Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Застосунок для автоматизованого керування персоналом з використанням ReactJS та NodeJS»

Виконав: студент 2 курсу, групи 1АКІТ-22м,
спеціальність 151– «Автоматизація та комп'ютерно-інтегровані технології».
_____ Юрій МЕЛЬНИК

Керівник к.т.н., доцент кафедри АІІТ
_____ Ярослав КУЛИК
«4» грудня 2023 р.

Опонент: к.т.н., доцент кафедри КН
_____ Руслан БЕЛЬЕЦЬКИЙ
«7» грудня 2023 р.

Допущено до захисту
Завідувач кафедри АІІТ

_____ д.т.н., проф. Олег БІСІКАЛО
«11» грудня 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти другий (магістерський)
Галузь знань 15 – Автоматизація та приладобудування
(шифр і назва)
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)
Освітня програма Інтелектуальні комп'ютерні системи
(назва освітньо-професійної програми)

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
20.09 2023 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Мельнику Юрію Валентиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Застосунок для автоматизованого керування персоналом з використанням ReactJS та NodeJS»
Керівник роботи Ярослав КУЛИК. к.т.н., доцент кафедри АІІТ
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
Затверджені наказом ВНТУ від «18» 09 2023р. № 247
Затверджене протоколом №1 засідання кафедри АІІТ від «30» 08 2023р.
2. Термін подання студентом роботи 05.12. 2023 р.
3. Вихідні дані до роботи: глобальна комп'ютерна мережа; операційна система MacOS; підтримка усіх доступних браузерів; протокол передачі даних HTTPS; фреймворк клієнта React JS та Ant Design; фреймворк серверу ExpressJS; середовище розробки Node.js;
4. Зміст текстової частини
Вступ. Дослідження аналогів. Порівняння методів. Вибір технологій. Порівняння методів розробки. Написання серверної частини. Створення клієнтського інтерфейсу. Економічна частина. Висновки. Список використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)

Діаграма розгортання, Діаграма роботи Node.js, Діаграма взаємодії вибраних технологій та користувача, Діаграма лінії життя Prisma OR Діаграма діяльності користувача із доступним функціоналом, Діагра класів.

6. Консультанти розділів роботи

Розділ	Ім'я, ПРІЗВИЩЕ та посада Консультанта	Підпис, дата	
		завдання видав	виконан прийня
Спеціальна частина	к.т.н., доцент кафедри АІТ Ярослав КУЛИК		
Економічна частина	к.е.н., доцент, Володимир КОЗЛОВСЬКИЙ	21.09.2023	06.12.20

7. Дата видачі завдання 21.09.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми МКР	22.09.23	29.09.23	Викон
2.	Дослідження аналогів. Порівняння методів	30.09.23	12.10.23	Викон
3.	Вибір технологій. Порівняння методів розробки	13.10.23	24.10.23	Викон
4.	Написання серверної частини	25.10.23	09.11.23	Викон
5.	Створення клієнтського інтерфейсу	10.11.23	20.11.23	Викон
6.	Економічна частина	21.11.23	01.12.23	Викон
7.	Оформлення матеріалів до захисту МКР	02.12.23	04.02.23	Викон

Студент _____ (підпис) Юрій МЕЛЬНИК (прізвище та ініціали)

Керівник роботи _____ (підпис) Ярослав КУЛИК (прізвище та ініціали)

АНОТАЦІЯ

УДК: 005.95

Мельник Ю.В. Застосунок для автоматизованого керування персоналом з використанням ReactJS та NodeJS. Магістерська кваліфікаційна робота зі спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології, освітня програма – Інтелектуальні комп’ютерні системи. Вінниця: ВНТУ, 2023. 157 с.

На укр.мові. Бібліогр.: 31 назва; рис.:48; табл.: 6.

У даній роботі було розглянуто проблематику даної сфери, а також дослідження інснуючих аналогів в Україні та світі, значення переваг та недоліків, спроектувано модель додатку, визначено технології для використання та послідовність його розробки, розроблено додаток для управління персоналом на основі бібліотеки React для клієнтської частини, Ant Design для стилізації, Node.js як середовище розробки, Express.js для серверної розробки з використанням Prisma ORM та бази даних SQLite. Також проведено аналіз витрат на розробку та оцінку економічного ефекту від можливої реалізації застосунку.

Ключові слова: управління, персонал, співробітники, користувач, технології.

ABSTRACT

Melnyk Y.V.. Application for automated personnel management using ReactJS and NodeJS. Master's thesis on specialty 151 - Automation and computer-integrated technologies, educational program - Intelligent computer systems. Vinnytsia: VNTU, 2023. 157 c.

In Ukrainian language. Bibliography: 31 titles; fig.:48; tabl.: 6.

In this work, the problems of this field were considered, as well as the research of existing analogues in Ukraine and the world, the importance of advantages and disadvantages, the application model was designed, the technologies for use and the sequence of its development were determined, an application for personnel management was developed based on the React library for the client part, Ant Design for styling, Node.js as development environment, Express.js for back-end development using Prisma ORM and SQLite database. An analysis of development costs and assessment of the economic effect of the possible implementation of the application was also carried out.

Keywords: management, personnel, employees, user, technologies.

.

ЗМІСТ

ВСТУП

Актуальність. Застосунки для керування персоналом залишаються актуальними в сучасному бізнес-середовищі. Ось деякі аспекти їх актуальності:

- **Автоматизація бізнес-процесів**: Застосунки для керування персоналом допомагають автоматизувати і оптимізувати багато рутинних завдань, пов'язаних з управлінням персоналом, такі як ведення списків працівників, ведення обліку робочого часу, обробка заявок на відпустки і оплату;
- **Ефективне управління даними**: Застосунки для керування персоналом дозволяють збирати і аналізувати дані про працівників, що допомагає приймати більш обгрунтовані рішення щодо управління персоналом;
- **Покращення комунікації**: Багато застосунків мають вбудовані інструменти для спілкування та співпраці між працівниками і керівництвом. Це сприяє покращенню комунікації в організації і зменшенню можливих недорозумінь;
- **Збереження даних та впорядкування інформації**: Застосунки дозволяють зберігати важливі документи та інформацію про працівників в одному місці, що робить процес управління персоналом більш ефективним і організованим;
- **Підвищення продуктивності**: Застосунки можуть надавати інструменти для відстеження продуктивності працівників і надавати рекомендації щодо їх покращення.

У світі, де бізнес-середовище постійно змінюється, і роль персоналу важлива для успішності організації, застосунки для керування персоналом лишаються актуальними та важливими інструментами для компаній будь-якого розміру і галузі. Вони допомагають підвищити ефективність управління персоналом і досягти стратегічних цілей бізнесу.

Мета – зменшення витрат часу на управління персоналом.

Об'єктом дослідження є управління працівниками організації, що включає в себе їхні особисті дані, контактну інформацію, кваліфікації, досвід та інші деталі.

Предметом дослідження є компанії, працівники та їх детальна інформація**.**

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

1. **Аналіз аналогів та порівняння методів**: Дослідження аналогів на ринку, які використовуються як системи для керування персоналом. Оцінка конкурентів та знаходження своєї конкурентної переваги;
2. **Вибір технологій**: Проведення аналіз сучасних методів розробки застосунків, особливостей та перспектив їх використання в технологіях, які стосуються web-додатків в галузі інформаційних систем та технологій. Порівняння переваг та недоліків технологій, які виконують власну функцію та на основі цього зробити висновок про вибір технологій для подальшої розробки;
3. **Створення запитів та зв'язку серверної та клієнтської частини**: Створення запитів для додавання співробітників, видалення зі списку та зміна інформації про співробітника з використанням Prisma як ORM (Object-Relational Mapping), серверної логіки за допомогою Express.js для обробки запитів від клієнтської сторони та Node.js як середовища виконання;
4. **Створення клієнтського інтерфейсу**: Розробка клієнтського інтерфейсу з використанням бібліотеки React, створення сторінок для перегляду списків співробітників, додавання в список, видалення зі списку та зміна інформації про співробітника, налаштування середовища розробки, React як клієнтського інтерфейсу.

Методи дослідження. компоненти web-систем, які забезпечують взаємозв'язок між їх частинами та розробку в області конструювання web-

додатків для подальшої побудови сервісу.

Науково-технічний результат:

Розроблено застосунок для керування персоналом, який допомагає автоматизувати і оптимізувати багато рутинних завдань, пов'язаних з управлінням персоналом, такі як ведення списків працівників, виведення детальної інформації по кожному з них та виконувати операції над цими даними.

Практичною цінністю даної роботи є застосунок для керування персоналом, який допомагає автоматизувати і оптимізувати багато рутинних завдань, пов'язаних з управлінням персоналом, такі як ведення списків працівників, виведення детальної інформації по кожному з них та виконувати операції над цими даними.

Апробація. Дана робота була апробована на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-24)» у 2023 році [3].

1 ДОСЛІДЖЕННЯ АНАЛОГІВ. ПОРІВНЯННЯ МЕТОДІВ

1.1 Проблематика створення застосунку

Ефективна функціонування будь-якої організації обумовлене різноманітними чинниками, проте одним із найсуттєвіших є належна організація системи управління персоналом.

На теренах колишнього Радянського Союзу розвиток цього підходу відбувався затримано в порівнянні з заходом. З матеріальних причин багато компаній дотримувалися формату, де керівник відділу особисто відповідає за управління своїм персоналом, або де керівництво концентрується на потребах і продуктивності лише свого відділу. Однак це призводило до деградації корпоративної культури, втрати мотивації у співробітників та загального зниження ефективності праці, породжуючи напружені відносини в колективі [2].

Зараз розв'язання проблеми стає очевидним: впровадження раціональної системи управління персоналом, за якою відповідають кваліфіковані фахівці. Під управлінням персоналу розуміється спрямована на керівництво людьми діяльність з метою втілення проєктів організації, з врахуванням акценту на працю, досвід та таланти працівників, а також на їхнє задоволення від власної роботи. Управління персоналом – це не лише строга регламентація дій, але й вміння взаємодіяти з працівниками, поважати їх, мотивувати та направляти їхню працю на користь компанії.

В управлінській сфері часто використовують два підходи: "управління персоналом" і "управління людськими ресурсами". Важливо зазначити, що ці концепції відрізняються одна від одної. Перше визнає людину як самостійну особистість зі своїми потребами і правами, при цьому враховуючи інтереси персоналу при встановленні комерційних цілей [3-4].

– При відборі персоналу приділяється увага не лише їхнім професійним навикам, але й особистим якостям;

- Колектив структурується так, щоб зберігати баланс між молодими фахівцями та тими, хто має вже досвід;
- Діяльність співробітників систематично оцінюється, а на підставі цих оцінок розробляються стратегії для їхнього професійного зростання та розвитку;
- Заохочується здорова і прозора конкуренція, що сприяє розвитку кадрового потенціалу компанії;
- В організації панує довіра до кожного співробітника, проте існує система адекватної перевірки результатів їхньої роботи;
- Компанія завжди готова до ситуацій, коли когось із співробітників не може бути на робочому місці, і може швидко здійснити заміщення.
- Для кожного працівника передбачені можливості для підвищення його професійної кваліфікації;
- Усі кадрово-управлінські процеси компанії відповідають чинним правовим актам.

Управління персоналом є складним і системним процесом, а сучасні фахівці визнають наявність значної кількості моделей цієї діяльності [5].

Стиль управління напряму впливає на розвиток співробітників усередині компанії, на успішність організації та на її корпоративну культуру і утримання фахівців. Процес управління включає функції планування своїх та підлеглих завдань, організацію роботи команди, керівництво та контроль з урахуванням досягнення цілей. Стиль управління персоналом – це спосіб, яким менеджер працює для досягнення мети та виконання своїх обов'язків. Управління співробітниками – це складний процес, який охоплює багато психологічних аспектів.

Ефективний менеджер – це той, хто може адаптувати свій спосіб управління залежно від різних факторів, зберігаючи при цьому увагу на успішному досягненні своїх цілей. Фактори, що впливають на стилі управління, включають внутрішні чинники, такі як організаційна культура,

політика компанії, пріоритети та рівень кваліфікації персоналу, і зовнішні чинники, такі як закони про зайнятість, економічні умови, конкуренція та вплив постачальників та споживачів.

Згідно з дослідженням Gollup, тільки 30% співробітників в США працюють продуктивно, а в усьому світі цей показник становить лише 13%. Даний дослідження також підтверджує, що майже 70% розбіжностей у відгуках про залученість співробітників пов'язані із стилем управління їхніх керівників.

Отже, низька залученість співробітників має негативний вплив на декілька аспектів:

- Зниження продуктивності;
- Зниження якості кінцевого результату роботи;
- Підвищення плинності персоналу;
- Підвищення відсотків прогулів;
- Зниження рентабельності.

Суть полягає в тому, що неправильне управління може демотивувати співробітників, що призводить до зниження їхньої результативності, продуктивності та збільшення обороту персоналу. Щоб запобігти цьому, важливо вибрати відповідний стиль менеджменту [6].

Застосунок для керування співробітниками є програмним забезпеченням, розробленим для допомоги підприємствам та організаціям у керуванні їхнім персоналом та людськими ресурсами. Цей застосунок може мати різні функції та можливості, залежно від потреб конкретного бізнесу, але основна мета полягає в автоматизації та оптимізації процесів управління персоналом.

Керування співробітниками - це складне завдання, і воно може стикатися з численними проблемами. Деякі з найрозповсюдженіших проблем у цій галузі включають:

- **Неефективна комунікація:** Недостатня або неефективна

комунікація може призвести до непорозумінь, конфліктів та низької продуктивності. Важливо встановлювати чіткі та відкриті лінії зв'язку в організації;

- **Несумісність цілей:** Коли цілі співробітників не співпадають з цілями організації, це може призвести до конфліктів та низької мотивації. Важливо встановити зв'язок між особистими та корпоративними цілями;
- **Відсутність мотивації:** Співробітники можуть втратити мотивацію через низьку заробітну плату, невизнання їхнього внеску або невірну організаційну культуру;
- **Конфлікти в колективі:** Внутрішні конфлікти між співробітниками або командами можуть сприймати продуктивність та створювати негативну атмосферу в організації;
- **Відсутність розвитку:** Співробітники можуть втрачати інтерес до роботи, якщо не надається можливість розвивати свої навички та здійснювати кар'єрний ріст;
- **Непрацюючі системи управління персоналом:** Застосунки та процеси управління персоналом можуть бути застарілими або неефективними, що ускладнює управління співробітниками.
- **Висока текучість кадрів:** Часта зміна співробітників може призвести до втрати досвіду і накладати фінансові та часові витрати на пошук і найм нових працівників;
- **Недостатність оцінки та зворотного зв'язку:** Відсутність процесу оцінки роботи співробітників та зворотного зв'язку може призвести до низької якості роботи і невірного розвитку кар'єри;
- **Незабезпеченість безпеки праці:** Невідповідність нормам безпеки на робочому місці може призвести до травм та порушень законодавства;
- **Зміни в організації:** Реорганізація, об'єднання або зміни в

керівництві можуть вплинути на співробітників та створити невизначеність.

Створення застосунка для керування співробітниками може мати багато переваг для підприємства та його керівництва [7-9]. Ось деякі з основних причин, навіщо це може бути корисним:

- **Ефективність та оптимізація процесів**: Застосунок для керування співробітниками може допомогти автоматизувати багато рутинних задач, що пов'язані з управлінням персоналом, таких як реєстрація годин роботи, облік відпусток, управління звільненням та інше. Це дозволяє підприємству ефективно використовувати ресурси та збільшити продуктивність;
- **Збільшення залученості співробітників**: Застосунок може надати співробітникам більше контролю над їхніми особистими даними, робочим графіком, відпустками та іншими аспектами їхньої робочої діяльності. Це може підвищити їхню задоволеність роботою та залученість;
- **Аналітика та звітність**: Застосунок може надавати керівництву доступ до важливої інформації про робочий процес, такої як витрати на персонал, продуктивність, відомості про вакансії та інше. Це може допомогти приймати більш обгрунтовані рішення щодо управління персоналом;
- **Забезпечення безпеки та дотримання нормативів**: Застосунок може допомагати в забезпеченні безпеки праці та дотриманні всіх законодавчих нормативів, пов'язаних із зайнятістю та умовами праці;
- **Зручність та доступність**: Застосунок дозволяє керівництву та співробітникам отримувати доступ до важливої інформації з будь-якого місця та в будь-який час, що особливо важливо в умовах віддаленої роботи або великої кількості філіалів.

Загалом, створення застосунку для керування співробітниками може покращити ефективність та продуктивність підприємства, полегшити роботу

керівництва та співробітників, і покращити загальну якість управління персоналом.

Аналогів застосунку для керування співробітниками існує велика кількість, оскільки кожен з них вирішує свою проблему з власним функціоналом.

Причини виникнення аналогів:

- **Різноманітність бізнес-потреб**: Різні компанії мають різні потреби у керуванні своїми співробітниками в залежності від розміру, галузі, географічного розташування та інших факторів. Щоб задовольнити різні потреби, існує багато програм, які спеціалізуються на певних аспектах управління персоналом;
- **Інновації і технологічний прогрес**: Ринок програм для управління співробітниками постійно змінюється і розвивається. Разом з тим розвиваються і нові технології, що дають можливість розробляти більш ефективні та функціональні рішення;
- **Конкуренція**: Конкуренція між розробниками програм для управління співробітниками сприяє інноваціям і покращенню функціональності. Це означає, що компанії намагаються пропонувати кращі рішення та конкурувати за клієнтів;
- **Специфічні потреби галузей**: Деякі галузі мають унікальні вимоги до управління співробітниками. Наприклад, галузі з високим рівнем регулювання, такі як фармацевтика або фінанси, можуть потребувати спеціалізованих рішень для забезпечення дотримання нормативів;
- **Географічна розташованість та масштаби підприємства**: Великі міжнародні компанії можуть вимагати систем, які можуть легко масштабуватися і працювати в різних частинах світу з урахуванням місцевих законів та культурних відмінностей [10].

З цього аналізу проблематики, можна зробити висновок, що дана проблема є серйозною і впливає на роботу всіх співробітників, що впливає на роботу компанії. В залежності від кількості співробітників, масштабів впливу

компанії, географічного положення та інших чинників, існує багато застосунків, сервісів та додатків, що повністю чи частоково вирішують дану проблему.

1.2 Дослідження аналогів

На сучасному ринку існує безліч сервісів для керування персоналом, які можуть бути аналогами один одного або мати свої унікальні особливості. Ось декілька популярних аналогів сервісів для керування персоналом:

- BambooHR;
- Workday;
- Zenefits;
- Gusto.

На українському ринку також існують сервіси для керування персоналом і HR-рішення, які призначені для організацій українського бізнес-середовища. Ось деякі з них:

- Factorial;
- Beekeeper;
- PeopleForce;
- Hurma;
- HR.amoCRM.

Всі ці аналоги були розроблені великими компаніями та корпораціями для керуваннння співробітниками у власній мережі. Для малих або середніх компаній подібних аналогів немає на українському ринку, а дозволити собі сервіси, які розраховані на великі компанії вони не можуть, оскільки це дорого і користуватися таким сервісом – невигідно з боку бюджету таких компаній. Дані аналоги використовують CRM-системи для контролю та створеннння даних, але ці системи використовують багато ресурсів, потрібен стабільний

високошвидкісний доступ до інтернету, потребують частої підтримки та дорогі в обслуговувані [8-10].

1.3 Порівняння застосунків

Для вибору оптимальних технологій для подальшої розробки потрібно розібрати переваги та недоліки світових та українських серсів для керування персоналом.

BambooHR – це приклад хмарної системи для управління персоналом, яка включає в себе інструменти для найму, звільнення, відпусток, атестації та аналітики. Найпопулярний в світі аналог.

Використовують компанії:

- Overstock.com;
- Foursquare;
- ClassPass;
- Patreon;
- Freshbooks.

Переваги BambooHR:

- Хмарне рішення;
- Інтуїтивний інтерфейс;
- Набір функцій для HR-управління.

Недоліки BambooHR:

- Вартість;
- Потреба в Інтернет-підключенні;
- Можливі обмеження для великих компаній;
- Інтеграція з іншими системами може бути складною.

WorkDay – система, яка пропонує більш широкий спектр рішень для управління персоналом, включаючи грошовий обіг, фінанси, робочий час та

інші функції.

Використовують компанії:

- Amazon;
- IBM;
- Target;
- HP Inc.;
- Procter & Gamble.

Переваги WorkDay:

- Хмарне рішення;
- Висока ступінь масштабованості;
- Широкий функціональний спектр для управління персоналом та фінансами;
- Інтуїтивний інтерфейс.

Недоліки WorkDay:

- Високі витрати на впровадження та підтримку;
- Складність інтеграції з існуючими системами;
- Залежність від Інтернет-підключення для доступу до системи;
- Може бути надто потужним для менших компаній.

Zenefits – сервіс, що спеціалізується на управлінні персоналом, включаючи підбір, страхування, оплату та інші HR-функції.

Використовують компанії:

- AgencyEA;
- Coney island prep;
- Firacley tile;
- Lensrentals;
- meUdines.

Переваги Zenefits:

- Хмарне рішення;
- Інтуїтивний інтерфейс;

– Можливість інтеграції з іншими системами.

Недоліки Zenefits:

– Складність налаштування та впровадження для деяких організацій;
– Може не відповідати всім потребам великих підприємств зі складною структурою;
– Потребує доступу до Інтернету для роботи.

Gusto – це рішення призначене для малих і середніх підприємств і включає в себе управління заробітною платою, податки, відпустки і інші аспекти управління персоналом.

Використовують компанії:

– NerdWallet;
– TechRepublic;
– U.S.News;
– SoftwareReviews.

Переваги Gusto:

– Хмарне рішення;
– Інтуїтивний інтерфейс;
– Широкий функціонал для управління заробітною платою, обліком кадрів та оподаткуванням;
– Можливість інтеграції з іншими програмами та послугами.

Недоліки Gusto:

– Вартість може бути високою для деяких користувачів;
– Можливість інтеграції з іншими системами може бути обмеженою у порівнянні з іншими рішеннями;
– Деякі функції менш розвинуті, ніж у деяких конкурентів;
– Вимагає доступу до Інтернету для роботи.

Factorial – цей хмарний сервіс надає рішення для управління персоналом, включаючи облік годин роботи, відпустки, найм, звільнення та інші HR-процеси.

Використовують компанії:

- Talent[ly];
- American Sailing;
- Extropy;
- ACL.

Переваги Factorial:

- Хмарне рішення;
- Інтуїтивний інтерфейс;
- Набір функцій для управління персоналом, включаючи облік кадрів, найм, звільнення, відпустки та інше;
- Здатність інтеграції з іншими системами та додатками.

Недоліки Factorial:

- Вартість може бути високою для деяких користувачів;
- Можливі обмеження для великих підприємств зі складною структурою;
- Залежність від Інтернет-підключення для доступу до системи;
- Інтеграція з деякими іншими системами може бути складною.

Beekeeper – мобільне рішення для спілкування та співпраці між співробітниками, а також включає в себе елементи управління персоналом.

Використовують компанії:

- Pike;
- ABB;
- Flagger Force;
- Cargill.

Переваги Beekeeper:

- Мобільна орієнтація;
- Зручний месенджер;
- Інтеграція з іншими системами;
- Захист даних.

Недоліки Beekeeper:

- Вартість;
- Залежність від Інтернет-підключення;
- Проблеми з інтеграцією;
- Багато непотрібних функцій, що дизорієнтують в інтерфейсі.

PeopleForce – це хмарна платформа для HR-управління, яка включає в себе інструменти для найму, звільнення, обліку робочого часу та інше.

Використовують компанії:

- Work.ua;
- Robota.ua.

Переваги PeopleForce:

- Хмарне рішення;
- Інтуїтивний інтерфейс;
- Набір функцій для управління персоналом, включаючи облік кадрів, найм, звільнення, відпустки та інше;
- Можливість інтеграції з іншими системами та додатками.

Недоліки PeopleForce:

- Вартість може бути високою для деяких користувачів;
- Залежність від Інтернет-підключення для доступу до системи;
- Інтеграція з деякими іншими системами може бути складною;
- Може бути обмежений функціонал для великих компаній зі складною структурою.

HURMA - це електронна платформа, яка надає широкий спектр функцій для управління та автоматизації бізнес-процесів в галузі HRM (управління людськими ресурсами).

Використовують компанії:

- Nova Poshta;
- Ezdorovya;
- Amomedia;

– Uniweb;

– Віяр.

Переваги HURMA:

– Автоматизація HR-процесів;

– Залучення співробітників до управління;

– Аналітика та звітність;

– Система безпеки та конфіденційності;

– Широкий функціонал.

Недоліки HURMA:

– Вартість;

– Складність внедрення;

– Не підходить для всіх компаній;

– Потребує навчання.

HR.amoCRM – це один з модулів CRM-системи amoCRM, який надає інструменти для управління персоналом та набор функцій для найму та обліку кадрів.

Використовують компанії:

– Horoshop;

– Rozetka;

– Prom;

– Weblium;

– PrestaShop.

Переваги HR.amoCRM:

– Інтеграція з CRM-системою amoCRM;

– Управління персоналом, навчання та розвиток співробітників;

– Облік кадрів та структура організації;

– Пошук та найм нових співробітників;

– Можливість аналізу даних та статистики.

Нелоліки HR.amoCRM:

- Залежність від CRM-системи amoCRM;
- Можливі обмеження у функціоналі порівняно зі спеціалізованими системами управління персоналом;
- Потребує доступу до Інтернету для роботи;
- Може вимагати додаткової настройки та інтеграції для вирішення конкретних бізнес-задач.

1.4 Висновки

Використовуючи інформацію, викладену в даному розділі, можна зробити висновок, що аналогів на світовому та українському ринку багато, їх основі переваги та недоліки дуже схожі, а саме:

Переваги:

- Інтуїтивний інтерфейс;
- Широкий набір функцій;
- Використовують хмарні технології.

Недоліки:

- Висока ціна;
- Багато сторонніх функцій, які не такі важливі для управління персоналом;
- Залежність від інтернет-з’єднання.

З цього переліку можна зробити висновок, що дані аналоги розроблені спеціально для великих компаній та корпорацій. Для малих або середніх компаній подібних аналогів немає на українському ринку, а дозволити собі сервіси, які розраховані на великі компанії вони не можуть, оскільки це дорого і користуватися таким сервісом – невигідно з боку бюджету таких компаній. Дані аналоги використовують CRM-системи для контролю та створенння даних, але ці системи використовують багато ресурсів, потрібен стабільний

високошвидкісний доступ до інтернету, потребують частої підтримки та дорогі в обслуговувані. Для малих або середніх компаній потрібно застосовувати технології, які менше потребують ресурсів для використання, запуску та зберігання даних, такі як бібліотеки та фреймворки, оскільки вони теж забезпечують зручність використання, але вони легше підтримуються та не потрібно платити за послуги, оскільки весь функціонал йде безкоштовно.

2 ВИБІР ТЕХНОЛОГІЙ. ПОРІВНЯННЯ МЕТОДІВ РОЗРОБКИ

У даному розділі було проведено аналіз сучасних методів розробки застосунків, особливостей та перспектив їх використання в технологіях, які стосуються web-додатків є актуальною задачею в галузі інформаційних систем та технологій. Основним та одним з найбільш популярним та зручним методом розробки є фреймоврки. З результатів дослідження в попередньому розділі, можна зробити висновок, що саме фреймворки є найбільш зручним та корисним інструментом для побудови застосунку для керування співробітниками.

2.1 Вибір мови програмування

На клієнтській частині для реалізування логіки вибір є лише між 2 мовами програмування: JavaScript та TypeScript. Для вибору мови потрібно розглянути переваги та недоліки кожної з мов.

Переваги JavaScript:

- **Популярність**: JavaScript є однією з найпопулярніших мов програмування, і має велике та активне спільноту розробників. Це означає, що ви знайдете велику кількість ресурсів та бібліотек для використання;
- **Виконання на клієнті**: JavaScript виконується на стороні клієнта (у браузері), що дозволяє реалізовувати інтерактивність та динаміку на веб-сайтах без необхідності відправляти дані на сервер;
- **Асинхронність**: JavaScript має підтримку асинхронного програмування, що дозволяє взаємодіяти з серверами та іншими ресурсами без блокування інтерфейсу користувача;
- **Зручність для веб-розробки**: JavaScript добре підходить для веб-розробки і може бути використаний для створення різних елементів,

таких як веб-сторінки, веб-додатки, ігри, анімація та багато інших.

Недоліки JavaScript:

- **Сумнівна якість коду**: JavaScript не накладає обмежень на структуру коду, що може призвести до менш читабельного та підтримуваного коду, особливо в великих проєктах;
- **Варіабельна реалізація в браузерах**: Хоча мова JavaScript стандартизована, різні браузери можуть мати власні реалізації, що вимагає додаткового завдання вирішення проблем сумісності;
- **Безпека**: JavaScript виконується в браузері користувача, що може бути використано для атак, таких як впровадження зловмисного коду (крос-сайтові атаки);
- **Обмежена можливість доступу до ресурсів ОС**: JavaScript у браузері має обмежений доступ до ресурсів операційної системи, що обмежує можливість виконання деяких завдань[23].

Переваги TypeScript:

- **Статична типізація**: TypeScript дозволяє визначати типи для змінних, аргументів функцій і об'єктів. Це допомагає виявляти помилки на етапі розробки та полегшує підтримку коду, особливо великих проєктів;
- **Підсилення інструментів розробки**: Більшість сучасних IDE та текстових редакторів підтримують TypeScript і надають автодоповнення, перевірку типів та інші корисні функції;
- **Читабельність коду**: Додавання типів до коду може покращити читабельність та зрозумілість коду для команди розробників;
- **Підтримка ECMAScript**: TypeScript підтримує всі функції JavaScript, а також нові функції та стандарти ECMAScript. Ви можете використовувати сучасний синтаксис JavaScript разом із статичною типізацією;
- **Сумісність з JavaScript**: Ви можете використовувати існуючий

JavaScript-код у TypeScript-проєктах, що полегшує перехід на TypeScript.

Недоліки TypeScript:

– **Додатковий обсяг коду**: Додавання типів може призвести до додаткового обсягу коду, що може бути заважаючим в невеликих проєктах;
– **Вивчення кривої**: Новачкам може знадобитися час, щоб вивчити TypeScript та набути досвіду в створенні і використанні типів;
– **Оновлення типів бібліотек**: Інколи оновлення типів для сторонніх бібліотек може бути неспростуваною задачею, особливо якщо бібліотеки не підтримуються активно;
– **Можлива перевірка на виконання**: Якщо ви використовуєте TypeScript, це не гарантує, що всі помилки будуть виявлені на етапі виконання.

Діаграма популярності мов програмування в 2023 році зображено на рисунку 2.1.

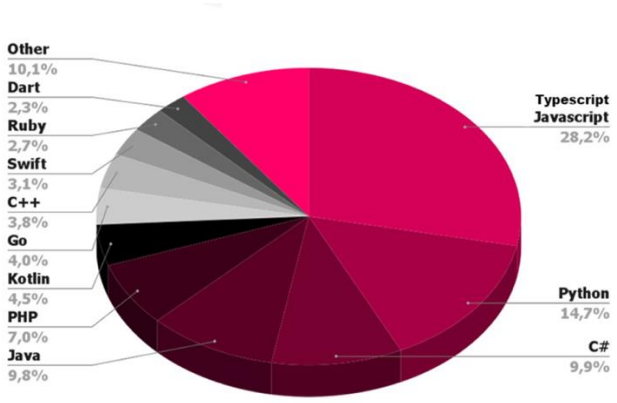

Рисунок 2.1 – Діаграма популярності мов програмування в 2023 році

З порівнянь переваг та недоліків даних мов програмування можна зробити висновок, що для виконання даної робота краще підійде TypeScript для написання клієнтських компонентів, оскільки статична типізація виявляють потенційні помилки на етапі розробки та більшість сучасних IDE

та текстових редакторів підтримують TypeScript і надають автодоповнення, перевірку типів та інші корисні функції. Для файлів, які містять логіку, яка буде виконуватися на сервері, краще використовувати звичайний JavaScript, тому що функції, які використовуються в цих фреймворках, вже типізонані і створювати додаткове навантаження, таке як типізація, не вважається за потрібне.

2.2 Використання фреймворків для розробки

Фреймворк (в інформаційних системах) – це структура програмної системи, що полегшує розробку і об'єднання різних компонентів великого програмного проєкту.

Фреймворки одразу адаптуються під розширення екрану, мають бібліотеки готових компонентів, приблизний час створення сайту – 5-6 годин.

Оскільки більшість функцій отримуємо вже готовими, розробник витрачає менше часу, тому може більше сконцентруватися на клієнтській частині, тим саме збільшуючи комфортність при використанні.

На даний час ця технологія достатньо швидко розвивається і фрейморки почали об’єднувати в стеки.

Прикладами стеків фреймворків є MERN, MEVN, MEAN [11]. Створенням сайту за допомогою стеків займає більше часу, оскільки різні фреймворки мають свої особливості – 1-2 дні.

Стеки потребують більше ресурсів, їх важче підтримувати та налаштувати між собою технології може зайняти багато часу.

З боку клієнта - це велика функціональність на сайті, зручність та швидкість користування, збільшуючи цим його бажання залишитися саме на цьому сайті.

З боку розробника - велика варіативність розташування та використання компонентів саме в тих місцях, де він чи клієнти вважають за потрібне [12-16].

Переваги:

– стандартна структура;
– можна застосовувати, коли присутня нестандартна логіка чи структура;
– швидкість роботи;
– витримують більше навантаження;
– високий рівень безпеки.

Недоліки:

– суворі правила використання;
– кожен фреймворк має власний стиль написання.

Діаграма популярності фреймворків в 2023 році зображено на рисунку 2.2.

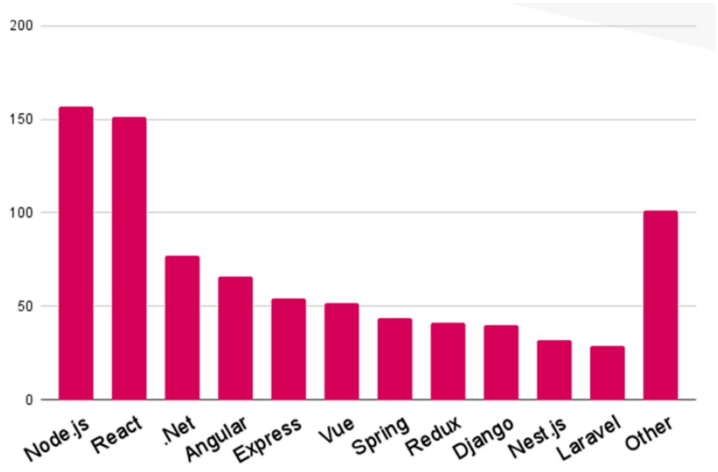

Рисунок 2.2 – Діаграма популярності фреймворків в 2023 році

2.3 Вибір фреймворка для клієнтської частини

Аналогів фреймворків для клієнтської частини є багато, але вони більшість схожі один на одного, оскільки розробники фреймворків надихалися один одним при створенні.

Ось найвідоміші клієнтські фреймворки і бібліотеки для розробки:

- **React**: Розроблений Facebook, React — це фреймворк для створення користувацьких інтерфейсів. Він дозволяє створювати компоненти, які можуть оновлюватися в реальному часі при зміні даних [27-30];
- **Angular**: Розроблений Google, Angular — це фреймворк для створення масштабованих веб-додатків. Він надає набір інструментів для розробки, включаючи маршрутизацію, управління станом та багато інших функцій;
- **Vue.js**: Цей фреймворк дозволяє створювати користувацькі інтерфейси та додавати інтерактивність до веб-додатків. Він також здатний управляти станом додатку та компонентами.

Є ще інші фреймворки, але вони програють в зручності, швидкості написання та розумінні побудови архітектури [13].

Переваги React:

- **Популярність**: React – один з найпопулярніших JavaScript-фреймворків для створення користувацьких інтерфейсів [28];
- **Гнучкість**: React є бібліотекою, а не фреймворком, що означає, що ви можете вибирати інші бібліотеки та інструменти, щоб доповнити його функціональність;
- **Компонентний підхід**: React сприяє створенню повторно використовуваних компонентів, що спрощує розробку та обслуговування коду;
- **Велика спільнота**: Має широку спільноту розробників та багато ресурсів, документацію і сторонніх бібліотек [29-30];
- **Підтримка CSS-in-JS та інших стилізаційних рішень**: React легко інтегрується з різними методами стилізації;
- **Швидка розробка**: Можливість автоматичної підтримки Hot Module Replacement (HMR) спрощує процес розробки.

Недоліки React:

– **Велика кількість виборів**: З великою свободою вибору можуть виникнути проблеми вибору найкращого підходу для вашого проєкту.

Слід зауважити, що всі недоліки стосуються лише розробників, а не клієнтів.

Переваги Angular:

– **Двостороннє зв'язування (Two-Way Data Binding)**: AngularJS забезпечує потужний механізм для автоматичного оновлення інтерфейсу користувача під час зміни даних моделі і навпаки;
– **Модульність**: AngularJS дозволяє розбити ваш додаток на окремі модулі, що спрощує організацію коду та сприяє повторному використанню компонентів;
– **Багатофункціональність**: Фреймворк надає багато готових рішень, таких як маршрутизація, валідація форм, обробка подій тощо.
– **Спільнота та ресурси**: У часи активного розвитку AngularJS існувала велика спільнота розробників та багато ресурсів, таких як документація та бібліотеки;

Недоліки Angular:

– **Великий обсяг коду**: AngularJS має велику кількість бібліотек, що призводить до збільшення обсягу завантаженого коду для користувачів і більшого часу на завантаження сторінок;
– **Застаріла технологія**: AngularJS був розроблений до появи сучасних стандартів веб-розробки, і він не враховує багатьох передових технік і підходів;
– **Підтримка і безпека**: Офіційна підтримка AngularJS завершилася, що може призвести до проблем із безпекою та сумісністю у майбутньому;
– **Важкість міграції**: Перехід з AngularJS на більш сучасний фреймворк може бути складним і витратним процесом через значні різниці у структурі та підходах.

З огляду на те, що AngularJS більше не підтримується, розробники зараз віддають перевагу сучасним фреймворкам, таким як Angular (без "JS"), React, або Vue.js, які надають більше можливостей і більш сучасну архітектуру для веб-розробки.

Переваги Vue.js:

- **Легкий для вивчення**: Vue.js має дуже дружній API та документацію, що робить його легким для вивчення, навіть для початківців у веб-розробці;
- **Компонентна архітектура**: Vue.js активно сприяє розділенню веб-додатків на компоненти, що полегшує розробку, підтримку та повторне використання коду;
- **Технологія двостороннього зв'язування**: Vue.js надає можливість просто створювати двостороннє зв'язування даних між моделлю та представленням, що полегшує взаємодію з інтерфейсом користувача;
- **Можливості директив**: Vue.js має багатий набір вбудованих директив, таких як v-for, v-if, і v-bind, що спрощують маніпуляцію DOM і взаємодію з компонентами.

Недоліки Vue.js:

- **Менший екосистема порівняно з React і Angular**: У порівнянні з React і Angular, екосистема Vue.js може бути менш розгалуженою, і для певних завдань може знадобитися створити власні рішення або використовувати менш популярні бібліотеки;
- **Менше великих підприємств і проєктів**: Хоча Vue.js активно використовується в різних проєктах, він може не мати таку широку підтримку великих корпорацій, як Angular або React;
- **Обмеження в можливостях SSR та SSG**: У порівнянні з Next.js, Vue.js має менше можливостей для рендерингу на стороні сервера (SSR) і статичної генерації (SSG);
- **Менше інструментів для станових управлінців**: В реактивному

стилі Vue.js для керування станом використовується Vuex, але інші фреймворки, такі як React, можуть мати більше інструментів та бібліотек для цієї цілі.

Слід зауважити, що React включає в себе рендер на стороні клієнта (CSR, Client-Side Rendering), а не на стороні сервера (SSR, Server-Side Rendering), як відбувається на сайтах, які використовують сайти на мові програмування PHP.

Переваги CSR над SSR:

- Швидкість і відзивчивість: CSR може бути швидшим у порівнянні з SSR, оскільки весь процес рендерингу відбувається на боці клієнта, і не потрібно чекати на відповідь від сервера для кожного запиту [27];
- Більша інтерактивність: CSR дозволяє створювати більш інтерактивні і динамічні сторінки, так як JavaScript може взаємодіяти зі сторінкою після завантаження;
- Зменшення серверного навантаження: Оскільки рендеринг відбувається на клієнтському боці, SSR може зменшити навантаження на сервер та покращити масштабованість;
- Кешування сторінок: CSR може легше використовувати кешування на стороні клієнта, що допомагає зменшити час завантаження для повторних відвідувачів;
- Більша гнучкість у виборі технологій: CSR дозволяє розробникам використовувати будь-які фреймворки або бібліотеки JavaScript для реалізації фронтенду;
- Можливість перевірки сторінок на SEO: CSR може бути легше налаштованим для оптимізації для пошукових систем (SEO) за допомогою певних технік, таких як prerendering.

Зазначені переваги CSR можуть бути особливо корисними для сучасних додатків, які вимагають більшої інтерактивності та швидкості завантаження. Сайти, написані на React, не потребують перезавантаження під час переходу

між сторінками, оскільки даний тип розробки використовує технологію SPA (Single Page Application), який дає можливість переходити між сторінками без їх перезавантаження.

Графік зростання популярності технологій для розробки клієнтської частини зображено на рисунку 2.3.

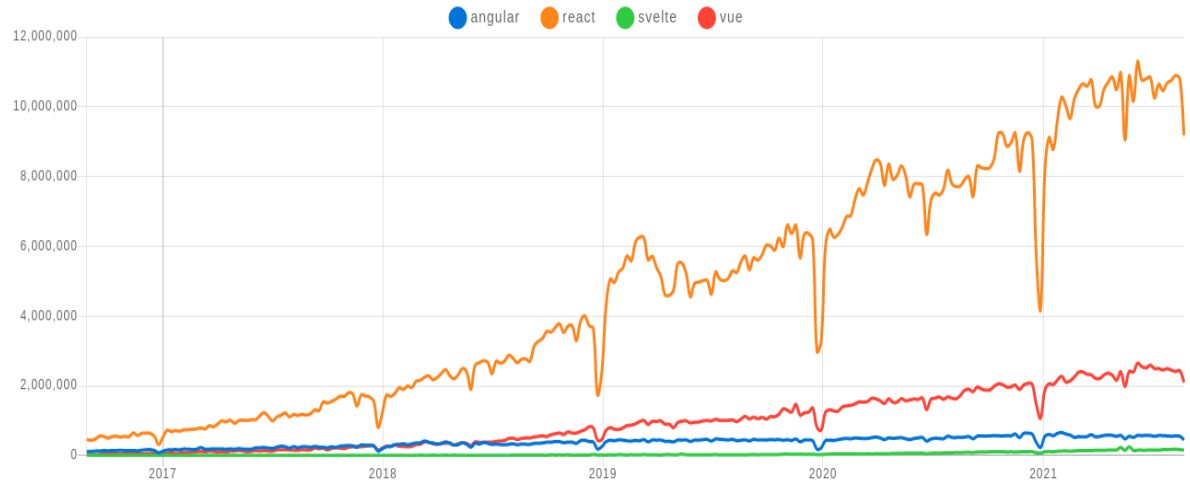

Рисунок 2.3 – Графік зростання популярності технологій для розробки клієнтської частини

Розглянувши дані технології, їх переваги та недоліки, можна зробити висновки, що технологія, яка найбільше підходить для виконання даної задачі є React, оскільки він забезпечує швидкість роботи програми, багато-модульність компонентів та переходи між сторінками без перезавантаження сторінки.

2.4 Вибір фреймворка для стилізації компонентів

Аналогів фреймворків для стилізації компонентів є багато, але вони більшість схожі один на одного, оскільки розробники фреймворків надихалися один одним при створенні [14].

Ось найвідоміші фреймворки і бібліотеки для стилізації компонентів:

Ant Design: популярна бібліотека для створення користувачських інтерфейсів, яка базується на React. Вона була розроблена компанією Alibaba і володіє великою спільнотою користувачів та активним розвитком. Ant Design надає готові компоненти, які допомагають розробляти стильні та функціональні інтерфейси для веб-сайтів і додатків [26].

Bootstrap: є одним із найпопулярніших і найвизнаніших CSS-фреймворків. Він має велику спільноту користувачів і надає готові класи та компоненти для створення респонсивних веб-сайтів. Bootstrap також включає JavaScript-компоненти, які допомагають в реалізації різноманітних функціональних елементів.

Tailwind CSS: відрізняється від інших фреймворків тим, що він надає велику кількість класів, які можна застосовувати безпосередньо до HTML-елементів для стилізації. Він спрощує процес створення власних стилів і дозволяє швидко змінювати дизайн.

Semantic UI: надає готові компоненти і класи з семантичною назвою, що полегшує розуміння їхнього призначення. Він також підтримує реактивний дизайн.

Materialize CSS: відповідає специфікаціям Material Design від Google. Він надає готові компоненти та стилі, які допомагають створювати додатки у стилі Material Design.

Milligram: ще один легкий фреймворк, який надає базовий набір стилів для швидкої розробки.

Bulma: легкий і простий використовувати CSS-фреймворк зі спрощеними класами для стилізації елементів. Він не включає JavaScript і дозволяє розробникам більше контролю над додатками.

Переваги Ant Design:

- Готові компоненти;
- Можливість налаштування стилів;
- Реактивний дизайн;

– Активна спільнота користувачів і розробників;
– Докладна документація [26].

Недоліки Ant Design:

– Розмір бібліотеки може бути великим для деяких проєктів;
– Інтеграція з іншими фреймворками або бібліотеками може бути складною;
– Деякі стилі можуть вимагати додаткових зусиль для кастомізації;
– Залежність від React, що обмежує використання бібліотеки в проєктах, де React не є основним фреймворком.

Переваги Bootstrap:

– Готові компоненти та стилі, що прискорюють розробку;
– Респонсивний дизайн, який легко адаптується до різних пристроїв;
– Велика і активна спільнота користувачів та розробників;
– Детальна документація і багато навчальних матеріалів;
– Підтримка для різних браузерів, включаючи старі версії Internet Explorer;
– Можливість кастомізації стилів і компонентів за допомогою Sass або Less;
– Адаптований до роботи з різними JavaScript-фреймворками.

Недоліки Bootstrap:

– Великий розмір завантажувального файлу, що може вплинути на швидкість завантаження сторінки;
– Зовнішній вигляд Bootstrap-сайтів може виглядати стандартизованим, оскільки багато проєктів використовують той самий набір компонентів і стилів;
– Не завжди підходить для проєктів із складними індивідуальними дизайнами;
– Може бути важко кастомізувати деякі аспекти дизайну, якщо ви не розбираєтеся в Sass або Less;

– Залежність від сторонніх стилів і JavaScript може призвести до конфліктів і проблем у розробці.

Переваги Tailwind CSS:

– Гнучкість і швидкість розробки;
– Адаптивний і респонсивний дизайн;
– Проста і читабельна HTML-розмітка;
– Можливість швидко кастомізувати стилі за допомогою конфігураційних файлів;
– Спільнота користувачів та багато розширень і плагінів для розширення функціональності;
– Можливість створення однорівневого коду з використанням JIT-компіляції для оптимізації розміру файлів CSS.

Недоліки Tailwind CSS:

– Може бути важко зрозуміти і вивчити на початку через велику кількість класів;
– Збільшує розмір HTML-файлів через використання багато класів.
– Персоналізація та кастомізація стилів може бути більш громіздкою порівняно з іншими методами;
– Розмір CSS-файлу може стати великим в розмірах для великих проєктів;
– Вимагає знань HTML та CSS для повноцінної розробки.

Переваги Semantic UI:

– Семантична назва класів;
– Готові компоненти;
– Легка настройка компонентів і стилів.

Недоліки Semantic UI:

– Великий розмір завантажувального файлу CSS та JavaScript;
– Велика кількість класів може бути важкою для вивчення та роботи з ними;

– Можливі проблеми зі сумісністю та інтеграцією з іншими бібліотеками або фреймворками;
– Обмежена активність спільноти порівняно з іншими більш популярними фреймворками.

Переваги Materialize CSS:

– Дизайн Material Design;
– Готові компоненти для швидкого створення інтерфейсів;
– Адаптивний дизайн і підтримка мобільних пристроїв;
– Легка настройка стилів і кольорів;
– Активна спільнота користувачів та підтримка;
– Детальна документація і приклади використання.

Недоліки Materialize CSS:

– Великий розмір завантажувального файлу CSS та JavaScript, що може вплинути на швидкість завантаження сторінки;
– Стилі Material Design можуть не підходити для всіх проєктів і вимагати додаткового кастомізації;
– Інтеграція з іншими бібліотеками або фреймворками може бути складною через унікальні класи та стилізацію Materialize CSS;
– Деякі функції та компоненти можуть бути обмеженими або вимагати додаткових плагінів для повноцінної роботи.

Переваги Milligram:

– Мінімалістичний і легкий;
– Простий і чистий код;
– Легка і швидка інтеграція в проєкти;
– Не вимагає багато часу для вивчення та освоєння.

Недоліки Milligram:

– Основні стилі обмежені, і фреймворк не має багато готових компонентів;
– Призначений в основному для невеликих і мінімалістичних проєктів;

- Вимагає додаткової кастомізації та розробки власних стилів для складних інтерфейсів;
- Може бути не підходити для проєктів, які вимагають широкого функціоналу і великої кількості компонентів.

Переваги Bulma:

- Легкість використання і навчання;
- Респонсивний дизайн і гнучкість;
- Стильна і сучасна зовнішність;
- Спільнота користувачів і наявність розширень;
- Модульність і можливість використання окремих компонентів.

Недоліки Bulma:

- Обмежена кількість готових компонентів порівняно з іншими фреймворками;
- Може потребувати більше додаткової роботи для створення складних дизайнів і інтерфейсів;
- Можливість конфліктів стилів при використанні разом з іншими бібліотеками або фреймворками;
- Менше популярний і менше розповсюджений порівняно з деякими іншими CSS-фреймворками.

Розглянувши дані технології, їх переваги та недоліки, можна зробити висновки, що технологія, яка найбільше підходить для виконання даної задачі є Ant Design, оскільки він побудований спеціально для бібліотеки React, а це включає в себе швидку та зручну інтеграцію в проєкт, великий список готових компонентів та докладна документація. Всі переваги даного інструменту повністю підходять для виконання даної задачі, а її недоліки не впливають на роботу застосунку [15].

2.5 Вибір платформи

У виборі середовища виконання є тільки 3 основних конкуренти:

- Node.js – це середовище виконання JavaScript, яке дозволяє виконувати код JavaScript на стороні сервера [18-22];
- Deno – це середовище виконання JavaScript і TypeScript, яке створено Райаном Далем, одним із співавторів Node.js;
- Bun — це середовище виконання JavaScript, менеджер пакунків, збірник тестів, створений з нуля за допомогою мови програмування Zig. Він був розроблений Джаредом Самнером для Node.js.

Переваги Node.js:

- **Швидкість виконання**: Node.js працює на основі двигуна V8 від Google, який відомий своєю високою швидкістю виконання коду JavaScript. Це робить Node.js відмінним вибором для створення швидких та ефективних додатків;
- **Однопоточність (Single-Threaded)**: Node.js використовує один потік для обробки багатьох запитів одночасно. Це дозволяє підтримувати велику кількість одночасних з'єднань без значного перевантаження системи [19, 21];
- **Наявність пакетного менеджера NPM**: Node.js включає в себе пакетний менеджер NPM (Node Package Manager), який дозволяє легко встановлювати сторонні бібліотеки та модулі, спрощуючи розробку;
- **Широкий вибір бібліотек і фреймворків**: Node.js має активну спільноту розробників, яка розробляє багато бібліотек і фреймворків для різних типів додатків, включаючи веб-сервери (наприклад, Express.js), інструменти для роботи з базами даних, реальним часом, тощо;
- **Ідеально підходить для реального часу і заснованих на подіях**

додатків: Завдяки асинхронності та подійно-орієнтованому програмуванню, Node.js ідеально підходить для створення додатків, які працюють в реальному часі, такі як чати, графіки онлайн, потокове відео, тощо.

Недоліки Node.js:

- **Однопоточність (Single-Threaded)**: Хоча однопоточність дозволяє Node.js обробляти багато запитів, вона також робить його менш придатним для довгооперацій них операцій, які можуть блокувати потік інші запити [20];
- **Нестабільність деяких пакетів**: Оскільки Node.js має активну спільноту розробників і численні пакети, деякі з них можуть бути нестабільними або застарілими;
- **Не найкращий вибір для CPU-інтенсивних завдань**: Оскільки Node.js використовує один потік, він може бути менш ефективним для виконання важких обчислювальних завдань порівняно з іншими технологіями [18].

Node.js є найбільшим середовищем виконанням з усіх, більшість сайтів, які використовують бібліотеки і фреймворки, використовують саме його [19-22].

Діаграма роботи Node.js зображено на рисунку 2.4.

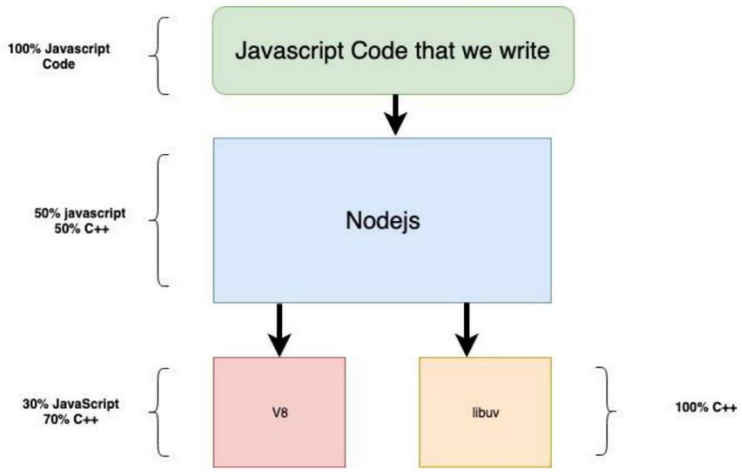

Рисунок 2.4 – Діаграма роботи Node.js

Переваги Deno:

- **Модульність та безпека за замовчуванням**: Deno ставить акцент на безпеку за замовчуванням і модульність. Усі зовнішні модулі завантажуються з вказанням шляху, а також їх виконання і обмеження доступу до системних ресурсів контролюється на рівні дозволів;
- **Вбудований пакетний менеджер**: Deno включає в себе вбудований пакетний менеджер, який дозволяє легко встановлювати залежності для вашого проєкту;
- **Підтримка TypeScript**: Deno підтримує TypeScript з коробки без необхідності встановлення додаткових пакетів чи налаштувань;
- **Async/await за замовчуванням**: Deno працює з асинхронним кодом на базі async/await за замовчуванням, що полегшує асинхронне програмування;
- **Можливість виконання на стороні клієнта (client-side)**: Deno може бути використаний на стороні клієнта браузера для виконання JavaScript-коду без потреби в інтерпретаторі JavaScript у браузері;
- **Активна спільнота та розвиток**: Незважаючи на те, що Deno відносно новий проєкт, він має активну спільноту розробників, і його розвиток продовжується.

Недоліки Deno:

- **Нестабільність API**: Оскільки Deno ще не досяг стабільної версії 1.0, API можуть змінюватися між версіями, що може створювати проблеми для проєктів, які використовують його;
- **Менша кількість бібліотек**: У порівнянні з Node.js, кількість готових бібліотек і розширень для Deno поки що обмежена, оскільки це ще досить новий проєкт;
- **Не така широка спільнота як у Node.js**: Node.js має велику і активну спільноту розробників, тоді як Deno ще не досяг такого рівня популярності.

З переваг Deno в сервісі нічого не використовується, окрімвбудованого пакетного менеджера, який є в усіх.

Переваги Bun:

– Більша швидкість виконання в деяких випадках;
– Все зберігається в одному місці;
– Немає необхідності з’єднувати серверні складові між собою.

Недоліки Bun:

– Велика кількість помилок;
– Неможлива інтеграція зі старих проєктів;
– Відсутність підтримки деяких пакетів.

На даний момент використовувати Bun не ркекомендується, оскільки версія 1.0 не стабільна.

З даних результатів, можна зробити висновок, що найбільш кращим варіантом буде використовувати саме Node.js як найбільш стабільне і популярне середовище виконання.

2.6 Вибір фреймворка для серверної частини

Аналогів фреймворків для серверної частини є багато, але вони більшість схожі один на одного, оскільки розробники фреймворків надихалися один одним при створенні [16].

Ось найвідоміші клієнтські фреймворки і бібліотеки для розробки:

– **Express.js**: Express є дуже популярним і мінімалістичним фреймворком для створення веб-серверів з використанням Node.js. Він є дуже гнучким і часто використовується для створення API-серверів;
– **Meteor**: Meteor - це фреймворк, який підтримує розробку повних стеків додатків, включаючи клієнтську, серверну і базу даних. Він

підтримує побудову веб-додатків в реальному часі;

- **Hapi.js**: Hapi - це фреймворк для створення серверів та додатків Node.js, який надає розширені можливості для контролю над конфігурацією і обробкою запитів.

Переваги Express.js:

- Простота використання та швидкість вивчення;
- Гнучкість і можливість налаштовувати сервер під свої потреби;
- Велика активна спільнота розробників і велика кількість розширень.;
- Підтримка middleware для обробки різних аспектів запитів;
- Висока продуктивність і швидкодія, що робить його ідеальним для створення API і веб-додатків з великим обсягом запитів.

Недоліки Express.js:

- Відсутність стандартної архітектури: Express не накладає жодних обмежень щодо архітектури додатка, що може вести до хаосу у великих проєктах;
- Вимагає великої кількості сторонніх бібліотек для створення повнофункціональних додатків (наприклад, для роботи з базою даних, аутентифікацією тощо);
- Іноді він може виглядати "розірваним" і нести в собі багато бізнес-логіки разом зі звичайним кодом для обробки запитів, що може призвести до неструктурованого коду.

Переваги Meteor:

- Швидкий розріст і розробка додатків;
- Реальний час і синхронізація даних;
- Повний стек технологій, включаючи фронтенд і бекенд;
- Простота встановлення і конфігурації проєкту;
- Велика кількість розширень і сторонніх пакетів, що полегшують розробку;
- Автоматична генерація коду для валідації даних та інших рутинних

завдань;

– Активна спільнота користувачів і підтримка з боку розробників.

Недоліки Meteor:

– Вагомий розмір початкового завантажу, оскільки Meteor включає в себе багато компонентів за замовчуванням;
– Обмежена підтримка баз даних, хоча Meteor підтримує MongoDB, інтеграція з іншими системами баз даних може бути складною;
– Високий рівень абстракції може змінювати стандартні способи розробки, що може бути незручним для деяких розробників;
– Велика кількість генерованого JavaScript-коду може призводити до збільшення завантаженого обсягу даних для клієнтів;
– Помірна підтримка серверних рендерінгів для SEO, хоча є рішення на базі сторонніх бібліотек.

Переваги Hapi.js:

– Контроль маршрутизації і обробки URL;
– Вбудована підтримка валідації та обробки даних;
– Повна підтримка WebSocket і реального часу;
– Потужна система плагінів для розширення функціональності;
– Велика кількість готових плагінів для різних завдань;
– Висока продуктивність і здатність до масштабування;
– Гнучкість у роботі з базами даних та іншими інтеграціями.

Недоліки Hapi.js:

– Вимагає додаткового часу на вивчення, порівняно з деякими іншими фреймворками;
– Може виглядати важким для простих завдань, оскільки призначений для складних додатків і API;
– Завантажений стартовий код, який включає багато налаштувань та опцій;
– Зменшена кількість готових рішень порівняно з більш популярними

фреймворками.

Розглянувши дані технології, їх переваги та недоліки, можна зробити висновки, що технологія, яка найбільше підходить для виконання даної задачі є Express.js, оскільки гнучкість, можливість налаштовувати сервер під свої потреби, висока продуктивність, швидкодія та підтримка middleware для обробки різних аспектів запитів – це те, що потрібно для створення такого застосунку.

Графік найпопулярніших бекенд-фреймворків злбражено на рисунку 2.5.

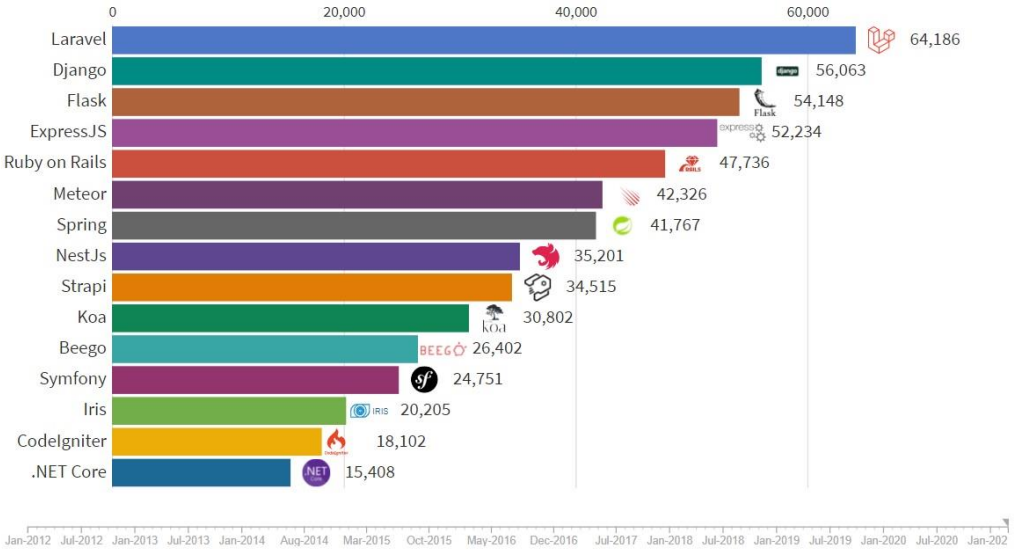

Рисунок 2.5 – Графік найпопулярніших бекенд-фреймворків

Слід зазначити, що на даному графіку Express.js є найпопулярнішим фреймворком для серверної частини саме для JavaScript, оскільки Laravel працює з додатками, написані на мові PHP, а Django та Flask – на Python.

2.7 Вибір ORM для застосунку

ORM (Object-Relational Mapping) - це підхід до роботи з базами даних в програмуванні. ORM дозволяє програмістам взаємодіяти з базами даних, використовуючи об'єктно-орієнтований підхід, замість традиційного роботи з

SQL-запитами і таблицями баз даних. Основна ідея ORM полягає в тому, щоб відобразити структуру бази даних на об'єкти в коді програми.

Основні переваги використання ORM включають:

- **Спрощення розробки**: ORM дозволяє програмістам працювати з об'єктами та класами, які відповідають структурі даних, що робить код більш зрозумілим і об'єктно-орієнтованим;
- **Незалежність від СУБД**: ORM може бути налаштованим для роботи з різними системами управління базами даних (СУБД), що дозволяє змінювати базу даних без зміни коду програми;
- **Зменшення можливостей для помилок**: ORM автоматично генерує SQL-запити, що зменшує можливість помилок, пов'язаних з неправильними запитами;
- **Підтримка відносин**: ORM дозволяє визначити відносини між таблицями в базі даних, спрощуючи отримання пов'язаних даних;
- **Кешування і оптимізація**: Деякі ORM-фреймворки підтримують автоматичне кешування та оптимізацію запитів, що може покращити продуктивність додатку.

Основні ORM для роботи з Node.js TypeScript:

- Prisma;
- Sequelize;
- TypeORM;
- Waterline;
- Objection.js.

Для кінцевого вибору ORM потрібно розглянути переваги та недоліки кожної з них.

Переваги Prisma:

- **Зручний DSL для запитів**: Prisma надає легкий для використання DSL для створення SQL-запитів. Це полегшує створення запитів та спрощує роботу з базою даних;

- **Статична типізація**: Prisma підтримує статичну типізацію, що дозволяє визначати типи для даних та використовувати перевірку типів під час компіляції. Це допомагає виявляти помилки на етапі розробки та полегшує роботу з даними;
- **Підтримка різних СУБД**: Prisma підтримує різні реляційні СУБД, включаючи PostgreSQL, MySQL, SQLite та інші;
- **Автоматичне міграції**: Prisma надає зручність для створення та виконання міграцій, що дозволяє зберігати схему бази даних синхронізованою з моделями даних;
- **Зручна робота з відносинами**: Prisma дозволяє легко визначати та використовувати відносини між таблицями бази даних, що спрощує отримання пов'язаних даних [24].

Недоліки Prisma:

- **Обмежена підтримка СУБД**: Перевагою Prisma є підтримка різних СУБД, але вона не підтримує всі наявні СУБД. Деякі менш популярні або екзотичні СУБД можуть не підтримуватися;
- **Залежність від Prisma Client**: Робота з Prisma передбачає використання Prisma Client для звернення до даних, що робить вас залежними від фреймворку;
- **Може бути зайвим для деяких проєктів**: Для деяких невеликих або простих проєктів використання Prisma може бути зайвим, оскільки це додає додатковий шар абстракції;
- **Потребує вивчення**: Як і будь-який інструмент, Prisma вимагає часу для вивчення та освоєння.

Слід зазначити, що Prisma є найпопулрнішим ORM-фреймворком для використання з високим рівнем підтримки та має статус найкращої ORM для Node.js.

Переваги Sequelize:

- **Підтримка багатьох СУБД**: Sequelize підтримує багато різних

систем управління базами даних, такі як MySQL, PostgreSQL, SQLite та більше. Це робить його гнучким у виборі СУБД для вашого проєкту.;

- **Зручний DSL для запитів**: Sequelize надає простий і зрозумілий DSL для створення SQL-запитів, що полегшує створення та виконання запитів до бази даних;
- **Підтримка транзакцій і операцій індексування**: Sequelize дозволяє легко виконувати операції з транзакціями та індексуванням в базі даних;
- **Підтримка асинхронного програмування**: Sequelize добре підходить для розробки асинхронних Node.js-додатків та підтримує використання обіцянь (Promises) та async/await для асинхронних операцій.

Недоліки Sequelize:

- **Повільність та зайві запити**: Sequelize може генерувати багато запитів SQL, що може призвести до поганої продуктивності великих систем з великою кількістю запитів;
- **Заворожений синтаксис**: Деякі розробники вважають синтаксис Sequelize непередбачуваним та складним для розуміння, зокрема для новачків;
- **Брак зручного статичного аналізу типів**: Sequelize не надає такого рівня статичного аналізу типів, як TypeScript разом із Prisma;
- **Велика настанова**: Деякі функції та опції Sequelize вимагають великої кількості настанови, щоб налаштувати та використовувати.

Переваги TypeORM:

- **Статична типізація**: TypeORM використовує TypeScript, що надає статичну типізацію для даних та запитів. Це допомагає виявляти помилки на етапі компіляції та полегшує підтримку коду;
- **Зручний DSL для запитів**: TypeORM надає зрозумілий та зручний

DSL для створення SQL-запитів. Це полегшує створення та виконання запитів до бази даних;

- **Підтримка різних СУБД**: TypeORM підтримує різні реляційні СУБД, такі як PostgreSQL, MySQL, SQLite та інші. Це робить його гнучким у виборі СУБД для вашого проєкту;
- **Можливість визначення сутностей (entities)**: TypeORM дозволяє визначати сутності, які відображають таблиці в базі даних. Це дозволяє робити роботу з даними більш об'єктно-орієнтованою;
- **Підтримка відносин**: TypeORM підтримує визначення відносин між таблицями в базі даних, що полегшує отримання пов'язаних даних.

Недоліки TypeORM:

- **Вивчення TypeScript**: Якщо ви не маєте досвіду в TypeScript, вам доведеться вивчити цю мову для роботи з TypeORM;
- **Додатковий обсяг коду**: Додавання типів та використання TypeScript може збільшити обсяг коду та зробити його більш об'ємним;
- **Потребує вивчення**: Як і будь-який інший ORM-фреймворк, TypeORM вимагає часу для вивчення та освоєння;
- **Залежність від фреймворку**: Робота з TypeORM призводить до залежності від фреймворку, що може ускладнити зміну технології в майбутньому.

Переваги Waterline:

- **Гнучкість**: Waterline розроблений з метою забезпечити гнучкість та підтримку різних баз даних. Він може працювати з реляційними (наприклад, MySQL, PostgreSQL) та NoSQL (наприклад, MongoDB) базами даних;
- **Уніфікований API**: Waterline надає однаковий API для взаємодії з різними системами управління базами даних. Це полегшує розробку додатків, які використовують різні типи баз даних;
- **Зручність для веб-розробки**: Waterline часто використовується в

Node.js-додатках для розробки веб-серверів і веб-додатків, де потрібна база даних;

- **Можливість визначення моделей даних**: Ви можете визначити моделі даних, які відображають ваші таблиці або колекції даних в базі.

Недоліки Waterline:

- **Обмежена функціональність**: У порівнянні з іншими ORM-фреймворками, такими як Sequelize або TypeORM, Waterline може бути менш функціональним і не мати такої багатої підтримки;
- **Замеджоване розвиток**: У даний час (за момент мого останнього оновлення в вересні 2021 року) Waterline може бути менш активно розвиваним та підтримуваним, ніж інші ORM-фреймворки;
- **Можливість обмеженої настройки і оптимізації**: Оскільки Waterline намагається бути універсальним ORM-фреймворком, у вас можуть бути обмежені можливості для настройки та оптимізації для конкретної бази даних;
- **Невелика спільнота користувачів**: У порівнянні з деякими іншими ORM-фреймворками, спільнота користувачів Waterline може бути меншою, що може ускладнити пошук допомоги та ресурсів.

Переваги Objection.js:

- **Зручний DSL для запитів**: Objection.js надає зрозумливий та зручний DSL для створення SQL-запитів та взаємодії з базою даних. Це полегшує створення складних запитів;
- **Підтримка статичної типізації**: Objection.js може використовувати TypeScript або Flow для статичної типізації, що допомагає виявляти помилки на етапі компіляції та полегшує роботу з даними;
- **Підтримка відносин між моделями**: Objection.js дозволяє визначати та використовувати відносини між моделями, що допомагає отримувати пов'язані дані з бази даних;

- **Підтримка транзакцій та міграцій**: Objection.js підтримує транзакції та міграції бази даних, що полегшує управління базою даних та зберігання схеми синхронізованою з моделями;
- **Зручність для веб-розробки**: Objection.js часто використовується для розробки веб-додатків та API, де потрібна взаємодія з базою даних.

Недоліки Objection.js:

- **Вивчення кривої**: Як і будь-який інший ORM-фреймворк, Objection.js вимагає часу та зусиль для вивчення та освоєння;
- **Може бути зайвим для деяких проєктів**: Для невеликих та простих проєктів використання Objection.js може бути зайвим, оскільки це додає додатковий шар абстракції;
- **Потребує встановлення та налаштування бази даних**: Ви повинні налаштувати підключення до бази даних для використання Objection.js.

На основі розглянутих переваг та недолік, можна зробити висновок, що технологія, яка найбільше підходить для виконання даної задачі є Prisma, оскільки вона володіє високим рівнем підтримки, сучасною архітектурою та вбудованій підтримці TypeScript. Найзручніше в розробці з використанням Prisma є те, що вона для побудови таблиць (якщо база даних є SQL) або документів (якщо база даних є NoSQL) використовує моделі, які є аналогічними з типами або інтерфейсами в TypeScript, тим самим унеможливлюють проблему з проблемою типізації при міграції даних (створення таблиць в базі даних на основі моделей, які описані в файлах) [17].

Діаграма лінії життя Prisma ORM зображено на рисунку 2.6.

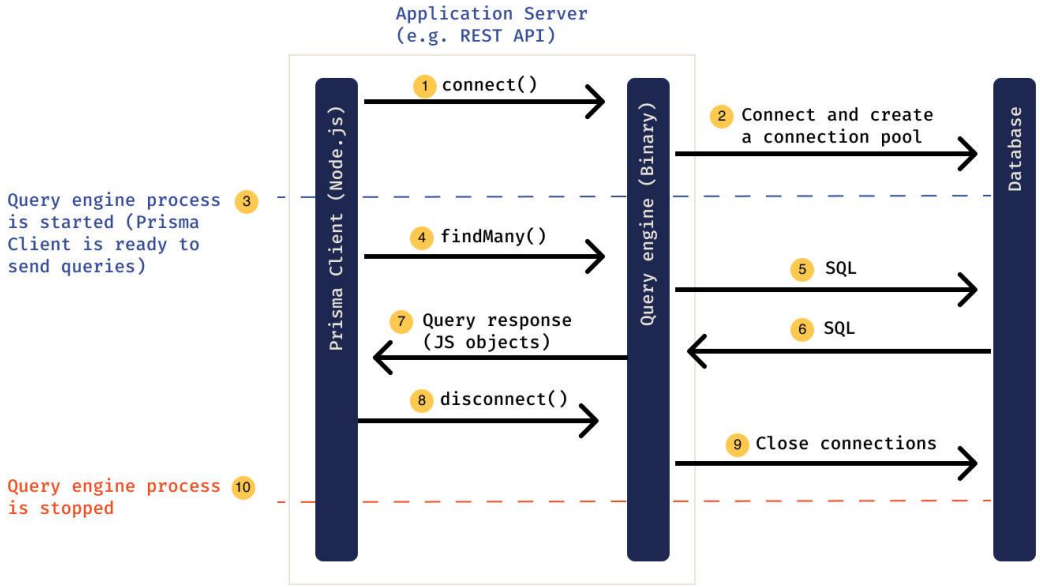

Рисунок 2.6 – Діаграма лінії життя Prisma ORM

2.8 Вибір бази даних

Баз даних є велика кількість, але в основному вони поділяються на реляційні (SQL) та нереляційні (NoSQL). Щоб обрати конкретну базу даних потрібно роглянути переваги та недоліки кожного з цих типів.

Переваги SQL:

- **Структурованість**: РБД дозволяють організувати дані в таблиці з рядками та стовпцями, що спрощує організацію та зберігання даних;
- **Цілісність даних**: Реляційні бази даних можуть забезпечити цілісність даних шляхом застосування правил цілісності та обмежень у базі даних;
- **Гнучкість**: РБД можуть обробляти великий обсяг даних і дозволяють виконувати складні запити за допомогою мови структурованих запитів SQL;
- **Надійність**: Вони забезпечують механізми резервного копіювання та відновлення, що робить їх надійними для зберігання важливих даних;
- **Нормалізація**: Реляційні бази даних дозволяють використовувати

процес нормалізації для зменшення дублікатів та забезпечення ефективного використання простору для зберігання.

Недоліки SQL:

- **Обмежена масштабованість**: Для дуже великих обсягів даних РБД можуть бути менш ефективними, оскільки їх обмеження може призвести до падіння продуктивності;
- **Висока складність**: Робота з РБД може бути складною через необхідність знань SQL та реляційної алгебри для ефективного використання;
- **Високі вимоги до обладнання**: Для забезпечення ефективної роботи РБД може знадобитися потужне обладнання, що може бути дорогим;
- **Відсутність гнучкості**: Деякі реляційні моделі можуть бути обмежені у виразі складних зв'язків та ієрархій даних;
- **Потреба у великій кількості з'єднань**: При складних запитах чи великих базах даних може виникати потреба в багатьох з'єднаннях таблиць, що може призвести до погіршення продуктивності.

Переваги NoSQL:

- **Масштабованість**: НРБД зазвичай добре масштабуються, що дозволяє їм ефективно обробляти великі обсяги даних та запитів;
- **Гнучкість**: Вони забезпечують більшу гнучкість для зберігання різноманітних типів даних, включаючи неструктуровані дані, які не можуть бути легко збережені в традиційних реляційних базах даних;
- **Швидкодія**: Нереляційні бази даних часто пропонують високу продуктивність та швидкодію обробки даних, що робить їх ідеальними для додатків, які потребують низького часу відгуку;
- **Простота масштабування**: Нереляційні бази даних часто працюють на розподіленому середовищі, що дозволяє легко масштабувати і збільшувати потужність за потреби;
- **Вартість**: В деяких випадках вони можуть бути більш ефективними з

точки зору вартості, оскільки вимагають менше обладнання та меншого обслуговування.

Недоліки NoSQL:

- **Обмежена підтримка операцій**: Деякі НРБД можуть не підтримувати повний набір операцій, який надається реляційними базами даних, таких як складні запити та транзакції;
- **Обмежена цілісність даних**: У деяких випадках НРБД можуть бути менш надійними з точки зору цілісності даних порівняно з традиційними реляційними базами даних;
- **Складність**: Деякі НРБД можуть мати складніші моделі даних або вимагати специфічного розуміння для ефективного використання;
- **Відсутність стандартів**: У світі НРБД немає єдиного стандарту, що може призвести до проблем інтеграції та перенесення даних між різними системами;
- **Обмежена підтримка інструментів**: У порівнянні з реляційними базами даних, НРБД можуть мати менший вибір інструментів та програмного забезпечення для розробки та адміністрування.

З цього порівняння можна зробити висновок, що SQL тип використовується, коли в нас відносно невелика кількість зв’язків, а NoSQL – коли даних дуже багато. Для подальшої роботи буде обрано SQL тип, оскільки проєкт буде використовувати невелику кількість зв’язків.

Серед баз даних, які використовують SQL-підхід буде обрано конкретну базу даних для подальшої роботи. Для цього потрібно провести аналіз серед існуючих аналогів:

- **MySQL**: Відкрита реляційна база даних, яка широко використовується для веб-додатків із невеликим та середнім обсягом даних;
- **PostgreSQL**: Можливо, найпотужніша відкрита реляційна база даних, яка має широкий набір функцій та дотримується стандартів SQL;

- **Microsoft SQL Server**: Повнофункціональна реляційна база даних, розроблена корпорацією Microsoft, яка підтримує широкий спектр бізнес-застосунків;
- **Oracle Database**: Масштабована реляційна база даних, яка часто використовується в корпоративних середовищах для обробки великих обсягів даних та важких завдань обробки;
- **SQLite**: Легка вбудовувана база даних, яка підходить для мобільних додатків та простих додатків, що вимагають невеликого обсягу даних.

Переваги MySQL:

- **Популярність та підтримка**: Існує велика спільнота користувачів, а також широкий набір документації та ресурсів для підтримки;
- **Відкритий код**: Будучи відкритою базою даних, MySQL є безкоштовним для використання та легко доступним для розширення або модифікації;
- **Простота використання**: Його легко встановити та налаштувати, і воно пропонує простий і зрозумілий SQL-інтерфейс;
- **Швидкодія**: MySQL здатний обробляти великий обсяг транзакцій та запитів, забезпечуючи швидкість та продуктивність;
- **Масштабованість**: Він може бути легко масштабований як вертикально, так і горизонтально, що дозволяє збільшувати обсяги даних та навантаження на сервер.

Недоліки mySQL:

- **Обмежена підтримка**: Незважаючи на широку спільноту, підтримка MySQL може бути обмеженою у порівнянні з комерційними альтернативами;
- **Складні операції**: Деякі складні операції можуть бути менш ефективними в MySQL порівняно з іншими базами даних;
- **Обмеження розширеності**: В деяких випадках розширення та модифікація MySQL може бути обмеженою порівняно з іншими

СУБД;

- **Обмежена підтримка даних**: MySQL може бути менш ефективним для зберігання та обробки неструктурованих даних порівняно з деякими НРБД;
- **Відсутність деяких функцій**: Деякі більш розширені функції, що доступні в інших реляційних базах даних, можуть бути відсутніми або обмеженими в MySQL.

Переваги PostgreSQL**:**

- **Розширені функції**: PostgreSQL надає широкий набір функцій, таких як географічні об'єкти, вбудовані рядкові функції, підтримка JSON та інші розширені можливості;
- **Відкритий код**: Як і MySQL, PostgreSQL є відкритою базою даних, що робить його доступним для використання та зміни безкоштовно;
- **Надійність**: PostgreSQL відомий своєю стабільністю та надійністю, зокрема завдяки механізмам транзакцій та відновлення;
- **Розширення**: Він дозволяє користувачам створювати свої власні функції, типи та мови програмування, що дозволяє розширювати його базовий функціонал;
- **Підтримка**: PostgreSQL має активну спільноту користувачів та розробників, які забезпечують підтримку, розвиток та оновлення системи.

Недоліки PostgreSQL:

- **Високі вимоги до ресурсів**: У порівнянні з деякими іншими СУБД, PostgreSQL може вимагати більше ресурсів для ефективної роботи та обробки великих обсягів даних;
- **Складність**: Іноді використання PostgreSQL може бути складним через великий набір функцій та складні механізми, які потребують розуміння для ефективного використання;
- **Вартість обслуговування**: Відкритий код не означає безкоштовного

обслуговування, і в деяких випадках підтримка та обслуговування PostgreSQL може бути вартіснішими у порівнянні з деякими іншими рішеннями;

– **Обмежена підтримка деяких інструментів**: У порівнянні з деякими комерційними СУБД, PostgreSQL може мати обмежений вибір інструментів та програмного забезпечення для адміністрування та управління.

Переваги Microsoft SQL Server:

– **Інтеграція з екосистемою Microsoft**: SQL Server добре інтегрується з іншими продуктами Microsoft, що робить його популярним в середовищі, де вже використовуються інші продукти Microsoft;
– **Надійність**: Він відомий своєю високою надійністю, швидкістю та продуктивністю, що робить його популярним в критичних застосунках та підприємницьких середовищах;
– **Розширені можливості аналітики**: SQL Server надає широкі можливості для аналітики даних, включаючи вбудовані засоби аналізу та звітності;
– **Безпека**: Він пропонує різноманітні механізми безпеки, включаючи механізми шифрування та контролю доступу, які роблять його популярним у сфері даних, де безпека має вирішальне значення;
– **Підтримка**: Microsoft має активну спільноту користувачів та широкий спектр підтримки для SQL Server, включаючи документацію, навчальні посібники та технічну підтримку.

Недоліки Microsoft SQL Server:

– **Вартість**: SQL Server є комерційною продукцією, що може вимагати великих витрат на ліцензії та підтримку, зокрема у великих підприємствах;
– **Обмежена переносимість**: Він може мати обмежену переносимість між різними платформами, оскільки він найбільш ефективно працює

в екосистемі Microsoft;

- **Високі вимоги до ресурсів**: У порівнянні з деякими іншими базами даних, SQL Server може вимагати більше ресурсів для ефективної роботи з великими обсягами даних;
- **Обмежена безпека**: У деяких випадках механізми безпеки SQL Server можуть бути менш гнучкими або складними у порівнянні з деякими іншими рішеннями для баз даних.

Переваги Oracle Database:

- **Масштабованість**: Oracle може легко масштабуватися для обробки великих обсягів даних та великих підприємств, що робить його популярним в критичних застосунках;
- **Надійність та продуктивність**: Він відомий своєю високою надійністю, продуктивністю та швидкістю, забезпечуючи ефективну роботу з великими обсягами даних;
- **Розширені можливості аналізу даних**: Oracle надає широкий набір аналітичних інструментів та можливостей, що дозволяють користувачам здійснювати складний аналіз даних та отримувати цінні інсайти;
- **Безпека**: Він пропонує різноманітні механізми безпеки, такі як шифрування даних та контроль доступу, що робить його популярним у сфері даних, де безпека є вирішальним чинником;
- **Підтримка**: Oracle має велику спільноту користувачів та широкий спектр підтримки, включаючи документацію, навчальні посібники та технічну підтримку.

Недоліки Oracle Database:

- **Вартість**: Oracle є комерційною продукцією, що може призвести до великих витрат на ліцензії та підтримку, особливо для великих підприємств;
- **Складність**: Використання та управління Oracle може бути складним

для новачків, оскільки воно має великий набір функцій та можливостей, які потребують розуміння;

- **Вимоги до ресурсів**: Oracle може вимагати великих ресурсів для ефективної роботи з великими обсягами даних та складних операцій;
- **Обмежена переносимість**: Він може мати обмежену переносимість між різними платформами, оскільки він найбільш ефективно працює в своєму власному середовищі.

Переваги SQLite:

- **Легкість використання**: SQLite пропонує простий та легкий у використанні SQL-інтерфейс, що робить його популярним серед початківців та додатків з невеликим обсягом даних [25];
- **Швидкодія**: Він відомий своєю високою швидкодією, що робить його ефективним для додатків, які потребують низького часу відгуку та обробки даних;
- **Легкість вбудовування**: SQLite може бути легко вбудована в інші додатки, що робить його популярним у вбудованих системах та мобільних додатках;
- **Невеликі вимоги до ресурсів**: Він вимагає менших обсягів пам'яті та обчислювальних ресурсів порівняно з більшими базами даних, що робить його ефективним для простих застосунків;
- **Безкоштовний та відкритий код**: SQLite є безкоштовним для використання та розповсюдження, що робить його доступним для широкого кола користувачів.

Недоліки SQLite:

- **Обмежені можливості масштабування**: Він може бути менш ефективним для обробки великих обсягів даних та багатокористувацьких середовищ через свої обмежені можливості масштабування;
- **Обмежена підтримка**: У порівнянні з більш великими СУБД, SQLite

може мати обмежені можливості та підтримку для складних операцій та бізнес-процесів;

– **Обмеженість функцій**: Він може мати обмежений набір функцій порівняно з більш спеціалізованими базами даних, що пропонують ширший спектр функціоналу та можливостей.

Слід зазначити, що SQLite – це база даних, яка найбільше підходить для роботи з Prisma ORM [24-25].

На основі проведеного аналізу, можна зробити висновок, що база даних SQLite найбільше підходить для подальшої роботи, оскільки вона швидка, легка, невимоглива до ресурсів, проста для інтеграції в проєкт, найбільше підходить для роботи з Prisma ORM та використовується для проєктів, в яких невелика кількість зв’язків.

Діаграма діяльності всіх вибраних технологій та користувача зображено на рисунку 2.7.

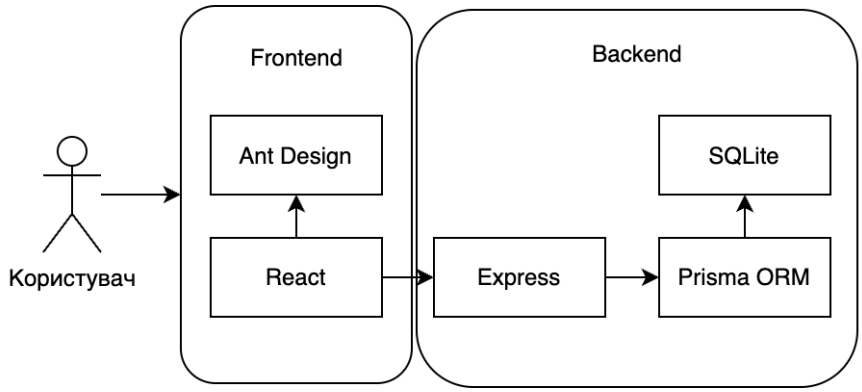

Рисунок 2.7 – Діаграма діяльності всіх вибраних технологій та користувача

2.9 Висновки

Проаналізувавшти можливі варіанти технологій для створення сайтів і застосунків, їх переваги та недоліки можна скласти порівняльну таблицю з технологій, які були обрані внаслідок дослідження технологій:

Таблиця 2.1 – Порівняльна характеристика технологій

	Переваги	Недоліки
React	– Популярність; – Гнучкість; – Компонентний підхід; – Велика спільнота; – Підтримка CSS-in-JS та інших стилізаційних рішень; – Швидка розробка.	– Велика кількість виборів.
Ant Design	– Готові компоненти; – Можливість налаштування стилів; – Реактивний дизайн; – Активна спільнота користувачів і розробників; – Докладна документація.	– Розмір бібліотеки може бути великим для деяких проєктів; – Інтеграція з іншими фреймворками або бібліотеками може бути складною; – Деякі стилі можуть вимагати додаткових зусиль для кастомізації; – Залежність від React.
Node.js	– Швидкість виконання; – Однопоточність ; – Наявність NPM; – Широкий вибір бібліотек і фреймворків;	– Нестабільність деяких; пакетів – Не найкращий вибір для CPU-інтенсивних завдань.

Продовження таблиці 2.1.

	Переваги	Недоліки
Express.js	– Простота використання та швидкість вивчення; – Гнучкість; – Велика активна спільнота розробників і велика кількість розширень; – Підтримка middleware; – Висока продуктивність і швидкодія.	– Відсутність стандартної архітектури; – Вимагає великої кількості сторонніх бібліотек для створення повнофункціональних додатків; – може нести в собі багато бізнес-логіки разом зі звичайним кодом для обробки запитів.
Prisma ORM	– Зручний DSL для запитів; – Статична типізація; – Підтримка різних СУБД; – Автоматичне міграції; – Зручна робота з відносинами.	– Обмежена підтримка СУБД; – Залежність від Prisma Client; – Може бути зайвим для деяких проєктів;
SQLite	– Легкість використання; – Швидкодія; – Легкість вбудовування; – Невеликі вимоги до ресурсів; – Найкраще підходить для роботи з Prisma ORM.	– Обмежені можливості масштабування; – Обмежена підтримка – Обмеженість функцій.

У даному розділі було проведено аналіз сучасних методів розробки сайтів, особливостей та перспектив їх використання в технологіях, які стосуються web-додатків є актуальною задачею в галузі інформаційних систем та технологій.

На основі порівняльного аналізу переваг і недоліків технологій розробки та фреймворків вибрані React для клієнтської частини, Ant Design для стилізації, Node.js як середовище розробки, Express.js для серверної розробки з використанням Prisma ORM та бази даних SQLite.

3 НАПИСАННЯ СЕРВЕРНОЇ ЧАСТИНИ

У даному розділі буде налаштовано серверну частину зв допомогою Express.js, налаштувано середовища розробки, підключено базу даних SQLite, створено моделі, на основі яких було створено таблиці та розгорнуто Prisma ORM.

3.1 Розгортання серверної частини

Для початку роботи потрібно створити серверне середовище, щоб запустити проєкт. Оскільки для робити над серверною частиною було обрано Express.js та Node.js, в даному фреймворку є готові шаблони для розгортання з базовими налаштуванями.

Для подальшої роботи потрібно локально встановити Node.js з їх офіційного сайту. Він надасть пакети npm (node package modules) з потрібними залежностями, модулями та плагінами для роботи.

Дана інсталяція дозволить нам використовувати заготовлені шаблони для роботи. Для того, щоб встпновити пакет в проєкт, потрібно прописати команду в терміналі:

npm i <назва_пакету>

Для того щоб розгорнути початкову збірку проєкту потрібно підключити пакет «express-generator». Після підключення даного пакету в терміналі стане доступна команда «express», яка згенерує базовий шаблон налаштувань. Після виконання даної команди, в директорії проєкту з’являться такі файли:

- bin – для кінцевого відвантаження коду проєкту та його мінімізації;
- public – для зберігання публічних файлів, такі як зображення, іконки та інші файли;
- routes – для зберігання файлів, які містимуть логіку для маршрутизації запитів;

– app.js – для підключення залежностей, які потрібні для налаштування серверної частини;
– node_modules – директорія, що містить в собі пакети з набором функцій та бібліотек, які були імпортовані після команди інсталяції. Вона містить більше директорій, ніж було імпортовано, оскільки одні залежності залежать від інших залежностей, тому імпорт йде рекурсивно для всіх пакетів;
– package.json – файл, який містить всю інформацію про залежності проєкту, скрипти та інформацію про власника.

Список залежностей у файлі package.json зображено на рисунку 3.1.

Рисунок 3.1 – Список залежностей у файлі package.json

Команда «express» автоматично імпортувала всі потрібні залежності в проєкт та підключила їх у файл app.js, окрім пакетів nodemon та concurrently. Ці пакети імпортовано окремо, оскільки вони знадобляться для подальшої розробки.

Вміст файлу app.js зображено на рисунку 3.2.

```
1   const express = require('express');
2   const cookieParser = require('cookie-parser');
3   const logger = require('morgan');
4   const cors = require('cors');
5   require('dotenv').config();
6
7   const app = express();
8
9   app.use(cors());
10  app.use(logger('dev'));
11  app.use(express.json());
12  app.use(express.urlencoded({ extended: false }));
13  app.use(cookieParser());
14
15  const PORT = process.env.PORT;
16
17  app.use('/api/user', require('./routes/users'));
18  app.use('/api/employees', require('./routes/employees'));
19
20  app.listen(PORT, () => console.log(`App listening on port ${PORT}`));
```

Рисунок 3.2 – Вміст файлу app.js

На рисунку 3.2 з 1 по 13 стрічки вміст файлу був згерерований автоматично, а з 14 стрічки і до кінця даного файлу було внесені зміни, які потрібні для подальшої роботи. Підключення відбувається за допоогою методу use().

На стрічці 15 було підключено порт, на якому відкриваєтсья проєкт. Дане значення отримується з файлу .env, в якому зберігаються змінні, які потрібні в усьому проєкті належать від середовище, в якому цей файл запускають.

На стрічках 17 і 18 було підключено прописані роути для обробки запитів за їх ендпоінтом (кінцевою точкою). В даному проєкті буде для кожної сутності свій ендпоінт. Для того, щоб показати, що ці запити пов’язані безпосередньо з отриманням та зміною даних з бази даних, було додано для кожного ендпоінта шлях, який починається з /api.

На стрічці 20 відбувається підключення до самого серверу за допомогою методу listen(), яка першим параметром приймає порт, за яким відбувається підключенням, а другим – функція, яка виконаєтсья у разі успішного підключення. В даному випадку просто виведеться в консоль «App listening on port 8000», оскільки підключення відбувається на порту 8000.

3.2 Підключення Prisma ORM

Для подальшої роботи потрібно налаштувати підключення в проєкт Prisma ORM.

Для цього потрібно виконати команду в терміналі:

npm i -D prisma,

де:

– команда «-D» – це позначення того, що дана залежність буде використовуватися лише на етапі розробки.

Далі потрібно вказати з якою саме базою даних буде відбуватися підключення з Prisma. Оскільки для даного проєкту було обрану базу даних SQLite, потрібно виконати команду в терміналі:

npm i init –datasourse-provider sqlite,

де:

– init – команда, означає зробити базову ініціалізацію проєкту;

– –datasourse-provider – обрати базу даних та створити таблиці сутностей;

– sqlite – назва бази даних.

В результаті виконання даної команди в директорії проєкту буде створено папку «prisma», яка міститиме в собі файл schema.prisma, яка містить в собі всі базові налаштування для зв'язку ORM з базою даних.

Підключення ORM до бази даних зображено на рисунку 3.3.

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}
```

Рисунок 3.3 – Підключення ORM до бази даних

Наступним етапом буде створення моделей, але для цього потрібно описати модель сутностей та їх зв’язки.

В даному проєкті буде використовуватися 2 моделі – User та Employee. Ще однією особливістю даної ORM є те, що можна перестворювати таблиці без втрати даних, які вони містили, за умови якщо таблиця не змінювалася або не мала зв’язків з таблицями, які змінилися. Оскільки один користувач може створити багатьох співробітників, то таблиці мають зв’язок «один-до-багатьох». Дану схему можна модернізувати до зв’язку «багато-до-багатьох», якщо виникне потреба редагувати інфорипцію про працівників в людини, яка її не зареєструвала.

UML-діаграма зв’язків між моделями зображено на рисунку 3.4.

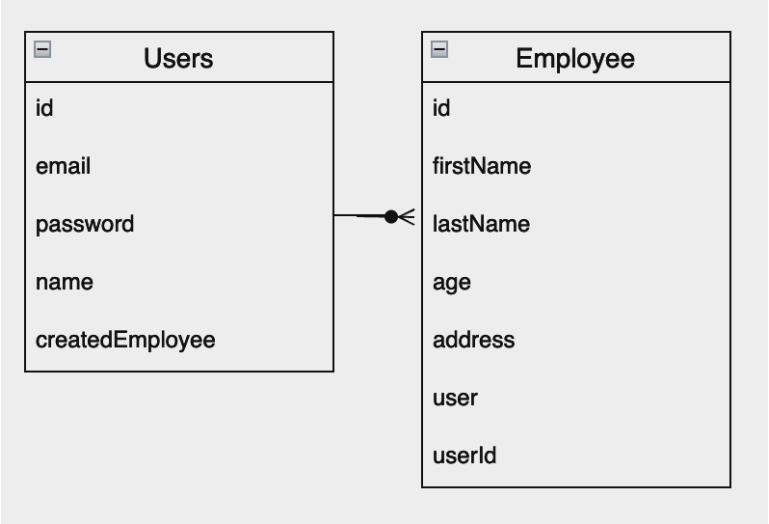

Рисунок 3.4 – Діаграма класів

Для того, щоб створити моделі в Prisma, їх потрібно типізувати.

Модель User має поля:

– id – унікальна стрічка, яка генерується;
– email – стрічка;
– password – стрічка;
– name – стрічка;
– createdEmployee – масив об’єктів, моделею для яких є Employee.

Оскільки зв’язок між схемами «один-до-багатьох», то в даному полі може міститися декілька об’єктів.

Модель Employee має поля:

- id – унікальна стрічка, яка генерується;
- firstName – стрічка;
- lastName – стрічка;
- age – число;
- address – стрічка;
- userId – число, яке пов’язує між собою зв’язок моделі Employee та моделі User;
- user – об’єкт, моделею для яких є User.

Для створення моделей в Prisma використовується оператор model. При використанні SQL баз даних в кожної таблиці повинен бути первинний ключ (primary key). Він створюється за допомогою анотації @id.

Оскільки при створенні будь-якого об’єкта на основі моделі ми не вказуємо поле id, то воно має задаватися за змовчуванням. В Prisma моделях це робиться за допомогою анотаці @default(), яка параметром приймає значення за змовчуванням.

В даній роботі унікальні значення будуть генеруватися за допомогою бібліотеки uuid, яка вбудована в Prisma і її методом uuid().

Модель User містить поле createdEmployee – масив об’єктів, моделею для яких є Employee. Зв’язок між моделями «один-до-багатьох», де один користувач може створити багато працівників, тому значенням поля createdEmployee буде «Employee[]».

В моделі Employee поле user містить в собі зв’язок між моделями Employee та User. Даний зв’язок в Prisma створюється за допомогою анотації @relation(), яка приймає в себе 2 параметри – fields та references. Параметр fields як значення отримує масив полів з моделі, в якій він використовується, які пов’язані з моделею, з якою потрібно утворити зв’язок. Параметр references

fields як значення отримує масив полів з моделі, з якою утворюється зв’язок.

Налаштування моделі User зображено на рисунку 3.5.

```
model User {
  id              String     @id @default(uuid())
  email           String
  password        String
  name            String
  createdEmployee  Employee[]
}
```

Рисунок 3.5 – Модель User у файлі schema.prisma

Налаштування моделі Employee зображено на рисунку 3.6.

```
model Employee {
  id        String @id @default(uuid())
  firstName String
  lastName  String
  age       String
  address   String
  user      User   @relation(fields: [userId], references: [id])
  userId    String
}
```

Рисунок 3.6 – Модель Employee у файлі schema.prisma

Для того, щоб продовжити роботу, потрібно створити таблиці в базі даних SQLite на основі створених моделей. Даний процес називається міграцією.

Для того, щоб виконати міграцію потрібно виконати команду в терміналі:

npx prisma migrate dev –name init,

де:

– npx – набір пакетів, який встановлюється разом з Node.js та використовується так само, як і npm, але працює з іншими пакетами;

– prisma – звернення до пакету prisma;

– migrate – команда для міграції моделей;

– dev – місце, де створяться таблиці;

– --name – команда для створення назви міграції;

– init – назва міграції.

В результаті виконання даної команди в директорії prisma створиться папка migration, а в ній створиться папка, назва якої генереється за прнципом:

{дата створення}{випадкове число}_{назва міграції}

Всередині даної директорії створюється файл migration.sql, який містить вже згенеровані таблиці.

Згенерована таблиця User зображена на рисунку 3.7.

```
CREATE TABLE "User" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "email" TEXT NOT NULL,
    "password" TEXT NOT NULL,
    "name" TEXT NOT NULL
);
```

Рисунок 3.7 - Згенерована таблиця User

Згенерована таблиця Employee зображена на рисунку 3.8. Слід зауважити, що деякі коанди було додано для того, щоб в подальшому реагувати на зміни не тільки в таблиці Employee, а й у таблиці User. Поле зв’язку генерується за шаблоном:

{Назва_таблиці}_{назва_таблиці_з якою_зв’язок}{назва_поля}_fkey

```
CREATE TABLE "Employee" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "firstName" TEXT NOT NULL,
    "lastName" TEXT NOT NULL,
    "age" TEXT NOT NULL,
    "address" TEXT NOT NULL,
    "userId" TEXT NOT NULL,
    CONSTRAINT "Employee_userId_fkey" FOREIGN KEY ("userId") REFERENCES
    "User" ("id") ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Рисунок 3.8 - Згенерована таблиця Employee

Для подальшої роботи потрібно запустити команду:

npx prisma studio

Дана команда дозволить переглядати таблиці у браузері для контролю за об’єктами, які будуть створюватися. В директорії prisma створиться додатковий файл «prisma-cliens.js», в якій буде створено екземляр Prisma-клієнта. Вміст даного файлу зображено на рисунку 3.9.

```
const PrismaClient = require('@prisma/client').PrismaClient;

const prisma = new PrismaClient();

module.exports = {
  prisma
}
```

Рисунок 3.9 – Вміст файлу prisma-cliens.js

3.3 Створення запитів

Після підключення до сервера потрібно створити запити для створення, отримання, редагування та видалення сутностей.

Для розділення логіги виконання і обробки даних, в корені проєкту буде створено директорію «controllers», яка міститиме функції для логіки виконання.

Для кожної сутності буде створено файл, який міститиме функції для кожного з ендпоінтів.

3.3.1 Функції для файлу users.js

В файлі users.js буде створено 3 функції:

– login – для авторизації користувача;

– register – для реєстрації користувача;

– current – для перевірки користувача.

Для подальшої роботи потрібно імпортувати:

– prisma – об’єкт в якому містяться дані про базу даних;
– bcrypt – бібліотека для шифрування і розшифрування пароля користувача;
– jsonwebtoken – бібліотека для створення унікального індетифікатора для користувача.

Функція login є асинхронною і приймає 2 парамета:

– req – об’єкт, що містить інформацію про запит, який надіслав користувач;
– res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для початку потрібно перевірити чи всі поля присутні. Для цього в об’єкта res з його поля body отримаємо поля email та password. Якщо хоча б одного з цих полів немає ми надсилаємо помилку зі статусом 400 (Bad Requets) за допомогою методу status(), який міститься в об’єкті res та приймає код статусу. Оскільки ці операції є асинхронними, потрібно дочекатися результату виконання цього методу. Після того, як відповідь була отримана, потрібно скористатися методом json(), міститься в об’єкті res та приймає об’єкт, який потрібно перетворити в формат JSON. Якщо в нас статус код 4xx або 5xx, зазвичай надсилається об’єкт з полем «message», в якому описується помилка. В даному випадку повідомлення буде містити інформацію про те, що потрібно заповнити всі поля. В наступних випадках, коли буде надсилатися статус код 4xx або 5xx, буде застосовуватися даний метод з надсиланням помилки.

Якщо ці поля є, далі відбувається перевірка на існування такого корстувача в базі даних. Для цього скористаємся об’єктом prisma, який ми імпортували та звернемося до його поля user (дане поле автоматично згенерується на основі назви моделі).

Поля, які згенеровані на основі назви моделі містять в собі методи

доступу для отримання, створення, редагування та видалення інформації з таблиці.

Для того, щоб отримати користувача з таблиці, в об’єкта user потрібно викликати метод findFirst(), який параметром приймає об’єкт з різними полями. Пошук відбувається за допомогою поля where, який є об’єктом, що містить параметри пошуку, де ключі об’єкта – це поля з таблиці по яким буде відбуватися пошук, а значення – переданий аргумент.

Пошук буде відбуватися лише за полем email, оскільки поле password міститься в базі даних зашифрованим.

Наступним етапом йде перевірка на збіг паролів. Для цбого потрібно використати імпортовану бібліотеку bcrypt та її методом compare, який приймає в себе приймає 2 параметра:

- Поле password, яке було в req.body;
- Поле password, яке було отримано з запиту на отримання користувача.

Якщо ці поля не збігаються, ми надсилаємо помилку зі статусом 400 (Bad Requets) та повідомленням, що було введено неправильний email чи пароль.

Якщо паролі збігаються, ми надсилаємо відповідь зі статусом 200 (OK) та об’єкт який містить поля:

- id: user.id;
- email: user.email;
- name: user.name;
- token: jwt.sign({ id: user.id }, secret, { expiresIn: '30d' }).

Поля id, email і name – поля, які були отримані з об’єкта user, а токен – нове створене поле за допомогою імпортованої бібліотеки jsonwebtoken (скорочено jwt) та її вбудованого методу sign(), який створює токен.

Метод sign() приймає 3 параметри:

- payload – інформація, яку потрібно зашифрувати. В даному випадку це поле id, оскільки для авторизації користувача потрібне лише це поле;

- secret – це стрічка, яка може містити в собі будь-які символи та слугує в якості ключа для шифрування;
- options – необов’язковий параметр, який потрібен для налаштування токену.

Оскільки secret буде використовуватися і для валідації токену, його потрібно винести в файл .env, який був створений для зберігання констант, таких як порт.

В даному випадку було задано опцію «expiresIn: '30d'», що означає життя дії токену дорівнює 30 днів.

На всі інші необроблені помилки ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що щось трапилося.

Функція register є асинхронною і приймає 2 парамета:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для початку потрібно перевірити чи всі поля присутні. Для цього в об’єкта res з його поля body отримаємо поля email, password та name.

Якщо хоча б одного з цих полів немає ми надсилаємо помилку зі статусом 400 (Bad Requets) та повідомленням, що потрібно заповнити всі поля.

Якщо ці поля є, далі відбувається перевірка на існування такого корстувача в базі даних.

Для того, щоб отримати користувача з таблиці, в об’єкта user потрібно викликати метод findFirst(), який використовувався в функції логін з тими самими параметрами.

Якщо користувач з таким email вже існує, ми надсилаємо помилку зі статусом 400 (Bad Requets) та повідомленням, що користувач з таким email вже існує.

Якщо такого користувача не існує, потрібно записати дані з тіла запиту в базу даних у відповідну таблицю, але оскільки зберігати незашифровані паролі

ен безпечно, потрібно зашифрувати його перед записом. Для цього необхідно скоритатися імпортованою бібліотекою bcrypt та її методом hash(), якрий приймає в себе 2 параметри:

- пароль, який потрібно зашифрувати;
- salt – параметр, який визначає ступінь хешованості парою.

Для того, щоб згенерувати параметр salt, потрібно використати з бібліотеки cbrypt метод genSalt(), який параметром приймає число, яке і визначає ступінь захешованості.

Для запису в базу даних скористаємося імпортованим об’єктом prisma та його полом users, яке відповідає за всі операції з базою даних.

Об’єкт users для створення нових записів в базу даних використовує метод create(), який приймає параметром об’єкт, який містить в собі поля з об’єктами. Для запису використовується поле data, який в якості значення приймає об’єкт з даними, які потрібно записати.

Створений об’єкт потрібно записати в змінну user, оскільки дані з цього запиту знадобляться далі, оскільки після реєстрації потрібно створити токен авторизації. Даний процес є аналогічним, як і в функції login.

Якщо користувач не створився ми надсилаємо помилку зі статусом 400 (Bad Requets) та повідомленням, що користувач не створився.

На всі інші необроблені помилки ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що щось трапилося.

Функція current є асинхронною і приймає 2 парамета:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Дана функція повертає відповідь зі статусом 200 (OK) та об’єкт який містить поля з інформацією про користувача.

3.3.2 Створення проміжкової функції для авторизації

Оскільки створення користувача не потребувала в собі бути авторизованим, для їх виконпнне не потрібно використовувати проміжкову функції (middleware) для авторизації. Для користувача потрібна авторизація, оскільки тільки авторизований користувач може змінювати, створювати, переглядати та видаляти співробітників.

У Express.js, middleware - це функції, які мають доступ до об'єкту запиту (request object), об'єкту відповіді (response object) та наступної функції middleware в циклі запит-відповідь. Вони використовуються для виконання різноманітних завдань, таких як обробка запитів, перевірка автентифікації, обробка помилок тощо.

Ці функції фактично є проміжними етапами обробки запитів і можуть виконувати широкий спектр завдань, таких як перевірка даних, авторизація, логування, обробка помилок і багато іншого. Вони виконуються перед тим, як запит досягне фактичного маршруту, що обробляє його.

Такі функці приймають в себе від 2 до 4 параметрів, але зазвичай використовуються 3:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач;
- next – функція, яка завершить свою роботу та запустить наступну функцію.

Як можна помітити, функції login, register та current мали параметри дуже схожі, як і middleware-функції. Вони і є middleware-функції, але без параметра next, оскільки їх запускають в певній черзі, а ці функції стоять в кінці виконання, тому їм параметр next не потрібен.

Створений middleware буде відповідати за те, чи авторизований користувач.

Для того, щоб преевірити чи користувач створений потрібно перевірити чи є в нього токен, що записується при створенні коричтувача. Для цього в об’єкті req є об’єкт з полем headers, що містить в собі поле authtorization. Якщо це поле є, воно містить в собі значення ‘Bearer <токен>’. Щоб отримати саме значення токену, можна скористатися методом split(), який розділяє стрічку на масив і приймає параметром розділювач. В даному випадку позділювач є знаком пробілу. В результаті виконання цього методу, отримано масив з 2 елементами: слово ‘Bearer’ та сам токен. Потрібно взяти токен по індексу, в даному випадку це 1.

Далі, для перевірки валідності токену потрібно скористатися імпортованою бібліотекою jwt та її методом verify(), що приймає 2 параметри:

- токен;
- секретний ключ, який був створений при створенні токену.

Якщо токени не співпадають, ми надсилаємо помилку зі статусом 401 (Not Found) та повідомленням, що користувач не авторизований.

Дана функція поверне декодовані дані. В нашому випадку це об’єкт з полем id.

Оскільки після виконання потрібно передати далі об’єкти запиту і відповіді (req і res), перед відправкою потрібно передати значення поля id в тіло запиту.

Для цього за допомогою імпортованого об’єкту prisma і його поля users за допомогою методу findUnique(), який приймає параметром об’єкт, який містить поле те саме, що і метод findFirst(), а саме – where, де параметром пошуку буде поле id, а значенням – поле id з декодованого об’єкту.

Якщо користувач не знайдений, ми надсилаємо помилку зі статусом 401 (Not Found) та повідомленням, що користувач не авторизований.

В об’єкт запиту (req) в поле user потрібно записати той об’єкт, що знайшовся в результаті виконання запиту на пошук.

Після всіх цих операцій потрібно викликати метод next(), щоб запустити виконання наступної в ланцюгу викликів middleware-функцій.

Код даної функції зображено на рисунку 3.10.

```
const auth = async (req, res, next) => {
  try {
    let token = req.headers.authorization?.split(' ')[1];

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    const user = await prisma.user.findUnique({
      where: {
        id: decoded.id,
      },
    });

    req.user = user;

    next();
  } catch (error) {
    res.status(401).json({ message: 'Not authorized' });
  }
};
```

Рисунок 3.10 – Код функції auth

Дану функцію потрібно викликати до того, як будуть відпрацьовані функції, яким потрібна авторизація.

3.3.3 Функції для файлу employees.js

Файл employees.js імстить в собі функції:

– all;
– add;
– remove;
– edit;
– employee.

Функція all є асинхронною і приймає 2 парамета:

– req – об’єкт, що містить інформацію про запит, який надіслав

користувач;

- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для того, щоб отримати працівників з таблиці, в об’єкта employee потрібно викликати метод findMany().

Якщо працівників не було отримано, ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що щось пішло не так.

В разі успішного виконання ми надсилаємо відповідь зі статусом 200 (OK) та масив який містить об’єкти з інформацією про працівників.

Функція add є асинхронною і приймає 2 парамета:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для початку потрібно перевірити чи всі поля присутні. Для цього в об’єкта res з його поля body отримаємо поля firstName, lastName, address та age.

Якщо хоча б одного з цих полів немає ми надсилаємо помилку зі статусом 400 (Bad Requets) та повідомленням, що всі поля є обов’язкрвим для заповнення.

Об’єкт employees для створення нових записів в базу даних використовує метод create(), який приймає параметром об’єкт, який містить в собі поля з об’єктами.

Для запису використовується поле data, який в якості значення приймає об’єкт з даними, які потрібно записати. В нього ми передаємо всі поля, що надіслав користувач та записуємо поле userId, яке потрібно взяти з об’єкта req з його поля user, яке містить поле id. Це є цим полем, що записується в middleware-функції auth. Тут воно записується для того, щоб показати який користувач зареєстрував даного співробітника.

Створений об’єкт потрібно записати в змінну employee, оскільки дані з цього запиту знадобляться для тогог, щоб повернути їх користувачеві.

В разі успішного виконання запиту, протрібно повертути відповідь зі статусом 200 (OK) та об’єкт employee, який містить поля з інформацією про співробітника.

На всі інші необроблені помилки ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що щось трапилося.

Функція remove є асинхронною і приймає 2 парамета:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для початку потрібно дістати поле id з об’єкту req та його поля body.

Об’єкт employees для видалення записів з базу даних використовує метод delete(), який приймає параметром об’єкт, який містить в собі поля з об’єктами. Даний метод приймає параметром об’єкт, який містить поле те саме, що і метод findFirst(), а саме – where, де параметром пошуку буде поле id, а значенням – поле id, який отримуємо з req.body.

В разі успішного виконання запиту, протрібно повертути відповідь зі статусом 204 (No Content) та повідомленням про успішне видалення.

В разі невдачі, ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що не вийшло видалити працівника.

Функція edit є асинхронною і приймає 2 парамета:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для початку потрібно дістати поле data з об’єкту req та його поля body та поле id з об’єкту data.

Об’єкт employees для видалення записів з базу даних використовує метод delete(), який приймає параметром об’єкт, який містить в собі поля з об’єктами. Даний метод приймає параметром об’єкт, який містить поле те саме, що і метод

findFirst(), а саме – where, де параметром пошуку буде поле id, а значенням – поле id, який отримуємо з data. Окрім цього, потрібно вказати що саме потрібно замінити. Для запису використовується поле data, який в якості значення приймає об’єкт з даними, які потрібно записати. В нього ми передаємо об’єкт дата, що міститься в req.body.

В разі успішного виконання запиту, протрібно повертути відповідь зі статусом 204 (No Content) та повідомленням про успішне оновлення працівника.

В разі невдачі, ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що не вийшло оновити інформацію про працівника.

Функція employee є асинхронною і приймає 2 парамета:

- req – об’єкт, що містить інформацію про запит, який надіслав користувач;
- res – об’єкт, що містить інформацію про відповідь, яку отримав користувач.

Для початку потрібно дістати поле id з об’єкту req та його поля params.

Для цього за допомогою імпортованого об’єкту prisma і його поля employees за допомогою методу findUnique(), який приймає параметром об’єкт, який містить поле те саме, що і метод findFirst(), а саме – where, де параметром пошуку буде поле id, а значенням – поле id з об’єкту req.body.

Якщо користувач не знайдений, ми надсилаємо помилку зі статусом 500 (Internal Server Error) та повідомленням, що відбулася помилка при отриманні співробітника.

3.4 Підключення запитів до роутера

Для підключення цих функцій до самого роутера в директорії routes jgтрібно створити js-файли для кожної сутності.

Перейшовши в дані файли, потрібно імпортувати:

– express – бібліотека яка містить методи для створення маршрутизації;
– функції з директорії controllers;
– auth – функцію-middleware.

Для того, щоб створити роутер для підключення потрібно з бібліотеки express викликати метод Router(). Даний об’єкт містить в собі методи для оюробки REST API запитів, такі як POST, PUT, DELETE та інші.

Для підключення до ендпоінта функцій, що мають відпрацьовувати, потрібно викликати метод роутера, який називається так само як метод запиту (для POST – post, DELETE – delete), які приймають 2 параметри:

– ендпоінт, на який буде відбуватися запит;
– набір функцій, які будуть відпрацьовувати під час запиту.

Хоча функція має лише 2 параметри, передавати можна необмежену кількість функцій і слід пам’ятати, що вони виконуються в порядку зліва направо. Тому middleware-функції мають йти завжди перед функціями які безпосередньо повертають дані користувачеві.

Для динамічних параметрів, таких як id, потрібно використовувати синтаксис /:id, для того, щоб express зрозумів, що це не частина шляху, а динамічний параметр. Дані роутери потрібно експортувати до файлу app.js, і підключити їх. Їх підключення зображено на рисунку 3.2 на стрічках 17 і 18. Приклад підключеного роутера на прикладі роутера працівників зображенно на рисунку 3.11.

```
const express = require('express');
const router = express.Router();
const { add, edit, remove, all, employee } = require('../controllers/employees');
const { auth } = require('../middleware/auth');

router.get('/', auth, all);
router.get('/:id', auth, employee);
router.post('/add', auth, add);
router.post('/remove/:id', auth, remove);
router.put('/edit/:id', auth, edit);
```

Рисунок 3.11 – Приклад підключеного роутера

3.5 Висновки

В даному розділі було налаштовано серверну частину за допомогою Express.js, налаштувано середовища розробки, підключено базу даних SQLite, створено моделі, на основі яких було створено таблиці та розгорнуто Prisma ORM. За допомогою даних налаштувань можна розгортати базовий шаблон для клієнтської частини на основі бібліотеки React.

Дані налаштування забезпечують стабільну роботу серверної частини та прозору конфігурацію ендпоінтів для створення запитів.

На основі підключення серверної частини, бази даних та розгортання ORM системи можна розгортати шаблон на основі бібліотеки React JS. Оскільки клієнтська частина більш гнучка та легше масштабується, її потрібно розгортати після налаштувань інших частин додатку. Серверна частина написана на JavaScript, тому не потрібно додаткових налаштувань чи адапторів для з’єднання frontent та backend частин.

4 СТВОРЕННЯ КЛІЄНТСЬКОГО ІНТЕРФЕЙСУ

В даному розділі було розгорнуто базовий шаблон для налаштування клієнтської частини на основі бібліотеки React, налаштування Redux Toolkit – менеджеру для контролю та передачі стану та інформації по проєкту та налаштування сторінок авторизації, реєстрації, головної сторінки та сторінки співробітників.

Діаграма взаємодії корисувача із доступним функціоналом зображено на рисунку 4.1.

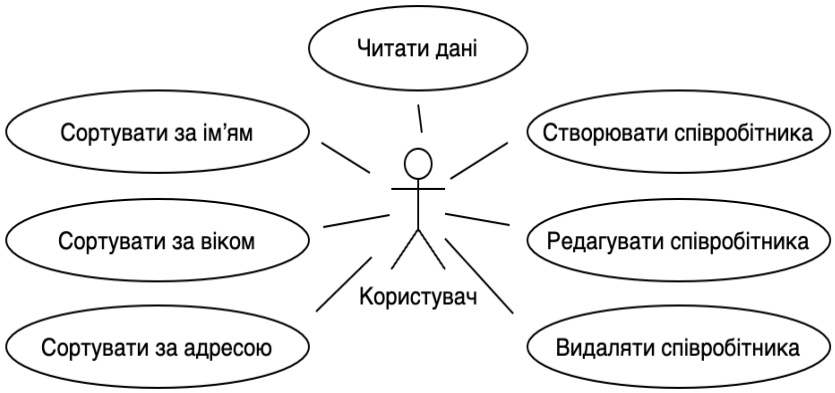

Рисунок 4.1 – Діаграма взаємодії корисувача із доступним функціоналом

4.1 Розгортання шаблону React

Для того, щоб розгорнути шаблон для налаштування клієнтської частини на основі бібліотеки React потрібно в корені проєкту прописати команду в терміналі:

npx create-react-app client –template redux-typescript,

де:

- npx – набір пакетів, який встановлюється разом з Node.js та використовується так само, як і npm, але працює з іншими пакетами;
- create-react-app – утиліта, яка дозволяє розгорнути шаблон для

налаштування клієнтської частини на основі бібліотеки React;

- client – назва директорії, в якій буде розгорнуто шаблон;
- --template – показник, який вказує на те, з якими початковими конфігураціями потрібно розгорнути даний шаблон;
- redux-typescript – назва шаблону.

Даний шаблон буде використовувати базове підключення бібліотеки Redux Toolkit, оскільки Toolkit вже йде за змовчуванням та весь код буде з використанням мови TypeScript.

Після запуску даної команди, ви директорії client буде така структура:

- node_modules – директорія, що містить в собі пакети з набором функцій та бібліотек, які були імпортовані після команди інсталяції;
- public – для зберігання публічних файлів, такі як зображення, іконки та інші файли;
- /src – директорія, в якій міститься всі файли для роботи з шаблоном
- .gitignore – файл зі списком директорій чи файлів, які не мають відслідковуватися системою Git;
- package-lock.json – файл, який містить в собі інформацію про всі пакети, які були встановлені через npm;
- README.md – файл розмітки, в якому міститься інформація про проєкт, як потрібно його запускати, які налашутвання потрібні та інфша інформація;
- tsconfig.json – файл з конфігурацією правил для використання мови TypeScript.

Директорія node_modules містить більше директорій, ніж було імпортовано, оскільки одні залежності залежать від інших залежностей, тому імпорт йде рекурсивно для всіх пакетів.

Для кожного шаблону вміст файлу .gitignore різний, але дуже схожий. В даному файлі містяться зазвичай node_module, package-lock.json, оскільки вони автоматично створюються на оновлюються після команди npm install

(скорочено npm i) та файли середовища, які не повинні бути загальодоступними.

Також вміст файлу залежить від пакетного менеджеру. В даному проєкті використовується npm, але може використовуватися yarn (пакетний менеджер, розроблений корпорацією Meta) та pnpm – менеджер, що схожий на npm. Різниця між ними лише в назві команд та швидкості роботи.

В свою чергу, директорія src має базову структуру:

- /app – директорія, яка містить базове налаштування з роботи Redux Toolkit (функції та підключення загального стору – місця, де зберігається інформація про стан проєкту);
- /features – директорія, яка містить налаштування окремого стору для кожної сутності;
- index.css – файл з загальними стилями;
- index.test.txs – файл для загальними тестами;
- index.tsx – файл для підключення самого проєкту в HTML-файл;
- App.tsx – файл з компонентом App, який містить в собі початкову розмітку;
- App.test.tsx – файл з тестами для компонента App;
- App.css – файл зі стилями для компонента App;
- logo.svg – шаблонна картинка;
- react-app-end.d.ts – файл, який підключає react-скрипти;
- reportWebWitals.ts – файл, який містить інформацію про перевірку на проблеми проєкту;
- setupTests.ts – файл, який містить тести для запуску проєкту.

Оскільки компонент App містить лише в собі лише шаблонну розмітку, даний компонент потрібно видалити з усіма файлами, що до нього прив’язані оскільки він не несе в собі цінності.

Для налаштування роутингу на сторінках потрібно скористатися бібліотекою react-router-dom. Її потрібно імпортувати з пакетного менеджера

npm.

В даному проєкті буде використовуватися 7 сторінок:

– home;
– employeeAdd;
– employeeEdit;
– employee;
– status;
– login;
– register.

Для того, щоб створити роутер, потрібно скористатися методом createBrowserRouter(), який параметром приймає масив об’єктів із полями path та element, де path – це шлях, по якому буде відмальовуватися сторінка, elements – компонент, який відмальовується на сторінці.

Налаштування роутера зображено на рисунку 4.2.

```
const router = createBrowserRouter([
  {
    path: Paths.home,
    element: <Employees />,
  },
  {
    path: Paths.login,
    element: <Login />,
  },
  {
    path: Paths.register,
    element: <Register />,
  },
  {
    path: Paths.employeeAdd,
    element: <AddEmployee />,
  },
  {
    path: `${Paths.employee}/:id`,
    element: <Employee />,
  },
  {
    path: `${Paths.employeeEdit}/:id`,
    element: <EditEmployee />,
  },
  {
    path: `${Paths.status}/:status`,
    element: <Status />,
  },
]);
```

Рисунок 4.2 – Налаштування роутера

Для підключення роутера безпосередньо в проєкт, потрібно імпортувати компонент RouterProvider з тієї ж бібліотеки і передати їй props (props – скор. properties – влстивість) router, значення якого є роутер, що був налаштований.

4.2 Підключення та налаштування Redux Toolkit

Перед підключенням Redux Toolkit слід зазначити, що шаблонні налаштування вже імпортовані в проєкт на етапі створення шаблону. Для цього він автоматично підключив пакети «react-redux» для зв’язку React-додатку та бібліотеки Redux та «reduxjs/toolkit» для отримання функцій, що економлять час для підключення та надають зручні функції для роботи з асинхронними запитами.

Для того, щоб мати доступ до даних всередині проєкту, Redux Toolkit використовує компонент Provider та його props store, який приймає в себе налаштування всього стору. Базова імплементація була створена з шаблоном.

Для утворення стору Redux Toolkit використовує метод configureStore(), який приймає параметром об’єкт з конфігураціями. Обов’язковим полем є reducer. Reducer (редуктор) визначає, як стан додатка змінюється відповідно до дій, які відбуваються у додатку. Він є чистою функцією, яка приймає поточний стан і дію (action) в якості аргументів і повертає новий стан.

У Redux Toolkit, редуктори створюються за допомогою функції createSlice, яка дозволяє визначити дії (actions) та редуктори (reducers) одночасно, зменшуючи багато повторюваного коду. Вони також автоматично генерують action creators, що полегшує роботу зі створенням дій та їх викликом.

Redux Toolkit дозволяє спростити управління станом за допомогою reducer-функцій, що полегшує розробку та підтримку Redux-додатків, допомагаючи уникнути зайвого бойлерплейту та спрощуючи процес взаємодії зі станом додатка.

В React використовується принцип хуків (hooks) – це функції, які створюють та змінюють стан компонента та виконують певні операції при появі (монтуванні) компоненті та його зникненню (демонтування).

Особливістю хуків є те, що вони починаються з префікса «use», їх не можна викликати по умові та вони мають виклакатися на верхньому рівні, тобто вони мають бути оголошені до будь-якої умови, яка може викликати розмонтування компоненту. Це пов’язано з тим, що React бібліотека «під капотом» будує чергу виконання хуків та монтування компонентів, тим самим забезпечує гарантовану роботу всього життєвого цклу компонентів.

Для роботи з React Redux надає 2 хуки:

- useDispatch();
- useSelector().

Виклик хука useDispatch() в результаті повертає функцію dispatch() для зміни стану в сторі. Вона приймає параеметром функцію, яка викликається, щоб змінити частину стору або весь стор.

Хук useSelector() приймає параметром функцію зворотнього виклику, що в свою чергу приймає загальний стор в якості аргументу та повертає частковий або повний стор.

Оскільки шаблон використовує TypeScript, Redux Toolkit окрім store налаштувань в файлі генерує ще 3 типи, і які будуть використані для подальшої роботи:

- AppDispatch – тип для useDispatch, щоб його типізувати для того, щоб валідувати аргументи функцій, які передаються параметром в dispatch();
- RootState – тип для всього стору;
- AppThunk – тип для роботи з асинхронними подіями в Redux Toolkit.

Окрім типів, генерується ще 2 хуки:

- useAppDispatch() – хук useDispatch() з типом AppDispatch;
- useAppSelector() – типізований хук useSelector.

Для типізації хука useSelector Redux використовує тип TypedUseSelectorHook<T>, де T – це тип, який потрібно піставити (такі типи називаються дженериками). В якості дженерика приймає тип стору (RootStore).

Кінцеві хуки зображено на рисунку 4.3.

```
export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
```

Рисунок 4.3 – Хуки useAppDispatch() та useAppSelector()

Для підключення результатів з асинхронних запитів потрібно створити екземпляр об’єкту з конфігураціями. В Redux Toolkit це створюється за допомогою методу createApi(), який імпортується з бібліотеки «reduxjs/toolkit» та приймає в якості параметра об’єкт з полями, серед який обов’язкові це baseQuery та endpoints.

Поле baseQuery приймає як значення конфігурацію, яка буде загальою для будь-якого запиту. Дана конфігурація створюється за допомогою методу fetchBaseQuery(), який імпортується з бібліотеки «reduxjs/toolkit» та приймає в якості параметра об’єкт з полями.

Перше поле – це baseUrl – частина url-стрічки, з якої будуть починатися всі запити

Друге поле – prepareHeaders – функція, що приймає параметром інформацію про запит, а іншим – інформацію про стор та методи для його обробки. В даній функції буде відбуватися перевірка на авторизацію користувача. Вона відбувається за допомогою токену, що створювався в попередньому розділі. Даний токен буде зберігатися як частина стору (коли буде відбуватися реєстрація чи логін, він буде зберігатися) або в браузерному сховищі localStorage (ця практика є більш поширеною щоб щоразу не звертатися до сховища).

В полі endpoinds описуються всі запити (URL, метод, тіло запиту та параметри). Для зручності можна описувати ендпоінти по різних файлах, щоб

було розділення по сутностях. В Redux це відбувається за допомогою функції injectEndpoints(), яка імпортується з бібліотеки «reduxjs/toolkit» та приймає в якості параметра об’єкт з 2 полями, головне з яких endpoints – функція, яка приймає параметром об’єкт та на основі цього об’єкта повертає інший об’єкт, поля якого є назвами ендпоінтрами а значеннями – налаштування кожного ендпоінта.

Налаштування базаового об’єкту для інтеграції в стор зображено на рисунку 4.4.

```
const baseQuery = fetchBaseQuery({
  baseUrl: 'http://localhost:8000/api',
  prepareHeaders: (headers, { getState }) => {
    const token = (getState() as RootState).auth.user?.token || localStorage.
    getItem('token');

    if (token) {
      headers.set('authorization', `Bearer ${token}`);
    }
    return headers;
  },
});

const baseQueryWithRetry = retry(baseQuery, { maxRetries: 1 });

export const api = createApi({
  reducerPath: 'splitApi',
  baseQuery: baseQueryWithRetry,
  refetchOnMountOrArgChange: true,
  endpoints: () => ({}),
});
```

Рисунок 4.4 – Налаштування базаового об’єкту для інтеграції в стор

Налаштування кожного ендпоінта відбувається за допомогою об’єкта builder, який передається в якості параметра дял функції зворотнього виклику. В залежності від методу запиту, потрібно використовувати різні методи. Якщо це метод на отримання (GET), тоді викликається метод query(), всі інші випадки – метод mutation(). Обидва ці методи приймають однакові параметри –об’єкт з полями, головне з яких query – функція, яка приймає параметром об’єкт, в який можна передавати тіло запиту, параметри та інші значення, що

можуть впливати на запит та повертає об’єкт з налаштуваннями запиту (тіло, метод, параметри та найголовніше url – обов’язкове поле, що показує на який саме ендпоінт потрібно відпрацьовувати).

Якщо в об’єкта builder викликається метод query(), то в об’єкті з налаштуваннями можна не вказувати метод, оскільки він буде GET.

Приклад налаштованого ендпоінта зображено на рисунку 4.5.

```
login: builder.mutation<ResponseLoginData, UserData>({
  query: (userData) => ({
    url: '/user/login',
    method: 'POST',
    body: userData,
  }),
}),
```

Рисунок 4.5 – Приклад налаштованого ендпоінта

Для сторінки реєстрації та авторизації буде створено 3 ендпоінти:

– login;
– register;
– current.

Для сторінки співробітника буде створено 5 ендпоінтів:

– getAllEmployees;
– getEmployee;
– editEmployee;
– removeEmployee;
– addEmployee.

Слід зауважити, що назви ендпоінтів подібні до тих запитів, які були створені в розділі 3.3.

Для того, щоб підключити даний об’єкт зі всіма ендпоінтами, потрібно імпортувати його в файл стору та підключити його в методі configureStore() ат в його поле reducer. Назва редуктора може бути будь-якою, але загальноприйняте правило, що воно міститься в полі reducerPath, а саме

значення – в полі reducer.

Додавання middleware в Redux відбувається в полі middleware в об’єкті, що приймає параметром функція configureStore() методом додавання middleware-функції в масив.

Для зручності роботи з токеном, потрібно додати middleware-функцію, яка буде перевіряти кспішність запитів та записувати токен в localStorage.

Створення middleware функцій, які опираються на результат іншої події, відбувається за допомогою функції createListenerMiddleware(), результат виконання якої повертає об’єкт з налаштуванням прослуховувача.

Для прослуховування в об’єкта потрібно викликати метод startListening(), який параметром приймає об’єкт з полями match та effect. Поле match приймає як значення подію, яку треба прослуховувати. Поле effect приймає функцію, яка має виконатися, коли подія відбудеться. Дана функція має параметри action та listenerApi. Об’єкт action містить в собі інформацію про подію, а listenerApi – набір функцій та об’єктів для керування стором.

Для скасування інших активних слухачів в наборі listenerApi потрібно викликати метод cancelActiveListeners(). Наступним кроком буде перевірка чи є в об’єкта action та його полі payload значення token. Якщо таке значення існує, то записуємо його в localStorage.

Код даної middleware-функції зображено на рисунку 4.6.

```
export const listenerMiddleware = createListenerMiddleware();

listenerMiddleware.startListening({
  matcher: authApi.endpoints.login.matchFulfilled,
  effect: async (action, listenerApi) => {
    listenerApi.cancelActiveListeners();

    if (action.payload.token) {
      localStorage.setItem('token', action.payload.token);
    }
  },
});
```

Рисунок 4.6 – Код middleware-функції для запису токену в localStorage

Останньою частиною для налаштування Redux в проєкті – підключення слайсів. Слайси – це основна одиниця організації коду та управління станом в додатку. Слайси є частиною нового підходу до роботи з Redux, який спрощує процес визначення дій, редукторів і селекторів стану.

Слайси в RTK включають в себе наступні ключові елементи:

- **Назва слайсу**: Це строка, яка ідентифікує певний "слайс" або частину вашого стану;
- **Початковий стан**: Це об'єкт, який визначає початковий стан або початкові дані для вашого слайсу;
- **Дії (actions)**: Це набір об'єктів, які описують можливі дії, які можуть вплинути на ваш стан;
- **Редуктори (reducers)**: Це функції, які обробляють дії і змінюють стан вашого слайсу;
- **Селектори (selectors)**: Це функції, які дозволяють вам вибирати та отримувати певні частини стану з вашого слайсу.

Створення слайсів дозволяє ефективно організовувати ваш код, спрощує роботу зі станом вашого додатку та сприяє швидкому розвитку. Вони дозволяють уникнути надмірного шаблонного коду та підтримують чистоту та зручність вашого коду.

В даному проекті буде 2 слайси – для авторивації та для співробітників.

Для авторизіації початковий стан буде сладатися з 2 полів – user та isAuthenticated. При будь-якій дії isAuthenticated буде мати значення true, а значення user – значення, яке передали в дію в якості параметру. Лише при дії виходу з системи (logout) весь стан стане порожнім.

Для співробітника буде лише 2 дії – отримати всіх співробітників та вихід з системи (logout). Початковий стан – список співробітників, який порожній. При успішній виконанні дії отримання співробітників, даний список запишеться в стан. При дії виходу з системи (logout) весь стан стане порожнім.

Дані слайси підключаються в методі configureStore() в параметрі в полі reducer.

4.3 Налаштування сторінок

Для розділення логіки для кожної зі сторінок, в папці client потрібно створити директорію pages, яка міститиме в собі директорії з взідним файлом для кожної зі сторінок. Дані компоненти використовуються для роутингу між сторінками при налаштуванні роутеру, зображеному на рисунку 4.1.

Для написання стилів в проєкті потрібно підключити фреймворк Ant Design. Щоб його підключити, потрібно в файл index.tsx, що знаходиться в корені папки client, імпортувати компонент ConfigProvider та об’єкт theme з пакету «antd». Компонент ConfigProvider приймає в себе props theme для налаштування теми. Даний props є об’єктом і в його поле algorithm потрібно присвоїти посилання на метод darkAlgorithm(), що знаходиться в імпортованому об’єкті theme.

Кінцева налаштована конфігурація точки входу в клієнтську частину зображено на рисунку 4.7.

```
const container = document.getElementById('root')!;
const root = createRoot(container);

root.render(
  <React.StrictMode>
    <Provider store={store}>
      <ConfigProvider
        theme={{
          algorithm: theme.darkAlgorithm,
        }}
      >
        <Auth>
          <RouterProvider router={router} />
        </Auth>
      </ConfigProvider>
    </Provider>
  </React.StrictMode>
);
```

Рисунок 4.7 – Кінцева налаштована конфігурація точки входу в клієнтську частину

Підключення усіх компонентів відбувається в одному місці – в HTML-розмітці в тегові з атрибутом id=root. В React йде підключення за допомогою методу createRoot(), імпортованого з React, та приймає параметром HTML-елемент, в якого буде вмонтовано весь додаток та повертає об’єкт, який містить метод render() для монтування розмітки в проєкт.

Для компонентів, які знадобляться в подальшій роботі, потрібно створити папку «components», яка міститиме в собі компоненти, створені та стилізовані вручну.

Перед розробкою сторінок потрібно створити загальний шаблон, який буде незмінним для кожної сторінки. Для даної задачі в фреймворку Ant Design існує компонент Layout, який містить в собі інші компоненти, такі як Header, Content, Footer та інші. Для подальшої роботи знадобляться лише Content та Header. Дані компоненти містять в собі базові стилі та слугують для розмітки, щоб клієнтська сторінка була семантично правильно побудована.

В бібліотеці React якщо компонент може приймати інші компоненти як дочірні, використовується зарезервований props, який називається children. Він показує в яке саме місце (чи місця) буде вмонтовано дочірній компонент. В даному випадку, children буде вмонтований саме всередину компонентів Contents та Headers. В компоненті Headers будуть вмонтовані компоненти, що не змінюють розмітку, то props children є непотрібним, тому що всі компоненти будуть вставлені бкзпосередньо в компонент Headers, який створюється вручну, так званий кастомний компонент (custom component), який створюється в папці /components.

Оскільки клієнтська частина використовує TypeScript, всі prosp потрібно типізувати. Компонент Layout, який був створений вчручну, приймає лише один props – children, тип якого ReactNode – це спеціальний тип, який означає, що це компонент, який можна вмонтувати в інший компонент.

Стилізація компонентів може бути здійснена багатьома різними способами. В даному проєкі буде використано підхід модульних стилів – це підхід, коли css-файли називаються з приставкою module перед розширенням

(наприклад header.module.css). Цей підхід працює подібно до звичайного css-файлу, але в css файлі потрібно імпортувати файл зі стилями і давати йому класи, а тут – файл імпортується як об’єкт і кожен клас є полем об’єкту. Це допомагає швидше стилізувати кожен компоеннт, незалежно від класу та рівня вкладеності.

Правильна архітектура потребує розділення файлів за сутностями, тому для кожного компонента в папці /components буде створено директорію, яка називається за назвою компонента та містить в собі файл з компонентом та файл його стилізації.

Компонент Layout містить в собі лише створений компонент Header та обгортку Layout, яка імпортована з пакету «antd», яка обгортає props children.

Компонент Header містить в собі:

- логотип (для прикладу було обрано іконку зі людьми);
- назву додатку (Employees);
- іконка виходу (якщо користувач авторизований);
- кнопка logout (якщо користувач авторизований);
- іконка входу (якщо користувач неавторизований);
- кнопка login (якщо користувач неавторизований);
- іконка users (якщо користувач неавторизований);
- кнопка Sign in (якщо користувач неавторизований).

Логотип та назва додатку мають міститися з лівого боку, кнопки керування авторизацією – з правого боку. В Ant Design це робиться за допомогою компонента Space, який розтягує блоки на всю ширину чи довжину.

Для подальшої роботи потрібно створити компонент кнопки, взявши з основу компонент Button з Ant Design, для того, щоб ми могли її перевикористати в тих місцях, де це потрібно.

Даний компонент буде приймати в себе такі props:

- children – дочірні компоненти (тип React.ReactNode);
- htmlType – тип кнопки за HTML-атрибутом type (тип 'button' або

'submit' або 'reset' або undefined, необов’язковий);

- onClick – функція, яка виконається на натиснення по елементу (() => void, функція без параметрів, яка нічого не повертає, необов’язковий);
- type – тип кнопки (тип 'primary' або 'link' або 'text' або 'ghost' або 'default' або 'dashed', необов’язковий);
- danger – показник помилки (тип boolean, необов’язковий);
- loading – показник завантаження (тип boolean, необов’язковий);
- shape – форма кнопки (тип 'circle' або 'default' або 'round' або undefined, необов’язковий);
- icon – компонент іконки (тип React.ReactNode, необов’язковий);
- style – об’єкт зі стилями (тип React.CSSProperties, необов’язковий).

Всі props, окрім children та style передаються в якості props безпосередньо в компонент Button з пакету «antd». Об’єкт styles передається на обгортку кнопки для можливості більш гнучкої стилізації, а children – передається в кнопку як місце, в яке мають потрапити внутрішні елементи. Обгорткою для кнопки, полів для введеня та інших елементів, які можуть бути використані в формі, потрібно обгорнути в компонент Item, який дістається з зовнішнього компоненту Form, який імпортується з пакету «antd».

4.3.1 Налаштування головної сторінки

Налаштування головної сторінки починається зі сторінки рєєстрації. На даній сторінці присутні форма для реєстрації та повідомлення, яке переадресовує користувача на сторінку логіна, якщо він має акаунт.

Форма для реєстрації включає в себе такі частини:

- поле Name;
- поле Email;
- поле Password;

– поле Confirm password;
– кнопка Sign In для відправки запиту.

Для позиціонування по центрі використовується компонент Row з бібліотеки «antd».

Для стилізації полів для введення потрібно створити власні поля на основі компонента Input, який теж імпортується з бібліотеки. Оскільки поле для введення паролю містить в собі додаткову логіку, то потрібно створити 2 компоненти.

Ці компоненти будуть стилізуватися за принципом, який був використаний при створенні кнопки – передача props та внутрішні налаштування за потреби.

Заичайні поля для введення будуть приймати в себе:

– name – ім’я поля (тип string);
– type – тип поля (тип string);
– placeholder – речення, що виводиться, копи поле не заповнене (тип string);
– required – показник, чи обов’язкове поле для заповнення (тип boolean);
– style – об’єкт зі стилями (тип React.CSSProperties, необов’язковий).

Компонент є частиною форми тому теж має обгортку Item з компонента Form.

Для стилізації та роботи поля для введення пароля потрібно передати 2 props:

– name – ім’я поля (тип string);
– placeholder – речення, що виводиться, копи поле не заповнене (тип string).

Валідація паролю відбувається в компоненті Item, що знаходиться в іншому компоненті Form. Item має props rules, що приймає як значення масив правил, за якими поле буде вважатися валідним.

Перше правило показує, що поле є обов’язковим та виводить відповідне повідомлення ористувачеві.

Друге правило утворюється внаслідок виклику внутрішньої функції, яка параметром приймає об’єкт з полями, але в даній перевірці потрібне лише один метод getFieldValue(), що приймає параметром ім’я поля та повертає його значення з форми. Внутрішня функція повертає об’єкт з полями, а для валідації полів надається метод validator(), що приймає параметром об’єкт правила та саме значення поля. І далі йде перевірка:

- якщо немає значення – поле валідне;
- якщо значення полів password і confirmPassword однакові – поле валідне;
- якщо значення полів password і confirmPassword різне – виводиться помилка, що значення не збігають;
- якщо значення поля коротше за 6 символів – виводиться помилка, що значення занадто коротке (додаткова перевірка).

При натисканні кнопки Sign in робиться запит на реєстрацію користувача. Якщо запит пройшов успошно – відбувається перехід (редірект) на головну сторінку. Якщо відбулася помилка – вона виводиться в спеціальному компоненті Alert, що відображає повідомлення.

Запити в формі компонента Form вшдбуваються за допомогою props onFinished, який приймає як значення функцію, що має виконатися (для форми HTML це аналог функції Submit()).

Сторінка авторизації має подібну логіку та компоненти, за 2 винятками:

- відсутнє поле Confirm password;
- якщо немає акаунта, повідомлення під формою може перекинути на сторінку реєстрації.

Сторінка для відображення списку співробітників містить в собі такі елементи:

- кнопка для додавання нового працівника;

– таблиця з пагінацією;

– форма для вибору кількості працівників на одній сторінці.

Кнопка для додавання нового працівника базується на кнопці, яку створювали вручну з текcом Add, іконкою додавання та функція на натисканні на неї пересилає на сторінку створення співробіників.

Базовий вигляд головної сторінки зображено на рисунку 4.8.

Рисунок 4.8 – Базовий вигляд головної сторінки

Таблиця з пагінацією створюється за допомогою компонента Table, який імпортується з пакету «antd» та приймає багато props, серед яких:

– dataSourse – дані для відображення;

– columns – масив з інформацією про кожну колонку;

– rowKey – унікальний ключ для кожного рядка таблиці;

– loading – показник завантаження;

– onRow – функція, яка приймає параметром об’єкт рядка та повертає об’єкт рядка з параметрами;

– pagination – об’єкт з налаштуваннями для пагінації таблииці (щоб відображалися не всі рядки, а лише конкретна кількість).

Значення dataSourse – дані, які прийшли з запиту, columns – масив, створений в компоненті з інформацією про кожну колонку, rowKey – id кожного рядка, loading – показник завантаження даних з запиту, onRow – функція, в яку на кожну ітерацію ми передаємо функцію onClick, яка робить редірект на сторінку редагування співробітника та pagination – об’єкт з полем pageSize, який рівний тому значенню, що введе користувач в форма (за

змовчуванням – 5).

Таблиця має вбудовану функцію сортуванню по стовпцях. Це забезпечується а допомогою поля sorted, що задається при налаштовуванні колонок таблиці.

Останній елемент – форма для зміни кількості співробітників. Дана форма складається з назви форми, поля для введення значення та кнопки підтвердження.

Для того, щоб коритувач міг вводити лише числа, тип в поля для введення є number. Потрібно додати перевірку, що число має бути цілим та більшим 0.

Майже всі компоненти в бібліотеці React мають свій стан. Для того, щоб створити сунтність стану, використовується хук useState(), який приймає параметром початковий стан та повертає значення стану та функцію, що цей стан змінює.

В компонентах за потреби потрібно створювати функцію, яка буде виконуватися, в залежності від того який стан чи стани змінилися. В даному компоненті йде перевірка чи конкретний користувач в мережі. Дані функції виконуються за допомогою хука useEffect, який параметром приймає функцію, яка буде виконуватисся при зміні значення, які дана функція приймає в якості залежностей другим аргументом – якщо користувача не існує, його редіректить на сторінку авторизації, якщо він існує – нічого не відбувається.

Сторінки авторизації, реєстрації та виводу таблиці співробітників – це одна й та ж сторінка з боку стилізації та логіки та точки зору побудови, але для роутингу – це різні сторінки, оскільки використовується різний шлях. Є можливість створити всі ці сторінки по одному маршруту, але це вважається поганою практикою, оскільки вона буде містити в собі логіку, яка вона не буде використовувати і тим самим збільшуючи розмір коду, який треба завантажити при першому перезавантаженні сторінки.

4.3.2 Налаштування сторінок співробітників

Сторінка співробітників включає в себе відображення інформації щодо одного співробітника, його редагування та можливість видалення (якщо даного співробітника було додано з того ж акаунта, що і відбувся вхід в систему).

На сторінку створення співробітника можна протрапити, натиснувши кнопку «Add» н головній сторінці.

Сторінка додавання співробітника містить в собі лише форму з полями:

- Поле First name;
- Поле Last name;
- Поле Age;
- Поле Address;
- Кнопка Add.

Всі поля для введення, крім Age мають тип text, а поле Age – тип number. Це було зроблено для того, щоб користувач не міг ввести значення, відмінні від числа.

Оскільки форма створення та форма редагування однакові за кіоткістю полів, є сенс створити компонент форми, який можна перевикористати на різних сторінках. Даний компонент приймає в себе такі props:

- onFinish – функція, яка виконається при відправці форми, яка приймає параметром об’єкт з значеннями форми (тип – (values) => void);
- btnText – текст кнопки відправлення (тип string);
- title – назва форми (тип string);
- error – повідомлення помилки, якщо вона присутня (тип string, необов’язковий);
- employee – об’єкт з інформацією про співробітника (тип Employee).

Тип Employee має поля, які були під час створення моделі Employee, тому що даний тип був згенерований Prisma ORM ісля того, як були створені

таблиці для бази даних.

Даний компонент має обгортку, яку утворює компонент Card, імпортований з бібліотеки «antd». Ще буде імпортовано компоненти Form, який використовується для форми, Space для зручнішої стилізації та Alert для виведення інформації про створення або помилку.

Поля форми утворюються за допомогою компонентів CustomInput та CustomButton, які створені в папці /components. Для повідомлення про можливу помилку використовується компонент Alert, який приймає props message, що містить стрічку повідомлення з помилкою. Додатково прописується стилізація для форми, щоб переписати деякі стандарнті стилі, які підтягуються з бібліотеки.

Форма додавання та редагування співробітника зображено на рисунку 4.9.

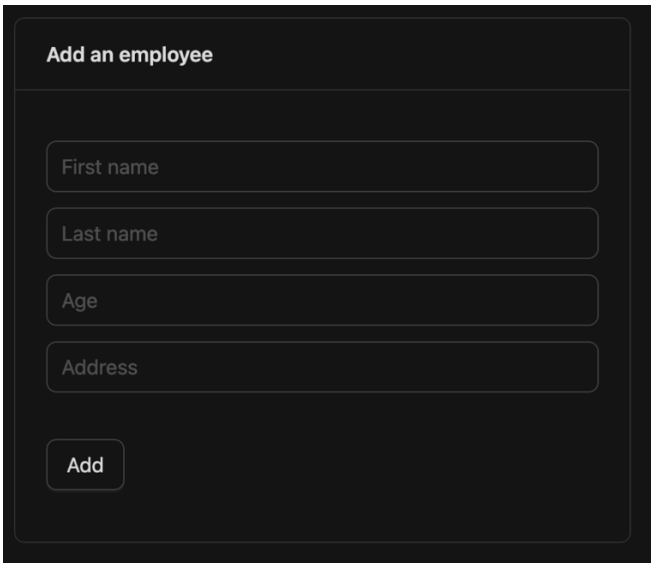

Рисунок 4.9 – Форма додавання співробітника

Слід зазаначити, що при редугуванні співробітника в відповідні поля підтягуються значення конкретного співробітника. Це відбувається, оскільки при переході на дану сторінку відбувається запит на отримання інформації.

Після успішного створення, редагування чи видалення співробітника відбувається перенаправлення на сторінку stutus з виведенням повідомлення

про успішне виконання операції та кнопкою для повернення на сторінку зі списком співробітників.

Повідомлення про успішне створення співробітника зображено на рисунку 4.10.

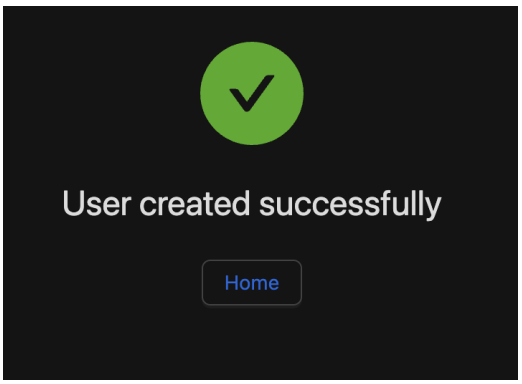

Рисунок 4.10 – Повідомлення про успішне створення співробітника

Сторінка перегляду інформації про співробітника містить таблицю з 2 колонок: назва поля та відповідне значення. Потрапити на дану сторінку можна, натиснувши на рядок в таблиці (в компоненті таблиці передавався props onRow та його поле onClick, яка містить функцію для перенаправлення).

Якщо це сторінка співробітника, якого було створено з акаунта, з якого було здійснено вхід в систему, під таблицею з інформацією буде ще додатково відображено 2 кнопки: зміни (Edit) та видалення (Delete). Після натискання кнопки Edit відбувається перенаправлення на сторінку редагування, а на кнопку Delete з’явиться модальне відно з попередженням чи дійсно користувач бажає видалити даного співробітника.

4.4 Висновки

В даному розділі було розгорнуто базовий шаблон для налаштування клієнтської частини на основі бібліотеки React, налаштування Redux Toolkit – менеджеру для контролю та передачі стану та інформації по проєкту та

налаштування сторінок авторизації, реєстрації, головної сторінки та сторінки співробітників.

Налаштування клієнтської частини включала в собі:

– створення шаблону бібліотеки React JS;
– підключення менеджержеру Redux Toolkit;
– підключення графічної бібліотеки Ant Design.

Застосунок містить в собі 7 сторінок:

– home;
– employeeAdd;
– employeeEdit;
– employee;
– status;
– login;
– register.

Для внутрішнього кешування, економії розміру js-файлів та зменшення кількості написання стилів, сторінки використовують однакові компоненти, які перевикористовуються на різних сторінках в застосунку. Якщо рахувати, що майже кожний компонент перевикористовувався, то загальний розмір js-файлу зменшився на 30% (в залежності від кількості використання компонентів).

5 ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит розробленого застосунку для керування персоналом

Як відомо, застосунки для керування персоналом широко використовуються у сучасному бізнес-середовищі, оскільки дають можливість автоматизувати бізнес-процеси та багато рутинних завдань; дозволяють збирати і аналізувати дані про працівників, що допомагає приймати більш обгрунтовані рішення щодо управління персоналом; покращують комунікації між працівниками та керівниками в підприємствах і організаціях; зберігають і впорядковують інформацію про працівників підприємства в одному місці; підвищують продуктивність праці працівників і надають інформацію щодо шляхів її підвищення тощо.

Тому метою виконаної нами магістерської роботи було створення застосунку для керування персоналом конкретного підприємства, використання якого дало б змогу суттєво зменшити витрати часу на управління персоналом підприємства. Для цього нами було проаналізовано існуючі аналоги; зроблено аналіз сучасних методів розробки застосунків, особливостей та перспектив їх використання в технологіях; розроблено клієнтський інтерфейс з використанням бібліотеки React, створено сторінки для перегляду списків працівників підприємства, їх додавання в список та видалення зі списку тощо.

В результаті було створено застосунок для керування персоналом підприємства, який допомагає автоматизувати і оптимізувати виконання багатьох рутинних завдань, пов'язаних з управлінням персоналом підприємства, виведенням детальної інформації по кожному з них та виконанням операцій над цими даними.

Все це сприятиме підвищенню ефективності управління персоналом підприємства і досягненню стратегічних цілей бізнесу.

Для встановлення комерційного потенціалу розробленого нами застосунку для управління персоналом було проведено його технологічний аудит, для чого було запрошено 3-х відомих експертів – кандидатів технічних наук, доцентів Кулика Я.А., Паламарчука Є.А. та Барабан М.В.

Встановлення комерційного потенціалу розробленого нами застосунку для управління персоналом було здійснено за критеріями, наведеними в таблиці 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1.

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
	Ринкові перспективи				
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне не-значне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансуваня ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансуваня є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансуванння
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та деше-ві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Запрошені експерти оцінили розроблений нами застосунок для управління персоналом таким чином (табл. 5.2):

Таблиця 5.2 – Результати технологічного аудиту розробленого нами застосунку для управління персоналом (за шкалою оцінювання «0»-«1»-«2»-«3»-«4»)

Критерії	Прізвище, ініціали експертів		
	Кулик Я.А.	Паламарчук Є.А.	Барабан М.В.
	Бали, що їх виставили експерти:		
1	3	3	2
2	2	3	3
3	3	3	3
4	3	3	2
5	3	3	2
6	2	3	3
7	3	3	2
8	3	3	2
9	3	3	3
10	3	3	2
11	3	3	3
12	3	3	2
Сума балів	$СБ_1 = 34$	$СБ_2 = 29$	$СБ_3 = 29$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{34+36+29}{3} = \frac{99}{3} = 33$		

Встановлення комерційного потенціалу розробленого нами застосунку для управління персоналом здійснювалося на основі рекомендацій, наведених в таблиці 5.3 [31].

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 33 бали, то це свідчить, що розроблений нами застосунок для управління персоналом має рівень комерційного потенціалу, який вважається «вище середнього».

Це пояснюється тим, що розроблений нами застосунок для управління персоналом має значно ширші функціональні можливості та дозволяє більш оперативно обробляти інформацію про працівників підприємства, що, у свою чергу, сприятиме більш кваліфікованому прийняттю управлінських рішень.

5.2 Розрахунок витрат на розроблення застосунку для управління персоналом)

На розроблення нашого застосунку було витрачено:.

А) Основна заробітна плата $З_о$ розробників, яка визначається за формулою (5.1):

$$З_0 = \frac{M}{T_p}\,[\text{грн}], \qquad (5.1)$$

де:

– де М – місячний посадовий оклад розробника, грн; приймемо, що

М = (6700…20000) грн/місяць;

– $Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 20 днів;

– t – число днів роботи розробників.

Зроблені розрахунки зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на оплату праці, грн
1.Науковий керівник магістерської роботи	18000	900,00	20 годин	≈ 3000
2. Магістрант-студент виконавець	2000 (беремо 6700)	335,00	60	≈ 20100
3. Консультант з економічної частини	17000	850,00	1,5 години	≈ 213 (при 6 годинному робочому дні)
Загалом				Зо = 23313 грн

Б) Додаткова заробітна плата $З_д$ розробників розраховується за формулою (5.2) як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_Д = \alpha \times З_0 = (0{,}1 \ldots 0{,}12) \times З_0 \quad (5.2)$$

Приймемо, що α = 0,108. Тоді для нашого випадку отримаємо:

$$З_Д = 0{,}108 \times 23313 = 2517{,}80 \approx 2518 \text{ (грн)}.$$

В) Нарахування на заробітну плату НЗПзп розробників (дослідників) розраховуються за формулою (5.3):

$$НЗП_{ЗП} = (З_0 + З_Д) \times \frac{\beta}{100}, \quad (5.3)$$

де:

– β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. β = 22%.

Тоді:

$НЗН_{зп}$ = (23313 + 2518) × 0,22 = 5682,82 ≈ 5683 (грн).

Г) Амортизація основних засобів А, які використовувались під час виконання даної роботи розраховується за формулою (5.4):

$$А = \frac{Ц \times Н}{100} \times \frac{Т}{12}, \quad (5.4)$$

де:

– Ц – загальна балансова вартість основних засобів, грн;

– Н – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що $Н_а$ = (5...25)%;

– Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	30000	25	3,0 (при 50% використ-танні)	≈ 938
2. Приміщення університету, кафедри	20000	4	3,0 (при 20% використ-танні)	40
Всього				А = 978 грн

Д) Витрати на матеріали М розраховуються за формулою (5.5):

$$М = \sum_i^n Н_і \times Ц_і \times К_і - \sum_і^n В_і \times В_і, \quad (5.5)$$

де:

– $Н_і$ – витрати матеріалу і-го найменування, кг;

– $Ц_і$ – вартість матеріалу і-го най-менування;

– Кі – коефіцієнт транспортних витрат, $К_і$ = (1,1…1,15);

– $В_і$ – маса відходів матеріалу і-го найменування;

– $Ц_в$ – ціна відходів матеріалу і-го найменування;

– n – кількість видів матеріалів.

Е) Витрати на комплектуючі К розраховуються за формулою (5.6):

$$К = \sum_{i}^{n} Н_і \times Ц_і \times К_і \text{ [грн]}, \quad (5.6)$$

де:

– $Н_і$ – кількість комплектуючих і-го виду, шт.;

– $Ц_і$ – ціна комплектуючих і-го виду;

– $К_і$ – коефіцієнт транспортних витрат, $К_і$ = (1,1…1,15);

– n – кількість видів комплектуючих.

Під час виконання роботи загальні витрати на матеріали та комплектуючі склали приблизно 300 грн.

Ж) Витрати на силову електроенергію $В_е$ розраховуються за формулою (5.7):

$$В_е = \frac{В \times П \times Ф \times К_П}{К_Д}, \quad (5.7)$$

де:

– В – вартість 1 кВт-год. електроенергії, в 2023 р. В ≈ 4,5 грн/кВт;

– П – установлена потужність обладнання, кВт; П = 0,5 кВт;

– Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 200 годин;

– $К_п$ – коефіцієнт використання потужності; $К_п$ < 1 = 0,7;

– $К_д$ – коефіцієнт корисної дії, $К_д$ = 0,6.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \times П \times Ф \times К_П}{К_Д} = \frac{4{,}5 \times 0{,}5 \times 200 \times 0{,}7}{0{,}6} = 525 \text{ (грн)}.$$

И) Інші витрати Вінш можна прийняти як (50…300)% від основної

заробітної плати розробників розрахувати за формулою (5.8), тобто:

$$В_{інш} = (0,5....3) \times З_о. \quad (5.8)$$

Для нашого випадку отримаємо:

$В_{інш} = 0,5 \times 23313 = 11656,5 \approx 11657$ (грн).

К) Сума всіх попередніх статей витрат складає витрати на виконання роботи безпосередньо розробником-магістрантом – В.

В = 23313+ 2518 + 5683+ 978 + 300 + 525 + 11657 = 44974 (грн).

Л) Загальні витрати на розроблення застосунку для управління персоналом Взаг розраховуються за формулою (5.9):

$$В_{заг} = \frac{В}{\beta}, \quad (5.9)$$

де:

– β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи.

Можна прийняти, що, $\beta \approx 0,95$, оскільки робота практично завершена.

Тоді: $В_{заг} = \frac{44974}{0,95} = 47341,05$ (грн) або приблизно 48 тисяч грн.

Тобто прогнозовані загальні витрати на розроблення застосунку для управління персоналом Взаг становлять приблизно 48 тисяч грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації нашої розробки

Економічний ефект від можливої комерціалізації розробленого нами застосунку для управління персоналом пояснюється його значно кращими функціональними можливостями, суттєвим покращенням управління персоналом підприємства, більш оперативною обробкою інформації про

працівників підприємства тощо.

Тому нашу розробку можна буде реалізовувати на ринку дещо дорожче, ніж аналогічні на гірші за подібними функціями розробки. Так, якщо подібна за функціями розробка у 2022 році коштувала для потенційного клієнта на ринку приблизно 7000 грн, то нашу розробку можна буде реалізовувати приблизно за 10 тисяч грн або на 3000 грн дорожче.

Аналіз місткості ринку показує, що сьогодні в Україні кількість реальних користувачів послугами подібних розробок може складати приблизно 5 осіб. Але у зв'язку зі стрімким розвитком процесів діджиталізації можна очікувати суттєве зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2024 року, то її результати будуть виявлятися протягом 2024-го, 2025-го та 2026-го років.

Прогноз зростання попиту на нашу розробку може становити по роках:

а) 2024 р. – приблизно +10 шт. до базового року;

б) 2025 р. – +20 шт. до базового року;

в) 2026 р. – +30 шт. до базового року.

Можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації, тобто виведення нашої розробки на ринок, за формулою (5.10) становитиме:

$$\Delta П_i = \sum_1^n (\Delta Ц_0 \times N + Ц_0 \times \Delta N)_i \times \lambda \times \rho \times (1 - \frac{\nu}{100}), \qquad (5.10)$$

де:

- $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів розробки у цьому році. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_o$ = (10000 – 7000) = 3000 (грн);
- N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 10 шт.;
- ΔN – покращення основного кількісного показника від

впровадження результатів нашої розробки. Таке покращення становитиме по роках, відповідно: у 2024 році – + 10 шт., у 2025 році – +20 шт, у 2026 році – +30 шт. (до базового 2023-го року);

- $Ц_о$ – основний якісний показник (тобто ціна), який визначає обсяг діяльності у році після впровадження результатів розробки, грн; $Ц_о$= 7000 грн;
- n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;
- λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;
- ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати $\rho = (0{,}2...0{,}5)$; візьмемо $\rho = 0{,}5$;
- υ – ставка податку на прибуток. У 2024-2026 роках υ = 18%.

Тоді можливе зростання чистого прибутку $\Delta П1$ для потенційного інвестора протягом першого року від можливого впровадження нашої розробки (2024 р.) складе:

$\Delta П_1 = [3 \times 10 + 10 \times 10] \times 0{,}8333 \times 0{,}5 \times (1 - \frac{18}{100}) \approx 44{,}41$ тисяч грн або приблизно 45 тисяч грн.

Можливе зростання чистого прибутку $\Delta П2$ для потенційного інвестора від можливого впровадження нашої розробки протягом другого (2025 р.) року складе:

$\Delta П_2 = [3 \times 10 + 10 \times 20] \times 0{,}8333 \times 0{,}5 \times (1 - \frac{18}{100}) \approx 78{,}58$ тисяч грн або приблизно 79 тисяч грн.

Можливе зростання чистого прибутку $\Delta П3$ для потенційного інвестора від можливого впровадження нашої розробки протягом третього (2026 року) складе:

$П_3 = [3 \times 10 + 10 \times 30] \times 0{,}8333 \times 0{,}5 \times (1 - \frac{18}{100}) \approx 112{,}74$ тисяч грн або приблизно 113 тисяч грн.

Приведена вартість зростання всіх чистих прибутків від можливого впровадження нашої розробки за формулою (5.11), становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.11)$$

де:

- $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої роботи, грн;
- т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;
- τ – ставка дисконтування. Приймемо τ = 0,10 (10%);
- t – період часу від моменту початку розроблення застосунку для управління персоналом до моменту отримання можливих чистих прибутків.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{45}{(1+0,1)^2} + \frac{79}{(1+0,1)^3} + \frac{113}{(1+0,1)^4} \approx 37 + 59 + 85 = 181 \text{ тисяча грн.}$$

Теперішня вартість інвестицій PV, що повинні бути вкладені в реалізацію нашої розробки: $PV = (1,0…5,0) \times В_{заг}$.

Для нашого випадку $PV = (1,0…5,0) \times 48 = 1 \times 48 = 48$ тисяч грн.

Розраховуємо абсолютний ефект від можливих вкладених інвестицій $Е_{абс}$ за фомрулою (5.12) складатиме:

$$Е_{абс} = ПП - PV, \quad (5.12)$$

де:

- ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливого впровадження нашої розробки, грн;
- PV – теперішня вартість інвестицій PV = 48 тисяч грн.

Абсолютний ефект від можливого впровадження нашої розробки (при прогнозованому ринку збуту) складе:

$Е_{абс}$ = 181 – 48 = 133 тисяч грн.

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною.

Далі за формулою (5.13) розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (5.13)$$

де:

– $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 133 тис. грн;

– PV –теперішня вартість початкових інвестицій PV = 48 тис. грн;

– $Т_ж$ – життєвий цикл розробки, роки;

– $Т_ж$ = 4 років (2023-й, 2024-й, 2025-й, 2026-й роки).

Для нашого випадку отримаємо:

$$Е_в = \sqrt[4]{1 + \frac{133}{48}} - 1 = \sqrt[4]{1 + 2,7708} - 1 = \sqrt[4]{3,7708} - 1 = 1,394 - 1 = 0,394 = 39,4\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність або мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою (5.14):

$$\tau_{мін} = d + f, \quad (5.14)$$

де:

– d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d = (0,10...0,12);

– f – показник, що характеризує ризикованість вкладень; f = (0,05...0,30).

Для нашого випадку отримаємо:

$$\tau_{мін} = 0,10 + 0,30 = 0,40 \text{ або } \tau_{мін} = 40\%$$

Оскільки величина $E_в = 39{,}4\% \approx \tau_{мін} = 40\%$, то потенційний інвестор у принципі може бути зацікавлений у фінансуванні та комерціалізації нашої розробки.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленого нами застосунку для управління персоналом.

Термін окупності $T_{ок}$ розраховується за формулою (5.15):

$$T_{ок} = \frac{1}{E_в}. \qquad (5.15)$$

Для нашого випадку термін окупності $T_{ок}$ коштів становитиме:

$$T_{ок} = \frac{1}{0,394} \approx 2,54 \text{ років} < 3 \text{ років},$$

що свідчить про потенційну доцільність комерціалізації розробленого нами застосунку для управління персоналом.

Далі проведено моделювання залежності величини внутрішньої дохідності вкладених потенційних інвестицій від рівня інфляції в країні. Як відомо, рівень інфляції в країні в 2024-му та наступних роках може зрости (наприклад, до 20%).

Прийнявши рівень інфляції у 20% отримаємо:

$$ПП = \frac{45}{(1+0,2)^2} + \frac{79}{(1+0,2)^3} + \frac{113}{(1+0,2)^4} \approx 32 + 46 + 55 = 133 \text{ тисячі грн.}$$

Тоді абсолютний ефект від можливого впровадження нашої розробки складе:

$$E_{абс} = 133 - 48 = 85 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

де:

- $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс} = 85$ тисяч грн;
- PV –теперішня вартість початкових інвестицій PV = 48 тисяч грн.

$$E_в = \sqrt[4]{1+\frac{85}{48}} - 1 = \sqrt[4]{1+1,7708} - 1 = \sqrt[4]{2,7708} - 1 = 1,29 - 1 = 0,29 = 29,0\%.$$

Оскільки величина $E_в$ = 29,0% $< \tau_{мін} = 40\%$, то потенційний інвестор може бути не зацікавлений у фінансуванні та комерціалізації нашої розробки.

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

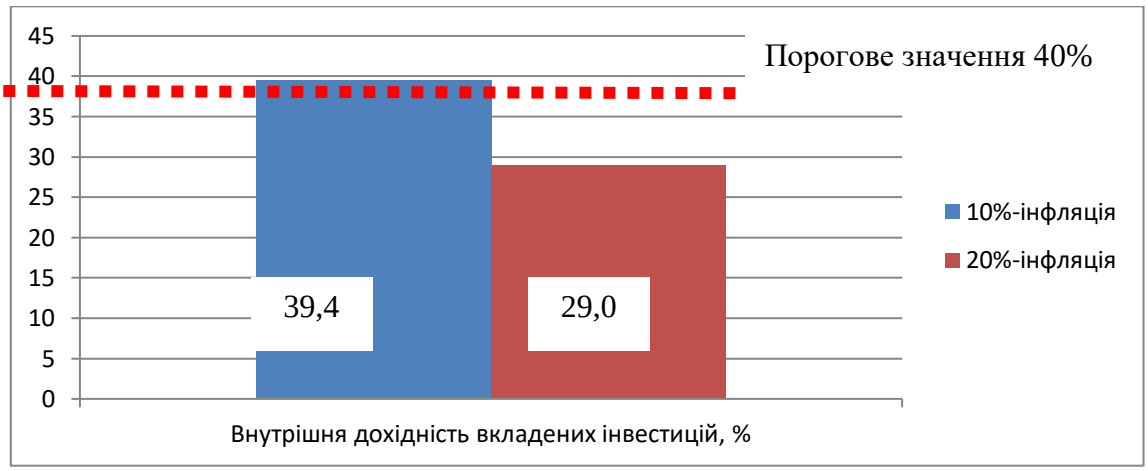

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій від рівня інфляції в країні

Аналіз діаграм на рис 5.1 показує, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в$ = 39,4%, що практично дорівнює пороговому значенню $\tau_{мін}$ = 40% і тому комерціалізація нашої розробки може бути доцільною; при рівні інфляції в 20% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію нашої розробки, становить $E_в$ = 29,0%, що значно менше порогового значення $\tau_{мін}$ = 40%, і тому комерціалізація нашої розробки може бути проблематичною. Для прийняття остаточного рішення потрібно провести додаткові обгрунтування і розрахунки (наприклад, знизити рівень прийнятого ризику, який ми прийняли за 30%, або підняти ціну реалізації нашої розробки, або суттєво збільшити попит на нашу розробку тощо).

Разом з тим, потрібно зазначити, що абсолютні величини прибутку, що їх може отримати потенційний інвестор, є дуже малими, і при таких цінах на

реалізацію розробки та величини попиту на неї інвестор наврядчи візьметься за її комерціалізацію (!).

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю 5.5:

Таблиця 5.5 – Результати виконаної економічної частини

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 50 тис. грн	48 тис. грн.	Досягнуто
2. Абсолютний ефект від впровадження розробки, тисяч грн	Не менше 100 тисяч грн	133 тисяч грн (при 10%-інфляції)	Виконано
3. Внутрішня дохідність інвестицій, %	не менше 40%	39,4% (при 10%-інфляції)	Досягнуто
4. Термін окупності інвестицій, роки	до 3-ти років	2,54 років (при 10%-інфляції)	Виконано

Таким чином, основні техніко-економічні показники розробленого нами застосунку для управління персоналом, визначені у технічному завданні, виконані.

ВИСНОВКИ

У першому розділі було проведено аналіз застосунків для керування персоналом. Розглянуто переваги та недоліки світових та українських аналогів. Для подальшої роботи було обрано використання фрейворків та бібліотек, оскільки вони теж забезпечують зручність використання, але вони легше підтримуються та не потрібно платити за послуги, оскільки весь функціонал йде безкоштовно.

У другому розділі було проведено аналіз сучасних методів розробки застосунків, особливостей та перспектив їх використання в технологіях, які стосуються web-додатків є актуальною задачею в галузі інформаційних систем та технологій. Розглянуто переваги та недоліки фреймворків, які можуть вирішити завдання. На основі порівняльного аналізу переваг і недоліків технологій розробки та фреймворків вибрані React для клієнтської частини, Ant Design для стилізації, Node.js як середовище розробки, Express.js для серверної розробки з використанням Prisma ORM та бази даних SQLite.

В третьому розділі було налаштовано серверну частину зв допомогою Express.js, налаштувано середовища розробки, підключено базу даних SQLite, створено моделі, на основі яких було створено таблиці та розгорнуто Prisma ORM.

В четвертому розділі було розгорнуто базовий шаблон для налаштування клієнтської частини на основі бібліотеки React, налаштування Redux Toolkit – менеджеру для контролю та передачі стану та інформації по проєкту та налаштування сторінок авторизації, реєстрації, головної сторінки та сторінки співробітників. Для внутрішнього кешування, економії розміру js-файлів та зменшення кількості написання стилів, сторінки використовують однакові компоненти, які перевикористовуються на різних сторінках в застосунку. Якщо рахувати, що майже кожний компонент перевикористовувався, то загальний розмір js-файлу зменшився на 30% (в залежності від кількості використання компонентів).

В п’ятому розділі було проведено економічну частину та було виконано основні техніко-економічні показники розробленого нами застосунку для управління персоналом, визначені у технічному завданні. З результатів потрібно відмітити, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в$ = 39,4%, що практично дорівнює пороговому значенню $\tau_{мін} = 40\%$ і тому комерціалізація нашої розробки може бути доцільною; при рівні інфляції в 20% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію нашої розробки, становить $E_в$ = 29,0%, що значно менше порогового значення $\tau_{мін}$ = 40%, і тому комерціалізація нашої розробки може бути проблематичною. Для прийняття остаточного рішення потрібно провести додаткові обгрунтування і розрахунки (наприклад, знизити рівень прийнятого ризику, який ми прийняли за 30%, або підняти ціну реалізації нашої розробки, або суттєво збільшити попит на нашу розробку тощо).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кулик Я.А, Мельник Ю.В. Застосунок для керування персоналом за допомогою React JS та NodeJS. Молодь в науці: дослідження, проблеми, перспективи (МН-24) : веб-сайт. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/index.
2. Бланшард, Кен, Ходжес, Філ. Сервант-лідерство. Київ: Шкільний світ. 2007, 247с.
3. Демарко Т., Лістер Т. "Peopleware: Productive Projects and Teams." Київ : Символ-Плюс, 2013. 272 с.
4. Клеппманн М. "Designing Data-Intensive Applications." CA : O'Reilly Media, 2017. 616 с.
5. Pic E. "The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses." UK : Crown Business, 2011. 336 с.
6. Сазерленд Д. "Scrum: The Art of Doing Twice the Work in Half the Time." UK : Penguin, 2014. 256 с.
7. Коне М. "Agile Estimating and Planning." NJ : Prentice Hall, 2005. 368 с.
8. Пінк Д. "Drive: The Surprising Truth About What Motivates Us." UK : Riverhead Books, 2009. 256 с.
9. Світцлер А., Максфілд Д., Макміллан Р. "Crucial Conversations: Tools for Talking When Stakes Are High." NY : McGraw-Hill Education, 2011. 288 с.
10. Делсім-Гіббс К. "Measure What Matters: Online Tools for Understanding Customers, Social Media, Engagement, and Key Relationships." NJ : Wiley, 2011. 272 с.
11. Холмс С. Стек MEAN. Mongo, Express, Angular: навч. посіб. Київ: Шкільний світ, 2017. 496 с.
12. Фланаган Д. Основи JavaScript. Київ: Шкільний світ. 2007, 1104с.
13. Крекфорд Д. JavaScript: The Good Parts. CA : O'Reilly Media, 2008. 176с.
14. Хавербейк М. Eloquent JavaScript. CA : No Starch Press, 2018. 472с.

15. Стефанов С. JavaScript Patterns. CA : O'Reilly Media, 2010. 236с.
16. Османі Е. Learning JavaScript Design Patterns. CA : O'Reilly Media, 2012. 150с.
17. Ambler S, Sadalage P. Refactoring Databases: Evolutionary Database Design. MA : Addison-Wesley Professional, 2006. 384с.
18. Mario Casciaro, Luciano Mammino. Node.js Design Patterns. Packt Publishing, 2014. 454с.
19. Cantelon M, Harter M. Node.js in Action. NY : Manning Publications, 2013. 416с.
20. Wandschneider M. Learning Node.js. MA : Addison-Wesley, 2013. 396с.
21. Wilson J. Node.js the Right Way. NC : Pragmatic Bookshelf, 2013. 334с.
22. Денис Завацький. Все що потрібно знати про Node.js: особливості, характеристики, область застосування : блог. URL: https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs (дата звернення: 15.10.2023).
23. Simpson K. You Don't Know JS. CA : O'Reilly Media, 2014. 98с.
24. Prisma. Блог : веб-сайт. URL: https://www.prisma.io/blog (дата звернення: 15.10.2023).
25. SQLite Tutorial. SQLite Tutorial : веб-сайт. URL: https://www.sqlitetutorial.net/ (дата звернення: 15.10.2023).
26. Kokane K. Introduction to Ant Design. 2020: веб-сайт, URL: https://blog.logrocket.com/introduction-to-ant-design/ (дата звернення: 15.10.2023).
27. CASES. Що таке React JS? Як почати вивчати Реакт? Навички для react developer : блог. URL: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer (дата звернення: 15.10.2023).
28. Дмитро Берестень. Відкрита JS бібліотека React? : блог. URL: https://web-developer.in.ua/assets/articles/react/react-js-library/react-js-library.html (дата звернення: 15.10.2023).

29. Дмитро Берестень. Як працює React.js під капотом? : блог. URL: https://web-developer.in.ua/assets/articles/react/react-works/react-works.html (дата звернення: 15.10.2023).
30. Дмитро Берестень. Вивчення розширених шаблонів React: Compound Components, Render Props та Hooks : блог. URL: https://web-developer.in.ua/assets/articles/react/react-compound/react-compound.html (дата звернення: 15.10.2023).
31. Козловський В. О., Лесько О.Й., Кавецький В.В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2021. 42 с.

ДОДАТКИ

Додаток А (Обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

З А Т В Е Р Д Ж У Ю
Завідувач кафедри АІІТ
д.т.н., професор Олег БІСІКАЛО
_____
(підпис)
«____» _____ 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
ЗАСТОСУНОК ДЛЯ АВТОМАТИЗОВАНОГО КЕРУВАННЯ ПЕРСОНАЛОМ З ВИКОРИСТАННЯМ REACTJS ТА NODEJS
08-31.МКР.013.02.000 ТЗ

Керівник к.т.н. доцент кафедри АІІТ
_____Ярослав КУЛИК
« _____» _____ 2023 р.
Розробив студент гр. 1АКІТ-22м
_____Юрій МЕЛЬНИК
«_____» _____ 2023 р.

Вінниця ВНТУ 2023

1 Назва та галузь застосування.

Застосунок для автоматизованого керування персоналом за допомогою ReactJS та NodeJS. Управління персоналом.

2 Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____2023р., та індивідуальне завдання на МКР, затверджене протоколом №__ засідання кафедри АІІТ від «__» _____ 2023р.
Термін виконання робіт:

3 Мета та вихідні дані для проведення робіт.

Метою магістерської кваліфікаційної роботи є зменшення витрат часу на управління персоналом.

4 Джерела розробки.

4.1 Кулик Я.А, Мельник Ю.В. Застосунок для керування персоналом за допомогою React JS та NodeJS. 2023, 2с.
4.2 Холмс Саймон. Стек MEAN. Mongo, Express, Angular: навч. посіб. Київ: Шкільний світ, 2017. 496 с
4.3 Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальностей 126 – «Інформаційні системи та технології», 151 – «Автоматизація та комп’ютерно-інтегровані технології» / Уклад. Р. Н. Квєтний, О. М. Бевз, О. В. Бісікало, Маслі Р.В. – Вінниця: ВНТУ, 2020. – 34 с.

5 Технічні дані.

5.1 Оперативна пам’ять – 2 GB,
5.2 Вільного місця – 1,1 GB,
5.3 Браузер – Google Chrome (не менше 8 версії)

6 Економічні показники.

До економічних показників входять:
-витрати на розробку 48 тисяч гривень
-мінімальна дохідність 39,4% при 10% інфляції
-термін окупності не більше 3-х років

7 Стадії розробки.

а) Дослідження аналогів. Порівняння методів. –
б) Вибір технологій. Порівняння методів розробки –
в) Написання серверної частини –
г) Створення клієнтського інтерфейсу –
е) Підтвердженння економічної доцільності –
є) Оформлення матеріалів до захисту МКР –

8 Порядок контролю та приймання.

Рубіжний контроль провести до «___» _____ 2023 р.
Попередній захист МКР провести «___» _____ 2023 р.
Захист МКР провести до «___» _____ 2023 р.

Розробив студент групи 1АКІТ-22м _____Юрій МЕЛЬНИК

Додаток Б

(Обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА
ДІАГРАМИ РОБОТИ ЗАСТОСУНКУ

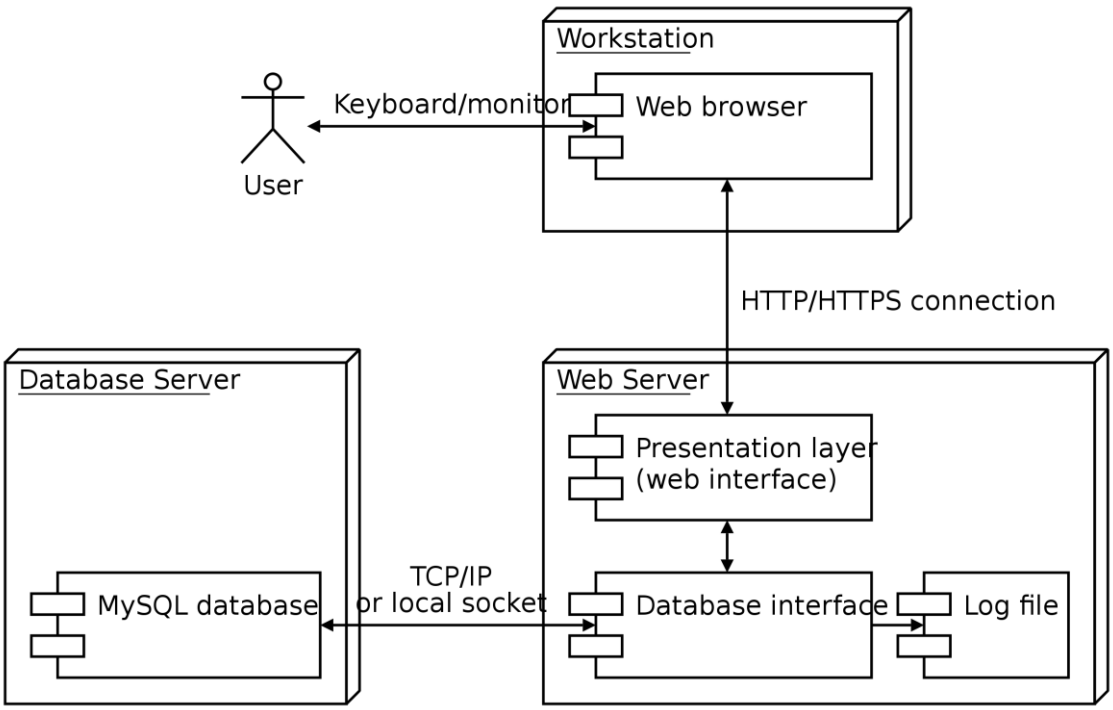

Рисунок Б.1 – Діаграма розгортання

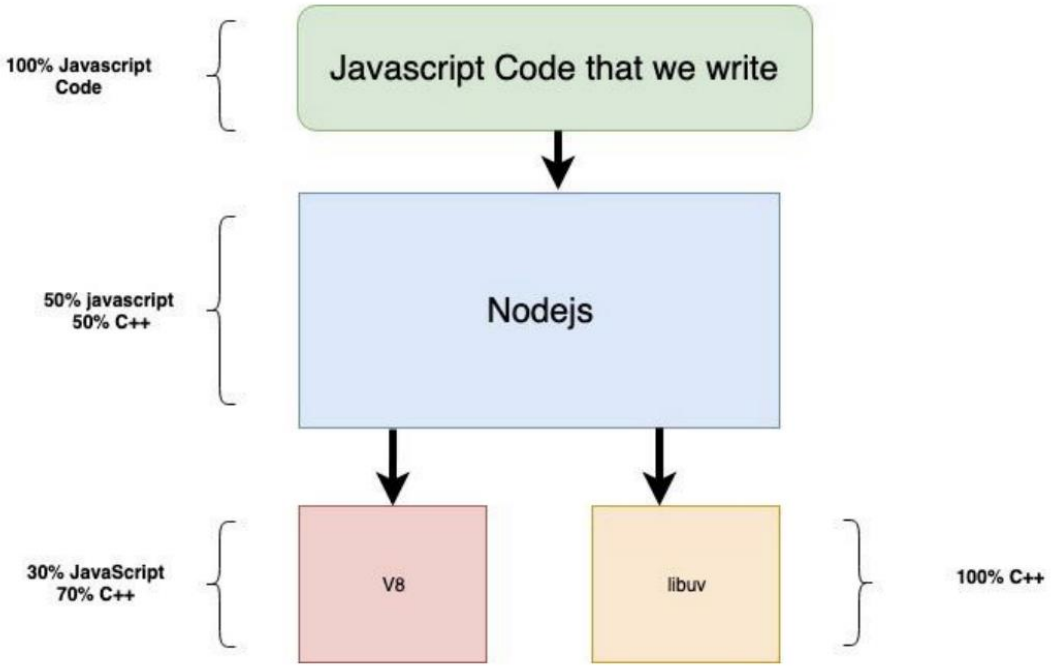

Рисунок Б.2 – Діаграма роботи Node.js

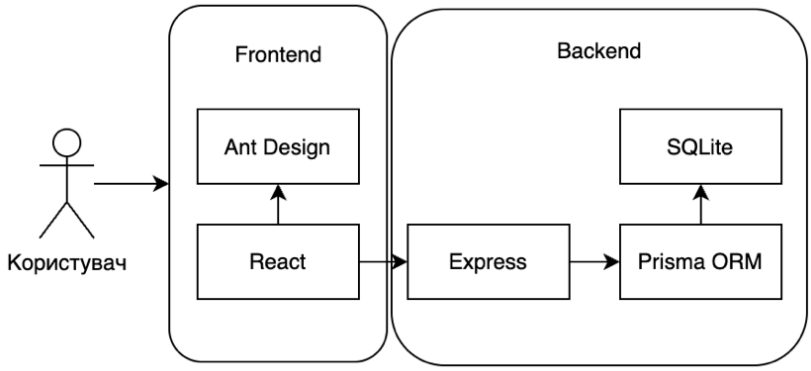

Рисунок Б.3 – Діаграма взаємодії всіх вибраних технологій та користувача

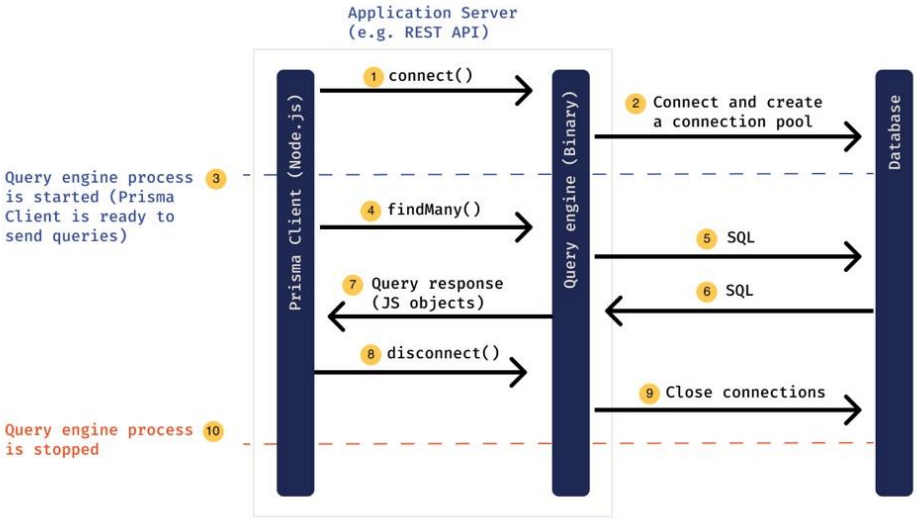

Рисунок Б.4 – Діаграма лінії життя Prisma ORM

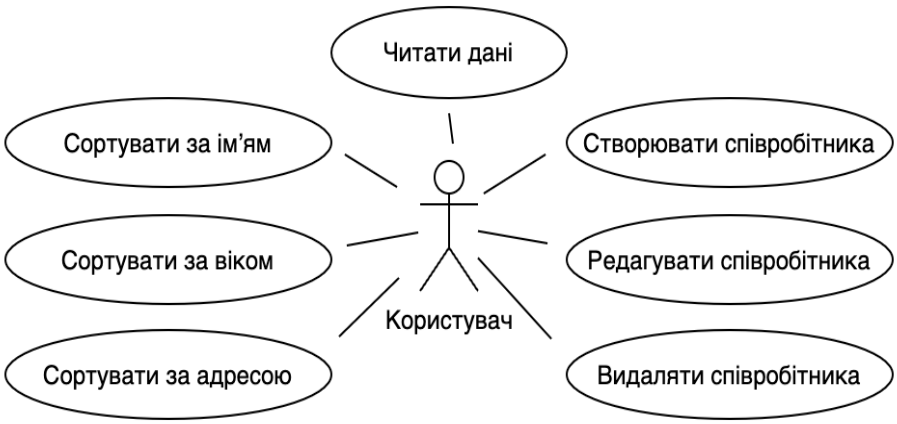

Рисунок Б.5 – Діаграма діяльності користувача із доступним функціоналом

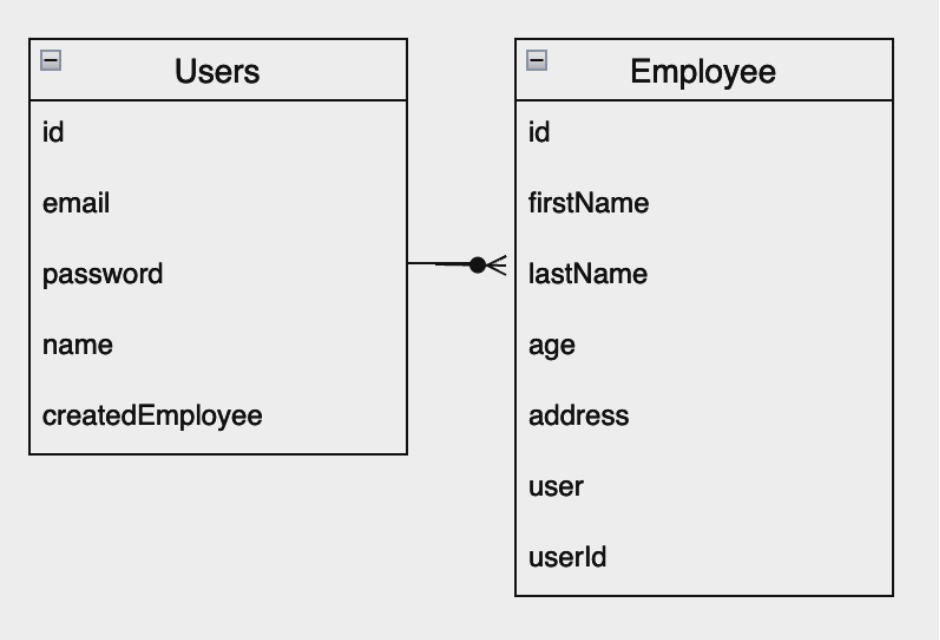

Рисунок Б.6 – Діаграма класів

Додаток В

(Обов'язковий)

Лістинг програми

Лістинг коду підключення шаблону React до HTML-документу

```
import './index.css';

import { ConfigProvider, theme } from 'antd';
import { RouterProvider, createBrowserRouter } from 'react-router-dom';

import { AddEmployee } from './pages/add-employee';
import { Auth } from './features/auth/auth';
import { EditEmployee } from './pages/edit-employee';
import { Employee } from './pages/employee';
import { Employees } from './pages/employees';
import { Login } from './pages/login';
import { Paths } from './paths';
import { Provider } from 'react-redux';
import React from 'react';
import { Register } from './pages/register';
import { Status } from './pages/status';
import { createRoot } from 'react-dom/client';
import reportWebVitals from './reportWebVitals';
import { store } from './app/store';


const router = createBrowserRouter([
 {
   path: Paths.home,
   element: <Employees />,
 },
 {
   path: Paths.login,
   element: <Login />,
 },
 {
   path: Paths.register,
   element: <Register />,
 },
 {
   path: Paths.employeeAdd,
   element: <AddEmployee />,
 },
 {
   path: `${Paths.employee}/:id`,
   element: <Employee />,
 },
```

```
  {
    path: `${Paths.employeeEdit}/:id`,
    element: <EditEmployee />,
  },
  {
    path: `${Paths.status}/:status`,
    element: <Status />,
  },
]);

const container = document.getElementById('root')!;
const root = createRoot(container);

root.render(
  <React.StrictMode>
    <Provider store={store}>
      <ConfigProvider
        theme={{
          algorithm: theme.darkAlgorithm,
        }}
      >
        <Auth>
          <RouterProvider router={router} />
        </Auth>
      </ConfigProvider>
    </Provider>
  </React.StrictMode>
);

reportWebVitals();
```

Лістинг коду домашньої сторінки

```
import { Card, Form, Row, Space, Typography } from 'antd';
import { useState, useEffect } from 'react';
import { useSelector } from 'react-redux';
import { Link, useNavigate } from 'react-router-dom';
import { useLoginMutation, UserData } from '../../app/serivices/auth';
import { CustomButton } from '../../components/custom-button';
import { CustomInput } from '../../components/custom-input';
import { ErrorMessage } from '../../components/error-message';
import { Layout } from '../../components/layout';
import { PasswordInput } from '../../components/password-input';
import { selectUser } from '../../features/auth/authSlice';
import { Paths } from '../../paths';
import { isErrorWithMessage } from '../../utils/is-error-with-message';

export const Login = () => {
  const navigate = useNavigate();
  const [error, setError] = useState('');
```

```
  const user = useSelector(selectUser);
  const [loginUser, loginUserResult] = useLoginMutation();


  useEffect(() => {
    if (user) {
      navigate('/');
    }
  }, [user, navigate]);


  const login = async (data: UserData) => {
    try {
      await loginUser(data).unwrap();

      navigate('/');
    } catch (err) {
      const maybeError = isErrorWithMessage(err);

      if (maybeError) {
        setError(err.data.message);
      } else {
        setError('Unknown error');
      }
    }
  };


  return (
    <Layout>
      <Row align='middle' justify='center'>
        <Card title='Login' style={{ width: '30rem' }}>
          <Form onFinish={login}>
            <CustomInput type='email' name='email' placeholder='Email' />
            <PasswordInput name='password' placeholder='Password' />
            <CustomButton type='primary' htmlType='submit' loading={loginUserResult.isLoading}>
              Login
            </CustomButton>
          </Form>
          <Space direction='vertical' size='large'>
            <Typography.Text>
              Don't have an account? <Link to={Paths.register}>Sign in</Link>
            </Typography.Text>
            <ErrorMessage message={error} />
          </Space>
        </Card>
      </Row>
    </Layout>
  );
};
```

Лістинг коду сторінки співробітників

```
import { useEffect, useState } from 'react';
import { Col, Form, Row, Space, Table, Typography } from 'antd';
import type { ColumnsType } from 'antd/es/table';
import { PlusCircleOutlined } from '@ant-design/icons';
import { CustomButton } from '../../components/custom-button';
import { Employee } from '@prisma/client';
import { Paths } from '../../paths';
import { useNavigate } from 'react-router-dom';
import { useGetAllEmployeesQuery } from '../../app/serivices/employees';
import { Layout } from '../../components/layout';
import { selectUser } from '../../features/auth/authSlice';
import { useSelector } from 'react-redux';
import { CustomInput } from '../../components/custom-input';

const columns: ColumnsType<Employee> = [
  {
    title: 'Name',
    dataIndex: 'firstName',
    key: 'firstName',
    sorter: {
      compare: (a, b) => a.firstName.localeCompare(b.firstName),
    },
    sortDirections: ['ascend', 'descend', 'ascend'],
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    sorter: {
      compare: (a, b) => +a.age - +b.age,
    },
    sortDirections: ['ascend', 'descend', 'ascend'],
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
    sorter: {
      compare: (a, b) => a.address.localeCompare(b.address),
    },
    sortDirections: ['ascend', 'descend', 'ascend'],
  },
];

export const Employees = () => {
  const [rowsPerPage, setRowsPerPage] = useState(5);
  const navigate = useNavigate();
  const user = useSelector(selectUser);
  const { data, isLoading } = useGetAllEmployeesQuery();

  useEffect(() => {
```

```
  if (!user) {
    navigate('/login');
  }
 }, [user, navigate]);

 const gotToAddUser = () => navigate(Paths.employeeAdd);
 const changeRowsPerPage = (data: { rows: string }): void => {
  if (+data.rows >= 0) setRowsPerPage(+data.rows);
 };

 return (
  <Layout>
   <Row style={{ marginBottom: '24px' }}>
    <Col span={8}>
     <CustomButton type='primary' onClick={gotToAddUser} icon={<PlusCircleOutlined />}>
      Add
     </CustomButton>
    </Col>
    <Col span={8} offset={8}>
     <Form
      onFinish={changeRowsPerPage}
      style={{ display: 'flex', gap: '10px', justifyContent: 'flex-end', alignItems: 'flex-end' }}
     >
      <Space direction='vertical'>
       <Typography>Rows per page</Typography>
       <CustomInput type='number' name='rows' placeholder='Rows' required={false} />
      </Space>
      <CustomButton type='primary' htmlType='submit' style={{ margin: '0' }}>
       Confirm
      </CustomButton>
     </Form>
    </Col>
   </Row>
   <Table
    loading={isLoading}
    rowKey={(record) => record.id}
    columns={columns}
    dataSource={data}
    pagination={{ pageSize: rowsPerPage }}
    onRow={(record) => {
     return {
      onClick: () => navigate(`${Paths.employee}/${record.id}`),
     };
    }}
   />
  </Layout>
 );
};
```

Лістинг коду сторінки співробітника

```
import { DeleteOutlined, EditOutlined } from '@ant-design/icons';
```

```
import { Descriptions, Divider, Modal, Space } from 'antd';
import { Link, Navigate, useNavigate, useParams } from 'react-router-dom';
import { useGetEmployeeQuery, useRemoveEmployeeMutation } from '../../app/serivices/employees';

import { CustomButton } from '../../components/custom-button';
import { ErrorMessage } from '../../components/error-message';
import { Layout } from '../../components/layout';
import { Paths } from '../../paths';
import { isErrorWithMessage } from '../../utils/is-error-with-message';
import { selectUser } from '../../features/auth/authSlice';
import { useSelector } from 'react-redux';
import { useState } from 'react';

export const Employee = () => {
  const navigate = useNavigate();
  const [error, setError] = useState('');
  const params = useParams<{ id: string }>();
  const [isModalOpen, setIsModalOpen] = useState(false);
  const { data, isLoading } = useGetEmployeeQuery(params.id || '');
  const [removeEmployee] = useRemoveEmployeeMutation();
  const user = useSelector(selectUser);

  if (isLoading) {
    return <span>Loading...</span>;
  }

  if (!data) {
    return <Navigate to='/' />;
  }

  const showModal = () => {
    setIsModalOpen(true);
  };

  const hideModal = () => {
    setIsModalOpen(false);
  };

  const handleDeleteUser = async () => {
    hideModal();

    try {
      await removeEmployee(data.id).unwrap();

      navigate(`${Paths.status}/deleted`);
    } catch (err) {
      const maybeError = isErrorWithMessage(err);

      if (maybeError) {
        setError(err.data.message);
      } else {
        setError('Unknown error');
      }
```

```
  }
 };

 return (
  <Layout>
   <Descriptions title='Employee information' bordered>
    <Descriptions.Item label='Name' span={3}>{`${data.firstName} ${data.lastName}`}</Descriptions.Item>
    <Descriptions.Item label='Age' span={3}>
     {data.age}
    </Descriptions.Item>
    <Descriptions.Item label='Address' span={3}>
     {data.address}
    </Descriptions.Item>
   </Descriptions>
   {user?.id === data.userId && (
    <>
     <Divider orientation='left'>Actions</Divider>
     <Space>
      <Link to={`/employee/edit/${data.id}`}>
       <CustomButton shape='round' type='default' icon={<EditOutlined />}>
        Edit
       </CustomButton>
      </Link>
      <CustomButton shape='round' danger onClick={showModal} icon={<DeleteOutlined />}>
       Delete
      </CustomButton>
     </Space>
    </>
   )}
   <ErrorMessage message={error} />
   <Modal
    title='Confirm deletion'
    open={isModalOpen}
    onOk={handleDeleteUser}
    onCancel={hideModal}
    okText='Confirm'
    cancelText='Cancel'
   >
    Are you sure you want to remove the employee from the table?
   </Modal>
  </Layout>
 );
};
```

Лістинг коду сторінки редагування сторінки

```
import { Employee } from '@prisma/client';
import { Row } from 'antd';
import { useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useEditEmployeeMutation, useGetEmployeeQuery } from '../../app/serivices/employees';
```

```
import { EmployeeForm } from '../../components/employee-form';
import { Layout } from '../../components/layout';
import { Paths } from '../../paths';
import { isErrorWithMessage } from '../../utils/is-error-with-message';


export const EditEmployee = () => {
  const navigate = useNavigate();
  const params = useParams<{ id: string }>();
  const [error, setError] = useState('');
  const { data, isLoading } = useGetEmployeeQuery(params.id || '');
  const [editEmployee] = useEditEmployeeMutation();

  if (isLoading) {
    return <span>Loading...</span>;
  }


  const handleEditUser = async (employee: Employee) => {
    try {
      const editedEmployee = {
        ...data,
        ...employee,
      };

      await editEmployee(editedEmployee).unwrap();

      navigate(`${Paths.status}/updated`);
    } catch (err) {
      const maybeError = isErrorWithMessage(err);

      if (maybeError) {
        setError(err.data.message);
      } else {
        setError('Unknown error');
      }
    }
  };


  return (
    <Layout>
      <Row align='middle' justify='center'>
        <EmployeeForm onFinish={handleEditUser} title='Edit employee' employee={data} btnText='Edit' error={error}
/>
      </Row>
    </Layout>
  );
};
```

Лістинг функції для авторизації користувача

```
const jwt = require('jsonwebtoken');
const { prisma } = require('../prisma/prisma-client');

const auth = async (req, res, next) => {
  try {
    let token = req.headers.authorization?.split(' ')[1];

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    const user = await prisma.user.findUnique({
      where: {
        id: decoded.id,
      },
    });

    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ message: 'Not authorized' });
  }
};
```

Лістинг коду підключення Prisma ORM до застосунку

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model User {
  id              String     @id @default(uuid())
  email           String
  password        String
  name            String
  createdEmployee  Employee[]
}

model Employee {
  id       String @id @default(uuid())
  firstName String
  lastName  String
  age       String
  address   String
  user      User   @relation(fields: [userId], references: [id])
  userId    String
}
```

Додаток Г

(Довідниковий)

Вихідні дані за результатами вхідних даних

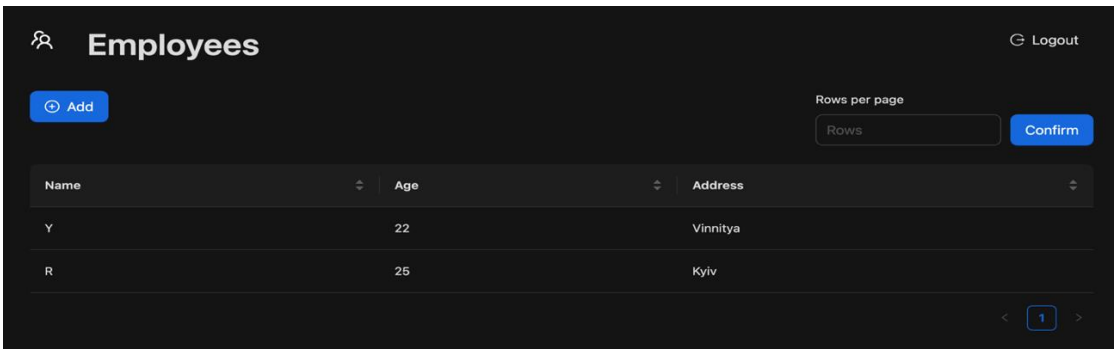

Рисунок Г.1 – Сторінка зі співробітниками

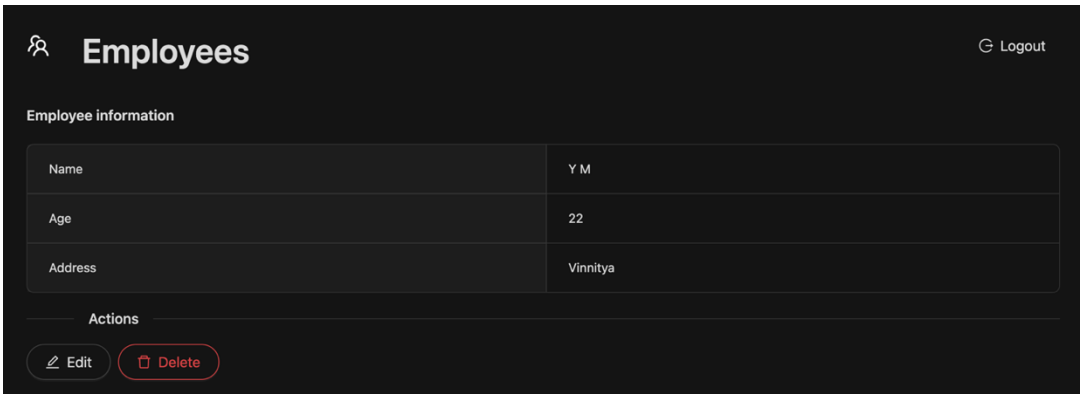

Рисунок Г.2 – Сторінка перегляду інформації про співробітника

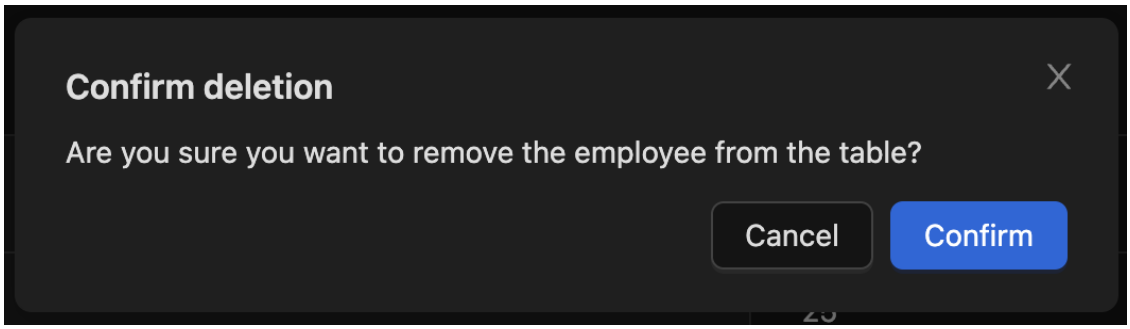

Рисунок Г.3 – Діалогове вікно про видалення співробітника

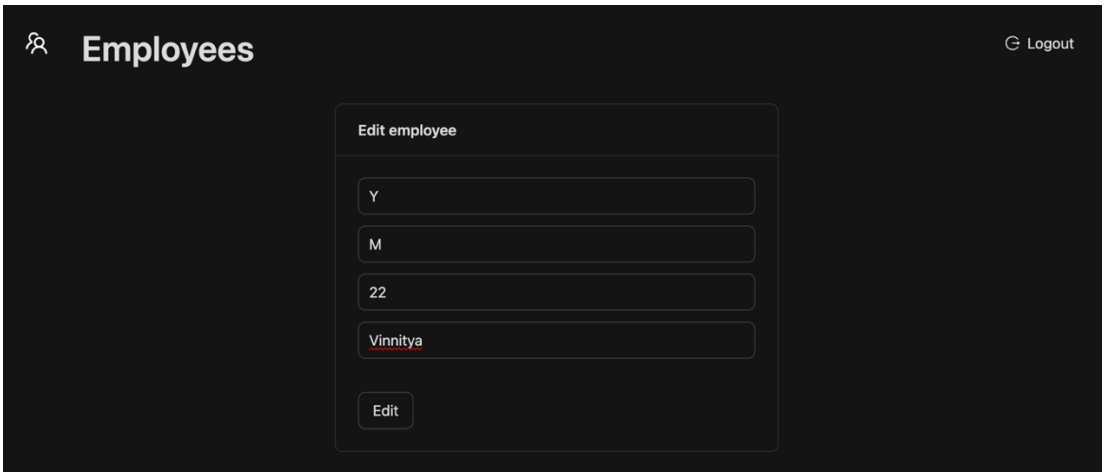

Рисунок Г.4 – Сторінка редагування інформації про співробітника

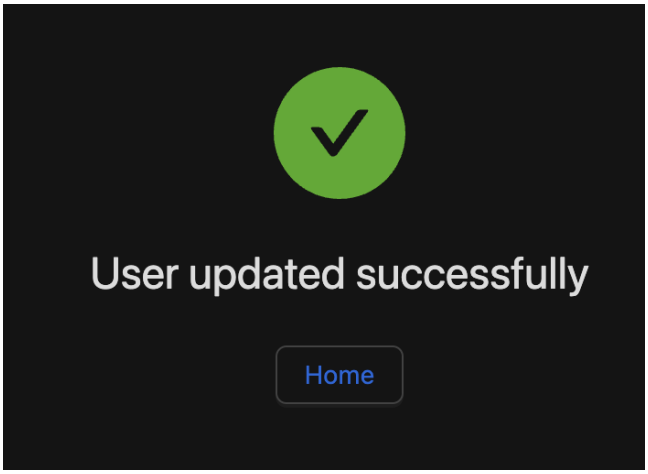

Рисунок Г.5 – Сторінка статусу про успішне редагування користувача

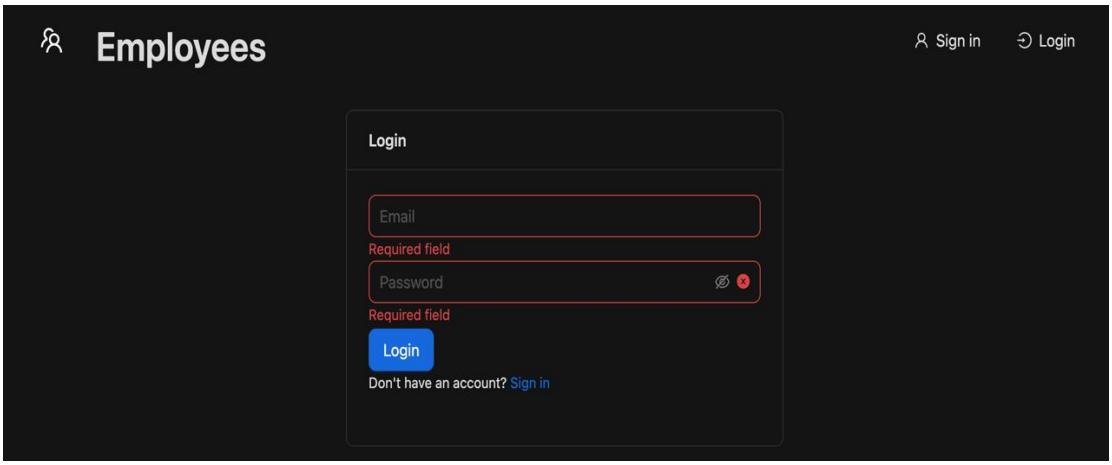

Рисунок Г.6 – Сторінка авторизації

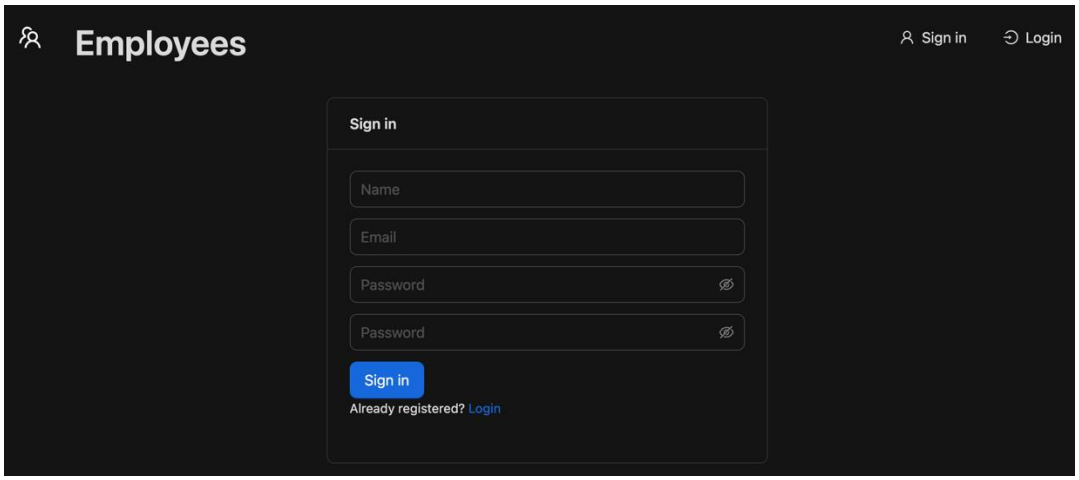

Рисунок Г.7 – Сторінка реєстрації

Додаток Д

(Довідниковий)

Діаграми популярності технологій

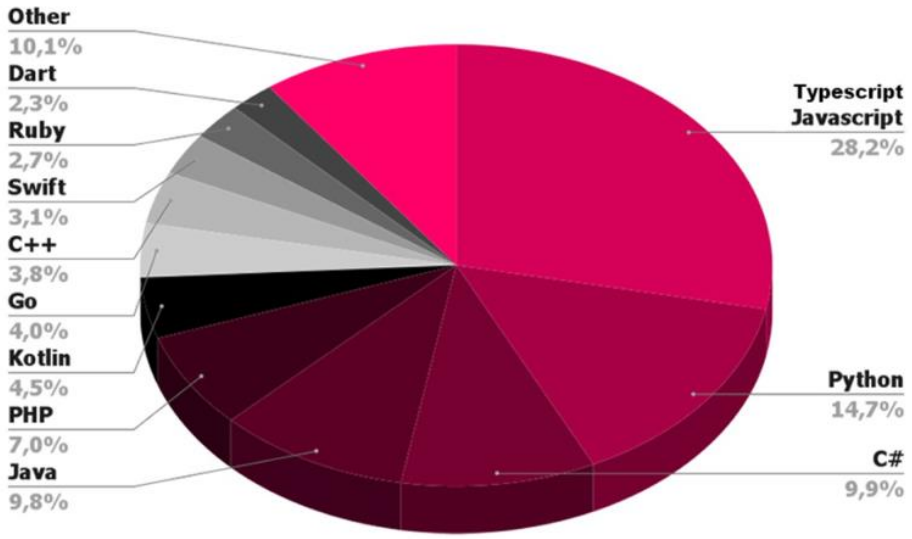

Рисунок Д.1 – Діаграма популярності мов програмування в 2023 році

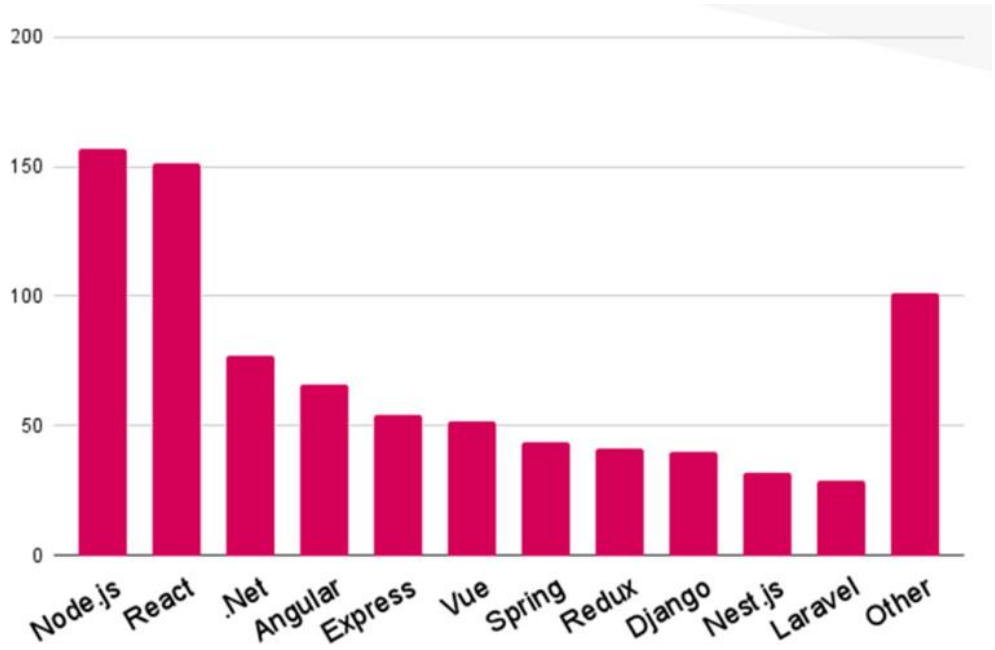

Рисунок Д.2 – Діаграма популярності фреймворків в 2023 році

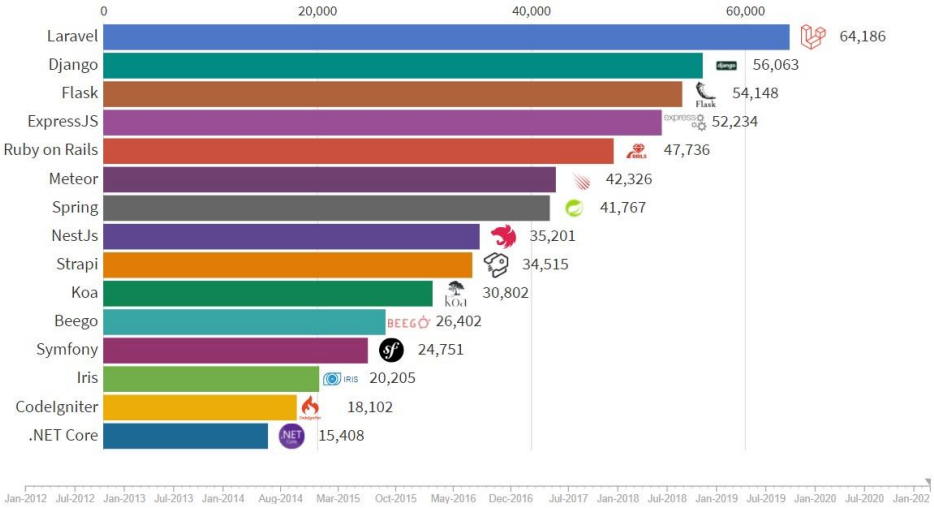

Рисунок Д.3 – Графік найпопулярніших бекенд-фреймворків

Додаток Е

(Довідниковий)

Набір даних для роботи

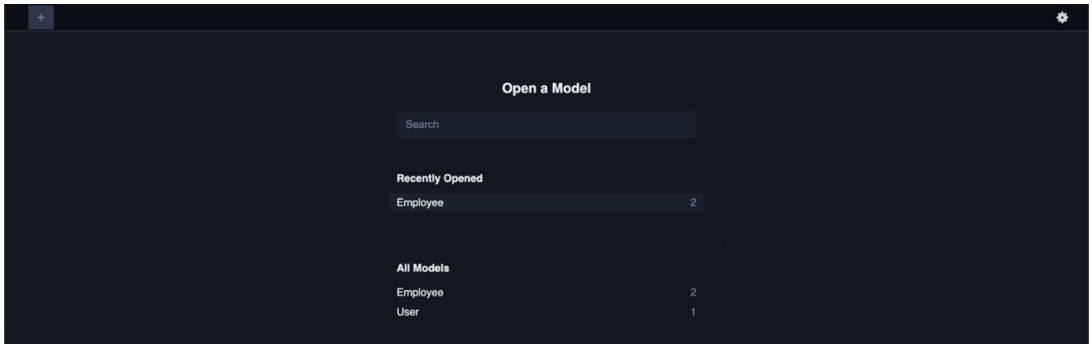

Рисунок Е.1 – Перегляд моделей у розширенні Prisma Client

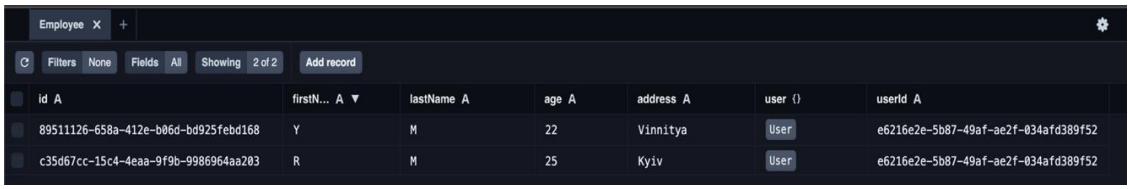

Рисунок Е.2 – Набір даних в таблиці Employee

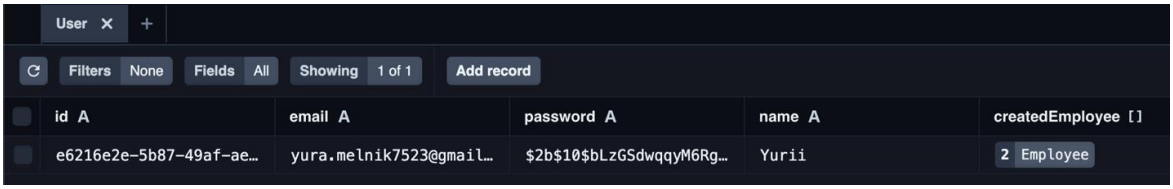

Рисунок Е.3 – Набір даних в таблиці User

Додаток Ж

(Обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Застосунок для автоматизованого керування персоналом з використанням ReactJS та NodeJS»

Тип роботи: магістреська кваліфікаційна робота
(БДР, МКР)

Підрозділ: кафедра АІІТ, ФІІТА, 1АКІТ-22м
(кафедра, факультет, навчальна група)

Науковий керівник Кулик Я.А, доц. каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності Unicheck

Оригінальність 99.7% Схожість 0.3%

Аналіз звіту подібності (відмітити потрібне)

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень

Особа, відповідальна за перевірку ___ Роман МАСЛІЙ
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи

Автор _____ Юрій МЕЛЬНИК
(підпис) (прізвище, ініціали)

Керівник роботи _____ Ярослав КУЛИК
(підпис) (прізвище, ініціали)