміст проекту

Складовими розробки є:

– аналіз існуючих програмних аналогів;

– вибір і обґрунтування методики побудови додатків і засобів їхньої реалізації;

– розробка структур даних і механізму їхньої взаємодії, робочих форм і засобів;

– розробка програми, яка реалізує алгоритми роботи компоненту.

– аналіз умов праці;

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

Компонент повинен використати існуючі угоди по стандартним викликам процедур, функцій, засобів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Автоматизовані робочі місця користувачів системи повинні задовольняти наступним умовам експлуатації:

– температура повітря: 18-22^0 С;

– відносна вологість повітря при 20^0 С до 80%;

– атмосферний тиск 107 кПа;

5.7 Вимоги до складу і параметрів технічних засобів

Компонент повинен бути реалізований на ЕОМ типу IBM PC в операційному середовищі Linux і орієнтований на сумісні з цією платформою зовнішні пристрої, мережне обладнання і прикладне програмне забезпечення.

5.8 Вимоги до інформаційної та програмної сумісності

Сумісність програмного забезпечення повинна бути забезпечена за рахунок використання бібліотеки MongoDB, яка сумісна з усіма останніми версіями операційної системи Linux.

5.8.1 Обладнання

Комп’ютер AMD E350/2 Gb/128 Gb/SVGA 14” 512 Mb 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 в вигляді опису схем і алгоритмів, інструкції користувача, а також текстів вхідних модулів програмного забезпечення в відповідності з 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 15 вересня 2021 року.

8 Вимоги щодо охорони праці

В частині охорони праці магістерської роботи повинен бути розглянутий аналіз умов праці програміста.

9 Перелік документів, які необхідно розробити

– структурна схема системи повна;

– функціональна схема системи;

– блок-схеми алгоритму роботи системи;

– діаграма процесів;

– пояснювальна записка.

10 Етапи розробки

На рівні проекту розробляються (терміни виконання етапів див. в "Завданні на магістерську дипломну роботу"):

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

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

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

10.4 Побудова схем взаємодії структур даних.

10.5 Створення прототипу компоненту. Створення програмного продукту.

10.6 Відлагодження компоненту, аналіз отриманих результатів.

10.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю і приймання

11.1 Подання магістерської дипломної роботи на попередній захист 04.12.2021 р.

11.1 Подання магістерської дипломної роботи на попередній захист 04.12.2021 р.

ВІДГУК

на магістерську дипломну роботу *Фесечка Дениса Вікторовича*

тема:

Долсідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS

Програмне забезпечення, створене в результаті виконання магістерської дипломної роботи, призначено для представлення веб-сайту компанії засобами фреймворку AngularJS. Реалізоване програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань з його сторони.

Розроблена магістерська дипломна робота представлена у вигляді розрахункової пояснювальної записки та графічної частини.

Всі аналізи і розробки системи виконані логічно і відповідають сучасному рівню розвитку науки і техніки. Всі необхідні матеріали та розрахунки викладені в повному обсязі.

Магістерська дипломна робота, яка розроблена студентом загалом відповідає вимогам, що ставляться до магістерських дипломних робіт за спеціальністю 123 “Комп’ютерна інженерія”. Вважаю, що магістерська дипломна робота заслуговує на високу оцінку, а студенту може бути присвоєно відповідну кваліфікацію.

Керівник магістерської дипломної роботи

_____ Босько В.В

(підпис)

РЕЦЕНЗІЯ

на магістерську дипломну роботу *Фесечка Дениса Вікторовича*

тема:

Долсідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS

Розроблена магістерська дипломна робота представлена у вигляді розрахункової пояснювальної записки та графічної частини, виконаної на п’ятьох аркушах формату А4.

В магістерській дипломній роботі викладено опис та обґрунтування вибраного рішення, розроблене програмне забезпечення для представлення веб-сайту компанії засобами фреймворку AngularJS. Всі аналізи і розробки системи виконані логічно і відповідають сучасному рівню розвитку науки і техніки. Необхідні матеріали та розрахунки викладені в повному обсязі.

В цілому магістерська дипломна робота виконана якісно та відповідає вимогам, що ставляться до магістерських дипломних робіт за спеціальністю 123 “Комп’ютерна інженерія”.

Вважаю, що магістерська дипломна робота заслуговує на високу оцінку, а студенту може бути присвоєно відповідну кваліфікацію.

Рецензент

_____

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-сайту компанії засобами фреймворку AngularJS

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 28

Літера: РП

Кропивницький – 2021 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);
```

Файл головного модуля на сервері

```
import { Module } from '@nestjs/common';
import { RedisModule } from 'nestjs-redis';
import { ConfigModule } from '@nestjs/config';
import { MongooseModule } from '@nestjs/mongoose';
import { MailerModule } from '@nestjs-modules/mailer';
import { HandlebarsAdapter } from '@nestjs-
modules/mailer/dist/adapters/handlebars.adapter';
import V1Module from '@v1/v1.module';
import AppController from './app.controller';
import AppService from './app.service';
import AppGateway from './app.gateway';
@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    MongooseModule.forRoot(process.env.MONGODB_URL as string, {
      autoReconnect: true,
      useCreateIndex: true,
      reconnectTries: Number.MAX_VALUE,
      reconnectInterval: 1000,
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }),
    RedisModule.register({
      url: process.env.REDIS_URL,
      onClientReady: async (client): Promise<void> => {
        client.on('error', console.error);
        client.on('ready', () => {
          console.log('redis is running on 6379 port');
        });
        client.on('restart', () => {
          console.log('attempt to restart the redis server');
        });
      },
      reconnectOnError: (): boolean => true,
    }),
    MailerModule.forRoot({
      transport: {
        host: process.env.MAILER_HOST,
        port: Number(process.env.MAILER_PORT),
        secure: false,
        auth: {
          user: process.env.MAILER_USERNAME,
          pass: process.env.MAILER_PASSWORD,
        },
      },
      defaults: {
        from: process.env.MAILER_FROM_EMAIL,
      },
      template: {
        adapter: new HandlebarsAdapter(),
        options: {
          strict: true,
        },
      },
    }),
    V1Module,
  ],
  controllers: [AppController],
  providers: [AppService, AppGateway],
})
export default class AppModule {}
```

Файл домашньої сторінки AngularJS

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

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

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

        vm.user = null;
        vm.allUsers = [];
        vm.deleteUser = deleteUser;

        initController();

        function initController() {
            loadCurrentUser();
            loadAllUsers();
        }

        function loadCurrentUser() {

UserService.GetByUsername($rootScope.globals.currentUser.username)
                .then(function (user) {
                    vm.user = user;
                });
        }

        function loadAllUsers() {
            UserService.GetAll()
                .then(function (users) {
                    vm.allUsers = users;
                });
        }

        function deleteUser(id) {
            UserService.Delete(id)
            .then(function () {
                loadAllUsers();
            });
        }
    }

})();
```