Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій

Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему «Розробка веб-додатку центру інформаційних технологій Університету митної справи та фінансів»

Виконав: студент групи К19-3

Спеціальність 122 «Комп'ютерні науки»

Назарян Артур Арменович
(прізвище та ініціали)

Керівник к.т.н. Ульяновська Юлія Вікторівна
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент_____
(місце роботи)

_____
(посада)

_____
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УНІВЕРСИТЕТ МИТНОЇ СПРАВИ ТА ФІНАНСІВ

Факультет: *інноваційних технологій*

Кафедра: *комп'ютерних наук та інженерії програмного забезпечення*

Освітньо-кваліфікаційний рівень: *бакалавр*

Спеціальність: *122 — Комп'ютерні науки*

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

К.т.н., доц. Ульяновська Ю.В.

_____

"_____" _____2022 року

ЗАВДАННЯ

на дипломну роботу студента

Назаряна Артура Арменовича

(прізвище, ім'я, по батькові)

1. Тема роботи *«Розробка веб-додатку центру інформаційних технологій Університету митної справи та фінансів»*

керівник роботи *к.т.н., доцент Ульяновська Ю.В.*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджено на засіданні кафедри комп'ютерних наук та інженерії програмного забезпечення від "___"_____20__року №___

2. Строк подання студентом роботи *20.05.2023*

3. Вихідні дані до проекту (роботи) *Аналіз існуючого та розробка нового веб-додатку (програмного забезпечення). Програмне забезпечення, яке буде розроблено у проекті, повинне обробляти дані після їх створення, редагування чи видалення викладачем, який має доступ до панелі керування.*

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати) *Розробка технічного завдання. Вибір програмного забезпечення*

для розробки та обґрунтування обраних засобів. Розробка програмного додатку. Опис розробленого програмного продукту. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень). *1. Тема дипломної роботи. 2. Актуальність дипломної роботи. 3. Дизайн розробленого застосунку. 4. UML діаграми програмного забезпечення. 5. Слайди, що описують процес розробки веб-додатку. 6. Висновки.*

6. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1.	Формування теми, мети, об'єкта та предмета кваліфікаційної роботи.	До 15.10.22	
2.	Укладання бібліографії та вивчення літературних джерел.		
3.	Дослідження предметної області, формулювання завдань кваліфікаційної роботи. Виконання першого розділу роботи.		
4.	Дослідження методів та засобів вирішення завдань дипломної роботи. Виконання другого розділу роботи.		
5.	Вирішення завдань дипломної роботи. Розробка програмного забезпечення (за необхідністю). Виконання третього розділу роботи (практичної частини) роботи.		
6.	Оформлення роботи та подання роботи на кафедру для перевірки	До 22.05.23	
7.	Одержання відгуку рецензента та подання до роботи. Попередній захист	29.05.23-2.06.23	
8.	Подання готової роботи та необхідних документів до роботи на кафедру	До 05.06.23	
9.	Захист	12.06.23-18.06.23	

Студент Назарян А. А.
(підпис) (прізвище та ініціали)

Керівник роботи Ульяновська Ю. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Назарян А. А. Розробка веб-додатку центру інформаційних технологій Університету митної справи та фінансів.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2023.

У даній кваліфікаційній роботі було розроблено веб-додаток центру інформаційних технологій Університету митної справи та фінансів. Веб-додаток складається з серверної (back-end) та клієнтської (front-end) частин. Додаток дозволяє анонімному користувачу переглянути новини та контактні дані центру інформаційних технологій. Кожна новина відноситься до відповідної категорії, яка може бути відредагована або видалена через панель керування. Передбачається, що будуть створені, як мінімум, наступні категорії: «Зустрічі з роботодавцями», «Наукові заходи», «Наші випускники» та «Студентське життя». Цей веб-додаток має сучасний дизайн, зручний, швидкий та має гнучку систему редагування динамічного контенту (новини, категорії новин, технології, партнери, контакти).

Додаток потребує наявність веб-браузеру, для перегляду сторінок, та, як мінімум, 4 гігабайти оперативної пам'яті.

Ключові слова: розробка, веб-додаток, веб-сайт, інформаційні технології.

ЗМІСТ

АНОТАЦІЯ 4

ВСТУП 7

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ 10

1.1 Аналіз публікацій щодо розробки веб-додатків 10

1.2 Аналіз методів розробки веб-додатків 11

1.3 Висновки 18

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ 20

2.1 Вибір програмних засобів для реалізації проекту 20

2.2 Засоби для розробки клієнтської частини 21

2.2.1 JavaScript/TypeScript 21

2.2.3 Ant Design 24

2.2.4 Vue.js 25

2.2.5 NPM 26

2.2.6 VS Code 27

2.3 Засоби для розробки серверної частини 28

2.3.1 Node.js 28

2.3.2 NestJS 29

2.3.3 SQL 31

2.3.4 SQLite 33

2.4 Аналіз розробки дизайну веб-додатку 34

2.5 Висновки 36

РОЗДІЛ 3 РОЗРОБКА ВЕБ-ДОДАТКУ ЦЕНТРУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 38

3.1 Розробка загальної архітектури проекту 38
3.2 Розробка дизайну проекту 41
3.3 Клієнтська (front-end) частина проекту 54
3.4 Серверна (back-end) частина проекту 60
3.5 Тестування 66
3.6 Висновки 71
ВИСНОВКИ 73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74
ДОДАТОК А 76
ДОДАТОК Б 82

ВСТУП

Актуальність дослідження. В умовах сучасних реалій системи освіти, коли студенти здебільшого проходять навчання дистанційно з використанням широкого спектру програмних засобів виникає необхідність усучаснити засоби інформування студентів. Також, за умов обмежених можливостей проведення вступних компаній у режимі присутності студентів в університеті – виникає потреба у наданні інформації про кафедру у онлайн режимі. У сучасному світі дуже важливо встигати за розвитком як технологій, методики навчання так і за інформаційними ресурсами.

У кафедри комп'ютерних наук та інженерії програмного забезпечення є сайт «Інформаційного центру», за допомогою якого студенти можуть дізнаватися інформацію про кафедру та її новини. Сайт вже виглядає застарілим та не є зовсім функціональним. З метою покращення якості сайту було вирішено написати новий використовуючи сучасні технології розробки.

Одним із ключових переваг розробки нового сайту на базі Vue.js є можливість розширення його функціональності у майбутньому без необхідності переписування коду. Також, завдяки гнучкій архітектурі Vue.js, розробка нового сайту може бути більш ефективною та швидкою, з використанням компонентів, які можна перевикористовувати у різних частинах сайту.

Використання Node.js для серверної частини веб-додатку дозволяє створювати високопродуктивні та масштабовані додатки. Node.js забезпечує високу швидкість роботи, а також забезпечує можливість використання однієї мови програмування на клієнтській та серверній сторонах, що зменшує час на розробку та підтримку коду.

Створення CRM системи для керування публічною інформацією (додавання новин, редагування сторінок, оновлення контактів тощо) на сайті дозволить зменшити ризик помилок та збільшити швидкість операцій зі зміною контенту на сайті.

Оновлення «Інформаційного центру» може стати ключовим кроком для кафедри у напрямку підвищення якості навчання та привабливості для потенційних абітурієнтів. Досягнення цієї мети допоможе перетворенню центру на офіційний сайт кафедри з усіма необхідними ресурсами та інформацією. Наявність такого веб-ресурсу для кафедри дозволить не тільки залучати нових студентів, але й підвищувати конкурентоспроможність у ринку вищої освіти, що в свою чергу сприятиме якості навчання та успіху студентів. Тому, інвестування часу та ресурсів у розвиток «Інформаційного центру» може стати вирішальним кроком для кафедри у досягненні її мети.

Мета роботи – спрощення обміну інформацією між університетом та студентами, залучення абітурієнтів до вступу на кафедру комп'ютерних наук та інженерії програмного забезпечення за рахунок автоматизації.

Методи дослідження – метод теорії інформації, обробка та аналіз інформації, методи проектування та розробки програмного забезпечення.

У відповідності з поставленою метою для вирішення технічної задачі в роботі вирішено такі завдання:

1. Проаналізувати технічні засоби, що використовуються для розробки, та обрати необхідні для створення програмного забезпечення веб-додатку центру інформаційних технологій.

2. Розробити вимоги до програмного забезпечення для веб-додатку центру інформаційних технологій на основі аналізу переваг та недоліків існуючих систем.

3. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.

4. Провести тестування

Об'єкт дослідження – веб-додаток центру інформаційних технологій.

Предмет дослідження – апаратно-програмне забезпечення веб-додатку центру інформаційних технологій.

Практичне значення одержаних результатів – підвищиться якість та швидкість комунікації між університетом та студентами. Також підвищиться імідж кафедри комп’ютерних наук та інженерії програмного забезпечення.

Результати роботи – розроблений сучасний веб-додаток центру інформаційних технологій.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз публікацій щодо розробки веб-додатків

Швидкість, з якою поширюється інформація сьогодні, є вражаючою. Інтернет став каталізатором до поширення інформації. В Інтернеті знаходяться мільйони веб-сайтів різного спрямування. Розвиток Інтернету нерозривно пов’язано з проектуванням сайтів. Масова поява сайтів спровокувала проблему їх якості. Популярність створення веб-ресурсів сприяла розробці різних систем і програм, які спрощують процес написання сайту. Також вони допомагають підвищити ефективність роботи, а також дозволяють розробнику сфокусуватися на основною логікою програми. Такі технології, як PHP, Java, Microsoft.Net, MySQL, Oracle, Microsoft SQL Server і розроблені на їх основі фреймворки — це каркаси системи або підсистеми, що можуть включати допоміжні програми, мови сценаріїв — і все, що полегшує розробку й об’єднання різних компонентів.

Використання фреймворків стає все більше і більше популярним, і це обгрунтовується тим, що розробка з допомогою фреймворку зменшує навантаження на процес розробки web-додатків, це досягається тим, що розробка з використанням фреймворку позбавляє від проблеми використання повторюваного коду. Без використання фреймворків, стає набагато складніше створювати web-додатки, супроводжувати і модернізувати їх. Між тим, використання фреймворків робить процес створення програми набагато більш легким і функціональним. Проаналізувавши інформацію з мережі Інтернет, можна побачити що існує сотні фреймворків для створення web-додатків. Тому досить складно зробити вибір фреймворку, так як кожен з них має велику кількість привабливих функцій та доповнень. А неправильний вибір фреймворку може стати основною причиною невдачі проекту [1].

У роботі [2] детально розглянуто проектування архітектури веб-застосунку для імітаційного моделювання управління транспортними засобами. Ця публікація виявилась надзвичайно корисною для розробки

центру інформаційних технологій Університету митної справи та фінансів, оскільки надала змогу ухвалити важливі рішення щодо архітектури веб-додатку, який повністю задовольняє потреби даного проекту. Проектування архітектури веб-застосунку було здійснено з урахуванням основних вимог до його функціональності, масштабованості та безпеки. Для забезпечення ефективності системи та швидкості обробки даних, було використано сучасні технології програмування та бази даних.

В роботі [3] було розглянуто реалізацію технології слабозв'язананих компонентів програмного забезпечення (ПЗ) на основі інверсії управління (IoC), тобто реалізацію механізму Dependency Injection (DI), з метою проєктування розподілених інформаційних систем покращеного захисту персональних даних. Також було розглянуто приклади кібератак, що дало змогу запобігти їх у веб-додатку.

Для проектування бази даних серверної частини веб-додатку центру інформаційних технологій університету було проаналізовано, як були спроектовані інші бази даних у існуючих роботах [4] – [5]. Розглянуто приклади проектування баз даних модуля студента у системі підтримки вивчення дисциплін та інтелектуальної системи діагностики захворювань.

1.2 Аналіз методів розробки веб-додатків

Веб-додатки стали необхідною складовою сучасного інтернет-середовища, і їх розробка вимагає використання ефективних та передових методів. У даному підрозділі було проведено аналіз різних методів розробки веб-додатків з метою визначення найбільш підходящого підходу для реалізації поставлених завдань.

Основні підходи розробки веб-додатків:

а) Традиційний підхід. Цей підхід до розробки веб-додатків базується на використанні таких технологій, як HTML, CSS та JavaScript. Цей підхід є добре відомим та широко використовуваним у веб-розробці, особливо на початкових етапах розвитку інтернету. За допомогою цих мов програмування створюється

фронтенд додатку, тобто та частина, яка відповідає за візуальне представлення і взаємодію з користувачем.

Для бекенду, тобто логічної та обчислювальної частини додатку, використовуються серверні технології, такі як PHP, Java або .NET. Ці мови програмування дозволяють взаємодіяти з базами даних, обробляти дані, забезпечувати безпеку та інші серверні операції.

Традиційна розробка веб-додатків має свої переваги та недоліки. До переваг можна віднести:

1) Доступність. Існує велика кількість розробників, які володіють цими технологіями, що робить їх широко використовуваними та дозволяє легко знайти команду для розробки проекту.

2) Відкритість. Технології, такі як HTML, CSS та JavaScript, є відкритими та стандартизованими, що сприяє переносимості додатків та їх сумісності з різними браузерами.

3) Широке використання. Традиційний підхід до розробки веб-додатків на базі HTML, CSS та JavaScript є добре відомим і широко використовується у галузі веб-розробки. Це означає, що знайдення програмістів та фахівців з веб-розробки для таких проектів є досить простим завдяки їх популярності та широкому поширенню.

4) Велика кількість ресурсів: Традиційна розробка веб-додатків має велику кількість документації, статей, підручників та онлайн-курсів, що допомагає розробникам здобути необхідні знання та навички. Також існує велика спільнота розробників, яка готова надати допомогу та підтримку у вирішенні проблем.

5) Гнучкість. Традиційна розробка дозволяє розробникам максимально контролювати кожен елемент веб-додатку. Вони можуть вільно налаштовувати розмітку, стилі та функціональність за своїми потребами. Це дає більшу свободу в реалізації задуманих ідей та дизайну.

Проте традиційна розробка також має свої недоліки:

1) Масштабованість. При збільшенні розміру проекту можуть виникати проблеми з масштабованістю. Це пов'язано зі зростанням складності коду, погіршенням структури додатку та збільшенням часу розробки і підтримки.

2) Підтримка. У традиційному підході може бути складно підтримувати та розширювати веб-додатки, особливо якщо вони розвиваються протягом тривалого часу або є багато розробників, які приєднуються до проекту.

3) Швидкість розробки. Традиційна розробка може бути часо- та працезатратною. Це пов'язано з необхідністю писати багато коду вручну та вирішувати деталі, що можуть забирати багато часу.

4) Оновлення і розгортання. Внесення змін та оновлення великих традиційних веб-додатків може бути складним завданням. Вони можуть вимагати перезавантаження сервера або зупинки додатку, що призводить до перерв у роботі користувачів.

б) Фреймворки на основі MVC (Model-View-Controller). Фреймворки на основі архітектурного шаблону Model-View-Controller (MVC) є популярним вибором для розробки веб-додатків. Цей шаблон включає в себе розділення додатку на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Це спрощує організацію та підтримку додатку, а також дозволяє використовувати перевикористовування коду та підтримувати розширюваність. Кожен з цих компонентів має свою роль та відповідальність, що сприяє чіткому розділенню логіки додатку:

1) Модель (Model). Модель відповідає за управління даними та бізнес-логікою додатку. Вона представляє собою схему даних, об'єкти, методи та функції, необхідні для роботи з даними. Модель забезпечує доступ до даних, їх збереження, зчитування, оновлення та видалення. Цей компонент відокремлений від інтерфейсу користувача та не залежить від способу їх відображення.

2) Представлення (View). Представлення відповідає за відображення даних та інтерфейсу користувача. Воно відображає дані, які надає модель, та забезпечує їх візуалізацію для користувача. Представлення може включати HTML-шаблони, CSS-стилі та інші компоненти, які відповідають за відображення даних у зручному та зрозумілому для користувача вигляді.

3) Контролер (Controller). Контролер відповідає за керування взаємодією між моделлю та представленням. Він обробляє вхідні дані, які надходять від користувача або зовнішніх джерел, та виконує відповідні дії. Контролер приймає запити від користувача, взаємодіє з моделлю для отримання або зміни даних, а потім передає оновлені дані до відповідного представлення для відображення оновленої інформації користувачу.

Компоненти моделі, представлення та контролера взаємодіють між собою за допомогою механізму сповіщень або подій. Наприклад, коли користувач взаємодіє з інтерфейсом користувача, контролер отримує цю дію та ініціює відповідні зміни в моделі. Після цього контролер оновлює відповідне представлення, щоб відобразити нові дані або стан додатку.

Фреймворки на основі MVC надають розробникам ряд переваг:

1) Розділення відповідальностей. Шаблон MVC дозволяє чітко розділити логіку додатку на окремі компоненти. Це полегшує розробку, тестування та підтримку додатку, оскільки кожен компонент виконує свою конкретну роль. Зміни в одному компоненті не впливають на інші, що дозволяє зберігати код додатку чистим і легко зрозумілим.

2) Підвищена повторна використовуваність. Завдяки розділенню логіки на модель, представлення та контролер, компоненти можуть бути використані повторно у різних частинах додатку або навіть в інших проектах. Це сприяє ефективному використанню ресурсів та прискорює розробку нових функціональностей.

3) Легкість у тестуванні. Кожен компонент моделі, представлення та контролера можна тестувати окремо, що спрощує процес виявлення та

виправлення помилок. Розробники можуть написати автоматизовані тести для перевірки правильності роботи кожного компонента та їх взаємодії.

4) Швидкість розробки. Використання фреймворків на основі MVC дозволяє прискорити процес розробки веб-додатків. Завдяки чіткому розділенню логіки та готовому набору інструментів, розробники можуть швидко створювати функціональні блоки та компоненти додатку. Багато частин загального функціоналу можуть бути вирішені за допомогою вже наявних рішень та модулів, що економить час розробки.

5) Розширюваність. Фреймворки на основі MVC надають зручні механізми для розширення та розвитку веб-додатків. Завдяки чіткому розділенню компонентів, розробники можуть додавати нові функції або змінювати існуючі без значних змін у загальній структурі. Це робить додаток більш масштабованим та гнучким, дозволяючи швидко реагувати на зміни вимог та впроваджувати нові можливості.

6) Стандартизація. Використання фреймворків на основі MVC сприяє стандартизації процесу розробки. Вони встановлюють загальні правила та практики, які розробники можуть дотримуватися. Це спрощує співпрацю в команді розробників, полегшує обмін кодом та сприяє підтримці та розумінню проекту в майбутньому.

Спільнота розробників. Фреймворки на основі MVC часто мають активну та велику спільноту розробників. Це означає, що розробники можуть швидко отримати допомогу, поради та рішення для своїх проблем. Спільнота також сприяє поширенню знань, обміну досвідом та розвитку фреймворку. Найпопулярнішими фреймворками на основі MVC є Ruby on Rails, Laravel, Django та ASP.NET MVC.

Незважаючи на багато переваг, фреймворки на основі архітектурного шаблону MVC також мають свої недоліки:

1) Складність. Використання архітектурного шаблону MVC може призвести до певної складності розробки, особливо для початківців. Розробникам потрібно мати глибоке розуміння всіх компонентів (моделі,

представлення та контролера) та взаємодії між ними, щоб правильно розподілити логіку та забезпечити взаємодію між компонентами.

2) Розмір коду. Застосування архітектурного шаблону MVC може призвести до збільшення розміру кодової бази. Кожен компонент має свою власну логіку та взаємодію з іншими компонентами, що може привести до зайвого дублювання коду або зайняти більше місця.

3) Складність управління станом. У деяких випадках, особливо в більших додатках, управління станом може стати складною задачею. При зростанні кількості компонентів та їх взаємодії може виникати проблема зі збереженням та оновленням стану додатку, що може призвести до складності в розумінні та налагодженні коду.

4) Перекриття функцій. У деяких випадках може виникати перекриття функцій між компонентами, особливо між контролером та представленням. Незрозумілість або неправильна організація взаємодії може призвести до неправильного функціонування додатку або змін в одному компоненті, що впливають на інші компоненти.

в) Клієнт-серверний підхід з використанням Vue.js та NestJS. Клієнт-серверний підхід є широко використовуваним у сучасних веб-додатках. Він передбачає розділення додатку на дві основні частини: клієнтську та серверну. Клієнт-серверний підхід зазвичай використовується для створення багатосторонніх веб-додатків, де клієнтська частина відповідає за взаємодію з користувачем та відображення даних, а серверна частина забезпечує обробку запитів, доступ до бази даних та бізнес-логіку.

Один з популярних підходів до реалізації клієнт-серверного підходу включає використання Vue.js на стороні клієнта та NestJS на стороні сервера.

Vue.js є прогресивним JavaScript-фреймворком, який надає багато переваг у розробці інтерфейсу користувача. Однією з головних переваг Vue.js є його компонентний підхід до розробки. Замість традиційного підходу до розробки, де інтерфейс користувача розбивається на різні елементи, Vue.js дозволяє створювати невеликі компоненти, які можна повторно

використовувати у різних частинах додатку. Це полегшує розробку, підтримку та масштабування коду, оскільки розробник може фокусуватись на створенні окремих компонентів, що робить робочий процес більш організованим та ефективним.

Крім того, Vue.js пропонує потужні механізми зв'язування даних та реактивного оновлення. Завдяки цим механізмам, зміни в даних автоматично відображаються на веб-сторінці без необхідності вручну оновлювати її. Це дозволяє створювати динамічні інтерфейси, де зміни в одному компоненті автоматично відображаються в інших компонентах, що значно полегшує роботу з даними та реагуванням на події.

NestJS, з іншого боку, є прогресивним фреймворком для створення масштабованих та ефективних серверних додатків на базі Node.js. Він використовує TypeScript як основну мову програмування та пропонує архітектурні принципи, взяті з фреймворка Angular. NestJS надає розробникам зручність в створенні розширюваних серверних додатків, підтримку різноманітних баз даних та роботу з HTTP-запитами.

Переваги клієнт-серверного підходу:

1) Розділення відповідальностей. Клієнт-серверний підхід дозволяє чітко розділити відповідальності між клієнтською та серверною частинами додатку. Vue.js використовується для реалізації клієнтського інтерфейсу та взаємодії з користувачем, тоді як NestJS забезпечує обробку запитів, доступ до бази даних та інші серверні операції. Це спрощує розробку, підтримку та масштабування додатку.

2) Масштабованість. Клієнт-серверний підхід дозволяє ефективно масштабувати додаток. Завдяки розділенню на клієнтську та серверну частини, можна гнучко масштабувати кожну з них окремо. Наприклад, можна встановити більше серверних ресурсів для обробки великого навантаження або використовувати CDN (Content Delivery Network) для розповсюдження статичного контенту з клієнтської частини.

3) Більша продуктивність. Використання Vue.js та NestJS дозволяє побудувати додаток з високою продуктивністю. Vue.js забезпечує швидке реагування інтерфейсу та швидке оновлення даних завдяки вбудованому системі реактивності. NestJS, зі свого боку, базується на ефективній архітектурі та використовує неблокуючий I/O, що дозволяє обробляти багато запитів одночасно без блокування потоку.

Недоліки клієнт-серверного підходу:

1) Складність в розробці. Розробка клієнт-серверних додатків може бути складнішою порівняно зі створенням простіших односторонніх додатків. Вимагається глибоке розуміння як клієнтської, так і серверної сторони, а також їх взаємодії. Це може призвести до підвищеної складності розробки та більшого часу на початкове налаштування.

2) Синхронізація даних. У клієнт-серверному підході потрібно забезпечити синхронізацію даних між клієнтом та сервером. Синхронізація даних є важливим аспектом в клієнт-серверному підході до розробки веб-додатків. Цей процес включає в себе ряд викликів та викликів до сервера для забезпечення актуальності та цілісності даних на обох сторонах. Це може включати обмін даними в реальному часі, вирішення конфліктів при одночасній модифікації даних, кешування та інші аспекти. Неправильне керування синхронізацією може призвести до втрати даних, некоректної взаємодії або низької продуктивності додатку.

3) Більший обсяг коду. Використання клієнт-серверного підходу зазвичай супроводжується збільшенням обсягу коду проекту.

Для створення даного веб-застосунку центру інформаційних технологій було використано саме клієнт-серверний метод розробки веб-додатків.

1.3 Висновки

З розвитком технологій інформаційно-комунікаційних систем, все більше установ освіти використовують різноманітні веб-додатки для підвищення якості та ефективності навчального процесу. Використання веб-

додатку для центру інформаційних технологій має бути корисним і ефективним рішенням.

Якщо брати до уваги те, що вед-додаток не виключає використання усіх інших методів комунікації, такі як пошта, або соціальні мережі – лише він має місце для розробки задля покращення обміну інформацією.

Переваги наявності веб-додатку для комунікації з викладачами та студентами включають зручність та ефективність у взаємодії, відсутність обмежень в часі та місці, можливість спільної роботи та обміну інформацією в режимі реального часу.

Також веб-додаток дозволяє викладачам та адміністраторам швидко та зручно оновлювати інформацію на сайті та публікувати новини та оголошення для всіх користувачів. Крім того, веб-додаток забезпечує збереження всієї інформації в одному місці, що сприяє зручності та організованості роботи.

Отже, розробка веб-додатку для центру інформаційних технологій Університету митної справи та фінансів є важливим кроком для покращення навчального процесу, сприяє зручності та ефективності взаємодії між студентами, викладачами та адміністрацією університету.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ

2.1 Вибір програмних засобів для реалізації проекту

Клієнтська частина:

1. JavaScript/TypeScript. Для розробки клієнтської частини була використана мова програмування JavaScript та її розширення TypeScript. JavaScript є широко використовуваною мовою для розробки веб-додатків, а TypeScript додає до неї можливості статичного типування, що поліпшує безпеку та підтримку коду.

2. SCSS/SASS. Для розробки стилів і оформлення веб-додатку була використана препроцесорна мова SCSS або її розширення SASS. SCSS/SASS дозволяють писати стилізований код з використанням змінних, вкладених стилів та інших зручних функцій, що спрощує розробку та підтримку CSS.

3. Ant Design. Для швидкого та стильного розроблення інтерфейсу користувача була використана компонентна бібліотека Ant Design. Ant Design надає готові компоненти та стилі, що допомагають створювати красивий та сучасний інтерфейс з легкістю.

4. Vue.js. Для реалізації динамічної інтерактивності та компонентного підходу до клієнтської частини було використано фреймворк Vue.js. Vue.js дозволяє легко створювати реактивні компоненти, управляти станом додатку та ефективно взаємодіяти з серверною частиною.

5. NPM. Для управління залежностями та пакетами був використаний менеджер пакетів NPM (Node Package Manager). NPM дозволяє швидко та зручно встановлювати, оновлювати та керувати пакетами, необхідними для розробки проекту.

6. VS Code. Для написання коду та розробки проекту був використаний текстовий редактор VS Code (Visual Studio Code). VS Code є потужним та розширюваним інструментом, який надає зручний інтерфейс розробки, автодоповнення коду, відладку, інструменти контролю версій та інші корисні функції. Він підтримує широкий спектр розширень, що дозволяє налаштувати його під власні потреби розробки.

Серверна частина:

1. Node.js. Для реалізації серверної частини проекту було використано середовище виконання JavaScript — Node.js. Node.js дозволяє виконувати JavaScript на сервері, що забезпечує ефективну обробку запитів, взаємодію з базою даних та реалізацію серверної логіки.

2. NestJS. Для побудови серверної архітектури та реалізації API був використаний фреймворк NestJS. NestJS пропонує модульну структуру, засновану на паттерні MVC (Model-View-Controller), що сприяє структурованому та організованому розробленню серверної частини. Він надає гнучкість вибору бази даних, підтримує валідацію даних, роутінг, автентифікацію та інші корисні функції.

3. SQL. Для роботи з базою даних SQLite в серверній частині проекту була використана мова структурованих запитів SQL (Structured Query Language). SQL є стандартною мовою для взаємодії з реляційними базами даних. З його допомогою можна створювати, змінювати та видаляти дані, виконувати складні запити та керувати структурою бази даних.

4. SQLite. Для збереження та управління даними була обрана система управління базами даних SQLite. SQLite є легким та простим у використанні рішенням, що забезпечує зручне збереження даних в локальній базі. Він підтримує мову SQL та має низькі вимоги до ресурсів, що робить його популярним в розробці малих та середніх проектів.

2.2 Засоби для розробки клієнтської частини

2.2.1 JavaScript/TypeScript

Мови програмування JavaScript і TypeScript були використані для написання динамічної логіки та функціоналу веб-додатку.

JavaScript – це динамічна мова комп'ютерного програмування. Він легкий і найчастіше використовується як частина веб-сторінок, реалізація яких дозволяє взаємодіяти з користувачем і створювати динамічні сторінки. Це мова програмування, що інтерпретується, з об'єктно-орієнтованими можливостями.

Спочатку JavaScript був відомий як LiveScript, але Netscape змінила свою назву на JavaScript, можливо через хвилювання, викликаного Java. JavaScript вперше з'явився у Netscape 2.0 у 1995 році під назвою LiveScript. Універсальне ядро мови було вбудоване у Netscape, Internet Explorer та інші веб-браузери.

Клієнтський JavaScript – найпоширеніша форма цієї мови. Сценарій повинен бути включений до HTML-документа або на нього має бути посилання, щоб код міг бути інтерпретований браузером. У нашому випадку JavaScript буде маніпулювати не HTML об’єктами а JXL.

Це означає, що веб-сторінка не обов'язково має бути статичним JXL, але може включати програми, які взаємодіють із користувачем, керують мобільним додатком та динамічно створюють JXL -контент.

Клієнтський механізм JavaScript забезпечує багато переваг у порівнянні з традиційними серверними сценаріями CGI. Наприклад, можна використовувати JavaScript, щоб перевірити, чи правильно введено користувачем дійсну адресу електронної пошти в полі форми.

Код JavaScript виконується, коли користувач надсилає форму, і якщо всі записи дійсні, вони будуть відправлені на веб-сервер.

JavaScript можна використовувати для перехоплення ініційованих користувачем подій, таких як натискання кнопок, навігація за посиланнями та інші дії, які ініціює користувач явно або неявно.

JavaScript вважається легким через те, що він мало використовує ЦП, простий у реалізації та має мінімалістичний синтаксис. Мінімалістичний синтаксис, наприклад, немає типів даних. Тут усе сприймається як об'єкт. Його дуже легко освоїти, оскільки синтаксис схожий на C++ і Java.

JavaScript не споживає багато ресурсів процесора, тому чудово підходить для написання мобільних додатків. Це не створює надмірного навантаження на процесор або оперативну пам'ять. JavaScript працює в браузері, незважаючи на те, що він має складні парадигми та логіку, що означає, що він використовує менше ресурсів, ніж інші мови. Наприклад,

NodeJS, варіант JavaScript, не тільки виконує більш швидкі обчислення, але й використовує менше ресурсів, ніж аналоги, такі як Dart або Java.

Крім того, в порівнянні з іншими мовами програмування, він має менше вбудованих бібліотек або фреймворків, що є ще однією причиною його легкості. Однак це має недолік, який полягає в тому, що нам необхідно включати зовнішні бібліотеки та фреймворки.

JavaScript одночасно компілюється та інтерпретується. У більш ранніх версіях JavaScript використовувався тільки інтерпретатор, який виконував код швидко і невчасно відображав результат. Но з часом продуктивність стала проблемою, так як інтерпретація досить повільна. Тому в нових версіях JS, можливо, після V8 також був включений JIT-компілятор для оптимізації виконання та більш швидкого відображення результатів. Цей JIT-компілятор генерує байт-код, відносно легкого кодування. Цей байт-код являє собою набір високо оптимізованих інструкцій.

V8 спершу використовує інтерпретатор для інтерпретації коду. При подальшому виконанні механізму V8 працює шаблони, такі як часто виконувані функції, часто використовувані змінні, і компілює їх для підвищення ефективності.

TypeScript — це мова програмування, яка позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript. TypeScript є розширенням JavaScript, яке додає можливість типізації та поліпшену розробку коду. Він є зворотньо сумісним із JavaScript. Скомпільовану програму на TypeScript можна виконувати в будь-якому сучасному веб-браузері.

2.2.2 SCSS/SASS

SASS — скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS). SASS призначений для підвищення рівня абстракції коду та спрощення файлів CSS. SASS має два синтаксиси:

1. SASS (оригінальний) — відрізняється відсутністю фігурних дужок, в ньому вкладені елементи реалізовані за допомогою відступів, а правила відокремлюються переведенням рядка;

2. SCSS (новий) — використовує фігурні дужки (подібно до CSS).

Основні особливості SCSS включають:

1. Змінні. SCSS дозволяє оголошувати та використовувати змінні. Це дозволяє зручно зберігати значення кольорів, розмірів шрифтів, відступів та інших параметрів. Змінні можна легко змінювати на всьому проекті, що полегшує зміну стилів.

2. Міксіни. Міксіни в SCSS це набір стилевих правил, які можна використовувати повторно. Їх можна використовувати для задання часто вживаних стилів, таких як заокруглення куточків, тіні, анімація та багато іншого. Міксіни спрощують процес розробки та зменшують дублювання коду.

3. Вкладеність. SCSS дозволяє вкладати стилі один в одного. Це дозволяє створювати більш організований та зрозумілий код. Вкладеність також спрощує зміну стилів, оскільки вони автоматично оновлюються в усіх вкладених елементах.

4. Імпорти. SCSS дозволяє розбити стилі на окремі файли і імпортувати їх до головного файлу стилів. Це полегшує організацію та управління стилями, особливо в великих проектах.

5. Підтримка CSS. SCSS підтримує всі основні функції та можливості CSS. Можна використовувати стандартні CSS-правила, псевдокласи, медіа-запити тощо.

2.2.3 Ant Design

Ant Design є компонентною бібліотекою, яка надає готові компоненти та стилі для розробки інтерфейсу користувача. Використання Ant Design спрощує процес створення стильного та сучасного інтерфейсу, оскільки вона пропонує широкий вибір готових компонентів, які можна легко налаштувати та використовувати.

Особливості та переваги Ant Design:

1. Готові компоненти. Ant Design надає багато готових компонентів, таких як кнопки, форми, таблиці, меню, модальні вікна, картки та багато іншого. Ці компоненти розроблені з дотриманням сучасних дизайнерських тенденцій та забезпечують стандартизований та однорідний вигляд і поведінку інтерфейсу.

2. Кастомізація. Ant Design дозволяє легко налаштувати зовнішній вигляд компонентів. Завдяки можливості змінювати кольори, типографіку, розміри та інші параметри, можна легко адаптувати компоненти до власного стилю та вимог проекту.

3. Система сітки. Ant Design надає потужну систему сітки, яка допомагає створювати респонсивні (адаптивні) інтерфейси, які гнучко пристосовуються до різних пристроїв і розмірів екранів.

4. Документація та підтримка. Ant Design має детальну документацію, яка пояснює використання компонентів та надає приклади їх використання. Крім того, спільнота користувачів Ant Design є активною та підтримується розробниками, що дозволяє отримувати допомогу та поради при роботі з бібліотекою.

2.2.4 Vue.js

Vue.js є прогресивним веб-фреймворком, який спрощує розробку клієнтської частини веб-додатків. Він зосереджується на створенні інтерфейсу користувача та забезпечує ефективну організацію коду та взаємодію з даними.

Основні особливості та переваги Vue.js:

1. Компонентна архітектура. Vue.js пропонує компонентну архітектуру, яка дозволяє розбити інтерфейс на незалежні компоненти. Компоненти можна повторно використовувати, що спрощує розробку та підтримку коду. Крім того, компоненти дозволяють легко розподіляти завдання між різними членами команди розробників.

2. Двостороннє зв'язування даних. Vue.js надає можливість зв'язувати дані між компонентами та шаблонами. Це означає, що зміна даних

в одному компоненті автоматично оновлює відображення даних в інших компонентах, що робить реактивну розробку більш простою та продуктивною.

3. Директиви. Vue.js має потужну систему директив, які дозволяють змінювати поведінку HTML-елементів. Директиви дозволяють вставляти умови, обробники подій, анімацію та інші функції безпосередньо в шаблоні, що полегшує контроль над інтерфейсом та взаємодією з користувачем.

4. Роутінг (маршрутизація). Vue.js має вбудовану систему маршрутизації (Vue Router), яка дозволяє створювати односторінкові додатки зі змінними URL-адресами. Це дозволяє створювати дружні до SEO додатки з розширеною навігацією та структурою сторінок. За допомогою Vue Router можна визначати шляхи (routes) і пов'язані з ними компоненти. При зміні URL-адреси Vue Router автоматично відображає відповідний компонент, що дозволяє створювати багатосторінкові додатки з багаторівневою навігацією.

2.2.5 NPM

NPM – найбільший у світі реєстр програмного забезпечення. Розробники з відкритим вихідним кодом з усіх континентів використовують npm для спільного використання та запозичення пакетів, і багато організацій також використовують npm для управління приватною розробкою.

NPM складається з трьох окремих компонентів:

1. Вебсайт
2. Інтерфейс-командного рядка (CLI)
3. Реєстр

Веб-сайт використовують для пошуку пакетів, налаштування профілів та управління іншими аспектами роботи з npm. Наприклад, користувач може налаштувати організації для керування доступом до загальнодоступних або приватних пакетів.

CLI запускається з терміналу, і саме так більшість розробників взаємодіють із NPM.

Реєстр є великою загальнодоступною базою даних програмного забезпечення JavaScript та метаінформації, що оточує його.

NPM використовують для:

1. Адаптування пакетів коду програм або включення пакетів як є.
2. Завантаження автономних інструментів, які можна використовувати зараз.
3. Запуску пакетів без завантаження за допомогою npx.
4. Передачі коду будь-якому користувачу NPM у будь-якому місці.
5. Обмежування використання коду конкретними розробниками.
6. Створення організацій для координації обслуговування пакетів, написання коду та розробників.
7. Управління кількома версіями коду та залежностями коду.
8. Оновлення програми під час оновлення базового коду.
9. Знаходження інших розробників, які працюють над схожими проблемами та проектами.

2.2.6 VS Code

Microsoft Visual Studio (VS) Code — це безкоштовний редактор коду, який можна використовувати для написання коду будь-якою мовою програмування без необхідності перемикання редакторів.

Одна з найкращих особливостей VS Code у тому, що він абсолютно безкоштовний. Все, що потрібно зробити, це обрати версію для операційної системи комп'ютера, завантажити та встановити редактор.

VS Code підтримує операційні системи Windows, Linux та Mac OS. Існують також збірки Insiders для перших користувачів, які оновлюються майже щодня.

VS Code — це частина Microsoft Visual Studio, яка є повним інтегрованим середовищем розробки (IDE). У нього є версія спільноти (community version), яка також безкоштовна, але також має платні підписки, які включають доступ до більш сучасних інструментів розробки та тестування, підтримку, навчання та Microsoft Azure (відкривається в новій вкладці) для створення веб-додатків.

Більшість редакторів коду мають підсвічування синтаксису, але VS Code також має IntelliSense, який розширює можливості автодоповнення коду, підказок за кодом та інформації про параметри. Це означає, що в міру того, як ви друкуєте, програма буде відображати контекстне меню з різними параметрами, які допоможуть вам не друкувати.

З VS Code легко працювати із системами контролю версій, такими як Git. Ви зможете розміщувати файли та робити комміти, а потім відправляти та виймати зміни до вибраного вами віддаленого репозиторію коду прямо з редактора.

Інтерфейс VS Code також включає ще одну панель інструментів у крайньому лівому кутку з кнопками для зміни виду. За промовчанням користувач буде в режимі провідника для роботи з файлами та папками, але є також поля для пошуку, керування версіями, запуску та налагодження та розширень. Натискання будь-якої з кнопок для вже включеного представлення призведе до повного закриття панелі, щоб збільшити область кодування.

Як і в більшості хороших редакторів коду, однією з найсильніших сторін VS Code є його налаштування. За допомогою так званих розширень можна змінити тему, додати нові мови та налагоджувачі, а також підключитися до різних служб.

2.3 Засоби для розробки серверної частини

2.3.1 Node.js

Node.js – це кросплатформове середовище виконання JavaScript з відкритим вихідним кодом. Це найпопулярніший інструмент практично для будь-якого проекту.

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

Програма Node.js виконується в одному процесі без створення нового потоку для кожного запиту. Node.js надає набір примітивів асинхронного введення-виводу у своїй стандартній бібліотеці, які запобігають блокуванню

JavaScript, і, як правило, бібліотеки в Node.js написані з використанням неблокуючих парадигм, що робить поведінку блокування швидше винятком, ніж нормою.

Коли Node.js виконує операцію введення-виводу, наприклад, читання з мережі, доступ до бази даних або файлової системи, замість блокувати потік і витрачати цикли ЦП на очікування, Node.js відновлює операції, коли повертається відповідь.

Це дозволяє Node.js обробляти тисячі одночасних підключень до одного серверу, не обтяжуючи себе керуванням паралельними потоками, що може бути значним джерелом помилок.

Node.js має унікальну перевагу, тому що мільйони розробників інтерфейсів, які пишуть JavaScript для браузера, тепер можуть писати код на стороні сервера на додаток до коду на стороні клієнта без необхідності вивчення зовсім іншої мови.

У Node.js нові стандарти ECMAScript можна використовувати без проблем, тому що вам не потрібно чекати, поки всі ваші користувачі оновлять свої браузери - ви самі вирішуєте, яку версію ECMAScript використовувати, змінюючи версію Node.js, і ви також можете увімкнути певні експериментальні функції запустивши Node.js з прапорами.

2.3.2 NestJS

NestJS є фреймворком для створення масштабованих та ефективних серверних додатків на платформі Node.js. Він базується на концепції архітектури MVC (Model-View-Controller) та пропонує розширення для роботи з TypeScript.

NestJS надає зручні інструменти для створення модульної та організованої структури проекту. Він підтримує використання декораторів для опису різних елементів додатку, таких як контролери, провайдери, маршрутизатори та фільтри. Це дозволяє розбити додаток на незалежні модулі та забезпечити чистоту коду.

Одним з ключових переваг NestJS є вбудована система впровадження залежностей (Dependency Injection). Вона дозволяє зручно керувати залежностями між компонентами додатку та спрощує процес їх створення та використання. Це покращує розширюваність та тестування додатків.

Нижче зображено взаємодію модулів у NestJS (див. рис. 2.1).

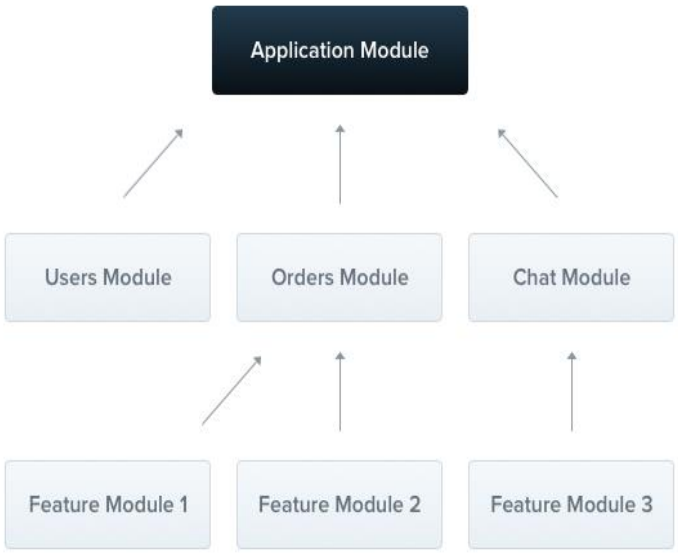


Рисунок 2.1 – Взаємодія модулів у NestJS

NestJS також надає потужні функції для обробки HTTP-запитів, роботи з маршрутами, валідації даних, авторизації та інших важливих аспектів розробки серверних додатків. Він підтримує різні шаблони проектів та дозволяє легко інтегруватися з іншими популярними бібліотеками та фреймворками.

Завдяки своїй гнучкості та розширюваності, NestJS є потужним інструментом для розробки серверної частини проекту. Він допомагає побудувати добре структурований, ефективний та легко супроводжуваний код, що відповідає сучасним стандартам розробки серверних додатків.

NestJS є відмінним вибором для розробки серверних додатків, оскільки він надає багато інших корисних функцій та особливостей:

1. Модульність. NestJS сприяє побудові додатків зі структурованим підходом до модулів. Це дозволяє розділити функціональність додатку на незалежні модулі, що спрощує управління та розширення проекту.

2. Middleware (проміжне програмне забезпечення). NestJS підтримує використання middleware, що дозволяє обробляти запити перед тим, як вони досягнуть контролера. Це важливий механізм для виконання певних операцій, таких як логування, авторизація, обробка помилок та інші.

3. WebSocket. NestJS надає підтримку реалізації веб-сокетів за допомогою WebSocket шлюзу. Це дозволяє створювати багатокористувацькі додатки в режимі реального часу з двостороннім зв'язком між клієнтом і сервером.

4. Тестування. NestJS підтримує тестування додатків за допомогою вбудованих інструментів тестування та фреймворку Jest. Це дозволяє легко створювати автоматизовані тести для перевірки працездатності та коректності серверного коду.

5. Конфігурація. NestJS пропонує механізм конфігурації, що дозволяє зручно налаштовувати параметри додатку. Це дозволяє легко використовувати різні середовища (наприклад, розробка, тестування, продакшн) з різними налаштуваннями.

6. Сумісність з існуючим кодом. NestJS дозволяє інтегруватися з існуючим JavaScript-кодом, що є великою перевагою для команд, які вже мають проекти на Node.js.

2.3.3 SQL

SQL (Structured Query Language) є мовою запитів, яка використовується для роботи з реляційними базами даних. В серверній частині проекту, зокрема з використанням NestJS та SQLite, SQL використовується для взаємодії з базою даних.

Спочатку SQL був основним способом роботи користувача з базою даних і давав змогу виконувати такий набір операцій:

1. Створення в базі даних нової таблиці.
2. Додавання в таблицю нових записів.
3. Зміна записів.
4. Видалення записів.
5. Вибірка записів з однієї або декількох таблиць (відповідно до заданої умови).
6. Зміна структур таблиць.

Згодом SQL ускладнився – збагатився новими конструкціями, забезпечив можливість опису та управління новими збереженими об'єктами (наприклад, індекси, уявлення, тригери і збережені процедури) - і став набувати рис, властивих мовам програмування.

Мова SQL являє собою сукупність операторів, інструкцій, обчислювальних функцій.

Оператори SQL поділяються на:

1. оператори визначення даних (Data Definition Language, DDL).
2. оператори маніпуляції даними (Data Manipulation Language, DML).
3. оператори визначення доступу до даних (Data Control Language, DCL).
4. оператори управління транзакціями (Transaction Control Language, TCL).

Використання SQL дозволяє розробникам створювати потужні запити, які можуть отримувати, оновлювати та видаляти дані з бази даних згідно з потребами проекту. SQL дозволяє зручно організовувати дані, створювати зв'язки між таблицями, виконувати складні пошукові запити та агрегації, забезпечувати цілісність та безпеку даних.

2.3.4 SQLite

SQLite – це легка та вбудована СУБД (система управління базами даних), яка зберігає дані в локальних файлах без необхідності використання окремого сервера баз даних. Вона використовується в серверній частині проекту для зберігання, організації та доступу до даних.

Слово «вбудований» означає, що SQLite не використовує парадигми клієнт-сервер, тобто рушій SQLite не є окремим процесом, що працює окремо і з яким взаємодіє програма, а являє собою бібліотеку, з якою програма компонується, і рушій стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включно з визначеннями, таблицями, індексами і даними) в єдиному стандартному файлі на тому комп'ютері, на якому виконується програма. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції запису весь файл, що зберігає базу даних, блокується.

Основні особливості SQLite:

1. Вбудована СУБД. SQLite включена безпосередньо в додаток, що дозволяє працювати з базою даних без необхідності налаштування та підтримки окремого сервера.
2. Легкість використання. SQLite проста у використанні і не потребує значних зусиль для налаштування та налагодження. Вона має простий інтерфейс командної строки та багато доступних бібліотек для різних мов програмування.
3. Підтримка стандартного SQL. SQLite підтримує велику частину стандарту SQL, включаючи створення таблиць, вибірку, вставку, оновлення та видалення даних, операції з'єднання, групування та багато іншого.
4. Ефективність. SQLite працює швидко та ефективно, особливо для менших та середніх обсягів даних.

5. Надійність. SQLite гарантує цілісність даних та відновлення після збоїв. Вона підтримує транзакції ACID (атомарність, консистентність, ізольованість, довіреність) для забезпечення цілісності та безпеки даних.

2.4 Аналіз розробки дизайну веб-додатку

Аналіз розробки дизайну веб-додатку є важливою частиною процесу створення веб-додатків, оскільки він визначає зовнішній вигляд і користувацький досвід додатку. У цьому розділі буде проведений аналіз розробки дизайну веб-додатку з акцентом на вибір фреймворка Ant Design та використання альтернативних інструментів для розробки дизайну. Весь дизайн проєкту було розроблено використовуючи спеціальну компонентну бібліотеку для користувацьких інтерфейсів – Ant Design.

Ant Design є одним з популярних фреймворків для розробки дизайну веб-додатків. Одна з головних причин вибору Ant Design полягає у його широкому спектрі готових компонентів, які забезпечують швидку та зручну розробку дизайну веб-додатку. Даний фреймворк має багатофункціональні та досить добре стилізовані компоненти, що дозволяють розробникам ефективно будувати користувацький інтерфейс. Одна з переваг Ant Design полягає у його модульності. Він дозволяє використовувати лише ті компоненти, які потрібні для конкретного проекту, що полегшує розробку та підтримку.

Також, Ant Design надає можливість налаштування компонентів за допомогою тем, стилів і властивостей, що дозволяє адаптувати дизайн до індивідуальних потреб проекту.

Дизайн, який був розроблений з використанням Ant Design, має сучасний вигляд і відповідає актуальним тенденціям у дизайні веб-додатків. Фреймворк забезпечує чистий та естетичний дизайн, що позитивно впливає на користувацький досвід та враження від веб-додатку. Крім цього, Ant Design підтримує розробку респонсивного дизайну, що дозволяє оптимізувати веб-додаток для різних пристроїв і розмірів екранів. Це особливо важливо в сучасному світі, де веб-додатки доступні на різних пристроях.

Адаптивний дизайн, забезпечений Ant Design, дозволяє веб-додатку центру інформаційних технологій гнучко реагувати на зміни розміру екрану, забезпечуючи зручне та зручне відображення контенту незалежно від пристрою, на якому він відображається. Це забезпечує однаково хороший користувацький досвід незалежно від того, чи використовує користувач ноутбук, планшет або смартфон.

Крім розробки респонсивного (адаптивного) дизайну, Ant Design також надає широкий спектр інструментів та ресурсів для стилізації та налаштування дизайну веб-додатку. За допомогою Ant Design, розробники можуть легко змінювати кольори, шрифти, розміри елементів і властивості компонентів, щоб відповідати вимогам проекту та створювати унікальний дизайн веб-додатку.

Однак, важливо враховувати, що вибір фреймворка Ant Design не є обов'язковим і може бути залежний від конкретних вимог і особливостей проекту. Існують інші популярні фреймворки та інструменти для розробки дизайну веб-додатку, такі як Bootstrap, Material UI, Semantic UI, які також мають свої переваги і можуть бути вибрані залежно від потреб проекту.

Хоча фреймворк Ant Design має багато переваг, варто враховувати його потенційні недоліки:

1. Великий розмір. Ant Design має досить великий розмір з урахуванням всіх компонентів і стилів. Це може призвести до тривалого завантаження сторінок, особливо на мобільних пристроях з обмеженими швидкостями інтернет-з'єднання. Розмір фреймворка може вплинути на продуктивність веб-додатку, особливо якщо використовуються лише деякі компоненти.

2. Навчання та вирішення проблем. Ant Design має свою власну документацію та специфіку використання. Навчання розробників, як працювати з фреймворком і вирішувати можливі проблеми, може зайняти деякий час. Крім того, якщо виникають проблеми або помилки, їх вирішення може бути складнішим через специфіку фреймворка.

3. Залежність від React. Ant Design побудований на основі бібліотеки React, тому використання цього фреймворку вимагає наявності або вивчення React. Це означає, що для розробки з використанням Ant Design необхідно мати знання та досвід у роботі з React. Для команд, що вже використовують інші бібліотеки або фреймворки, перехід на Ant Design може бути зв'язаним з великими зусиллями по переконвертації існуючого коду або навчання нових інструментів.

Було обрано саме Ant Design, тому що він більше підходить для веб-додатку центру інформаційних технологій, оскільки має відповідну колірну палітру, а також дизайн компонентів, що добре підходить під університетську тематику.

2.5 Висновки

В результаті аналізу розробки веб-додатку були вибрані відповідні технології для різних аспектів проекту.

У клієнтській частині веб-додатку було використано Vue.js, що є потужним фреймворком JavaScript для розробки користувацького інтерфейсу. Вибір Vue.js обумовлений його простотою використання, швидкістю розробки та активною спільнотою, яка надає постійну підтримку та оновлення.

Серверна частина веб-додатку була розроблена за допомогою фреймворка NestJS, який базується на мові програмування Node.js. NestJS надає потужні можливості для створення розширюваних та масштабованих серверних додатків. Вибір NestJS обумовлений його архітектурою, яка сприяє швидкій розробці та забезпечує добру організацію коду. Активна спільнота розробників надає підтримку та допомогу при роботі з фреймворком.

Для розробки користувацького інтерфейсу була обрана компонентна бібліотека Ant Design, яка пропонує велику кількість готових компонентів і підтримує розробку респонсивного дизайну. Вибір Ant Design обумовлений його універсальністю, простотою використання та активною спільнотою розробників.

Використання цих технологій має значний позитивний вплив на розробку даного веб-застосунку. Активна спільнота розробників забезпечує підтримку, оновлення і надання прикладів коду, що сприяє швидкому розвитку та розв'язанню проблем. Дизайн веб-додатку був розроблений з урахуванням всіх вимог і потреб, що дозволяє створити привабливий та користувацький зручний інтерфейс для користувачів. Враховуючи вимоги центру інформаційних технологій, дизайн був розроблений з врахуванням корпоративного стилю, що забезпечує єдність та впізнаваність веб-додатку.

РОЗДІЛ 3 РОЗРОБКА ВЕБ-ДОДАТКУ ЦЕНТРУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

3.1 Розробка загальної архітектури проекту

Розробка загальної архітектури проєкту була важливим етапом у рамках цього дослідження, спрямованого на розробку веб-додатку центру інформаційних технологій університету. Веб-додаток було спроектовано з використанням сучасних технологій та підходів для забезпечення масштабування, гнучкості та ефективності.

Загальна архітектура проекту була побудована на основі принципів клієнт-серверної архітектури (див. рис. 3.1) та розділення відповідальностей між backend (серверна) та frontend (клієнтська) частинами додатку. Дане рішення дозволило досягти високої прогнозованості, розширюваності та підтримуваності проекту.

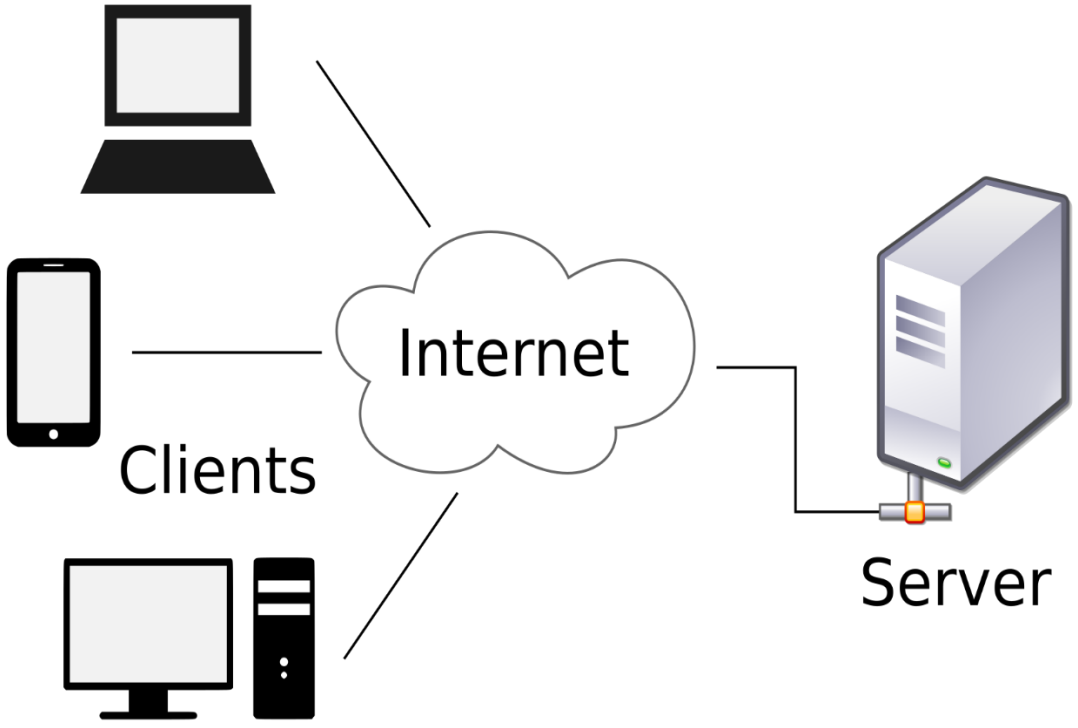


Рисунок 3.1 – Клієнт-серверна архітектура

Набір технологій, вибраний для реалізації даного проекту, включав у себе наступні компоненти: SCSS, JavaScript, TypeScript, NestJS (серверна частина), Vue.js (клієнтська частина), SQLite (вбудована реляційна система керування базами даних).

У backend (серверній) частині проекту був використаний фреймворк NestJS, який базується на програмній платформі Node.js та надає ефективний механізм для створення масштабованих та модульних додатків. Фреймворк – це програмне середовище, яке спрощує та прискорює створення програмного забезпечення. Розробка backend складалася з наступних компонентів:

1. Контролери (controllers), які відповідають за обробку запитів від клієнтів та передачу даних до сервісів.
2. Сервіси (services), які містять бізнес-логіку додатку та забезпечують взаємодію з базою даних та іншими сервісами.
3. Моделі (models), що відповідають за структуру та схему даних, використовуваних у додатку.

У якості СУБД (системи управління базами даних) було обрано SQLite з декількох причин:

1. Легкість використання. SQLite є простою і легкою у використанні СУБД. Вона не вимагає окремого серверу чи налаштувань, що спрощує її налаштування і розгортання. Вона інтегрована безпосередньо в додаток і працює на більшості платформ, включаючи мобільні пристрої і настільні комп'ютери.
2. Портативність. База даних SQLite зберігається в одному файлі, що дозволяє легко переносити базу даних між різними середовищами. Це особливо корисно для веб-додатків, які можуть бути розгорнуті на різних серверах або кластерах.
3. Надійність та стабільність. SQLite має доказану надійність та стабільність. Вона має механізми для запобігання втрати даних при

випадковому відключенні або збоях в роботі. Крім того, SQLite має високу швидкодію, що дозволяє ефективно виконувати запити до бази даних.

4. Підтримка. SQLite є відкритим джерелом і має активну спільноту розробників. Вона постійно оновлюється та вдосконалюється, що забезпечує підтримку нових функцій та виправлення помилок.

5. Ефективність. SQLite має невеликий обсяг пам'яті та низьку витрату ресурсів системи, що робить її ефективним використанням для невеликих та середніх проектів. Вона добре працює зі звичайними операціями бази даних: створення таблиць, додавання, оновлення та видалення записів.

У frontend (клієнтській) частині проекту був використаний фреймворк Vue.js, який надає потужні інструменти для розробки інтерфейсу користувача. Компонентний підхід фреймворку Vue.js (див. рис. 3.2) дозволяє розділити інтерфейс на незалежні компоненти, які можна повторно використовувати та дуже легко керувати ними.

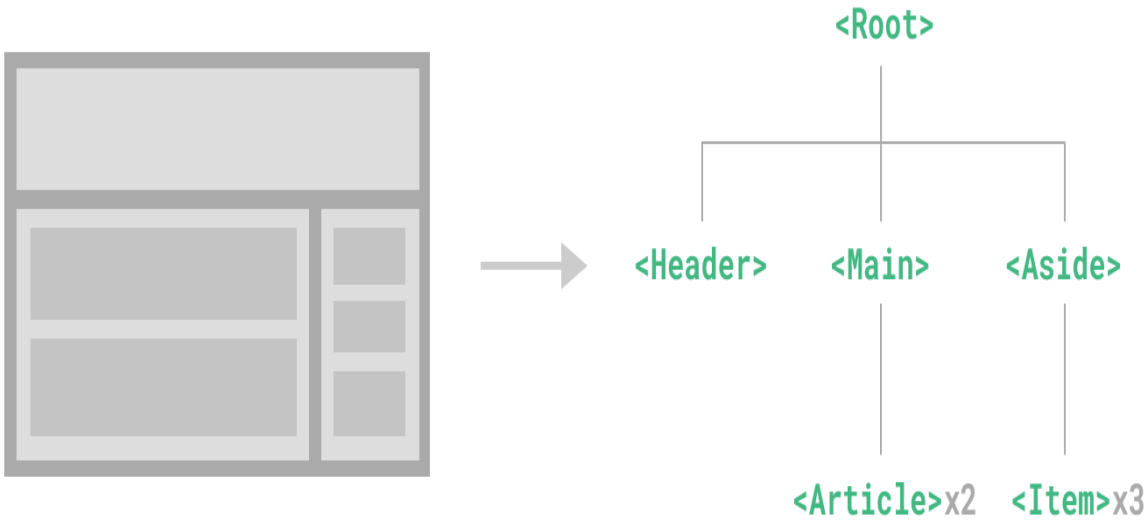


Рисунок 3.2 – Компонентний підхід фреймворку Vue.js

Архітектура frontend (клієнтської) частини проекту включала наступні складові:

1. Компоненти. Було створено набір компонентів, таких як кнопки, форми, таблиці тощо. Кожен компонент мав свою внутрішню логіку та представлення, що сприяло зручності розробки та підтримки.

2. Маршрутизація. Використання Vue Router дозволило налаштувати перехід між сторінками додатку та забезпечило коректну адресацію веб-додатку. Кожна сторінка була пов'язана з відповідним компонентом, що дозволяло легко керувати навігацією та взаємодією з користувачем.

3. Стан додатку. Для керування станом додатку було використано Pinia - становий менеджер для Vue.js. Це дозволило зберігати та оновлювати дані додатку в одному центральному місці та полегшити взаємодію між компонентами.

4. Взаємодія з backend (сервером). Для забезпечення комунікації між frontend та backend частинами додатку було використано API-виклики з використанням бібліотеки Axios. Це дозволило отримувати та відправляти дані між клієнтом та сервером за допомогою стандартних HTTP-запитів.

5. Стилізація. Для стилізації інтерфейсу було використано CSS-препроцесор SCSS, що дозволяло ефективно керувати стилями та забезпечувати легкість підтримки та розширення дизайну.

Ця архітектура проекту сприяла створенню ефективного та функціонального веб-додатку з покращеним користувацьким досвідом.

3.2 Розробка дизайну проекту

При розробці дизайну веб-додатку центру інформаційних технологій університету була врахована мета створити зручний, привабливий та інтуїтивно зрозумілий інтерфейс користувача. Розробка дизайну була проведена з урахуванням сучасних тенденцій у веб-дизайні, а також з дотриманням стилю університету.

Основні етапи розробки дизайну включали наступні кроки:

1. Аналіз вимог. Перед початком процесу розробки було проведено детальний аналіз вимог до інтерфейсу користувача. Було визначено основні

функціональні вимоги, потреби користувачів, а також враховано бізнес-цілі проекту. Цей аналіз допоміг зрозуміти, як краще пристосувати дизайн до потреб і очікувань цільової аудиторії.

2. Створення концепції. На основі аналізу вимог та розуміння потреб користувачів була розроблена концепція дизайну. Це включало визначення кольорової палітри, вибір шрифтів, створення макетів інтерфейсу та вивчення сучасних трендів у веб-дизайні. Концепція була затверджена та стала основою для подальшої розробки.

3. Створення макетів. На основі затвердженої концепції були розроблені детальні макети інтерфейсу. Були визначені компоненти, їх розташування на сторінцях та стиль оформлення. Макети були створені з урахуванням принципів зручності використання, доступності та естетики.

4. Взаємодія та анімація. Для надання веб-додатку центру інформаційних технологій привабливого та динамічного вигляду була розроблена система взаємодії та анімації. Це включало визначення ефектів при наведенні курсору, анімаційних переходів між сторінками та компонентами, а також динамічні елементи, які покращували взаємодію та залучення користувача.

5. Доступність та респонсивний (адаптивний) дизайн. При розробці дизайну була приділена увага доступності інтерфейсу для різних користувачів, включаючи людей з обмеженими можливостями. Були використані відповідні техніки та рекомендації для забезпечення доступності контенту та його взаємодії. Також був врахований респонсивний дизайн, щоб забезпечити оптимальне відображення та використання веб-додатку на різних пристроях та розмірах екранів.

6. Тестування дизайну. Розроблений дизайн був підданий тестуванню, щоб переконатися, що він відповідає вимогам, функціональності та користувацькому досвіду. Були проведені тестування залучених користувачів, аналіз поведінки та зворотній зв'язок, щоб виявити можливі недоліки та внести необхідні виправлення.

Сторінки front-end (клієнтської) частини веб-додатку виглядають по-різному, як для студентів, так і для адміністраторів, які успішно авторизувалися. Панель керування для адміністраторів веб-додатку складається з компонентів (кнопки, поля), які призначені для створення, редагування та видалення різних сутностей, зокрема особливостей на головній сторінці, новин і контактів.

Зважаючи на розробку дизайну проекту, було обрано бібліотеку компонентів Ant Design для Vue.js з кількох причин:

1. Компонентна бібліотека. Ant Design надає широкий набір готових компонентів, які можна легко використовувати в проекті: кнопки, форми, таблиці, модальні вікна, навігаційні елементи та багато іншого. Використання готових компонентів спрощує розробку, зменшує час, потрібний для реалізації інтерфейсу, і забезпечує єдність стилю та зовнішнього вигляду.

2. Сучасний та стильний дизайн. Ant Design пропонує сучасний та привабливий дизайн, який відповідає останнім трендам веб-дизайну. Чисті лінії, збалансована використання кольорів та уважний до деталей дизайн роблять проект привабливим для користувачів.

3. Підтримка розширення. Ant Design має гнучку архітектуру, яка дозволяє додавати власні стилі та розширювати компоненти за потребою. Це дає можливість налаштувати дизайн проекту відповідно до його унікальних вимог та бренду.

4. Документація та підтримка. Ant Design має добре організовану документацію, яка містить приклади використання, пояснення параметрів компонентів та рекомендації щодо їх використання. Також, фреймворк має активну спільноту, яка надає підтримку та розробляє нові функціональності.

Дизайн головної сторінки веб-додатку для неавторизованого користувача зображено на рисунках 3.3-3.5.

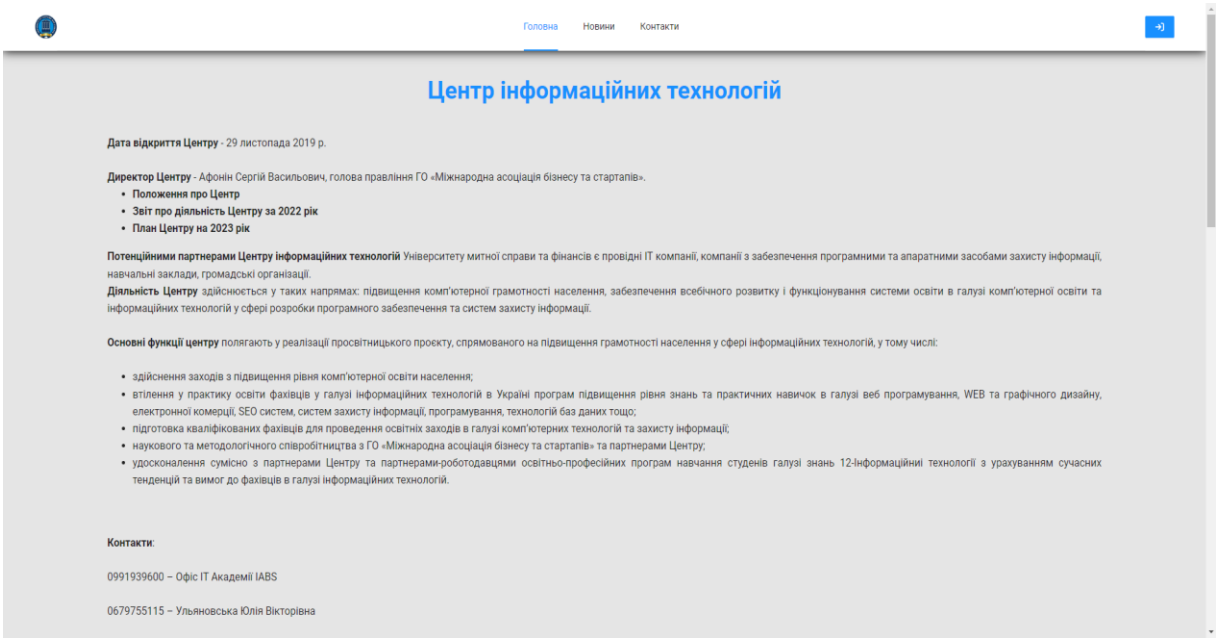


Рисунок 3.3 – Дизайн інформації про центр головної сторінки для неавторизованого користувача

Рисунок 3.3 містить дизайн головної сторінки для неавторизованого користувача з інформацією щодо центру: дата відкриття, директор, потенційні партнери, діяльність, основні функції та контакти.

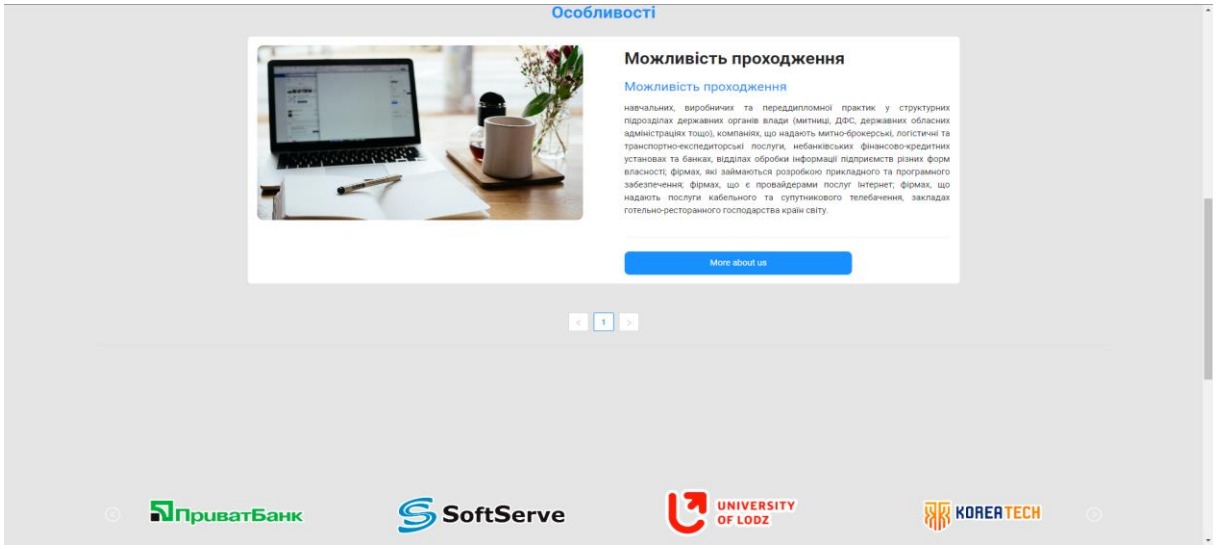


Рисунок 3.4 – Дизайн особливостей головної сторінки для неавторизованого користувача

Рисунок 3.4 містить дизайн головної сторінки з особливостями центру інформаційних технологій для неавторизованого користувача. Особливість складається з підзаголовку, заголовку, тексту та кнопки з посиланням, якщо воно присутнє. Нижче особливостей розташовані кнопки, які є елементами пагінації. Пагінація – це компонент бібліотеки Ant Design, який надає зручні кнопки для організації розподілу даних на сторінки у веб-застосунках.

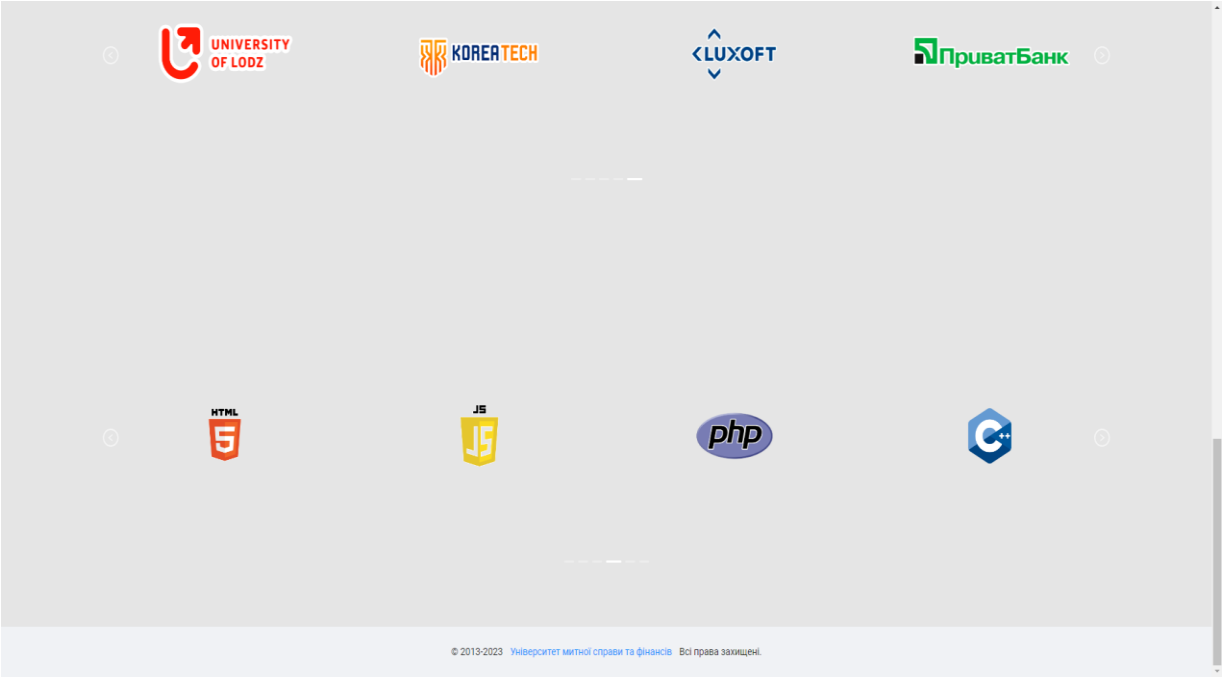


Рисунок 3.5 – Дизайн каруселей партнерів та технологій головної сторінки для неавторизованого користувача

Рисунок 3.5 містить дизайн головної сторінки з каруселлю партнерів та технологій центру. Карусель є компонентом бібліотеки Ant Design, який дозволяє створювати каруселі або слайдери з обраними елементами. Карусель може бути використана для створення слайд-шоу, банерів, галерей, каруселей товарів та багатьох інших сценаріїв. Карусель складається з елементів каруселі (слайдів), стрілок вліво/вправо та індикатору поточної сторінки, що розташований знизу.

Дизайн головної сторінки веб-додатку для авторизованого користувача зображено на рисунках 3.6-3.9.

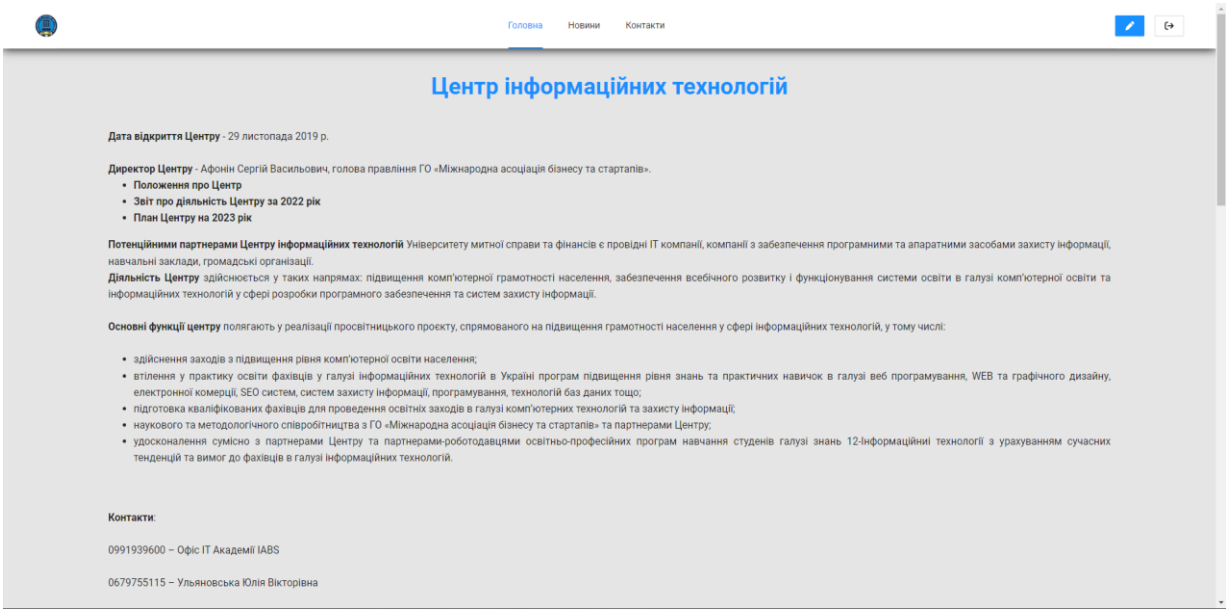


Рисунок 3.6 – Дизайн інформації про центр головної сторінки для авторизованого користувача

Рисунок 3.6 містить дизайн головної сторінки для авторизованого користувача з інформацією щодо центру, яка не відрізняється від інформації, що відображається для неавторизованого користувача.

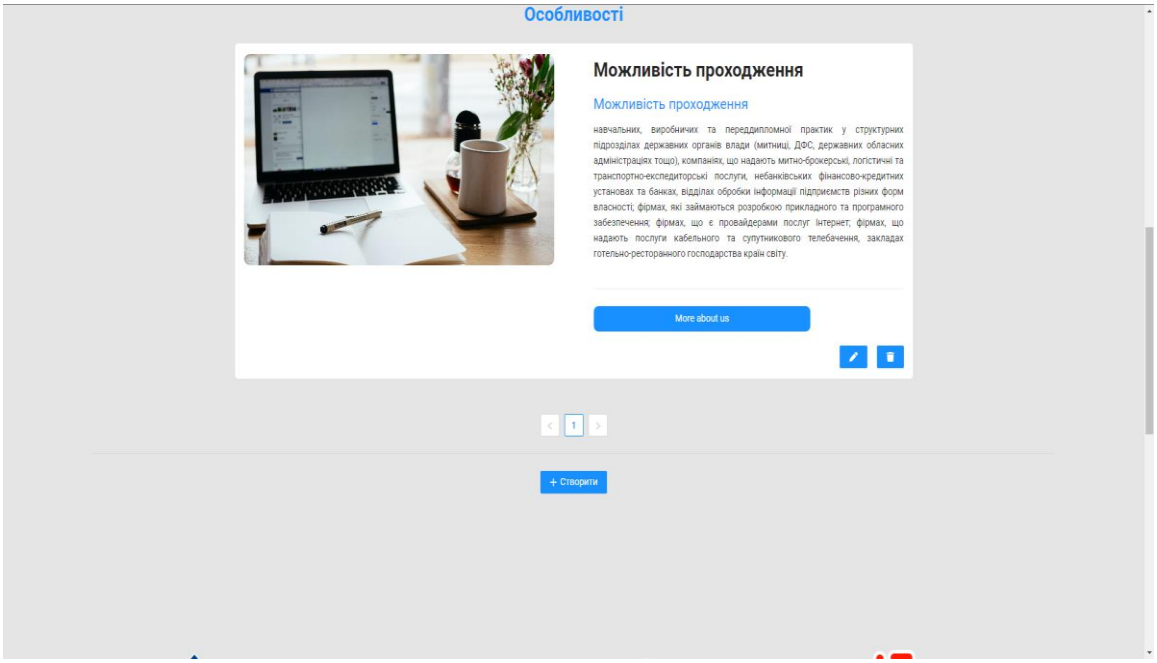


Рисунок 3.7 – Дизайн особливостей головної сторінки для авторизованого користувача

Рисунок 3.7 містить дизайн головної сторінки з особливостями центру інформаційних технологій для авторизованого користувача. Особливість складається з підзаголовку, заголовку, тексту та кнопки з посиланням, якщо воно присутнє. На відміну від дизайну для неавторизованого користувача, особливість ще має 2 кнопки для редагування та видалення. Також нижче з'являється кнопка для створення нової особливості.

Рисунок 3.8 – Дизайн партнерів головної сторінки для авторизованого користувача

Рисунок 3.8 містить дизайн партнерів головної сторінки з каруселлю партнерів для авторизованого користувача. Нижче каруселі розташовані кнопки для створення нових партнерів та редагування існуючих. Через редагування також існує можливість видалити бажані елементи каруселі, якщо це потрібно.

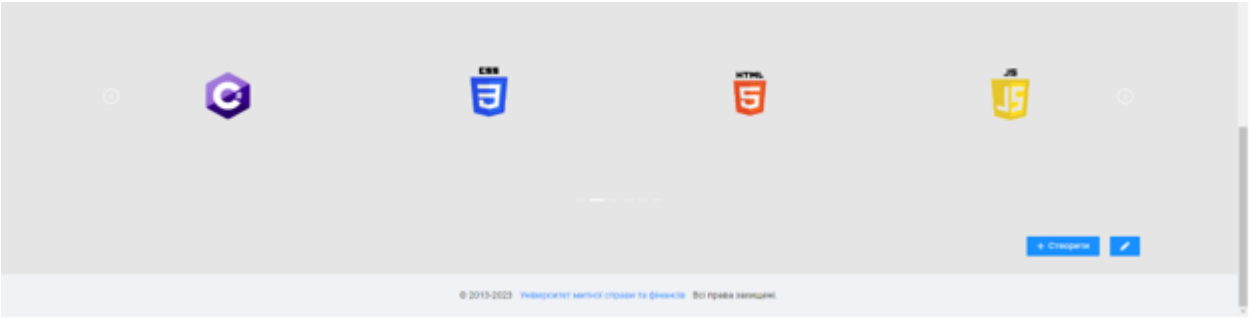


Рисунок 3.9 – Дизайн технологій головної сторінки для авторизованого користувача

Рисунок 3.9 містить дизайн партнерів головної сторінки з каруселлю технологій для авторизованого користувача. Нижче каруселі розташовані кнопки для створення нових технологій та редагування існуючих.

Дизайн сторінки «Зустрічі з роботодавцями» для неавторизованого користувача зображено на рисунку 3.10.

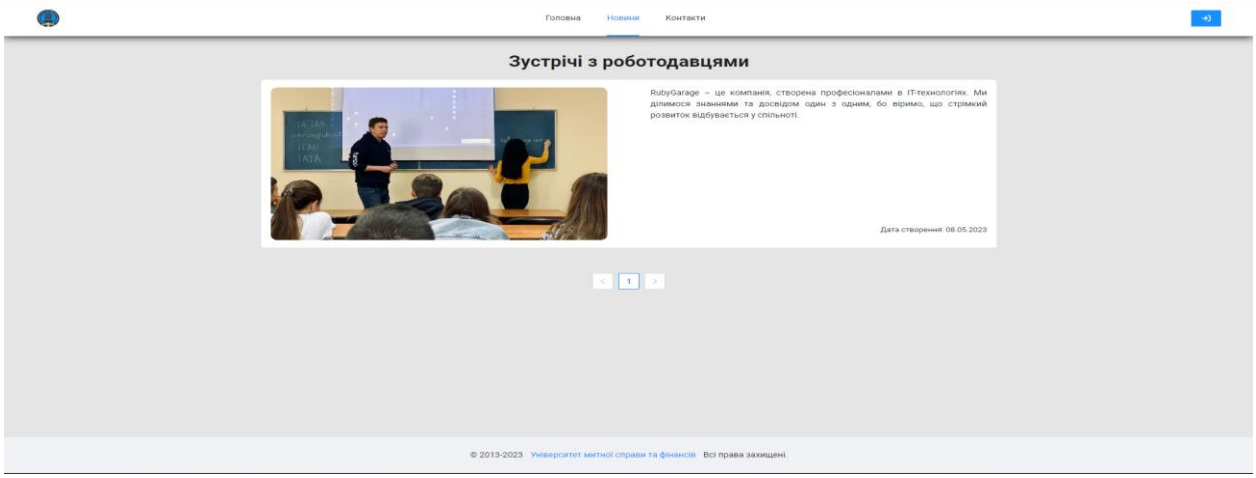


Рисунок 3.10 – Дизайн сторінки «Зустрічі з роботодавцями» для неавторизованого користувача

Рисунок 3.10 містить дизайн сторінки «Зустрічі з роботодавцями» для неавторизованого користувача. Сторінка складається з компонентів новин, які мають пагінацію та однакові для всіх сторінок (категорій) новин: «Зустрічі з роботодавцями», «Наукові заходи» та «Студентське життя». Компонент новин містить зображення з лівої частини, а текст та дату створення – з правої частини.

Дизайн сторінки «Зустрічі з роботодавцями» для авторизованого користувача зображено на рисунку 3.11.

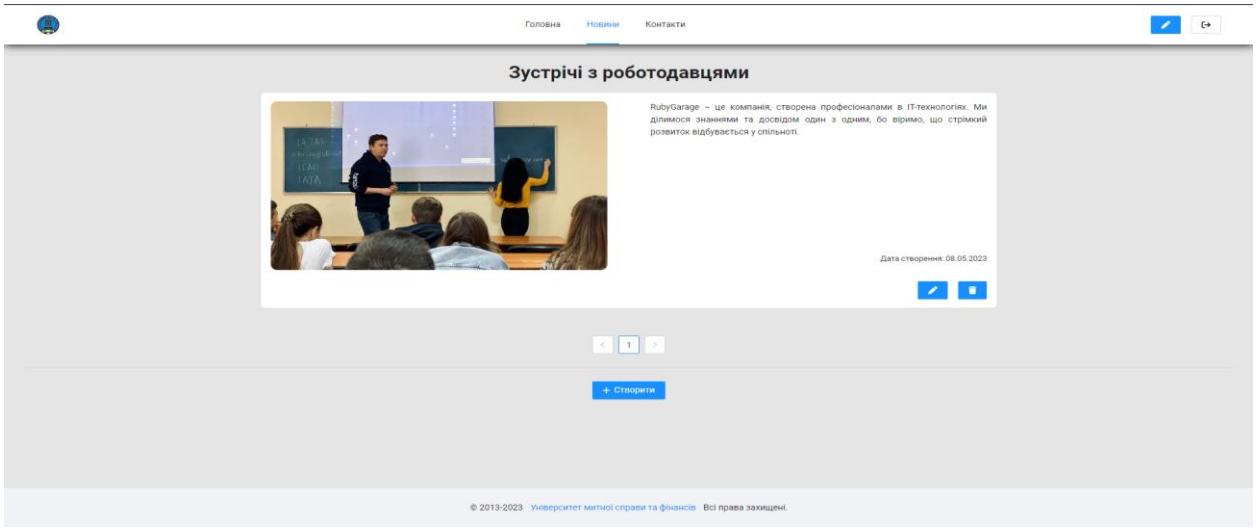


Рисунок 3.11 – Дизайн сторінки «Зустрічі з роботодавцями» для авторизованого користувача

Рисунок 3.11 містить дизайн сторінки «Зустрічі з роботодавцями» для авторизованого користувача. Сторінка складається з компонентів новин, які мають пагінацію. З правої частини компонента можна помітити 2 кнопки для редагування та видалення новини.

Дизайн сторінки «Наукові заходи» для неавторизованого користувача зображено на рисунку 3.12.

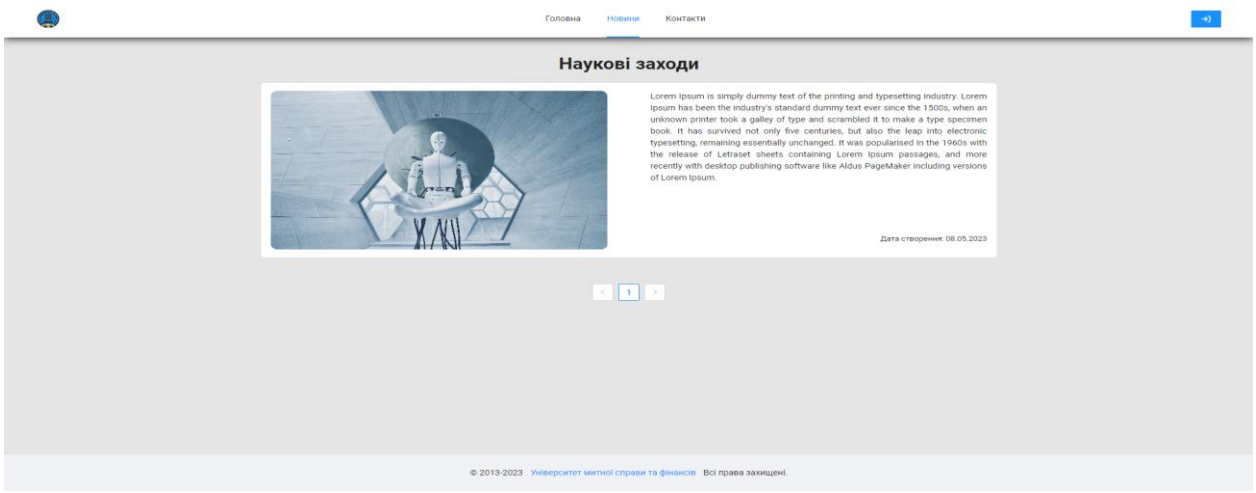


Рисунок 3.12 – Дизайн сторінки «Наукові заходи» для неавторизованого користувача

Рисунок 3.12 містить дизайн сторінки «Наукові заходи» для неавторизованого користувача. Як можна було помітити, всі сторінки новин мають ідентичний дизайн, але віздрізняються назвою сторінки (категорією) та вмістом новин.

Дизайн сторінки «Наукові заходи» для авторизованого користувача зображено на рисунку 3.13.

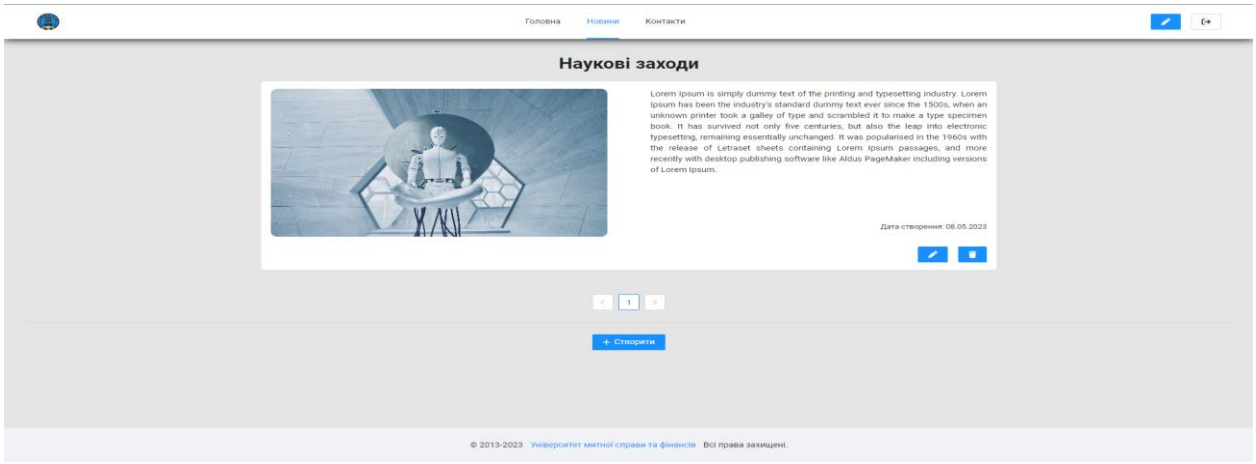


Рисунок 3.13 – Дизайн сторінки «Наукові заходи» для авторизованого користувача

Рисунок 3.13 містить дизайн сторінки «Наукові заходи» для авторизованого користувача. З правої частини компонента новин можна помітити 2 кнопки для редагування та видалення новини.

Дизайн сторінки «Студентське життя» для неавторизованого користувача зображено на рисунку 3.14.

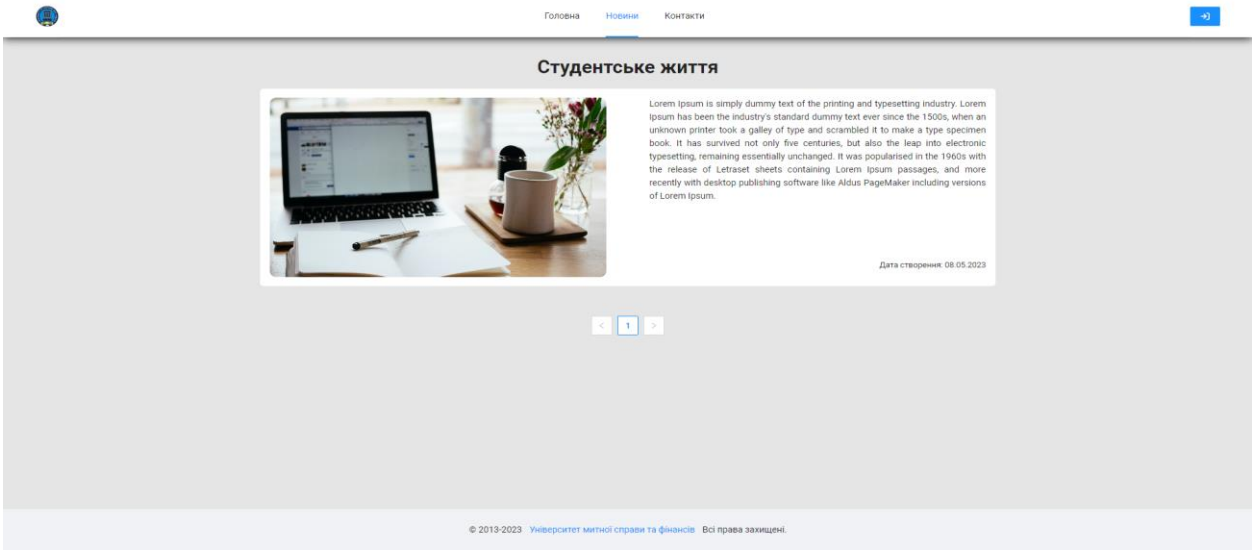


Рисунок 3.14 – Дизайн сторінки «Студентське життя» для неавторизованого користувача

Рисунок 3.14 містить дизайн сторінки «Студентське життя» для неавторизованого користувача. Одна сторінка пагінації дозволяє виводити по дві новини на сторінку. Для зміни поточної сторінки можна використовувати кнопки, які містять стрілки вліво та вправо.

Дизайн сторінки «Студентське життя» для авторизованого користувача зображено на рисунку 3.15.

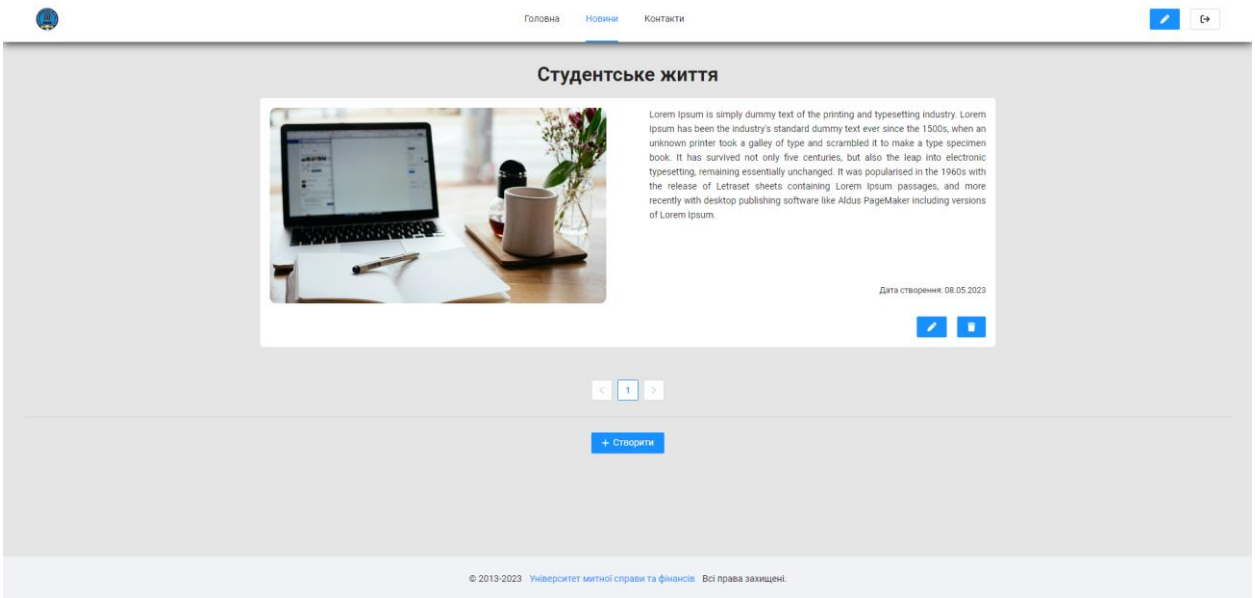


Рисунок 3.15 – Дизайн сторінки «Студентське життя» для авторизованого користувача

Рисунок 3.15 містить дизайн сторінки «Студентське життя» для авторизованого користувача. З правої частини компонента новин можна помітити 2 кнопки для редагування та видалення новини.

Дизайн сторінки «Контакти» для неавторизованого користувача зображено на рисунку 3.16.

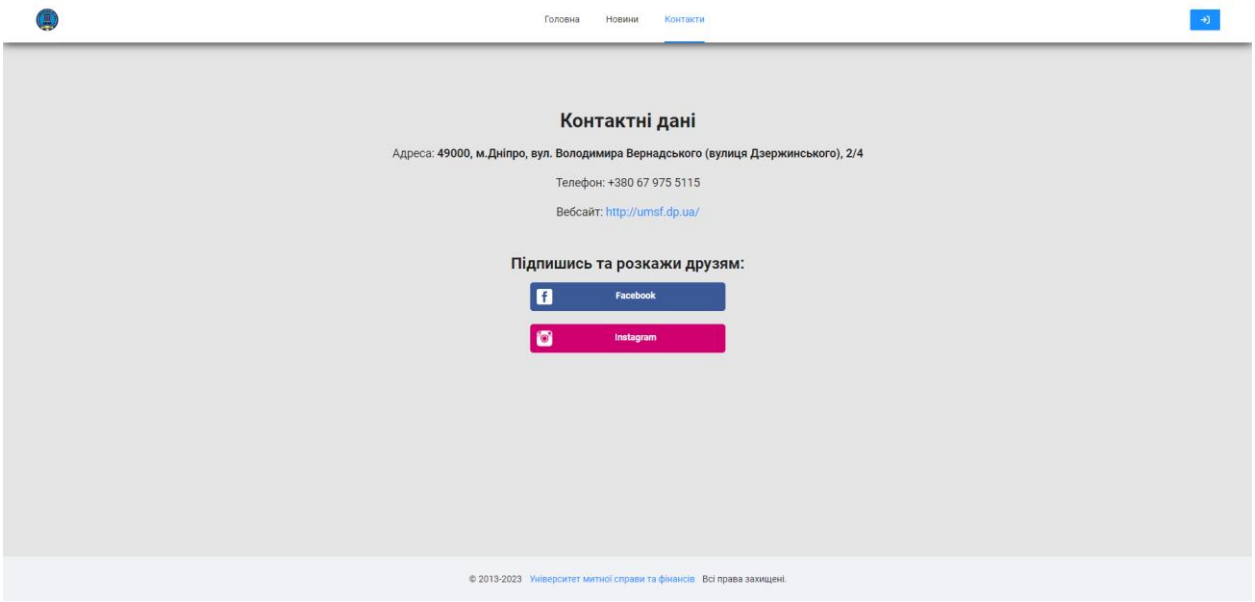


Рисунок 3.16 – Дизайн сторінки «Контакти» для неавторизованого користувача

Рисунок 3.16 містить дизайн сторінки «Контакти», що складається з заголовку сторінки, адреси, телефону, вебсайту та посилань на соціальні мережі центру інформаційних технологій.

Дизайн сторінки «Контакти» для авторизованого користувача зображено на рисунку 3.17. Знизу контактних даних є горизонтальна смуга і кнопка, що дозволяє редагувати адресу, телефон та посилання на вебсайт, якщо це необхідно.

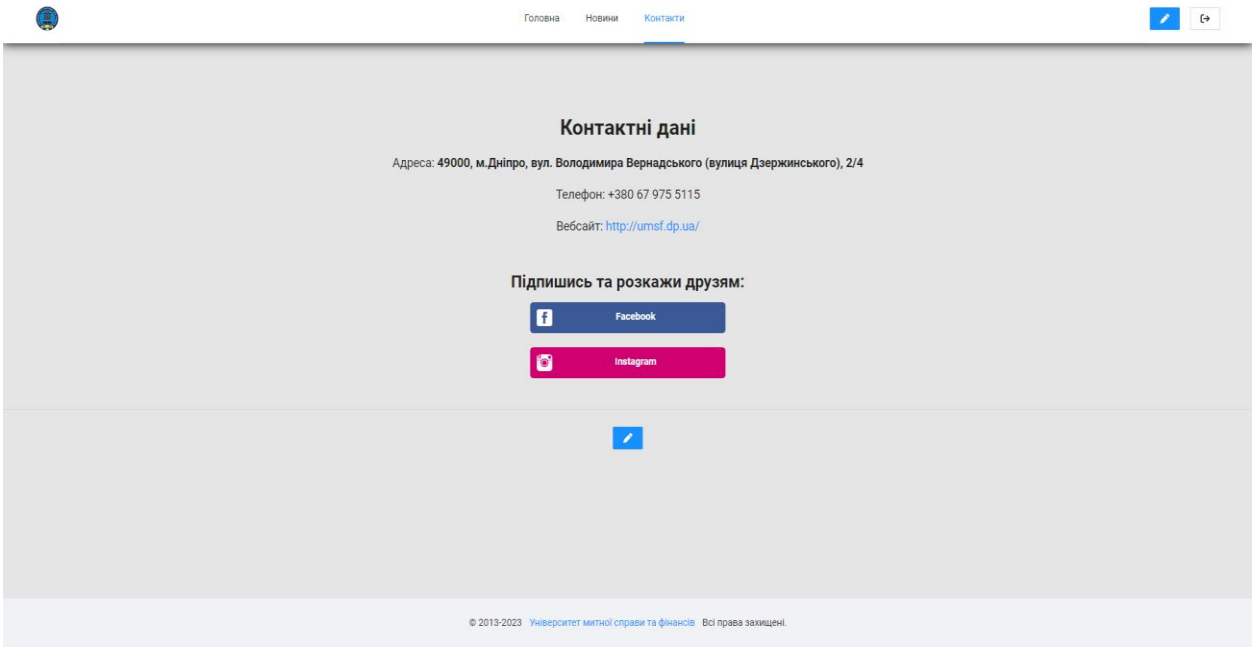


Рисунок 3.17 – Дизайн сторінки «Контакти» для авторизованого користувача

Навігаційне меню було розташовано у верхній частині сайту оскільке це має декілька переваг:

1. Легка доступність. Розташування меню у верхній частині сайту робить його легко видимим і доступним для користувачів, тому що вони можуть швидко знайти потрібний пункт меню без необхідності прокручування сторінки.

2. Чітка навігація. Меню у верхній частині сайту зазвичай має добре впорядковану структуру, що дозволяє зручно організувати основні розділи або сторінки. Користувачі можуть швидко зорієнтуватися і знайти потрібну інформацію.

3. Постійна видимість. Меню, розміщене у верхній частині сайту, залишається видимим на кожній сторінці. Це дозволяє користувачам легко переміщатися між розділами сайту, незалежно від того, на якій сторінці вони знаходяться.

Кольори на сайті були грамотно підібрані з метою створення естетичного та привабливого вигляду сайту. Вибір кольорів на сайті був здійснений з урахуванням наступних факторів:

1. Гармонія. Кольорова палітра була створена з дотриманням принципів гармонії кольорів. Кольори були взаємно підібрані таким чином, щоб вони доповнювали один одного і створювали збалансований вигляд.

2. Брендування. Кольори відповідають ідентичності та брендуванню сайту. Було враховано кольори, які пов'язані з логотипом сайту.

3. Емоційний вплив. Вибрані кольори мають певний емоційний вплив на користувачів. Теплі та насичені кольори можуть викликати почуття енергії та позитиву, тоді як нейтральні та приглушені кольори можуть створювати відчуття спокою та надійності.

4. Контрастність та читабельність. Кольорова палітра була підібрана з урахуванням контрастності та читабельності. Важливо, щоб текст та інші важливі елементи були чітко видимі на задньому фоні.

5. Створення атмосфери. Вибрані кольори мають сприяти створенню певної атмосфери на сайті. Наприклад, яскраві та живі кольори можуть створювати веселе та динамічне враження, тоді як відтінки синього або зеленого можуть створювати відчуття спокою та гармонії.

3.3 Клієнтська (front-end) частина проекту

Для реалізації клієнтської частини проекту було використано мову програмування TypeScript та один з найпопулярніших front-end фреймворків – Vue.js. Одним з основних чинників вибору цих інструментів розробки була їх універсальність та легкість використання.

Нижче зображено функціональну схему front-end частини проекту (див. рис. 3.18).

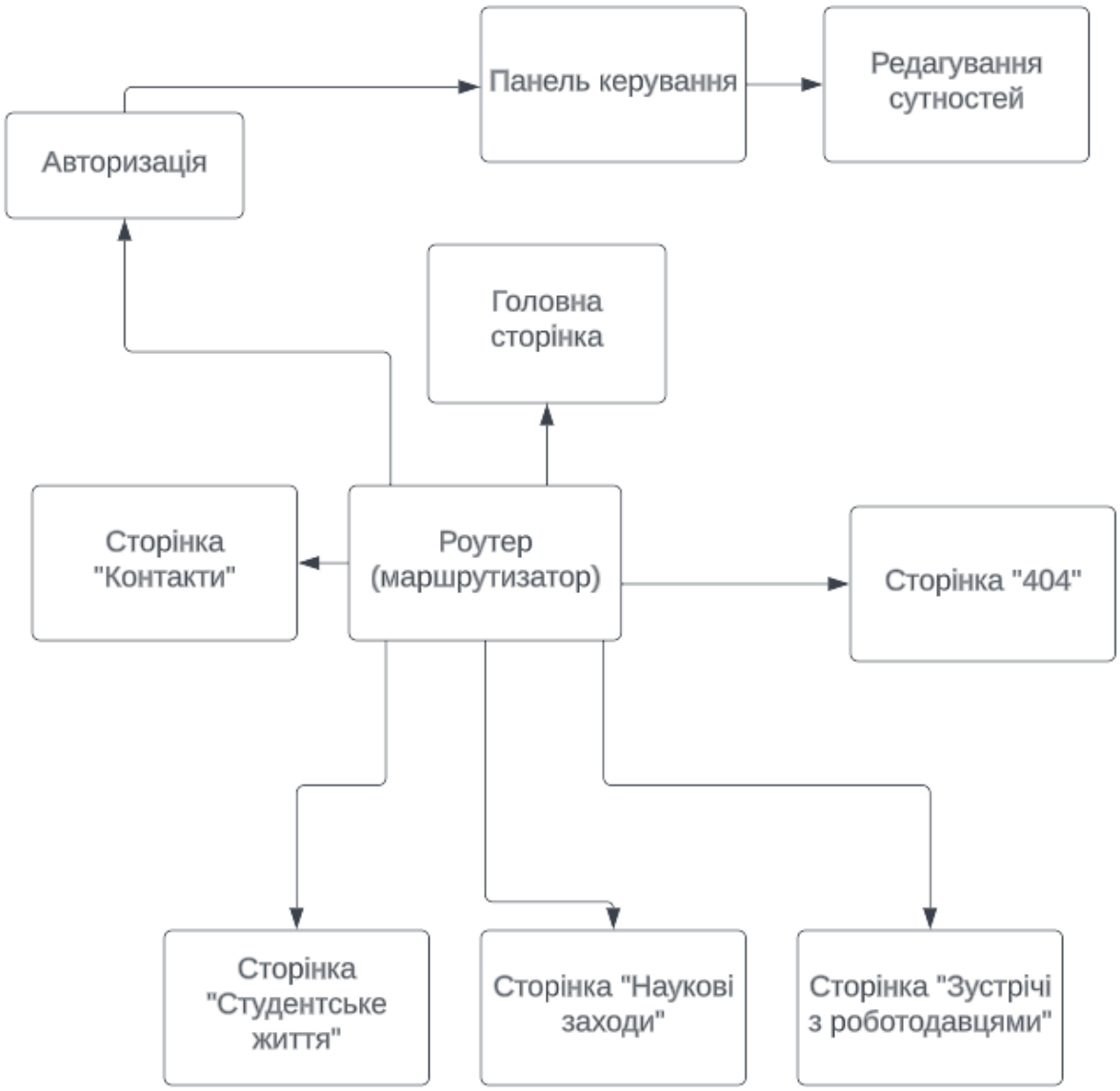


Рисунок 3.18 – Функціональна схема front-end частини проекту

Front-end частина проекту складається з наступних сторінок:

1. Головна
2. Авторизація
3. Контакти
4. 404
5. Студентське життя
6. Наукові заходи

7. Зустрічі з роботодавцями

Авторизація у проекті представлена у вигляді модального вікна, яка взаємодіє з роутером. Роутер перевіряє, чи був користувач успішно авторизований. Якщо так, то маршрутизатор надає доступ до панелі керування, що в свою чергу дає можливість редагувати сутності (новини, контакти).

Потрібно розуміти, що весь додаток відбудовується лише в одному (кореневому) компоненті – App.vue (див. рис. 3.19), який в свою чергу отрисовує менеджер макета (див. рис. 3.20), що динамічно показує потрібні сторінки.

Рисунок 3.19 – Компонент App.vue

```vue
<script setup lang="ts">
import { markRaw, ref, watch, type Component } from 'vue';
import { useRoute } from 'vue-router';

import { layouts } from '@/layouts/data';
import DefaultLayout from '@/layouts/Default/Index.vue';

const layout = ref<Component>();
const route = useRoute();

watch(
  () => route.meta?.layout as keyof typeof layouts,
  (metaLayout: keyof typeof layouts) => {
    try {
      const component = metaLayout && layouts[metaLayout];
      layout.value = markRaw(component || DefaultLayout);
    } catch (e) {
      layout.value = markRaw(DefaultLayout);
    }
  },
  { immediate: true }
);
</script>

<template>
  <component :is="layout"> <router-view /> </component>
</template>

```

Рисунок 3.20 – Компонент LayoutManager.vue (менеджер макета)

Компонент LayoutManager.vue перевіряє, який макет був обраний для поточної сторінки та відрисовує його. Якщо для сторінки не був обраний макет, то то встановлюється макет за замовчуванням «DefaultLayout».

Файлову структуру front-end частини веб-додатку можна побачити на рисунку 3.21.

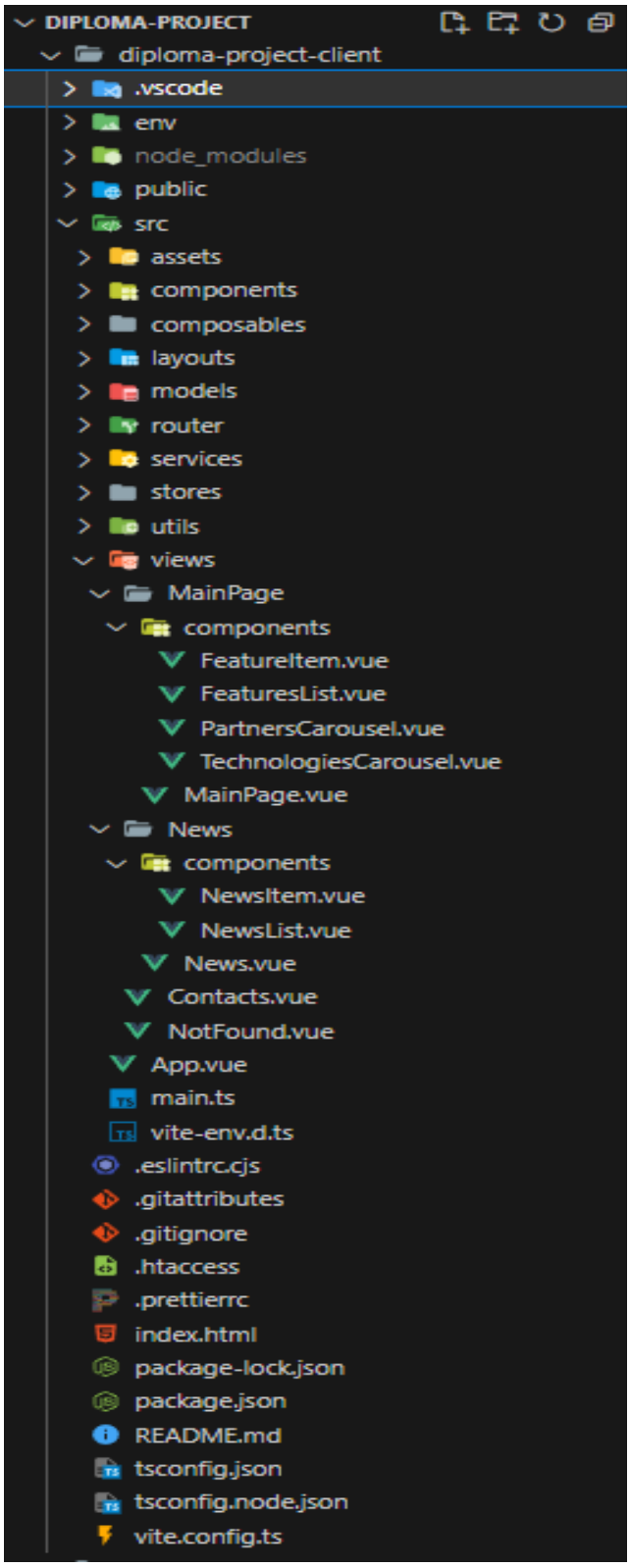


Рисунок 3.21 – Файлова структура front-end частини веб-додатку

Даний додаток містить шапку (header), який в свою чергу містить логотип сайту, навігаційне меню та кнопки для авторизації, якщо користувач не авторизувався, інакше – кнопки редагування категорій новин та вихід з

облікового запису. Окрім шапки, додаток також містить підвал (footer). За логіку відображення поточної сторінки, шапки та підвалу сайту на кожній зі сторінок відповідає стандартний макет (див. рис. 3.22), який відображається, якщо він був обраний компонентом LayoutManager.vue.

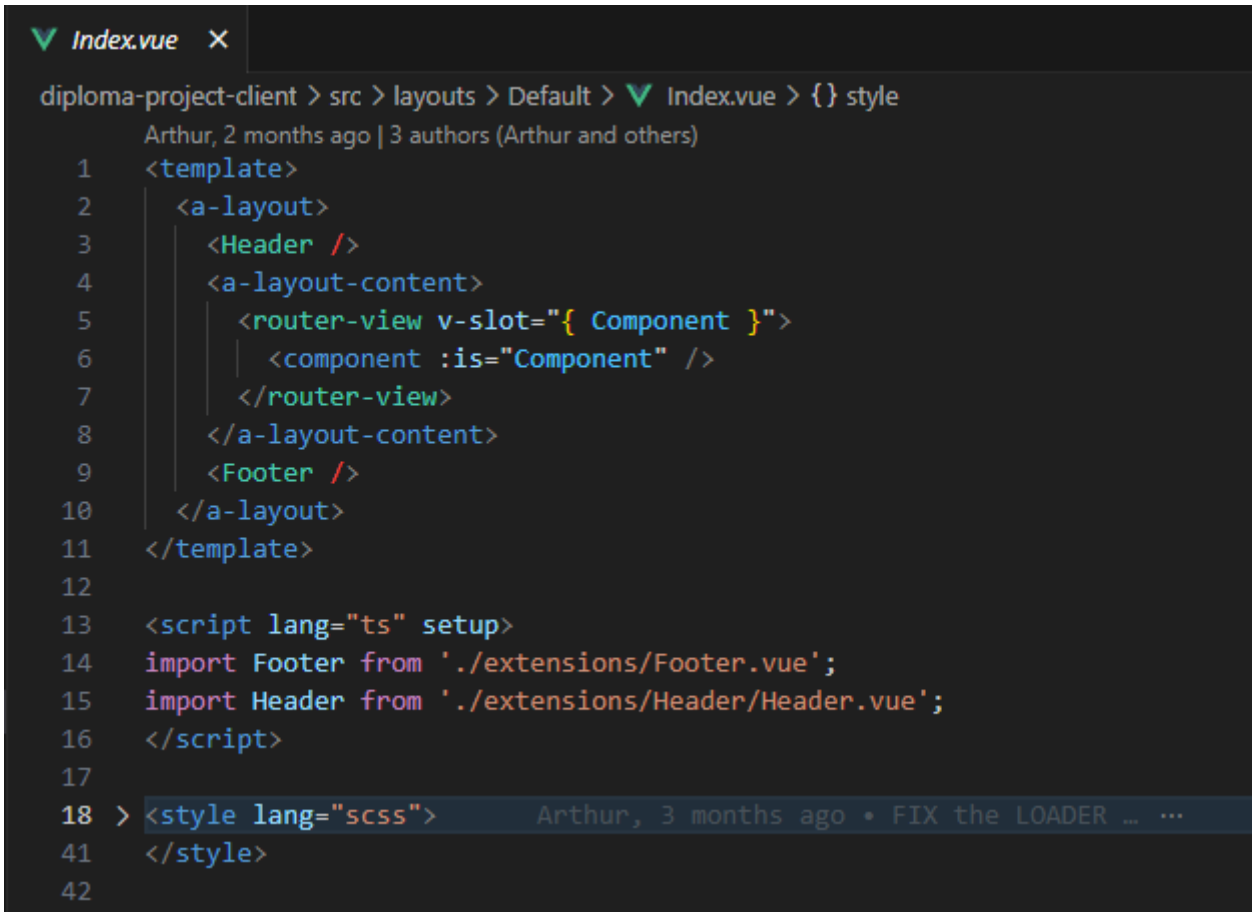
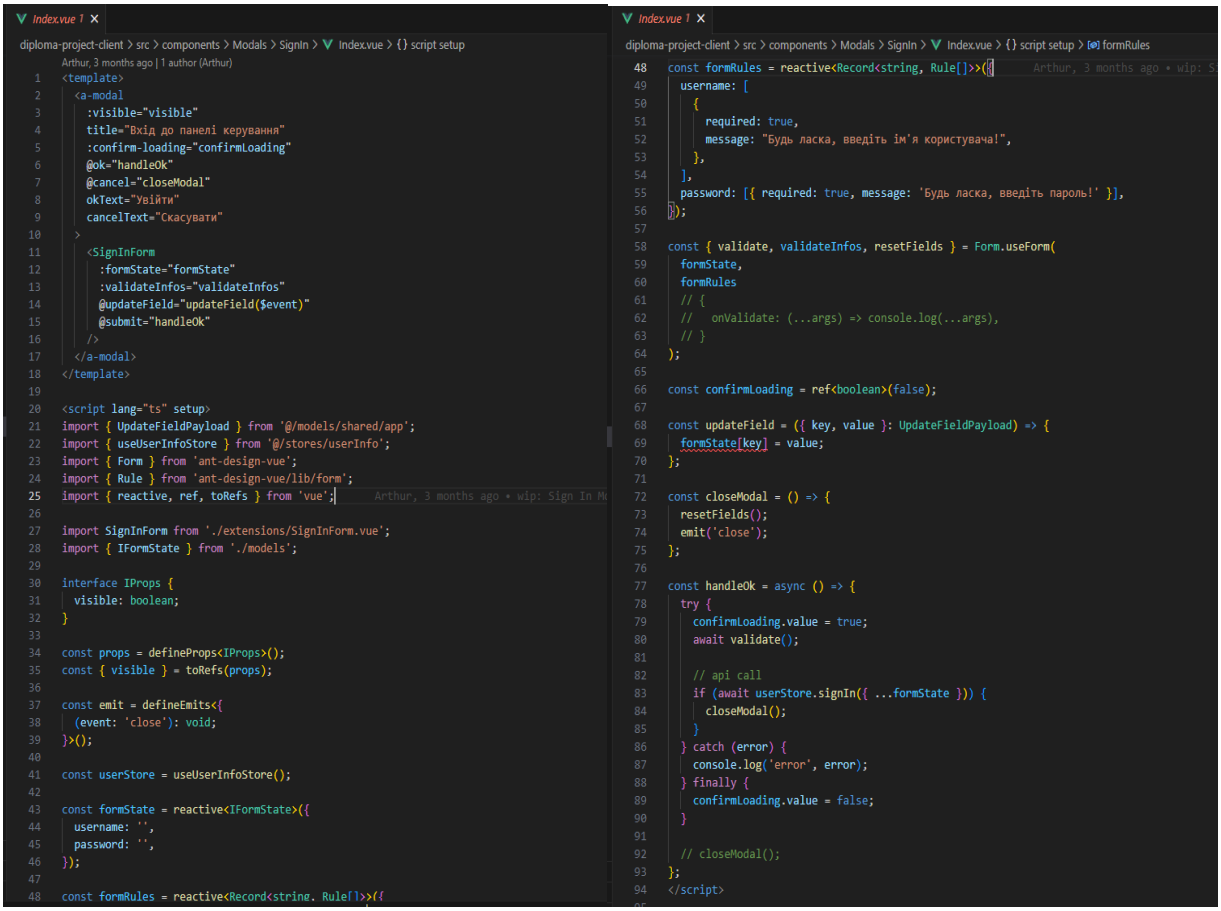


Рисунок 3.22 – Компонент стандартного макета

Одразу після запуску веб-додатку відкривається головна сторінка, що містить інформацію про центр інформаційних технологій, особливості, список партнерів та технологій. Для успішного проходження процесу авторизації потрібно ввести правильні дані до полів (ім’я користувача та пароль) модального вікна авторизації. Нижче зображено логіку роботи модального вікна авторизації (див. рис. 3.23).



```vue
<template>
  <a-modal
    :visible="visible"
    title="Вхід до панелі керування"
    :confirm-loading="confirmLoading"
    @ok="handleOk"
    @cancel="closeModal"
    okText="Увійти"
    cancelText="Скасувати"
  >
    <SignInForm
      :formState="formState"
      :validateInfos="validateInfos"
      @updateField="updateField($event)"
      @submit="handleOk"
    />
  </a-modal>
</template>

<script lang="ts" setup>
import { UpdateFieldPayload } from '@/models/shared/app';
import { useUserInfoStore } from '@/stores/userInfo';
import { Form } from 'ant-design-vue';
import { Rule } from 'ant-design-vue/lib/form';
import { reactive, ref, toRefs } from 'vue';

import SignInForm from './extensions/SignInForm.vue';
import { IFormState } from './models';

interface IProps {
  visible: boolean;
}

const props = defineProps<IProps>();
const { visible } = toRefs(props);

const emit = defineEmits<{
  (event: 'close'): void;
}>();

const userStore = useUserInfoStore();

const formState = reactive<IFormState>({
  username: '',
  password: '',
});

const formRules = reactive<Record<string, Rule[]>>({
  username: [
    {
      required: true,
      message: "Будь ласка, введіть ім'я користувача!",
    },
  ],
  password: [{ required: true, message: 'Будь ласка, введіть пароль!' }],
});

const { validate, validateInfos, resetFields } = Form.useForm(
  formState,
  formRules
  // {
  //   onValidate: (...args) => console.log(...args),
  // }
);

const confirmLoading = ref<boolean>(false);

const updateField = ({ key, value }: UpdateFieldPayload) => {
  formState[key] = value;
};

const closeModal = () => {
  resetFields();
  emit('close');
};

const handleOk = async () => {
  try {
    confirmLoading.value = true;
    await validate();

    // api call
    if (await userStore.signIn({ ...formState })) {
      closeModal();
    }
  } catch (error) {
    console.log('error', error);
  } finally {
    confirmLoading.value = false;
  }

  // closeModal();
};
</script>
```

Рисунок 3.23 – Компонент модального вікна авторизації

Завдяки роботі роутера можна переходити між сторінками веб-додатку та взаємодіяти з тим, що потрібно.

3.4 Серверна (back-end) частина проекту

Для реалізації серверної частини проекту було використано мову програмування TypeScript та один з найпопулярніших back-end фреймворків – NestJS.

Файлову структуру back-end частини веб-додатку можна побачити на рисунку 3.24.

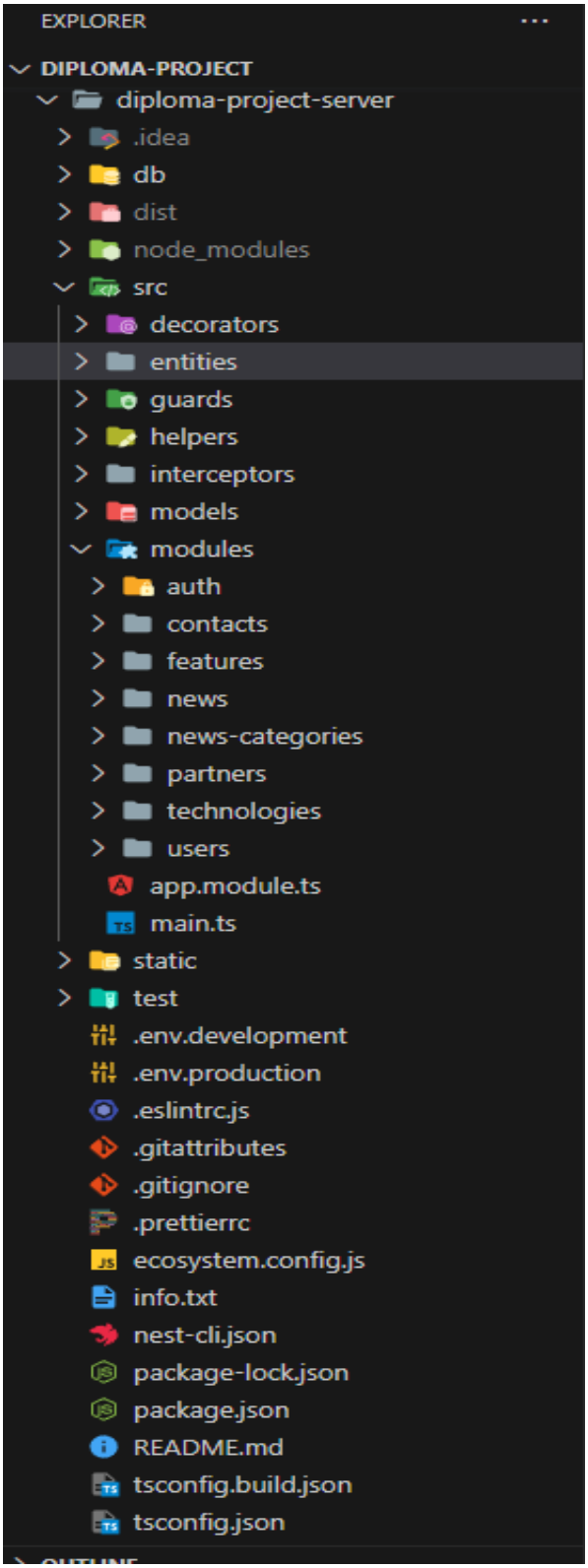


Рисунок 3.24 – Файлова структура back-end частини веб-додатку

Папка «src» містить код проекту. Файл main.ts – це основний файл запуску, а app.module.ts – головний модуль застосунку (див. рис. 3.25).

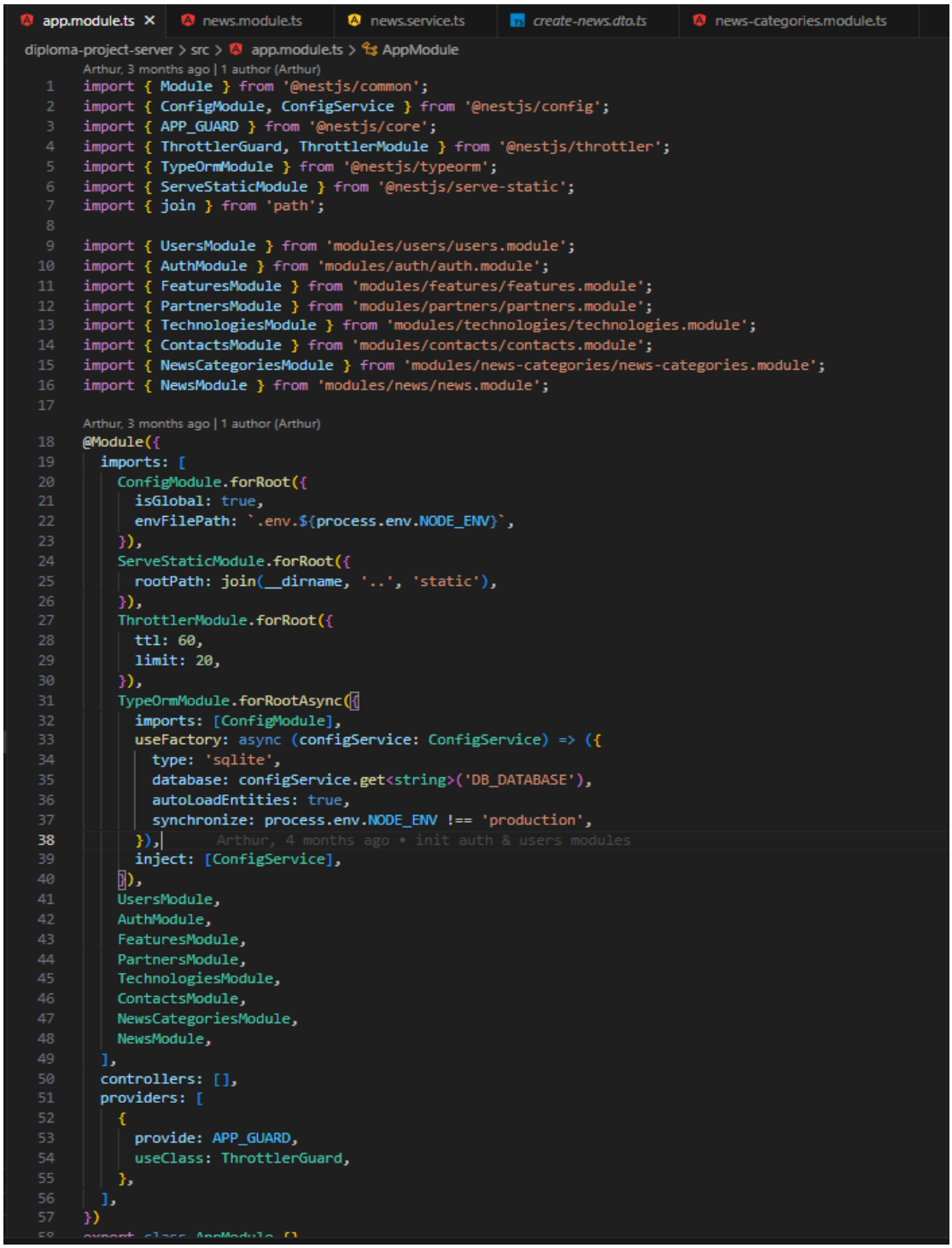


Рисунок 3.25 – Головний модуль серверної частини веб-додатку

Модуль складається з сутностей, контролера та сервісу. Головний модуль, в свою чергу, містить інші модулі додатку:

1. Users. Модуль відповідає за створення нового облікового запису, отримання користувача за допомогою ідентифікатора або ім’я користувача, оновлення даних та видалення облікового запису.

2. Auth. Модуль відповідає за взаємодію з користувачем: створення, авторизацію, вихід. Для реалізації цього функціоналу він використовує методи модулю Users.

3. Features. Модуль відповідає за створення, редагування та видалення особливостей, що відображаються на головній сторінці веб-додатку.

4. Partners. Модуль відповідає за створення, редагування та видалення партнерів, що відображаються на головній сторінці веб-додатку.

5. Technologies. Модуль відповідає за створення, редагування та видалення технологій, що відображаються на головній сторінці веб-додатку.

6. Contacts. Модуль відповідає за створення, редагування та видалення контактів, що відображаються на сторінці «Контакти» веб-додатку.

7. NewsCategories. Модуль відповідає за створення, редагування та видалення категорій новин. Приклади вже існуючих категорій: «Зустрічі з роботодавцями», «Наукові заходи», «Студентське життя».

8. News. Модуль відповідає за створення, редагування та видалення новин. Новини відносяться до відповідної категорії новин.

На рисунках 3.26-3.27 наведено блок-схеми принципу роботи реєстрації та авторизації веб-додатку.

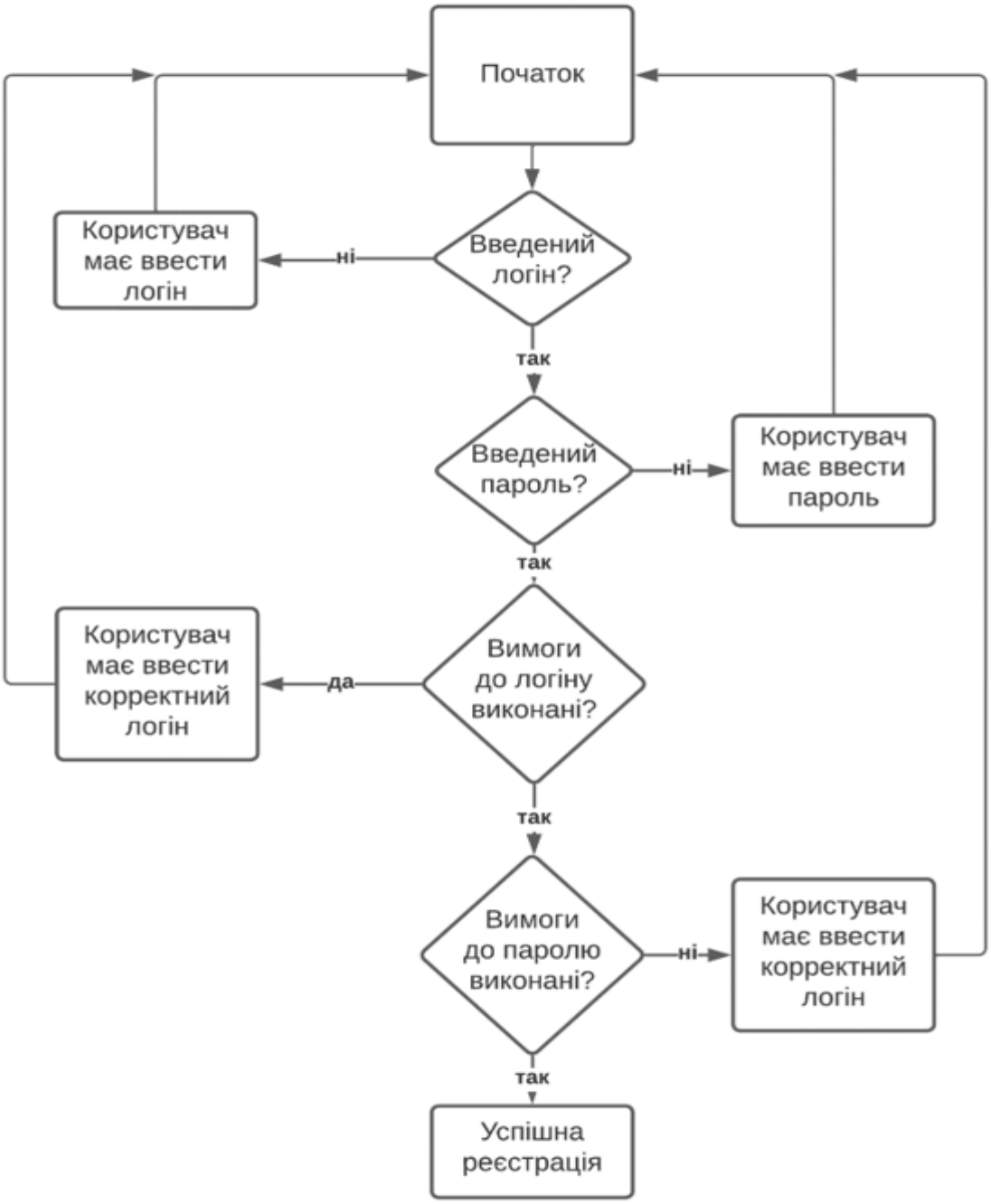


Рисунок 3.26 – Блок-схема принципу роботи реєстрації веб-додатку

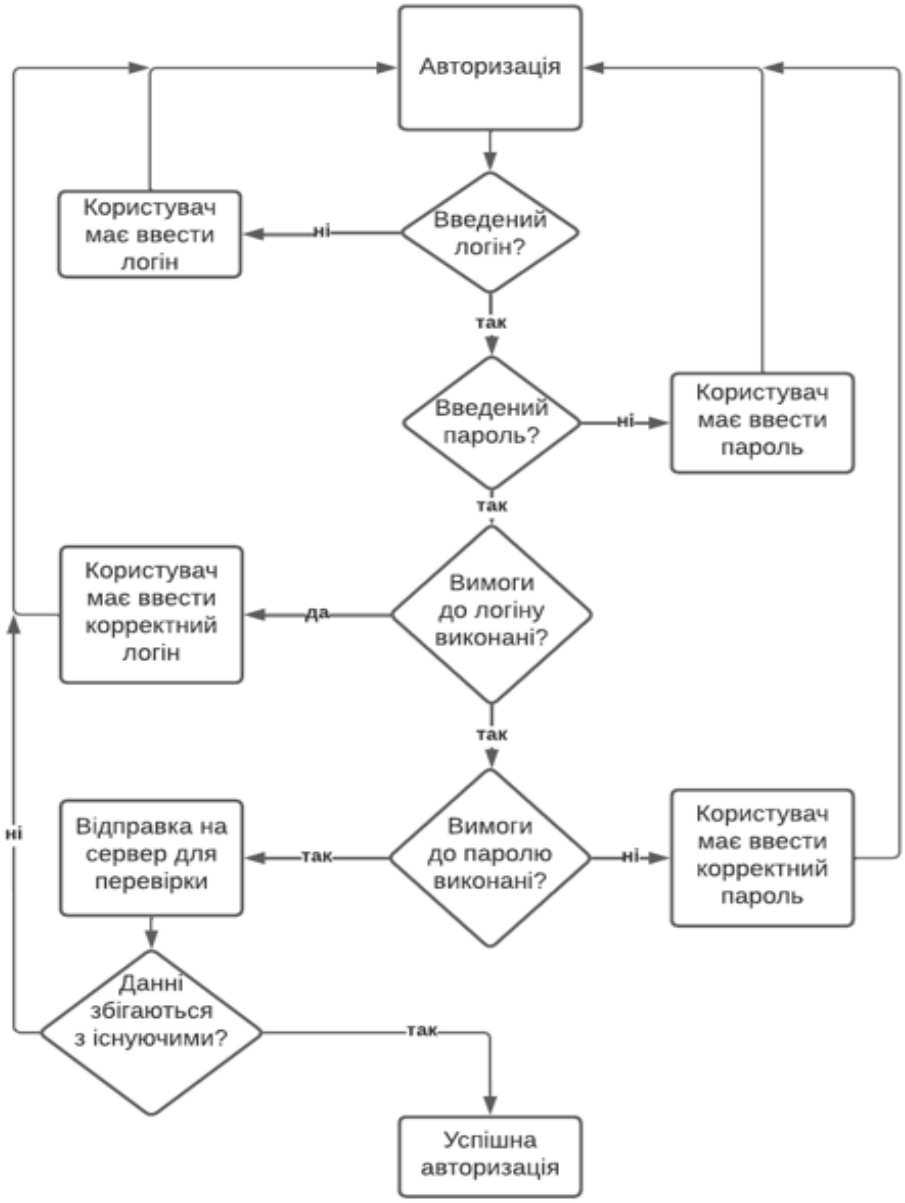


Рисунок 3.27 – Блок-схема принципу роботи авторизації веб-додатку

ER-діаграма, або діаграма сутностей-зв'язків, є важливим інструментом для проектування баз даних, оскільки ER-діаграма допомагає ідентифікувати сутності (або об'єкти), які потрібно зберігати у базі даних, і встановлює зв'язки між цими сутностями. Вона дозволяє чітко визначити, які дані повинні бути збережені і як вони пов'язані між собою. Також ER-діаграма допомагає зрозуміти структуру бази даних і виявити можливості для оптимізації. На рис. 3.28 зображено ER-діаграму back-end частини застосунку.

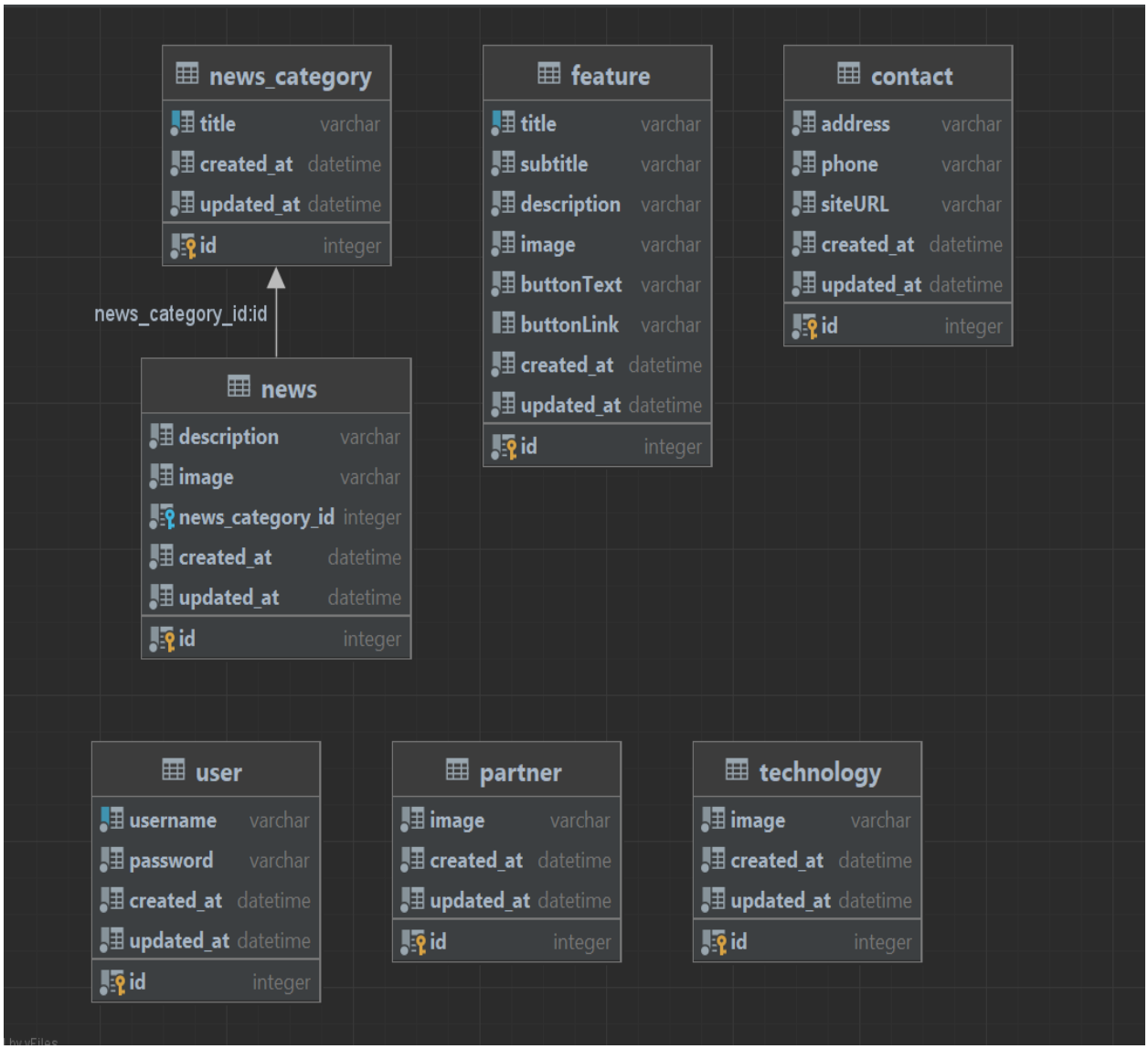


Рисунок 3.28 – ER-діаграма

Як видно на рисунку 3.28, присутній зв'язок «один-до-багатьох» між таблицями «news» та «news_category» бази даних. Жовтим ключем позначено primary key (первинний ключ), він ідентифікує кортеж відношення. Синім ключем позначено foreign key (зовнішній ключ). Зовнішній ключ — атрибут в деякому відношенні R, який відповідає первинному ключу іншого відношення або того ж таки відношення R.

3.5 Тестування

У цьому підрозділі буде описано процес тестування виконання запитів HTTP до серверної частини веб-додатку з використанням інструменту Postman. Postman – це API-платформа для розробників, яка дозволяє їм

тестувати свої API. Тестування серверної частини дозволяє перевірити, чи працює вона належним чином, відповідає вимогам функціональності та масштабованості.

Перед початком тестування було налаштовано тестове середовище, яке відтворює умови роботи сервера у реальному середовищі. Було встановлено Postman та налаштовані необхідні параметри для взаємодії з сервером.

У процесі тестування була врахована швидкість інтернет-з'єднання для більш точної оцінки продуктивності. Швидкість завантаження складала 95 Мбіт/с, а швидкість вивантаження – 94 Мбіт/с.

Для отримання найбільш правильного середнього часу відповіді від сервера було зроблено 10 ітерацій на кожен запит.

Середній час виконання http-запиту для авторизації – 59 мс. Це є нормальним часом роботи сервісу, який відповідає за авторизацію.

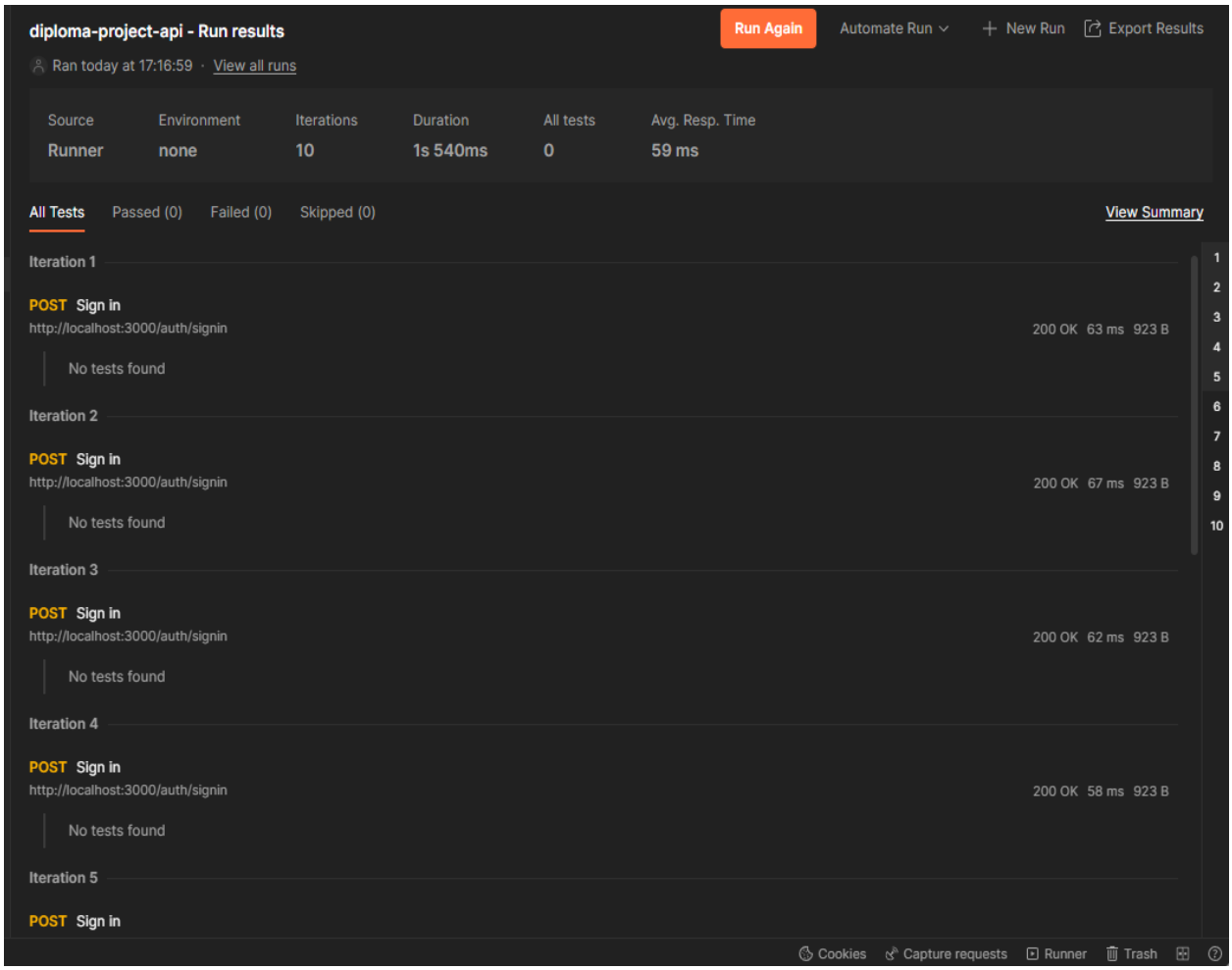


Рисунок 3.29 – Робота з HTTP-запитом для авторизації у Postman

Середній час виконання http-запиту для виходу з облікового запису – 2 мс. Це є досить швидким показником роботи сервісу, що відповідає за функціонал виходу з облікового запису.

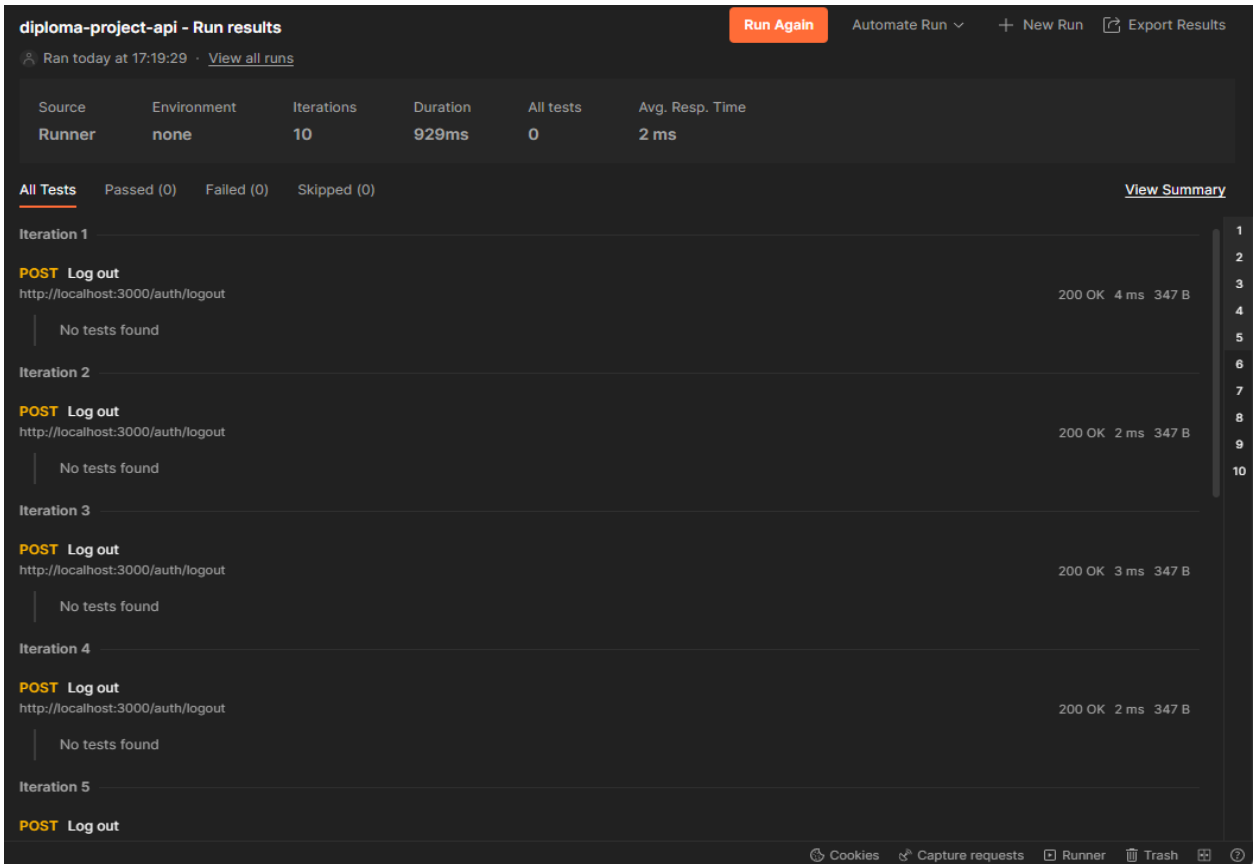


Рисунок 3.28 – Робота з HTTP-запитом для виходу у Postman

Середній час виконання http-запиту для отримання всіх особливостей (на головній сторінці) – 3 мс. Це є досить швидким показником роботи сервісу, що відповідає за функціонал виходу з облікового запису.

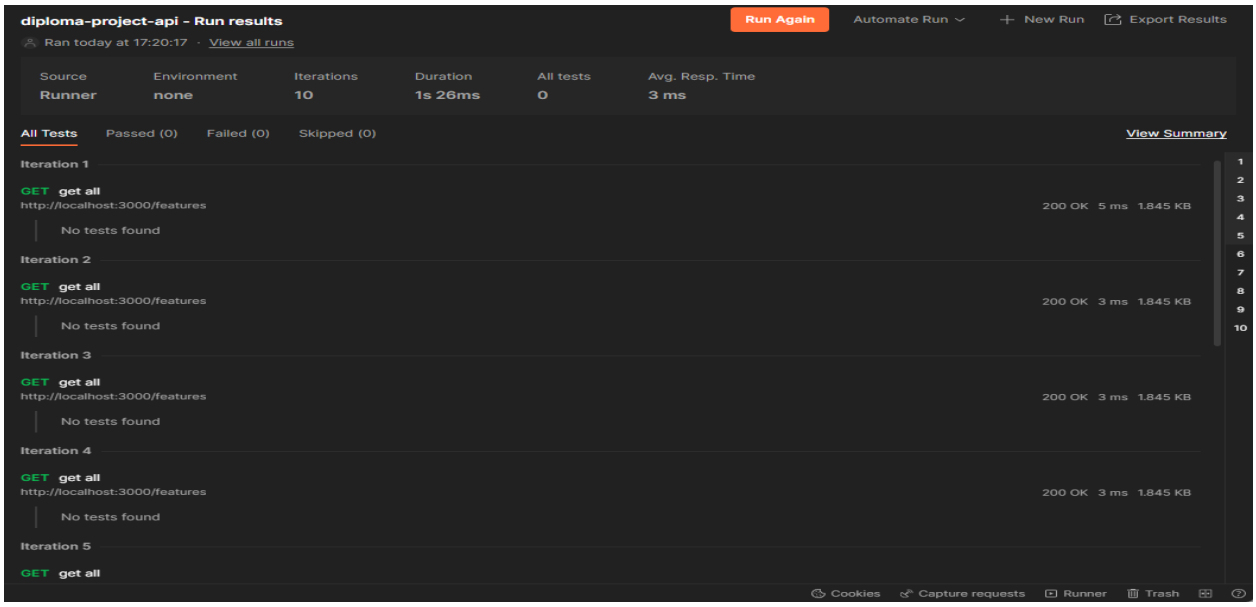


Рисунок 3.29 – Робота з HTTP-запитом для отримання всіх особливостей у Postman

Середній час виконання http-запиту для отримання всіх партнерів (на головній сторінці) – 3 мс. Це є швидким показником роботи сервісу, що відповідає за функціонал отримання всіх особливостей.

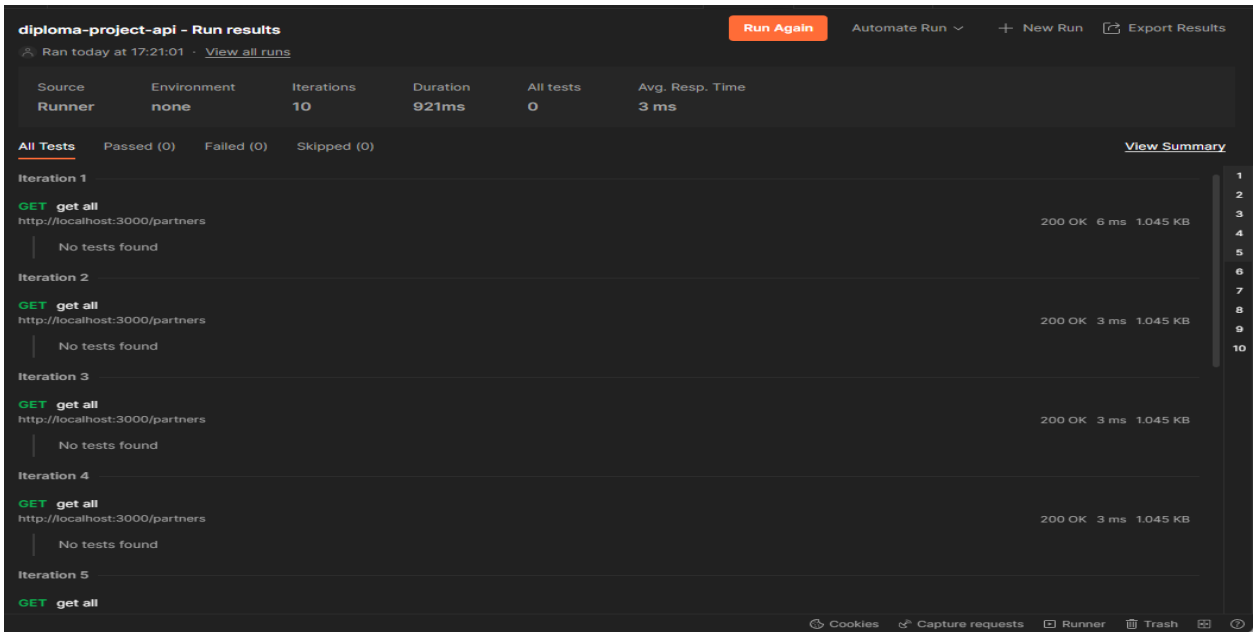


Рисунок 3.30 – Робота з HTTP-запитом для отримання всіх партнерів у Postman

Середній час виконання http-запиту для отримання всіх технологій (на головній сторінці) – 3 мс. Це є швидким показником роботи сервісу, що відповідає за функціонал отримання всіх технологій.

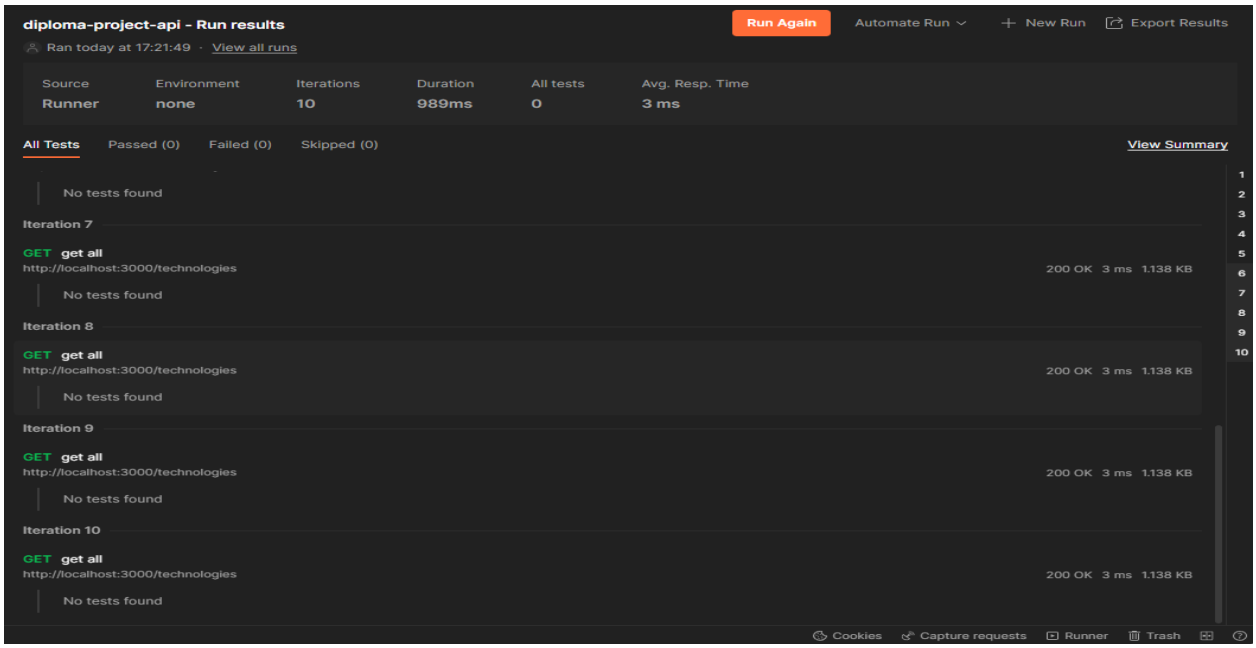


Рисунок 3.31 – Робота з HTTP-запитом для отримання всіх технологій у Postman

Середній час виконання http-запиту для отримання всіх контактів – 3 мс. Це є швидким показником роботи сервісу, що відповідає за функціонал отримання всіх контактів.

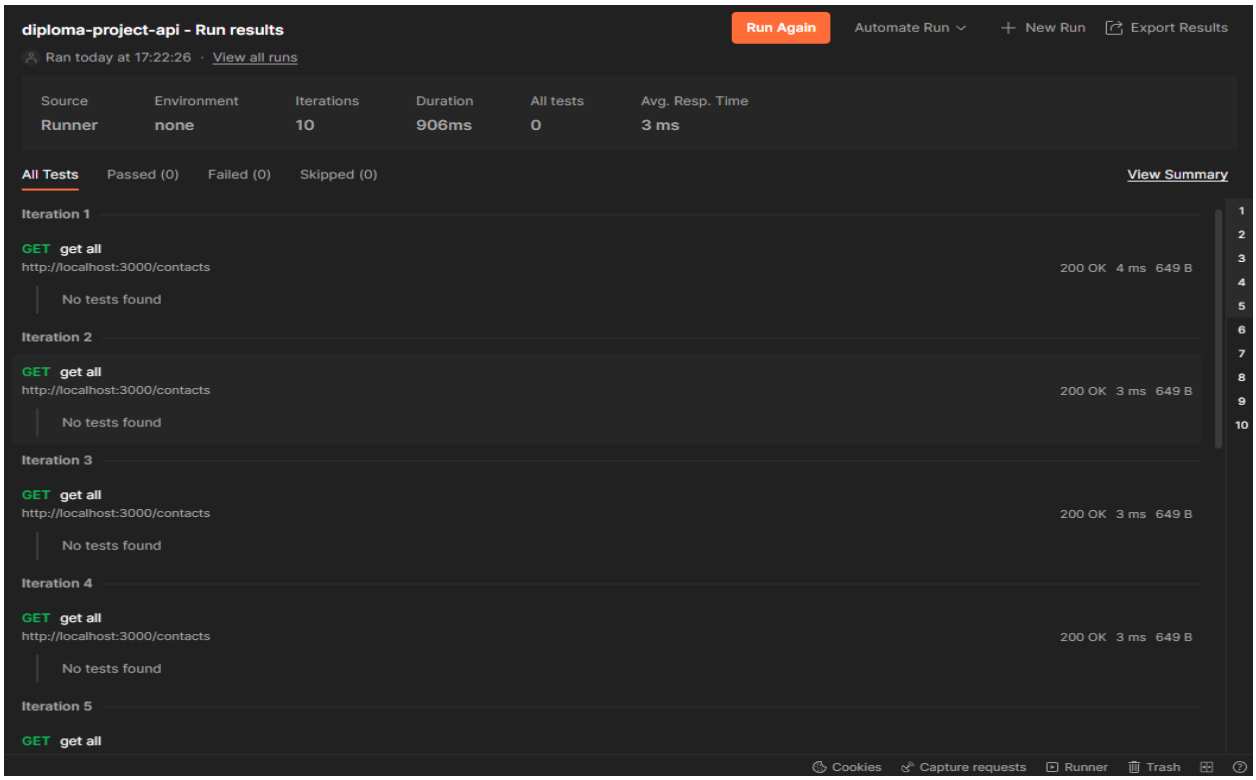


Рисунок 3.32 – Робота з HTTP-запитом для отримання всіх контактів у Postman

Час виконання кожного запиту був записаний і проаналізований для подальшого визначення можливих проблем або оптимізацій.

3.6 Висновки

У рамках практичної частини проекту було розглянуто декілька ключових аспектів розробки веб-додатків:

1. Розробка загальної архітектури проекту. В цьому етапі була проведена робота над визначенням загальної структури та організацією компонентів проекту. Було приділено достатню увагу плануванню та розподілу функціональності між клієнтською та серверною частинами додатку.

1. Розробка дизайну проекту. В цьому етапі було створено ефективний та зручний інтерфейс користувача, відповідний вимогам та потребам веб-додатку центру інформаційних технологій університету. Було

приділено увагу до візуального оформлення, навігації та зручності використання.

2. Клієнтська (front-end) частина проекту. В цьому етапі було проведено розробку клієнтської частини додатку з використанням Vue.js фреймворку. Були створені компоненти, забезпечена реактивність та зв'язування даних, що дозволило створити динамічний та ефективний інтерфейс користувача.

3. Серверна (back-end) частина проекту. В рамках цього етапу було розроблено серверну частину додатку з використанням фреймворку NestJS. Були налаштовані маршрутизація, з'єднання з базою даних та обробка запитів, що дозволило забезпечити правильну роботу додатку на серверній стороні.

4. Тестування. В цьому етапі було проведено тестування серверної частини додатку з використанням Postman. Були перевірені різні типи запитів та виміряно час їх виконання. Це дало можливість виявити та виправити різні помилки та неузгодженості в роботі серверної частини додатку.

ВИСНОВКИ

В ході розробки веб-додатку центру інформаційних технологій Університету митної справи та фінансів було виконано значну роботу зі створення загальної архітектури проекту, розробки дизайну, реалізації клієнтської (front-end) та серверної (back-end) частин проекту, а також проведення тестування.

Дизайн був розроблений з використанням бібліотеки для користувацьких інтерфейсів Ant Design. Ant Design складається з компонентів, що мають сучасний дизайн та добре поєднуються зі стилем сайтів для університетів.

Для розробки клієнтської частини застосунку був використаний JavaScript фреймворк – Vue.js.

Зв'язок бази даних із додатком та логіка роботи серверної частини були реалізовані на основі back-end фреймворку NestJS.

Розроблений проєкт було успішно протестовано з використанням програми для тестування API – Postman, що дало можливість оптимізувати запити до серверної частини та бази даних.

Для викладачів веб-застосунок дає можливість створювати новини, які відносяться до різних категорій, редагувати особливості, контакти, партнерів, технології тощо.

Для студентів цей додаток дає можливість переглянути головну сторінку (особливості, список технологій, партнерів), новини, та контакти інформаційного центру університету.

Отже, веб-додаток був успішно реалізований, враховуючи потреби та вимоги центру інформаційних технологій університету. Створений функціонал, який відповідає потребам студентів та викладачів, сприяє покращенню доступу до інформації та забезпеченню зручної взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Проценко М. М. Аналіз фреймворків як засобів розробки web-додатків. *Міжнародний науковий журнал "Інтернаука"*. 2016. №6(1). С. 86-89.
2. Фірсов О. Д., Ульяновська Ю. В., Мормуль М. Ф., Пікулін Д. О. Проектування архітектури веб-застосунку для імітаційного моделювання управління транспортними потоками. *Системи та технології.* 2022. №1(63). С. 70-87.
3. Зінченко А.Ю. Проектування розподілених інформаційних систем на основі використання технології слабозв'язаних компонентів. *Системи та технології*. 2022. №1(63). С. 5-14.
4. Поворознюк Н. І., Бобрівник К. Є., Грибков С. В. Проектування бази даних модуля студента у системі підтримки вивчення дисциплін. *Наукові праці Національного університету харчових технологій.* 2017. №23(1). С. 7-15.
5. Єпік М. О. Проектування бази даних інтелектуальної системи діагностики захворювань. *Вісник Херсонського національного технічного університету.* 2019. №2(69). С. 139-144.
6. Модель-вид-контролер — Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80 (дата звернення: 13.05.2023).
7. Клієнт-серверна архітектура — Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0 (дата звернення: 13.05.2023).

8. Components Basics | Vue.js. URL: https://vuejs.org/guide/essentials/component-basics.html (дата звернення: 13.05.2023).
9. Кейт Джонс. DOM Scripting: Web Design with JavaScript and the Document Object Model. / К. Джонс — Перше, 2005. — 368 с.
10. JavaScript — Вікіпедія. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення: 13.05.2023).
11. TypeScript — Вікіпедія. URL: https://uk.wikipedia.org/wiki/TypeScript (дата звернення: 13.05.2023).
12. Ant Design - The world's second most popular React UI framework. URL: https://ant.design/ (дата звернення: 13.05.2023).
13. Vue.js — Вікіпедія. URL: https://uk.wikipedia.org/wiki/Vue.js (дата звернення: 13.05.2023).
14. Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/ (дата звернення: 13.05.2023).
15. NestJS - A progressive Node.js framework. URL: https://nestjs.com/ (дата звернення: 14.05.2023).
16. SQL — Вікіпедія. URL: https://uk.wikipedia.org/wiki/SQL (дата звернення: 14.05.2023)
17. SQLite — Вікіпедія. URL: https://uk.wikipedia.org/wiki/SQLite (дата звернення: 14.05.2023)
18. Зовнішній ключ — Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%97%D0%BE%D0%B2%D0%BD%D1%96%D1%88%D0%BD%D1%96%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87 (дата звернення: 14.05.2023).

ДОДАТОК А

КОД FRONT-END ПРОЕКТУ

Link.vue

```
<template>
  <component
    :is="link ? 'a' : 'button'"
    class="ui-link"
    :href="link"
    target="_blank"
    rel="noopener noreferrer"
    :title="link"
  >
    {{ buttonText }}
  </component>
</template>
<script setup lang="ts">
import { computed } from 'vue';

const props = withDefaults(
  defineProps<{
    buttonText: string;
    href: string | null;
  }>(),
  {
    href: '',
  }
);

const link = computed<string | undefined>(() =>
```

```
  props.href?.length ? props.href : undefined
);
</script>
```

TextComponent.vue

```
<template>
  <div
    class="text custom-text-style-plain"
    v-if="type === 'html'"
    style="white-space: pre-wrap"
  >
    {{ text }}
  </div>
  <div
    v-else
    :class="`text custom-text-style-${type || 'plain'} ${
      color ? `${color}-text` : ''
    }`"
  >
    {{ text }}
  </div>
</template>
<script setup lang="ts">
defineProps({
  text: String,
  color: String,
  type: String,
});
</script>
```

LayoutManager.vue

```
<script setup lang="ts">
import { markRaw, ref, watch, type Component } from 'vue';
import { useRoute } from 'vue-router';

import { layouts } from '@/layouts/data';
import DefaultLayout from '@/layouts/Default/Index.vue';

const layout = ref<Component>();
const route = useRoute();

watch(
  () => route.meta?.layout as keyof typeof layouts,
  (metaLayout: keyof typeof layouts) => {
    try {
      const component = metaLayout && layouts[metaLayout];
      layout.value = markRaw(component || DefaultLayout);
    } catch (e) {
      layout.value = markRaw(DefaultLayout);
    }
  },
  { immediate: true }
);
</script>

<template>
  <component :is="layout"> <router-view /> </component>
</template>
```

Default.vue

```
<template>
  <a-layout>
    <Header />
    <a-layout-content>
      <router-view v-slot="{ Component }">
        <component :is="Component" />
      </router-view>
    </a-layout-content>
    <Footer />
  </a-layout>
</template>

<script lang="ts" setup>
import Footer from './extensions/Footer.vue';
import Header from './extensions/Header/Header.vue';
</script>
```

App.vue

```
<template>
  <LayoutManager />
  <notifications
    classes="vue-notification my-notification"
    position="top right"
  />
</template>

<script lang="ts" setup>
import LayoutManager from '@/layouts/LayoutManager.vue';
</script>
```

NotFound.vue

```
<template>
  <div class="not-found-page">
    <a-typography-title :level="2" class="title">
      Сторінка не знайдена!
    </a-typography-title>
    <Link
      button-text="Головна сторінка"
      href="/"
      target="_self"
      title="Перейти на головну сторінку"
    />
  </div>
</template>

<script lang="ts" setup>
import Link from '@/components/UI/Link.vue';
</script>

<style lang="scss" scoped>
.not-found-page {
  display: flex;
  padding: 100px 0;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  /* background-image: url(@/assets/img/not-found.jpg);
  background-size: cover;
  background-repeat: no-repeat;
  background-position: center left;
```

```
  height: 100%; */

  @media (max-width: 1400px) {
    padding: 50px 0;
  }

  @media (max-width: $max-width-tablet) {
    padding: 30px 0;
  }
}
</style>
```

ДОДАТОК Б

КОД BACK-END ПРОЕКТУ

main.ts

```
import { Logger, ValidationPipe } from '@nestjs/common';
import { NestFactory } from '@nestjs/core';
import { NestExpressApplication } from '@nestjs/platform-express';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import * as cookieParser from 'cookie-parser';

import { AppModule } from 'app.module';

async function bootstrap() {
  const origins: string[] = [
    process.env.CLIENT_URL,
    process.env.CLIENT_URL_LOCALHOST,
    process.env.SERVER_URL,
    // vite preview
    'http://127.0.0.1:4173',
  ];

  const app = await NestFactory.create<NestExpressApplication>(AppModule, {
    cors: {
      origin: function (origin, callback) {
        if (!origin || origins.indexOf(origin) !== -1) {
          callback(null, true);
        } else {
          callback(new Error('Not allowed by CORS'));
        }
      },
```

```
    credentials: true,
  },
});

const development = process.env.NODE_ENV === 'development';

console.log('development', development);
console.log('NODE_ENV', process.env.NODE_ENV);
console.log('CORS origins:', origins);

app.use(cookieParser());
app.useGlobalPipes(
  new ValidationPipe({
    transform: true,
    transformOptions: {
      enableImplicitConversion: true,
    },
  }),
);

const config = new DocumentBuilder()
  .setTitle('Diploma project API')
  .setDescription('Diploma project API')
  .setVersion('1.0')
  .addBearerAuth()
  .build();

const document = SwaggerModule.createDocument(app, config);
SwaggerModule.setup('api', app, document);
```

```
  const port = process.env.PORT || 3000;
  await app.listen(port);
  Logger.log(`The app is running on: http://localhost:${port}/api`);
}
bootstrap();
```

app.module.ts

```
import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { APP_GUARD } from '@nestjs/core';
import { ThrottlerGuard, ThrottlerModule } from '@nestjs/throttler';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';

import { UsersModule } from 'modules/users/users.module';
import { AuthModule } from 'modules/auth/auth.module';
import { FeaturesModule } from 'modules/features/features.module';
import { PartnersModule } from 'modules/partners/partners.module';
import { TechnologiesModule } from 'modules/technologies/technologies.module';
import { ContactsModule } from 'modules/contacts/contacts.module';
import { NewsCategoriesModule } from 'modules/news-categories/news-
categories.module';
import { NewsModule } from 'modules/news/news.module';

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      envFilePath: `.env.${process.env.NODE_ENV}`,
```

```
  }),
  ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'static'),
  }),
  ThrottlerModule.forRoot({
    ttl: 60,
    limit: 20,
  }),
  TypeOrmModule.forRootAsync({
    imports: [ConfigModule],
    useFactory: async (configService: ConfigService) => ({
      type: 'sqlite',
      database: configService.get<string>('DB_DATABASE'),
      autoLoadEntities: true,
      synchronize: process.env.NODE_ENV !== 'production',
    }),
    inject: [ConfigService],
  }),
  UsersModule,
  AuthModule,
  FeaturesModule,
  PartnersModule,
  TechnologiesModule,
  ContactsModule,
  NewsCategoriesModule,
  NewsModule,
],
controllers: [],
providers: [
  {
```

```
      provide: APP_GUARD,
      useClass: ThrottlerGuard,
    },
  ],
})
export class AppModule {}
```