ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-ДОДАТКУ TASK MANAGER ДЛЯ ПОСТАНОВКИ ЗАДАЧ НА ПРОЕКТІ МОВОЮ TYPESCRIPT**»

Виконав: студент 5 курсу, групи ППЗ-52 спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Гусак С.Р.

(прізвище та ініціали)

Керівник Шевченко С.М.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність – 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
інженерії програмного забезпечення

Негоденко О.В.

“____” _____2021 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Гусак Сергій Романович
(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка web-додатку «Task manager» для потановки задач на проекті мовою TypeScript»
Керівник роботи Шевченко С.М. к.п.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «12» березня 2021 року №65.
2. Строк подання студентом роботи «01» червня 2021 року
3. Вихідні дані до роботи: Наукові джерела з питань актуальності SPA. Мова програмування – TypeScript та актуальності фреймворку Vue.js.

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

1. Аналіз предметної області та постановка задачі забезпечення розповсюдження інформації в соціальних мережах.
2. Теоретичні основи для розробки програмного засобу.
3. Розробка програмного забезпечення.
4. Огляд програмної реалізації та тестування програмного забезпечення

5. Перелік демонстраційного матеріалу (назва основних слайдів)

1. Титульний слайд
2. Мета, об’єкт та предмет дослідження
3. Аналоги
4. Вимоги додатку
5. Використані технології
6. Архітектура додатку
7. Entity Relationship Diagram
8. Висновки

6. Дата видачі завдання «19» квітня 2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури	15 лютого – 18 лютого	
2	Опис предметної області	20 лютого – 18 лютого	
3	Постановка задачі	29 лютого – 1 квітня	
4	Програмна реалізація	2 квітня – 4 квітня	
5	Тестування розробленого ПО	5 квітня – 6 квітня	
6	Вступ, висновки, реферат	7 квітня – 10 квітня	
7	Попередній захист роботи		
8	Здача роботи в деканат		

Студент _____ Гусак С.Р
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шевченко С.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалавської роботи 50 с., 21 рис., 5 табл., 14 джерел, додатки на 11 аркушах.

Об'єкт сфери дослідження – програмне забезпечення для постановки задач на проекті.

Предмет дослідження – web-додаток (SPA – Single Page Application), що дає можливість постановки задач робітникам компанії.

Мета дослідження – забезпечення чіткого контролю графіку робітників компанії. Чітке уявлення про стан робочого процесу та процес розробки продукту як для розробників, так і для інших членів команди. Розподілення графіку.

Методи дослідження – методи теорії інформації, методи теорії тестування, методи постановки задач, методи планування графіку.

У роботі проведено аналіз існуючих рішень програмного забезпечення, які допомагають в керуванні проектами та постановкою задач. Проведено вибір програмних засобів та технологій для реалізації програмної частини web-додатку Task Manager.

Даний додаток написаний на мові TypeScript з використанням фреймворку Vue.js, мови SCSS та мови розмітки HTML. Для реалізацій Back-end задач використано Node.js. В якості бази даних використовується MySQL.

Ключові слова: web-додаток, TypeScript, Vue.js, MySQL, Node.js, Visual Studio Code.

ЗМІСТ

ВСТУП

Об'єкт сфери дослідження – програмне забезпечення для постановки задач на проекті.

Предмет дослідження – web-додаток (SPA – Single Page Application), що дає можливість постановки задач робітникам компанії.

Мета дослідження – забезпечення чіткого контролю графіку робітників компанії. Чітке уявлення про стан робочого процесу та процес розробки продукту як для розробників, так і для інших членів команди. Розподілення графіку.

Методи дослідження – методи теорії інформації, методи теорії тестування, методи постановки задач, методи планування графіку.

Task Manager (менеджер завдань) - це корисний інструмент для управління проектами. Ви можете кількома кліками роздавати завдання всім своїм співробітникам і стежити за їх виконанням.

Яскравим прикладом застосування додатку може бути його використання в IT-компанії, де керівник розподілить завдання між працівниками, які, в свою чергу, будуть розуміти свою задачу. На цей процес витрачається менше часу ніж на багатогодинні розмови. Робочий процес спрощується і стає більш організованим, адже вся інформація розкладена по поличках.

На відміну від web-додатку Task Manager, його аналоги являють собою платне програмне забезпечення, і цей факт не дає вільно користуватись функціоналом такого додатку без певних грошових вкладень.

Одною з найголовніших недоліків аналогів, також є їх важкість і багатофункціональність. З точки зору користувача, це не завжди є плюсом, адже кожному потрібно витратити час на те, щоб розібратись як працює додаток. Вони включають в себе функціонал без якого можна обійтись.

Для українського користувача недоліком ще може бути те, що багато з аналогів не мають українського перекладу. Хоча в деяких з них є російська мова, яка є зрозумілою більшості з нас, але ж найприємніше використовувати інтерфейс, побудований українським словотвором.

1 ОГЛЯД ТА АНАЛІЗ

1.1 Актуальність

Task Manager – це програмне рішення керування проектом. Функціонал додатку дозволяє керівнику або іншим співробітникам роздавати групові задачі, розділяти їх на етапи, контролювати прогрес виконання. В сучасному світі такі сервіси користуються доволі великою популярністю, особливо в IT-компаніях.

Сервіси такого типу дозволяються здійснювати управління і контроль над проектами з найменшими тратами часу. Керівнику, або іншому робітнику не потрібно йти в сусідній кабінет для того, щоб дізнатися, чим зайнятий той чи інший співробітник.

Впровадження і використання Task Manager було корисним і актуальним раніше. Але в умовах карантину, масового переходу на формат віддаленої роботи, їх застосування стає життєво необхідним. Адже набагато зручніше ставити і коригувати завдання, контролювати їх виконання, керувати командою в єдиному інтерфейсі, ніж окремо спілкуватися з кожним учасником проекту.

1.2 Огляд існуючих додатків для постановки задач на проекті

В сучасному ринку існує велика кількість різноманітних програм які дозволяють керувати проектами, постановкою задач і обліком робочого часу. Вони можуть бути як програми для Windows, Mac, Android/iOS так і web-додатком.

Розглянемо найпопулярніші з них:

1. Asana – система дрібних інструментів, які допомагають керувати задачами, взаємовідносини з клієнтами, проектами і багато інших функцій. Існує тільки англійська версія. Цікавою особливістю є те, що майже всім функціоналом можна керувати, використовуючи тільки клавіатуру. Головний інтерфейс показано на рис 1.1.

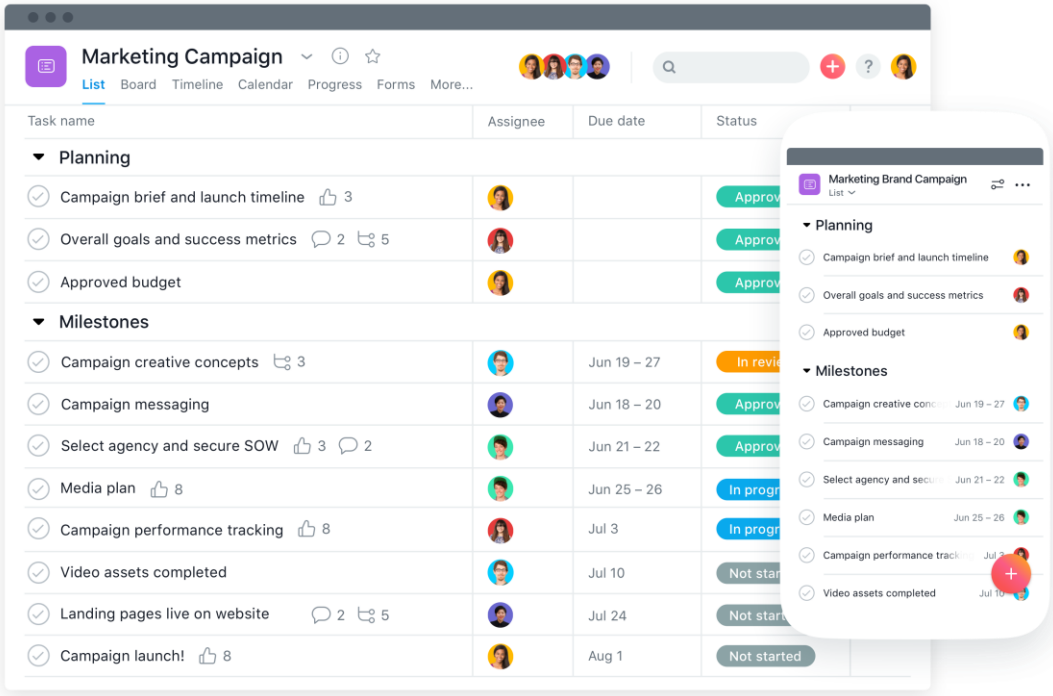


Рисунок 1.1 – Інтерфейс Asana

2. Basecamp 3 – об’єднує в собі задачі, групові чати, календар, сповіщення про події і роботу с файлами. Універсальна програма, яка може замінити декілька додатків для роботи з задачами, повністю організувати та автоматизувати роботу. На рис 1.2 показано приклад головної сторінки проекту.

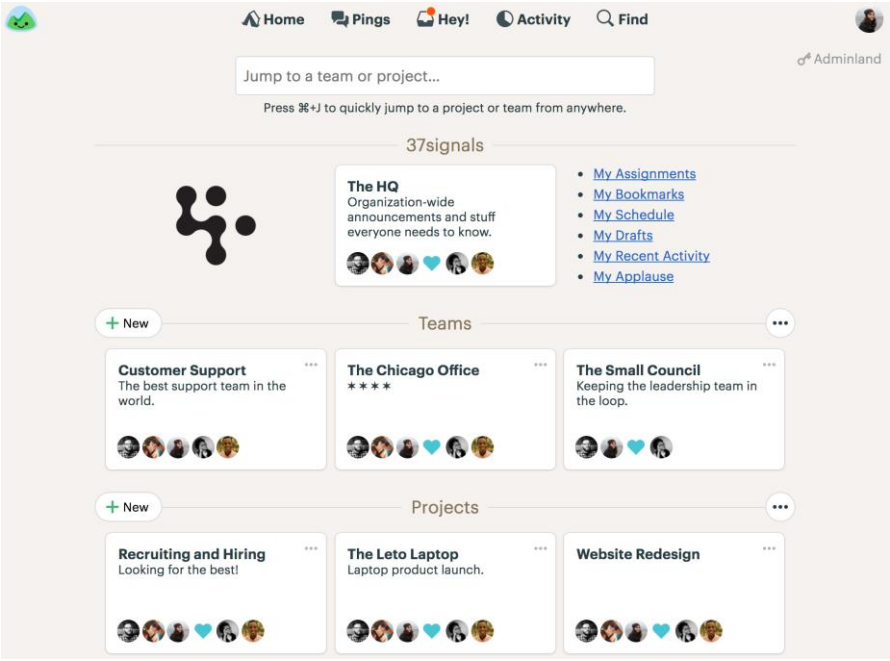


Рисунок 1.2 – Приклад головної сторінки Basecamp 3

3. Jira – найкраще всього підходить для компаній, що займаються розробкою програмного забезпечення. В цьому додатку багато дрібних, вузько направлених задач, які складним чином пов’язані одна з одною. Інтерфейс головної сторінки з інтегрованим і налаштованим проектом показано на рис 1.3.

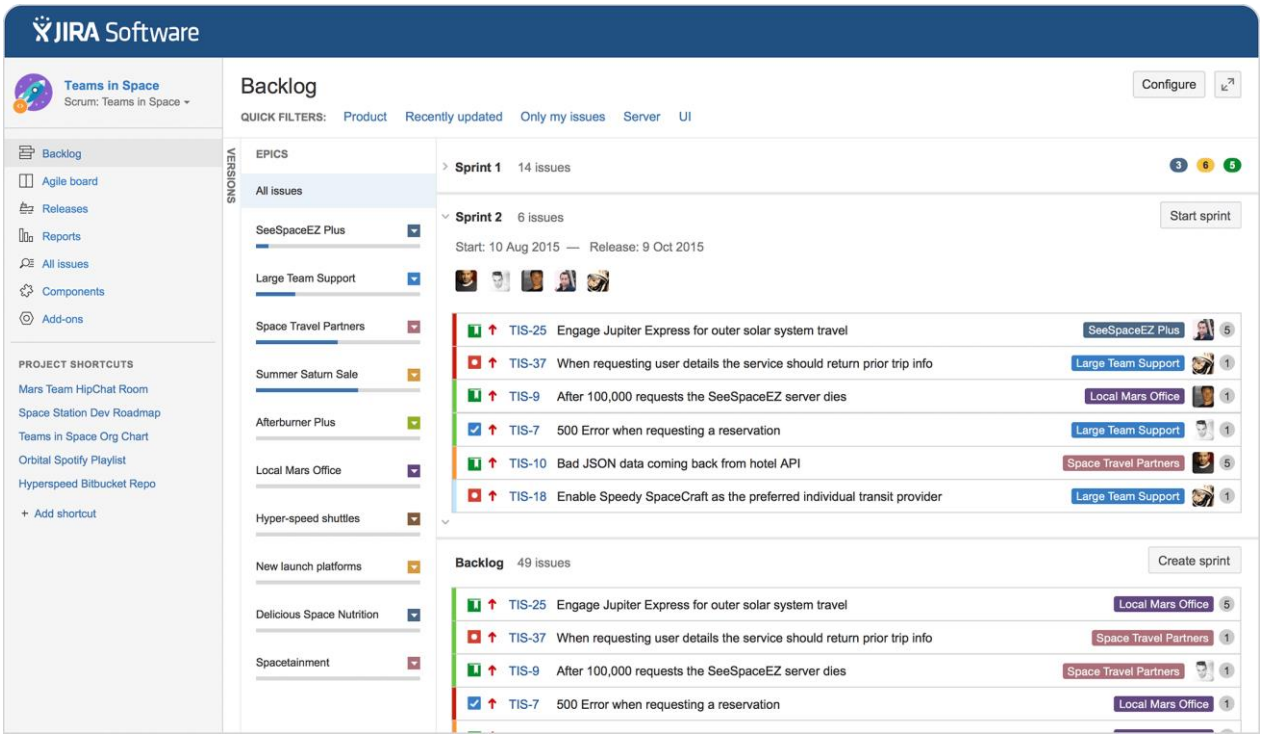


Рисунок 1.3 – Головна сторінка Jira з налаштованим проектом

Багато існуючих додатків для керування проектами локалізовані лише англійською мовою, що робить їх незручним для неангломовного користувача.

Всі вищеописані аналоги складні в освоєнні і потребують багато часу для повного розуміння функціоналу. Це основний недолік таких програм.

1.3 Аналіз програмних засобів

Для реалізації програмної частини розробки для мови TypeScript потрібно використовувати редактор коду. Існує багато різноманітних варіантів програмного забезпечення цього типу, але було обрано один з найпопулярніших і найбільш функціональних – Visual Studio Code (VS Code) рис 1.4.

VS Code – легкий, але потужний редактор вихідного коду, який являється desktop-додатком. Він доступний для Windows, macOS та Linux.

VS Code постачається з вбудованою підтримокою JavaScript, TypeScript та Node.js. Також він має багату екосистему розширень для інших мов, таких як C++, C#, Java, Python, PHP, GO та серидовищ використання, таких як .NET та Unity.

Перевагами цього редактору є:

- безкоштовність;
- потужність;
- простий інтерфейс;
- можливість налаштування сніпетів та установки плагінів для пришвидшення розробки;
- зв’язок проекту з GitHub;
- вбудована консоль з можливістю інтеграції Git Bash.

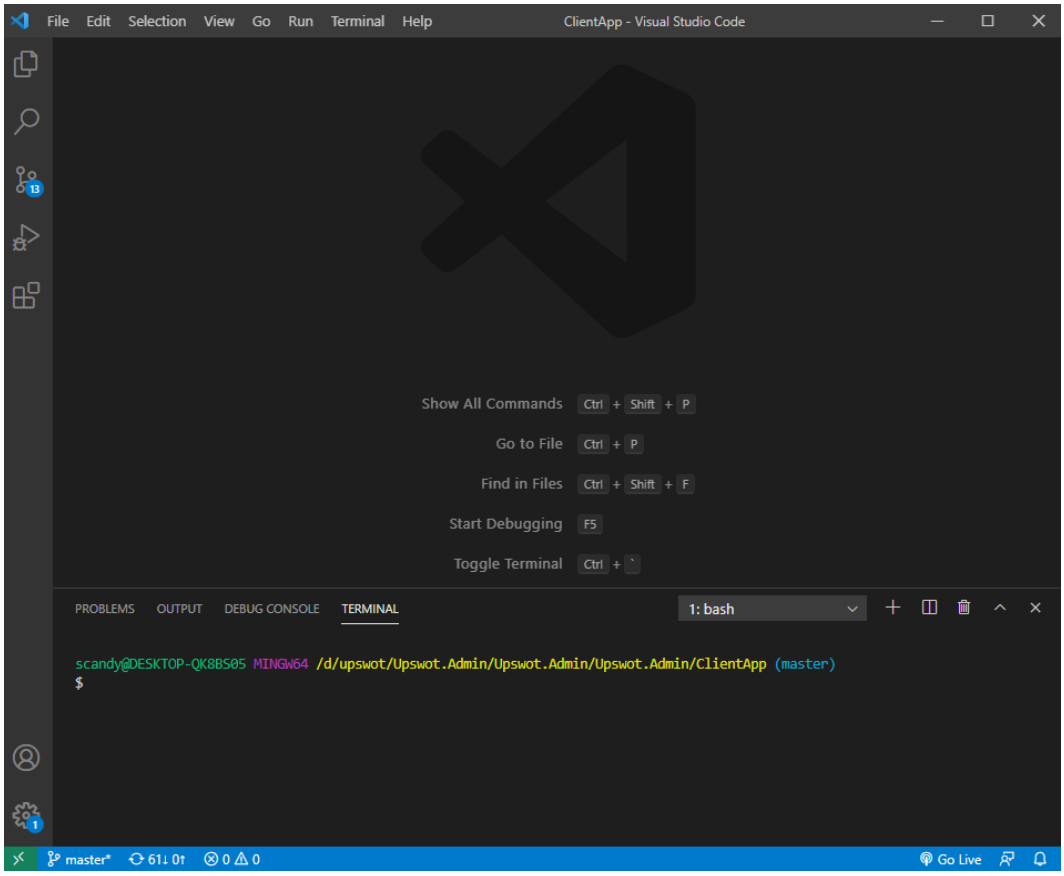


Рисунок 1.4 – Інтерфейс Visual Studio Code

Цікавим є те, що навіть Facebook обрав VS Code в якості основного редактору, що ще раз підкреслює якість обраного програмного засобу.

Для розробки додатку на TypeScript + Vue.js були використані наступні розширення:

- TSLint Vue;
- Vetur;
- vue-template-beauty.

Кожне розширення допомагає тримати свій код в «чистоті» з правильним форматуванням. Стилістику коду можна налаштовувати під свої потреби.

Вибір редактору коду являється дуже важливим етапом на початковому етапі розробки програмного продукту. Цей вибір впливає на швидкість розробки, через ряд факторів:

- наскільки простим є інтерфейс;
- доступ до файлів проекту;
- інтуїтивна зрозумілість;
- можливість роботи з розширеннями, що пришвидшують роботу.

При роботі з Visual Studio Code протягом декількох років, та паралельному використанні аналогів таких як Sublime Text 3, Nodepad++, WebStorm та інших, склалась думка про те, що саме VS Code – той редактор, який являється найзручнішим для використання. Швидкість розробки була найвищою, в порівнянні з аналогами.

Як висновок, можна сказати, що через ряд переваг та досвід використання аналогів, VS Code - один з найзручніших та потужніших редакторів коду, саме тому було обрано саме його.

2 ВИМОГИ ДОДАТКУ

2.1 Опис проекту

Мета цього кроку полягає в тому, щоб кожен крок в процесі роботи з додатком був чітким та повним. При розробці продукту потрібно відповідати поставленим вимогам. Вимоги є двух типів: нефункціональні та функціональні вимоги.

Нефункціональні вимоги – чіткі критерії того, як додаток повинен працювати, що він потрібен робити.

Функціональні вимоги – описують повний функціонал додатку, яку розробники повинні побудувати, щоб користувач міг виконати свої задачі в рамках бізнес-потреб.

2.1.1 Вимоги зовнішнього інтерфейсу

Hardware:

- крос-платформність;
- доступ до інтернету (в пристрою, з якого буде виконуватись робота з додатком).

Software:

- крос-браузерність і коректність роботи на наступних версіях браузерів, або новіших: Internet Explorer 11, Chrome 29, Firefox 20, Edge 12;
- адаптивність інтерфейсу.

2.1.2 Функціональні вимоги

Додаток включає в себе наступний функціонал:

1. Реєстрація – користувач повинен мати можливість зареєструватись в додатку для подальшої роботи з налаштуванням проекту. Поля для реєстрації повинні бути наступні:

- імя;

- прізвище;
- посада;
- email;
- пароль;
- підтвердження паролю.

2. Авторизація – використання функціоналу додатку повинно бути можливим тільки у авторизованого користувача. У вікні авторизації повинні бути наступні поля:

- email;
- пароль.

3. Відновлення паролю – користувач повинен мати можливість відновити пароль, у випадку, якщо він його забув.

4. Реєстрація компанії, до якої можна буде прив’язати проекти та користувачів.

5. Створення проекту – користувач повинен мати можливість створити новий проект з його назвою і описом. У вікні створення проекту повинні бути наступні поля:

- назва;
- опис.

6. Прив’язка користувача до створеного проекту – користувач повинен мати можливість додати іншого користувача в створений проект за допомогою Email;

7. Створення задачі – користувач повинен мати можливість створити нову задачу і прив’язати її до проекту. При створенні задачі у вікні повинні бути наступні поля:

- назва;
- опис;
- запланований час;
- кінцевий срок;

- тип задачі (баг, новий функціонал);
- пріоритет;
- відповідальний;
- проект (вибір проекту зі створених).

8. Перегляд проектів – користувач повинен мати можливість переглядати всі проекти до яких він прив’язаний.

9. Перегляд задач – користувач повинен мати можливість переглядати всі задачі, які прив’язані до проекту.

10. Фільтрація задач – користувач повинен мати можливість встановити фільтр по відповідальному до задачі користувачу.

11. Редагування створеної задачі – користувач повинен мати можливість змінювати назву та опис створеної задачі, відповідального, а також можливість встановити витрачений час на задачу.

12. Пошук задачі за назві – користувач повинен мати можливість пошуку задачі за назвою.

13. Встановлення статус задачі – користувач повинен мати можливість змінювати статус задачі на один з наступних:

- до виконання;
- в роботі;
- в тестуванні;
- виконано.

14. Видалення задачі – користувач повинен мати можливість видалити створену задачу.

15. Редагування профілю – користувач повинен мати можливість змінювати свій профіль.

2.1.3 Нефункціональні вимоги

Нефункціональні вимоги додатку включають в себе наступні пункти:

1. Вимоги кінцевого продукту:

- usability – додаток повинен бути простим, інтуїтивно зрозумілим, займати мінімально часу, для того, щоб розібратись, тобто мати user-friendly інтерфейс;
- продуктивність – додаток повинен бути оптимізованим і, як наслідок, швидким в завантаженні даних;
- вільний простір – додаток повинен мати вільний простір для зберігання інформації, для того, щоб система працювала стабільно;
- надійність – додаток не повинен показувати невірну інформацію користувачу.

2. Зовнішні вимоги:

- безпека – додаток має бути захищений від зовнішнього втручання та атак;
- конфіденційність – додаток має зберігати конфіденційність даних введених користувачем;
- взаємодія – додаток має забезпечувати доступ до компонентів програми без зміни ефективності та послідовності.

2.2 Використані технології

Успіх реалізації програми, а також подальша підтримка, у великому значені залежить від обраних технологій. Технології повинні мати можливість реалізації вищеописаних вимог.

Швидкість розробки також залежить від правильно обраних засобів реалізації. Наприклад, розробка функціоналу за допомогою одної технології може бути повільніше ніж іншою, тому цей крок є важливим в процесі розробки програмного продукту.

Щодо подальшої підтримки, потрібно розуміти, що написаний код програми буде змінюватись і для подальшого розуміння, потрібно, щоб все написане раніше було інтуїтивно зрозуміло. Саме для цього моменту, для

розробки Task Manager використовується саме мова програмування TypeScript, а не більш всім відомий JavaScript.

2.2.1 Мова TypeScript. Порівняння з JavaScript

TypeScript – мова програмування, яка створена людиною, що створила C#. Це результат того, що можна зробити, щоб допомогти розробникам при написанні коду на JavaScript. Можна взяти написаний на JavaScript код, додати декілька ключових слів і перетворити код в сильно типізовану об’єктно-орієнтовну кодову базу, з перевіркою синтаксису. Додавши крок компіляції, можна перевірити, що код написано надійно, який буде вести себе так, як і задумувалось.

Основною проблемою JavaScript для розробки великих проектів є його динамічність і не типізованість.

Щоб отримати представлення про перевагу TypeScript, розглянемо деякі моменти які TypeScript дає в таблиці:

- компіляція;
- сильна типізація;
- інтеграція з популярними бібліотеками;
- інкапсуляція;
- методи доступу public та private.

Відсутність компіляції в JavaScript можна розглядати як перевагу, але це не завжди так. Компілятор може знайти помилки, від маленьких (пропущена душка, або кома) до більш не невиразних, наприклад як використання одинарної лапки, в місці, де повинна використовуватись подвійна. Особливо компіляція потрібна у великих проектах, адже наскільки професійним не був би розробник, людині властиво робити помилки.

Використання кроку компіляції в робочому процесі спрощує роботу при управлінні великою базою коду. Існує стара приказка, в якій говориться, що ми повинні рано виходити з ладу і голосно виходити з ладу, а компілятор буде дуже голосно кричати на самій ранній можливій стадії, коли будуть виявлені помилки.

Це означає, що будь-яка фіксація змін у вихідному коді буде вільна від помилок виявлених компілятором.

Типізація є основною і найголовнішою перевагою. Наприклад розглянемо код на рис 2.1.

```
var example = 'this is a string';
console.log('example=' + example);

example = 1;
console.log('example=' + example);

test = function (a, b) {   return a + b; }
console.log('example=' + example);
```

Рисунок 2.1 – Приклад коду JavaScript

В першій стрічці ми створюємо змінну `test` і присвоюємо їй строкове значення. Щоб переконатись в тому, що ми записали – ми виводимо значення в консоль. Потім присвоюємо числове значення і знову виводимо в консоль. В третьому випадку ми присвоюємо змінній `test` функцію. Запускаючи код, ми побачимо наступне (рис 2.2).

```
example = this is string
example = 1
example = function (a, b) {
    return a + b;
}
```

Рисунок 2.2 – Результат запущеного коду з рис 2.1

Тут явно видно зміну значень, які внесено в змінну. Вона змінюється переходячи зі строкового значення в числове, а потім стає фунцією.

Зміна типу змінної є дуже небезпечним явищем і може привести до незчисленних проблем. Ось чому традиційні об’єктно-орієнтовні мови забезпечуються сильну типізацію. Другими словами, вони не дозволяють природі змінної змінюватись після оголошення. TypeScript вирішує цю проблему.

Синтаксис для перевірки типу дуже простий, див рис 2.3.

```
var example: string = 'this is a string';
example = 1;
example = function(a, b) { return a + b; }
```

Рисунок 2.3 – Приклад типізації TypeScript

Якщо запустити даний код – компілятор визве дві помилки, див рис 2.4

```
hello.ts(3,1): error TS2322: Type 'number' is not assignable to type 'string'.
hello.ts(4,1): error TS2322: Type '(a: any, b: any) => any' is not assignable to type
 'string'.
```

Рисунок 2.4 - Результат запущеного коду з рис 2.3

Було вказано, що змінна `example` - рядок. Ці дві помилки вказують на те, що в процесі компіляції відбулась зміна типу змінної. Таким чином вирішується проблема з типізацією і такий запис називається синтаксичним цукром, або більш формально анотація типу.

Сильно типізувати існуючі бібліотеки можна шляхом створення файлу визначення. TypeScript використовує файли з розширенням .d.ts як свого роду заголовки, за аналогією з такими мовами, як C++. Ці файли визначень містять інформацію, яка описує кожну доступну функцію і/або змінні, а також зв’язані з ними анотації типів.

Виходячи з вищеописаного можна зробити висновок, що використання TypeScript у великих проектах більш доцільно ніж JavaScript.

2.2.2 Vue.js

Для початку потрібно зрозуміти навіщо взагалі потрібні js-фреймворки.

Кодування цілком можна використовувати і без їх використання, але правильно підібрана середа може значно полегшити роботу. Більш того, вони безкоштовні та з відкритим кодом.

В першу чергу, використання фреймворку підвищить продуктивність. Можна розглядати це як обхідний шлях: менше коду прийдеться писати вручну, так як вже є готові до використання функції та шаблони.

Існує велика кількість таких фреймворків, але найпопулярніші та найбільш документовані – це: React, Angular, Vue.js.

Нижче проведемо більш детальний аналіз одної з цих технологій і буде зрозуміло, чому саме Vue.js було обрано в ролі основного фреймворку.

Vue.js – це прогресивний фреймворк для створення користувальницьких інтерфейсів. На відміну від інших монолітних каркасів, Vue розроблений з нуля, щоб бути поступово адаптованою. Цей фреймворк дозволяє впроваджувати інтерактивну поведінку та додаткові можливості в будь-який контекст, в якому виконується JavaScript, або, як в нашому випадку TypeScript.

Vue можна використовувати як на окремих сторінках, вирішуючи прості задачі, так і в якості фундаменту для повноцінних промислових додатків. Він і його додаткові бібліотеки дозволять створювати повноцінні важкі web-додатки.

Одною з примітних можливостей Vue – це ненав'язлива реактивність (див. рис. 2.5). Моделі являють собою прості об'єкти. У міру їх зміни оновлюється і уявлення дених, завдяки управління станом додатку стає простим і очевидним.

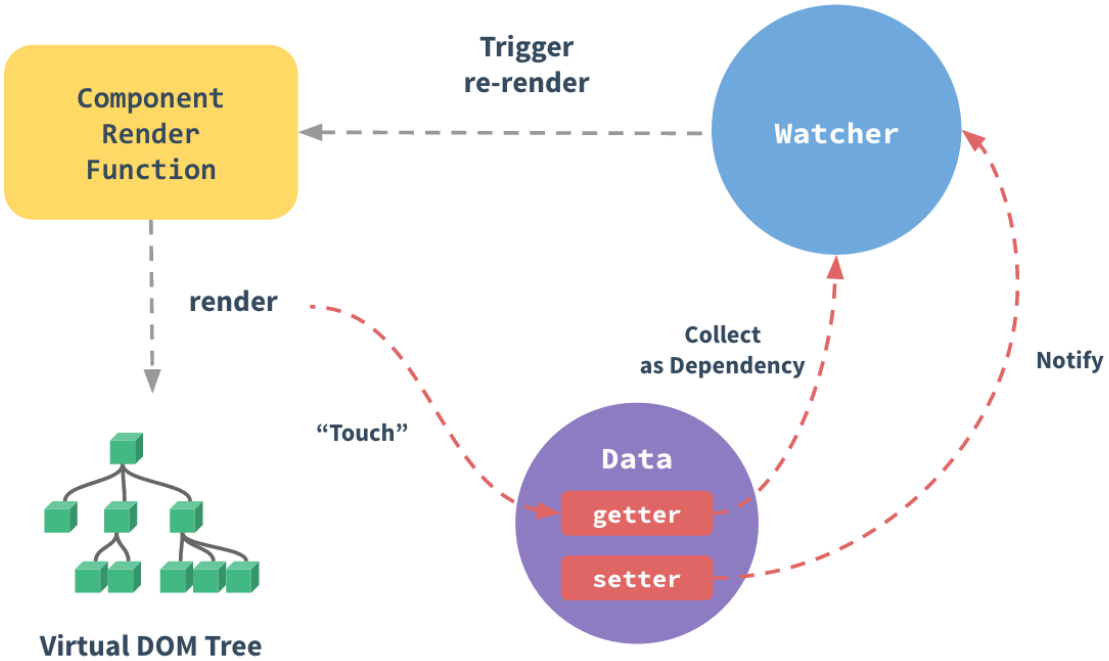


Рисунок 2.5 – Реактивність у Vue.js

Основна бібліотека орієнтована тільки на переглядовому шарі, її легко підібрати та інтегрувати з іншими бібліотеками. З іншого боку, Vue також прекрасно підходить для складних односторінкових додатків, коли вона використовується в поєднанні з сучасними інструментами та бібліотеками.

Vue.js було обрано через ряд його переваг:

- посилений HTML. Vue.js має багато схожих з Angular, іншим відомим фреймворком, характеристик. Це допомагає оптимізувати обробку HTML-блоків з використанням різних компонентів;
- детальна документація. Vue.js має дуже хорошу документацію, яка може пришвидшити швидкість навчання розробників і економію часу на розробку додатку з використанням базових знань HTML та JavaScript;
- чудова інтеграція. Vue.js можна використовувати як для створення односторінкових додатків, так і для більш складних web-додатків. Найважливіше, що невеликі інтерактивні частини можна легко інтегрувати в існуючу інфраструктуру, не надаючи при цьому негативного впливу на всю систему;
- велике масштабування. Vue.js допомагає розроблювати досить великі шаблони для багатократного використання, які можна розроблювати без трати великої кількості часу, через простоту структури.
- порівняно маленький розмір. Vue.js може займати приблизно 20 КБ і при цьому зберігати свою швидкість і гнучкість, що дозволяє досягти більш високої продуктивності, в порівняні з іншими фреймфорками.

2.2.3 Node.js

Вибір технології для backend задач є не менш важливою для web-додатку, ніж вибір інших технологій. Ядро платформи Node.js стрімко розвивається, а темп росту колекції користуватських модулів вражають увагу.

Платформа з’явилась в 2009 році і дуже стрімко розвивалась і також розвивається по сей день.

Node.js – асинхронна керована подіями виконавча платформа JavaScript з потужною, але компактною стандартною бібліотекою.

Її супроводом та підтримкою займається Node.js Foundation – галузевий консорціум з відкритою моделлю управління. Існує дві версії Node:

- поточна;
- LTS (Long Term Support), яка користується довготривалою підтримкою.

Я зупинився на Node.js через його переваги, які можна перечислювати доволі довго. Основною перевагою є – неблокуюча модель введення-виведення. Ця система керуюча подіями та працює асинхронно, будуючи чергу по пріоритетності.

Одною з найосновніших переваг також являється однопоточна модель програмування. Програмні потоки являються стандартним джерелом помилок, і хоч деякі з тих мов програмування, які з’явились недавно, включаючи Go та Rust, намагаються надати безпечні інструмети паралельного програмування, Node працює з моделлю, що використовується в браузері.

Браузерний код представляє собою послідовність команд, які виконуються одна за одною, код не виконується паралельно. Для призначених для користувача інтерфейсів така модель не має сенсу, оскільки користувач не хоче чекати завершення повільних операцій. Для вирішення цієї проблеми в браузерах використовуються події: коли користувач клацає по кнопці, то ініціюється подія і виконується функція, яка була визначена раніше, але ще не виконалась. Тим самим вирішуються деякі проблеми, які зустрічаються в багатопоточному програмуванні.

Коли до серверу одночасно підключається велика кількість людей, їй простіше впоратись з навантаженням, так як немає необхідності створювати окремий потік для кожного підключення.

Платформа Node.js побудована на базі JavaScript движка V8 від Google, який використовується у браузері Google Chrome. Дана платформа, в основному використовується для створення веб-серверів, як в нашому випадку, але сфера застосування на цьому не закінчується.

Розглянемо ряд інших переваг:

- одна мова програмування як на клієнті так і на сервері. Розробнику, який вивчив JavaScript, буде легше вивчити «надбудову» ніж іншу технологію;
- швидкість. Швидкість робочого прототипу, що справляється з навантаженням не віднімає багато часу. На першому етапі, коли формується кістяк майбутнього продукту, витрачається мінімум зусиль;
- синтаксис JavaScript. З вище описаного зрозуміло, що TypeScript мало того, що схожий з JavaScript, після компіляції він в нього перетворюється. Ця перевага є одною з основних, тому що не потрібно тратити час на вивчення іншої мови програмування, адже набагато швидше зрозуміти синтаксис вже знайомого JavaScript;
- технологія стрімко поліпшується. Над цим трудяться тисячі програмістів по всьому світу. Важко недооцінювати роль ком’юніті, адже це допомога зі складностями, дуже багато навчальних матеріалів, оперативно знайдені помилки і тд. І це все завдяки відкритому коду.

2.2.4 MySQL

MySQL – найбільш приспособлена в середі web СУБД (системою управління базами даних).

Базою даних називають структурований набір даних. Це може бути що завгодно: від простого переліку покупок до галереї зображень. Щоб додавати, звертатися і обробляти дані, збережені в комп'ютерній базі даних, Ви потребуєте в системі управління бази даних, типу MySQL. Так як комп'ютери дуже хороші при обробці великої кількості даних, бази даних грають центральну роль в обчисленнях, як автономні утиліти, або як частини інших пакетів прикладних програм.

MySQL виникла як спроба застосувати mSQL до власних розробок компанії: таблиць, для яких використовувалися ISAM - підпрограми низького рівня. В результаті був вироблений новий SQL-інтерфейс, але API-інтерфейс

залишився у спадок від mSQL. Звідки походить назва «MySQL» - достеменно не відомо. Розробники дають два варіанти: або тому, що практично всі напрацювання компанії починалися з префіксу My, або на честь дівчинки на ім'я My, дочки Майкла Монті Віденіуса, одного з розробників системи.

MySQL являє собою реляційну СУБД, яка зберігає дані в окремих таблицях. Це додає швидкодію та гнучкість. Таблиці пов’язані певними відносинами, що роблять можливим об’єднати дані з декількох таблиць в одному запиті.

SQL-частина MySQL орієнтована на Structured Query Language, найбільш загальна стандартизована мова, яка використовується, щоб звернутись до комп’ютерних баз даних.

Також MySQL являється Open Source Software, а це означає, що текст програми відкрити для читання і редагування всім бажаючим. Будь-який розробник може завантажити MySQL і використовувати його абсолютно безкоштовно, що є великим плюсом для комерційних проектів. Також, будь-який бажаючий може вивчати вихідний текст і змінювати його на свій розсуд.

MySQL дуже швидкий, надійний і легкий у використанні. Він має практичний набір властивостей, розроблених у близькому співробітництві з користувачами. MySQL був спочатку розроблений, щоб обробити дуже великі бази даних набагато швидше, ніж існуючі рішення і успішно використовувався у високо вимогливих промислових середовищах.

При постійній розробці MySQL сьогодні пропонує багатий і дуже корисний набір функцій. Можливості падключення, швидкодія і захист роблять MySQL підходящим для звернення до баз даних з Internet.

Відомо, що для виконання додатків клієнту на більшості хостинг-площадок провайдери надають невелику кількість ресурсів. Тому потрібна високоефективна СУБД, маючи при цьому високу надійність, адже більшість web-додатків повинні працювати 24/7.

Як висновок, відзначено ряд плюсів MySQL:

- простота. MySQL легко встановлюється. Існує багато сторонніх інструментів, включаючи візуальні, які полегшують початок роботи з базою даних;
- функціональність. MySQL підтримую більшу частину функціоналу SQL;
- швидкість. Через нехтування деякими стандартами, MySQL може працювати швидше.

Навіть, якщо брати в увагу деякі обмеження даної СУБД, то це не буде перешкодою для розробки web-додатку Task Manager. Весь необхідний для розробки функціонал присутній. Беручи до уваги також той факт, що MySQL одна з найпростіших в розумінні СУБД, то вибір пав саме на неї.

2.2.5 Інші технології

Для зручності і більшої швидкості розробки використано ще ряд технологій, які спрощують розробку. До них входить:

Vue CLI - повноцінна система для швидкої розробки на Vue.js, яка надає:

- інтерактивне створення проекту;
- велика колекція офіційних плагінів, інтегруючих найкращі інструменти екосистеми;
- швидке прототипування без конфігурацій;
- повноцінний графічний інтерфейс для створення та управління проектами.

Vue CLI прагне стати стандартним інструментарієм для екосистеми Vue, що забезпечує безперебійну роботу різних інструментів змірки, встановлює розумні значення за замовчуванням. Це дає розробнику можливість повністю зосередитись на розробці програми, а не проводити дні за її налаштуванням. В той же час залишається гнучкість налаштування конфігурації кожного інструменту без необхідності вилучення конфігурації в окремий файл.

BootstrapVue – інтерфейсна бібліотека, яка допомагає створювати чуйні, доступні та мобільні проекти за допомогою Vue.js. Bootstrap – це відкритий та безкоштовний HTML, CSS та JS фреймворк, який використовується веб-

розробниками для швидкої верстки адаптивних сайтів та web-додатків. По суті це всім відома бібліотека Bootstrap 4, але розрахована на підключення до проекту, який використовує Vue.js.

VueRouter – офіційна бібліотека маршрутизації для Vue.js. Оскільки web-додаток Task Manager – це SPA (Single Page Application), то ця бібліотека просто життєво-необхідна. Вона допомагає інтегруватись з Vue.js і дозволяє легко створювати SPA-додатки. Включає наступні можливості:

- модульна конфігурація маршрутизатору;
- вкладені маршрути/представлення;
- доступ до параметрів маршруту;
- анімація переходів на основі Vue.js;
- зручний контроль навігації;
- автоматичне проставлення активного CSS класу для посилань;
- режими роботи HTML5 history або хеш;
- налаштування поведінки прокрутки сторінки.

Node.js Express – фреймворк, який знадобився для створення API. Це гнучкий та мінімалістичний веб-фреймворк, що надає обширний набір функцій для web-додатків та мобільних додатків. На даний момент, це найпопулярніша, на даний момент структура.

Хоча сам Node.js Express є досить мінімалістичним, розробники створили сумісні проміжні пакети для вирішення практивно будь-якої проблеми web-розробки. Існують бібліотеки для роботи з файлами, cookie, сеансами, парметрами URL та багато інших.

Sequelize – ORM (абревіатура від Object Relational Mapping – Об'єктно-реляційна проекція), див. рис. 2.6, бібліотека, для додатку на Node.js, яка здійснює зіставлення таблиць в базі даних і відносини між ними з класами. При використанні цієї бібліотеки можна не писати SQL-запити, а працювати з даними як зі звичайними об’єктами.

Рисунок 2.6 – Технологія ORM

Axios – це JavaScript бібліотека, яка допоможе у виконанні HTTP-запитів. Основними перевагами бібліотеки являється пітримка «промісів» та автоматичне перетворення JSON-даних.

SASS – модуль для CSS. Метамова на основі CSS. В додатку допоможе збільшити рівень абстракції та забезпечить спрощення файлів каскадних таблиць стилів. SASS – це препроцесор, який дозволяє використовувати функції, недоступні в CSS, наприклад змінні, вкладеності, міксини, успадкування та інші приємні речу, які повертають зручність написання CSS. При написанні стилів, використовуючи SASS, препроцесор обробляє файл і зберігає його як простий CSS-файл, зрозумілий браузеру.

GitBash – це командний рядок, за допомогою якого можна використовувати функції Git. Він емулює середовище bash в Windows і дозволяє використовувати більшість стандартних команд Unix. Цей інструмент допоможе при роботі з модулями додатку.

Webpack – статичний модульний збирач для додатків на JavaScript (в нашому випадку для TypeScript). У випадку з розробкою додатку Task Manager, він буде використовуватись у Vue CLI для компіляції файлів в мінімізовані, для деплою.

Vuex (див. рис. 2.7) – патерн управління станом та бібліотека для додатків на Vue.js. В додатку Task Manager він служить централізованим сховищем для всіх даних компонентів програми з правилами, що гарантують, що стан може бути змінено тільки передбачуваним чином.

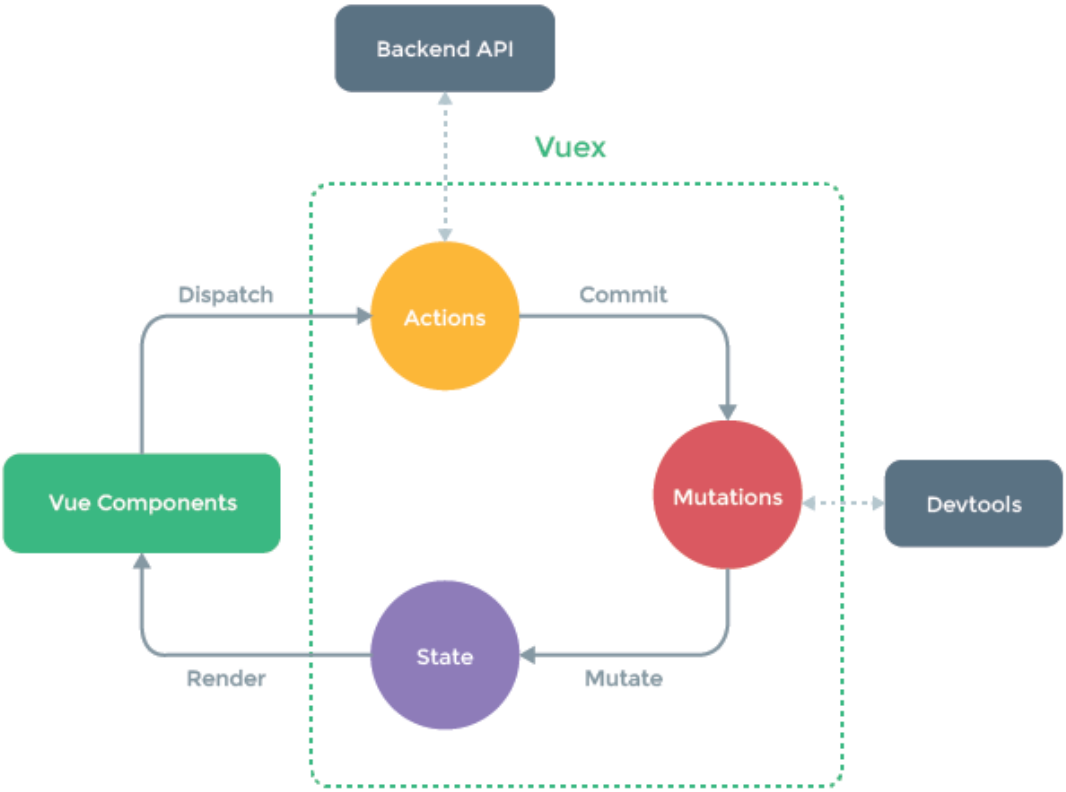


Рисунок 2.7 – Принцип роботи Vuex

2.3 Поняття односторінкового додатку

Почати потрібно з того, що існує два типи web-додатку:

- односторінковий, або ж більш відомо як SPA – Single Page Application;
- багатосторінковий, або MPA – Multi Page Application.

Забігши на перед, відразу відзначу, що Task Manager – це SPA, тобто односторінковий web-додаток. Залишилось розібратись чому саме цей тип було обрано.

Багатосторінковий додаток – це традиційний web-додаток, в якому з сервера запитується нова сторінка, для відображення кожний раз, коли відбувається обмін даними туди і назад. Об’єм контенту, який відображається, величезний, тому такі

додатки, як правило, багаторівневі. Мають в собі значну кількість посилань та складними призначеними для користувача інтерфейсами.

Односторінковий додаток (SPA) – це буквально одна сторінка. Вона постійно взаємодіє з користувачем, динамічно переписуючи список інформації для відображення, а не завантажуючи нові сторінки з серверу.

Грубо кажучи, у випадку з Vue.js, на сервері лежить один index.html файл, який замінює в собі контент, що відображається користувачу. У випадку з Vue.js, цей процес контролюється за допомогою вбудованого механізму розділення додатку на так звані «чанки», тобто на частини які підгружають потрібний функціонал по потребності.

Навіть не знаючи цього, кожний день більшість людей працює з такими додатками, розглянемо декілька прикладів:

- Trello – додаток для списків справ;
- Facebook – соціальна мережа;
- Gmail – поштовий клієнт;
- Twitter – соціальна мережа.

З вищеописаних прикладів, вже можна зрозуміти, що SPA – не тільки прості додатки, а й складні механізми, які вміщають в себе величезні комплекси фунціоналу.

Як описано вище, особливість архітектури SPA заключається в тому, що всі елементи, не обхідні для роботи додатку, технічно, знаходяться на одній сторінці. Даний тип додатків завантажує додаткові модулі після запиту користувача. Будь яка активність користувача, в таких додатках, фіксується для зручності навігації. Це дозволяє скопіювати посилання і відкрити додаток на конкретному етапі взаємодії на іншій вкладці, в іншому браузері або ж на іншому пристрої.

При завантаженні нових модулів у SPA, вміст сторінки оновлюється частично і немає необхідності повторно завантажувати елементи, які не змінювались. Це сильно збільшує швидкість відповіді і зменшує об’єм даних, що передається між браузером і сервером.

Таким чином, даний тип додатку, по способу взаємодії з користувачем, більше всього схожий на роботу desktop-додатків, але з різницею в том, що перший – знаходиться на сервері.

Як висновок, список переваг SPA буде наступний:

- доступність – можна отримувати моментальний доступ до функціоналу з будь-якого пристрою, без проблем з сумісністю, об’ємом пам’яті, потужістю або часом на інсталювання;
- універсальність – використовувати додаток можна з будь-якого пристрою, якщо на ньому є доступ до інтернету, якщо при розробці враховувались різні розширення екрану, тобто «адаптив», то використовувати такий додаток однаково зручно як з телефону так і з ПК;
- можливість використання великого об’єму даних – розмір додатку та даних, що використовуються обмежений не пам’яттю пристрою, а пам’яттю сервера;
- швидкість – одна сторінка з усім необхідним не тільки економить час на повторне завантаження даних, а й підвищує швидкість роботи;

З технічної точки зору, з появою JavaScript-фреймворків, розробка таких додатків стала навіть простішою ніж розробка традиційних MPA додатків. Такі фреймворки спрощують створення архітектури проекту і надають не мало готових елементів для роботи.

Одним з таких фрейворків є Vue.js, який підтримує мову TypeScript, детальний опис якого в розділі 2.2.2.

3 ПРОЕКТУВАННЯ ДОДАТКУ І АРХІТЕКТУРИ

3.1 Основний сценарій використання додатку

User Flow – призначений для користувача сценарій, що ілюструє порядок дій, який користувач виконує для рішення завдання.

Сценарій – це скелет інтерфейсу, що складається з кроків – окремих екранів. Для повного розуміння того, як використовується додаток, потрібно розглянути покрокові дії користувача, для роботи з додатком:

1. Користучачі, що являються робітниками компанії створюють свої облікові записи у формі реєстрації. Або ж одна відповідальна людина створює обліковий запис кожному із своїх робітників.
2. Користувач, що являється відповідальною особою, реєструє компанію, для подальшого використання функціоналу. Після цього, прив’язує до створеної компанії всі створені облікові записи робітників по email, який являється унікальним полем в базі даних.
3. Користувач, що зареєстрував компанію створює проекти, що розроблюються в робочому процесі.
4. До кожного проекту користувач створює ряд задач, описує їх та встановлює відповідального. В ході задачі в ній можуть змінюватись статус, відповідальний, витрачений на задачу час та опис задачі.

З вище описаними кроками налагоджується процес розстановки задач кожному робітнику і пришвидшується процес розробки як програмного продукту, так і будь-якого іншого процесу.

3.2 Архітектура додатку

Архітектура додатку описує її організацію з точки зору структури та поведінки шляхом представлення різних компонентів та взаємозв’язку між ними, див. рис 3.1.

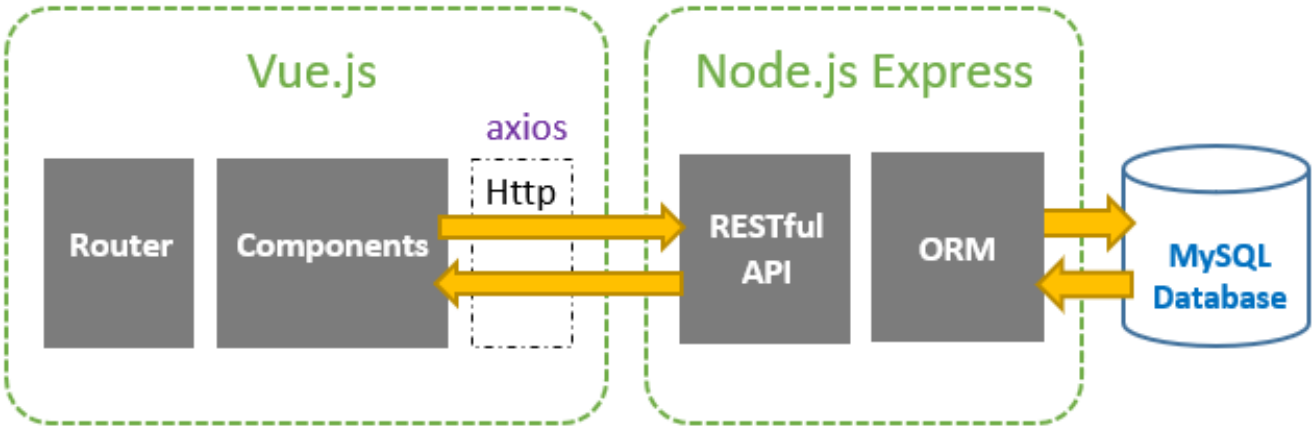


Рисунок 3.1 – Архітектура додатку Task Manager

Node.js Express експортує REST API та взаємодіє з базою даних MySQL за допомогою Sequelize ORM.

Vue Client надсилає HTTP-запити та отримує HTTP-відповіді, використовуючи Axios.

3.1.1 APIs

Абревіатура API розшифровується як «Application Programming Interface» (інтерфейс програмування додатків). Більшість великих компаній на певному етапі розробляють API для клієнтів або для внутрішнього використання, як це зроблено в додатку Task Manager.

API – це готовий код, який спрощує життя програмістам. Розробник може використовувати готовий код API, щоб написати додаток. Він допомагає організувати код і зробити так, щоб компоненти програмного забезпечення могли використовуватись неоднократно.

У випадку з розробкою web-додатку, API может передавати інформацію у відмінному від стандартного HTML формату, завдяки чому цим зручно користуватись.

Сторонні загальнодоступні API начастіше віддають дані в одному з двох форматів: XML або JSON. При розробці своїх API, використовується передача

інформації у форматі JSON. Такий формат є більш лаконічний і більш простий в читанні ніж XML.

XML набагато важче інтерпретувати в JavaScript ніж JSON, коли останній, в свою чергу, має компактну нотацію та ієрархічні дані.

В табл. 3.1 показано API, які будуть використовуватись в додатку Task Manager, для отримання, зміни, та видалення даних.

Таблиця 3.1 Список API

Метод	Посилання	Дія
POST	api/login	Авторизувати користувача
POST	api/registration	Зареєструвати користувача
POST	api/editProject	Змінити дані проекту
POST	api/editTask	Змінити дані задачі
POST	api/editUser	Змінити дані користувача
DELETE	api/deleteProject	Видалити проект
DELETE	api/deleteTask	Видалити задачу
DELETE	api/deleteUser	Видалити користувача
POST	api/addCompany	Створити компанію
GET	api/taskTypes	Отримати типи задач
GET	api/currentUser	Отримати дані авторизованого користувача
GET	api/userListInCompany	Отримати користувачів, які працюють в конкретній компанії. Id компанії береться з авторизованого користувача
POST	api/logout	Вийти з облікового запису
GET	api/project/:userId	Отримати проекти по UserId

Метод	Посилання	Дія
GET	api/task	Отримати список задач. Список задач буде складатись тільки з тих, які
GET	api/taskById/:id	Отримати задачу по Id
POST	api/addNewProject	Додати новий проект
POST	api/addNewTask	Додати нову задачу

3.3 Структура проекту

Структура проекту – важливий етап в процесі розробки.

В ході розробки структура папок для проекту розширюється і потрібно зробити максимально просте і зрозуміле «дерево» клієнта (див. рис. 3.2). Це потрібно для того, щоб в процесі розробки та розширення не виникало проблем. Структура клієнта складається за наступних частин:

- package.json – список модулів, які використовуються для клієнта;
- assets - містить в собі файли стилів та медіа файли;
- components – містить в собі окремі логічні компоненти додатку;
- directives – додаткові дерективи Vue.js, підключені до проекту;
- pages – окремі компоненти сторінок;
- plugins – підключені плагіни (у даному випадку BootstrapVue);
- main.ts – файл ініціалізації клієнту;
- models.d.ts – файл, в якому декларуються типи об’єктів;
- router.ts – vue-router файл, в якому прописуються присутні в додутку сторінки;
- babel.config.js – транспайлер, який потрібний для роботи з новим синтаксисом JavaScript, перетворення в зрозумілий для браузера запис. Іншими словами переписує код стандарту ES2015 на більш пізній;

- tsconfig.json – файл для налаштування TypeScript проекту. Він встановлює кореневий каталог проекта TypeScript, виконує налаштування параметрів компіляції та встановлює файли проекту;
- tslint.json – файл з правилами стилізації та форматування TypeScript коду;
- vue.config.js – налаштування Vue.js проекту, а також конфігурація webpack.

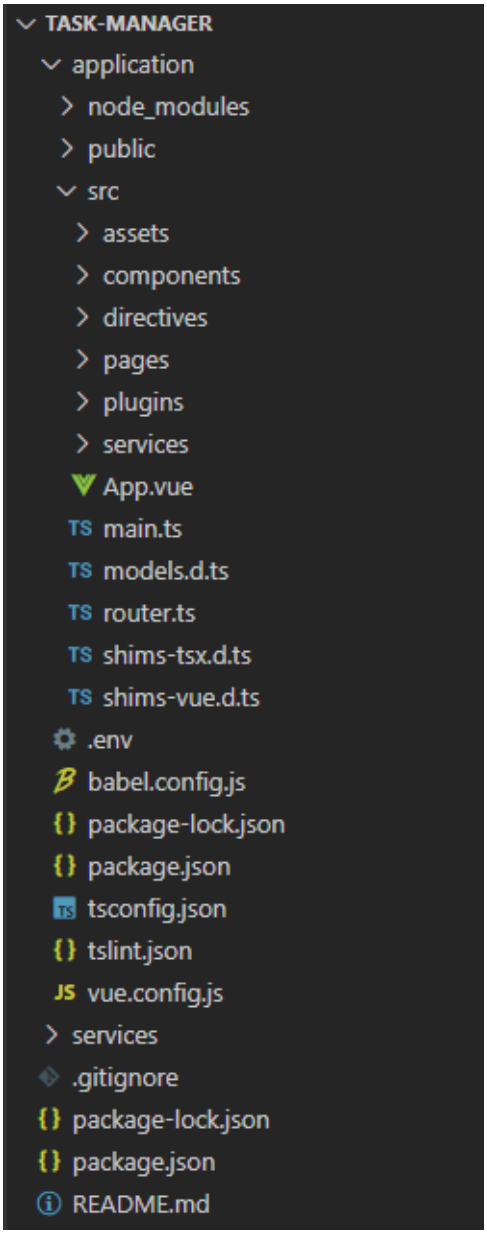


Рисунок 3.2 – Структура клієнту Task Manager

Для розуміння структури повністю, також створено структуру серверу, див. рис. 3.3. Вона складається з наступних компонентів:

- db.config.js – експортує параметри налаштування для підключення MySQL та Sequelize;
- express web-server у файлі server.js, де відбувається налаштування CORS, ініціалізація та запуск API Express REST;
- в папці models файли, у яких додається конфігурація баз даних, а також модель даних Sequelize;
- котролери в папці controllers;
- маршрути для обробки операцій в папці routes.

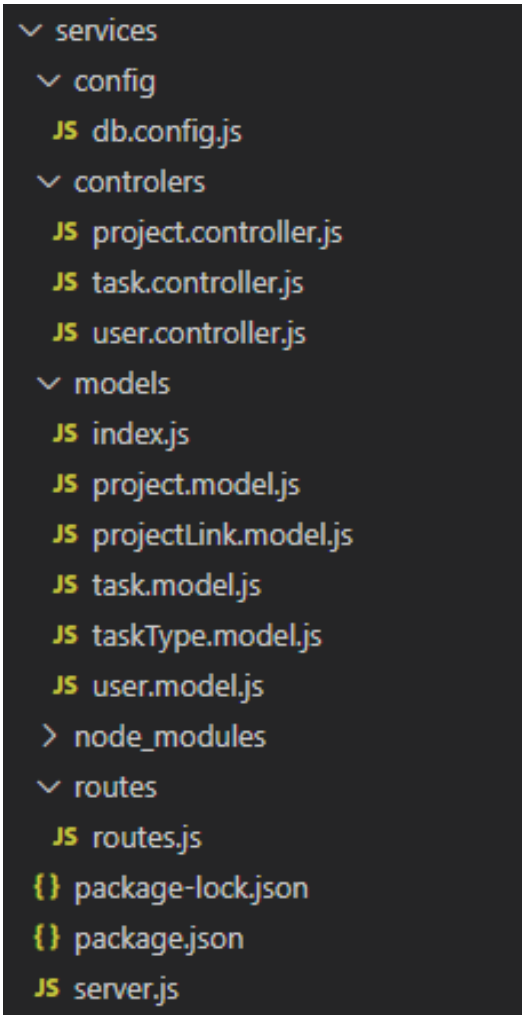


Рисунок 3.3 – Структура серверу Task Manager

3.4 Таблиці баз даних

Для того, щоб працювати з базою даних, зберігати там інформацію, потрібно визначити об’єкти.

Нижче за допомогою таблиці будуть описані сутності в базі даних. Докладна інформація про кожну сутність, назву, атрибути та опис того, для чого вона використовується див. табл. 3.2.

Таблиця 3.2 – Сутності бази даних Task Manager

Назва таблиці	Атрибути	Опис атрибута	Загальний опис об’єкта
User	Id	Ідентифікатор користувача. (Primary Key)	Сутність являє собою користувача з прив’язаною до нього інформацією.
	FirstName	Ім’я користувача.	
	LastName	Прізвище користувача.	
	Position	Посада користувача.	
	Email	Email користувача.	
	Password	Пароль користувача.	
	CompanyId	Компанія в якій працює робітник	
Company	Id	Ідентифікатор компанії. (Primary Key)	Сутність являє собою зареєстровані

Назва таблиці	Атрибути	Опис атрибута	Загальний опис об’єкта
	Name	Назва компанії.	компанії
ProjectLink	Id	Ідентифікатор. (Primary Key)	Розв’язочна таблиця між User і Project
	UserId	Foreign key з таблиці User, який вказує, до якого користувача прив’язаний проект.	
	ProjectId	Foreign key з таблиці Project	
Project	Id	Ідентифікатор проекту. (Primary Key)	Сутність являє собою створені проекти
	Name	Назва проекту	
	Description	Опис проекту	
Task	Id	Ідентифікатор задачі. (Primary Key)	Сутність являє собою створені задачі
	Name	Назва задачі	
	Description	Опис задачі	

Назва таблиці	Атрибути	Опис атрибута	Загальний опис об’єкта
	Deadline	Дата здачі задачі	
	TaskTypeId	Тип задачі. Foreign key таблиці TaskType.	
	PriorityId	Приорітет. Foreign key таблиці Priority.	
	SpentTimeInHour	Витрачений час в годинах.	
	UserId	Відповідальний користувач Foreign key з таблиці User.	
	ProjectId	Прив’язаний проект до задачі Foreign key з таблиці Project	
TaskType	Id	Ідентифікатор типу задачі. (Primary Key)	Довідник типів задач
	Name	Назва типу задачі	
Priority	Id	Ідентифікатор приорітету. (Primary Key)	Довідник приорітетів

Назва таблиці	Атрибути	Опис атрибута	Загальний опис об’єкта
	Name	Назва приорітету	

3.3.1 Діаграма відносин сутностей

Діаграма відносин сутностей – це блок-схеми, які показують, як сутності ставляться один до одного в системі. ER (Entity Relationship) діаграма – модель, яку найчастіше використовують для розробки реляційних баз даних.

Для додатку Task Manager використовується наступна ER діаграма (див. рис. 3.4).

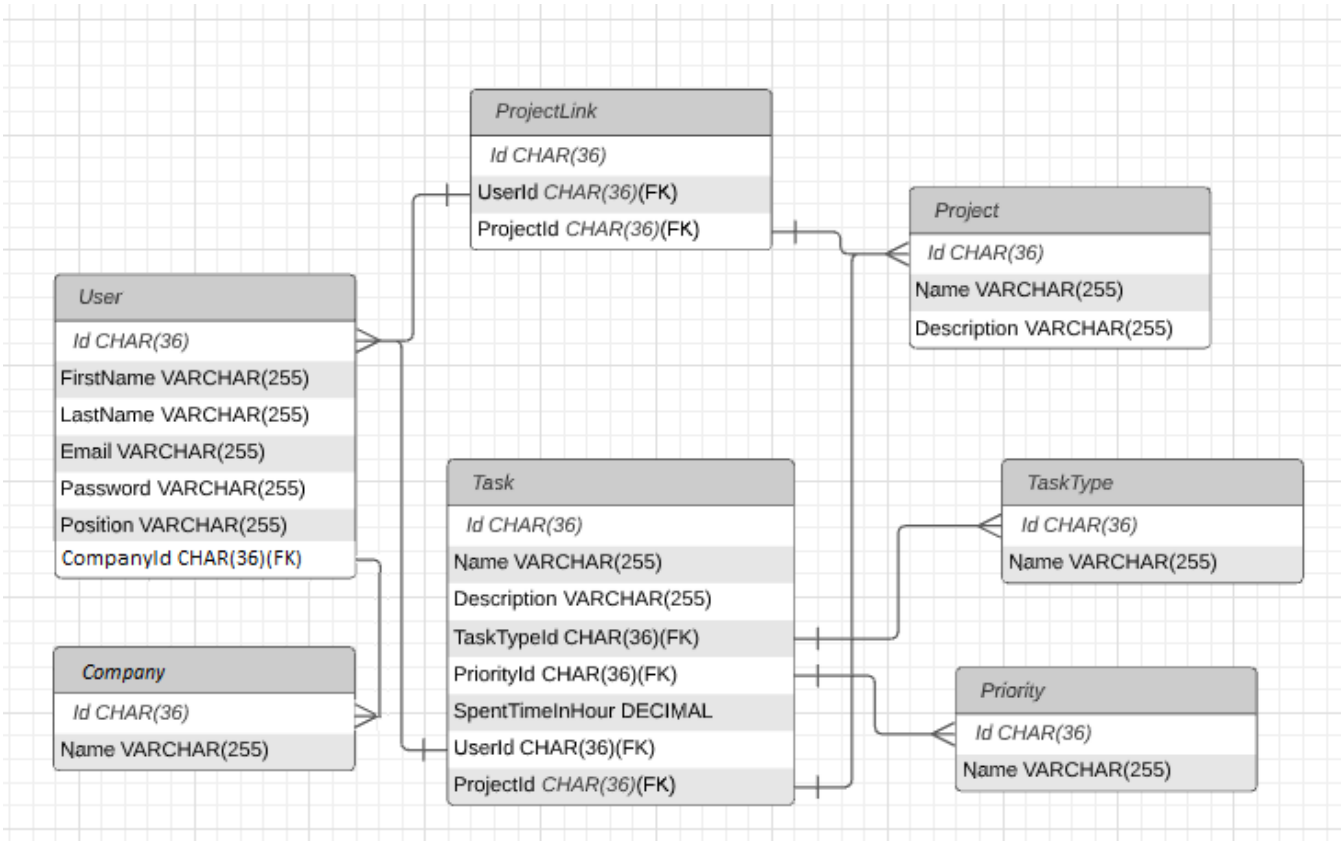


Рисунок 3.4 – Task Manager Entity Relationship Diagram

3.5 Інтерфейс

Інтерфейс web-додатку, написаного на TypeScript за домогою Vue.js, складається з компонентів, які динамічно змінюються в залежності від відображуваного фунціоналу.

Компоненти – це блоки коду, які підходять до багатократного використання. Вони можуть включати в себе опис зовнішнього вигляду частин додатку, а також реалізацію логічних можливостей проекту. Вони допомагають розробнику у створенні модульної кодової бази, яку зручно підтримувати.

В додатку Task Manager компоненти, по простому принципу, розділено на:

- компоненти сторінок, які використовуються для реєстрації окремих посилань у vue-router;
- загальні компоненти, які використовуються в компонентах сторінок. Такі компоненти являються логічними блоками, що можуть використовуватись не один раз.

Для розуміння інтерфейсу Task Manager, розглянемо його компоненти. В табл. 3.3 показано кожен компонент сторінки, що використовується в додатку, його назву та опис.

Таблиця 3.3 – Набір компонентів сторінок Task Manager

Назва	Опис
Login	Сторінка авторизації користувача.
SignUp	Сторінка реєстрації користувача.
HomePage	Головна сторінка, в яку користувач попадає після авторизації. В ній також знаходиться список проектів.
Profile	Сторінка перегляду/редагування профілю.
AddCompany	Сторінка реєстрації компанії.

Назва	Опис
TaskList	Сторінка зі списком задач.
Task	Сторінка перегляду/редагування задачі.
Project	Сторінка перегляду/редагування профілю.
ForgetPassword	Сторінка відновлення паролю.

В табл. 3.4 показано загальні компоненти. Як вище описано такі компоненти в додатку можуть використовуватись не один раз. Таблиця розділена на декілька колонок і включає в себе: назву компоненту, його опис та до якого компоненту він підключений, тобто батьківський компонент або декілька компонентів.

Таблиця 3.4 – Набір загальних компонентів Task Manager

Назва	Опис	Батьківський компонент(и)
BasicLayout	Основний контейнер в інтерфейсі авторизованого користувача.	HomePage, Profile, Project, TaskList, Task, AddCompany
Header	Верхня частина інтерфейсу авторизованого користувача.	BasicLayout
AuthForm	Форма для авторизації користувача.	Login
ForgetPassButton	Кнопка з логікою переходу на сторінку з відновленням паролю.	AuthForm
RegistrationForm	Форма реєстрації користувача.	SignUp
ForgetPasswordForm	Форма відновлення паролю.	ForgetPassword

Назва	Опис	Батьківський компонент(и)
AddCompanyForm	Форма реєстрації компанії	
TaskForm	Форма для створення\зміни задачі.	Task
DataPicker	Компонент для вибору дати, який знадобиться при виборі кінцевого сроку задачі.	TaskForm
TimePicker	Компонент з можливостю встановлення часу, який знадобиться для того, щоб встановлювати запланований або витрачений на задачу час.	TaskForm
DeleteBtn	Кнопка видалення запису з таблиці (проекту, задачі).	Task, Project
BaseSelect	Компонент з логікою «селекту», тобто можливістю вибору одного з варіантів.	TaskForm,
BaseButton	Кнопка зі стандартними стилями, яка використовується по всьому проекту. Компонент передає подію кліку, за допомогою цього можна зробити будь-яку дію. Також можна передавати стилі в залежності від того, яка кнопка потрібна.	AuthForm, TaskForm, Project, ForgetPasswordForm, R egistration Form

Назва	Опис	Батьківський компонент(и)
Footer	Нижня частина інтерфейсу зареєстрованого користувача.	BasicLayout
Logo	Окремий блок з логотипом, який включає в себе посилання на головну сторінку.	Header, Footer
ProjectCard	Блок для перегляду інформації про проект. Знаходиться на головній сторінці, як один із списку проектів.	HomePage
TaskSearch	Блок для пошуку задачі по назві задачі.	TaskList
TaskFilter	Блок для фільтрації задач по відповідальному користувачу.	TaskList
TaskCard	Блок на сторінці списку задач. Являє собою інформацію по одній окремій задачі.	TaskList

4 ЕТАП РЕАЛІЗАЦІЇ

Розробка web-додатку – це цілий комплекс заходів та дій з планування створення сайту в мережі Internet в залежності від поставлених цілей і завдань.

Різниця між звичайним сайтом та цілим SPA дуже велика, адже в останньому потрібно розроблювати набагато більше різного функціоналу, як і складного так і простого. Також для розробки SPA потрібно більше технологій, в той час як для розробки звичайного сайту може бути достатньо звичайного JavaScript.

Програмна реалізація веб-додатку Task Manger складається з певної кількості кроків:

- розробка дизайну;
- створення запланованої структури;
- розробка серверної частини;
- ініціалізація з базою даних;
- розробка інтерфейсу;
- «деплой» проекту.

4.1 Програмно-апаратна частина

Розробка програмно-апаратної частини, або ж Backend-розробка – це набір програмно-апаратних засобів, за допомогою яких реалізована логіка роботи додатку. Іншими словами, це те, що приховано від користувача і що відбувається поза браузером і його пристроєм.

Наприклад, якщо в пошукову стрічку того ж інтернет магазину ввести якусь річ, яку потрібно знайти, запит відправляється на сервер і в роботу вступає Backend-логіка.

4.1.1 Створення Node.js додатку

Для створення проекту, використовувався Git Bash (див. рис. 4.1), який використано в якості Unix-консолі, та сам Node.js.

Рисунок 4.1 – Консоль Git Bash

Спочатку, за допомогою консолі створювалась папка серверу, див. рис. 4.2.

```
$ mkdir server
$ cd server
```

Рисунок 4.2 – Створення папки додатку Node.js

Наступним кроком був додаток Node.js файлом package.json, див рис. 4.3.

```
npm init

name: (server)
version: (1.0.0)
description: Node.js Rest Apis with Express, Sequelize & MySQL.
entry point: (index.js) server.js
test command:
git repository:
keywords: nodejs, express, sequelize, mysql, rest, api
author: serhii husak
license: (ISC)

Is this ok? (yes) yes
```

Рисунок 4.3 – Ініціалізація Node.js

Для того, щоб налаштувати сервер, проведено встановлення необхідних модулів, а саме:

- express;
- sequelize;
- body-parser;
- cors.

4.1.2 Хешування паролю

Для того щоб зрозуміти навіщо хешувати пароль, потрібно зрозуміти, що взагалі таке хеш.

Криптографічний хеш-функція, частіше звана просто хешем, - це математичний алгоритм, що перетворює довільний масив даних в складається з букв і цифр рядок фіксованої довжини. Причому за умови використання того ж типу хешу довжина ця буде залишатися незмінною, незалежно від обсягу вступних даних. Крипостійкість хеш-функція може бути тільки в тому випадку, якщо виконуються головні вимоги: стійкість до відновлення хешованих даних і стійкість до колізій, тобто утворення з двох різних масивів даних двох однакових значень хеш-кодування.

Цікаво, що під дані вимоги формально не підпадає ні один з існуючих алгоритмів, оскільки знаходження зворотного хешу значення - питання лише обчислювальних потужностей.

Найбільш поширена сфера застосування хешування - зберігання паролів. Наприклад, якщо користувач забув пароль до web-додатку, то доведеться скористатися функцією відновлення пароля. В цьому випадку користувач, не отримає свій старий пароль, оскільки онлайн-сервіс насправді не зберігає паролі у вигляді звичайного тексту. Замість цього він зберігає їх у вигляді хеш-значень. Тобто навіть сам сервіс не може знати, як насправді виглядає ваш пароль. Виняток становлять лише ті випадки, коли пароль дуже простий і його хеш-значення широко відомо в колах хакерів.

Основною ідеєю хешування паролю являється його захист, якщо база даних якимось чином попаде в руки зловмисникам.

В web-додатку Task Manager було використано бібліотеку node.bcrypt.js, яка використовує хеш-функцію bcrypt для хешування та алгоритм «Eksblowfish», див. рис. 4.4. Хеш паролю, з використанням bcrypt сладає 60 символів.

```
EksBlowfishSetup (cost, salt, key)
  state ← InitState ()
  state ← ExpandKey (state, salt, key)
  repeat (2^cost)
    state ← ExpandKey (state, 0, salt)
    state ← ExpandKey (state, 0, key)
  return state
```

Рисунок 4.4 – Алгоритм Eksblowfish

4.2 Клієнтська частина

Клієнтська частина розробки додатку, або ж Frontend - це публічна частина web-додатків, з якої користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань, призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. По суті, фронтенд - це все те, що бачить користувач при відкритті web-сторінки.

У свою чергу, web-додаток - клієнт-серверний додаток, в якому клієнтом виступає в основному браузер, а сервером - web-сервер. Логіка web-додатки розподілена між сервером і клієнтом, зберігання даних здійснюється переважно на сервері, обмін інформацією відбувається по мережі. Простіше кажучи, це те, що бачить користувач і які дії виконує кожен раз, коли підключається до мережі інтернет і відкриває будь-який браузер.

Frontend-розробка - це робота зі створення публічної частини web-додатки, з якої безпосередньо контактує користувач, і функціоналу, який зазвичай виконується на стороні клієнта. Тобто, Fontend розробник працює над тим, щоб на

сайті кожна кнопка, іконка, текст і вікно не тільки стояли на своєму місці, не перекривали один одного і виглядали цілісно (це веб-верстка), але і щоб вони виконували своє пряме призначення - виробляли якісь дію (наприклад, щоб кнопка "купити" відкривала кошик, а "play" - запускала відтворення фільму або музики).

4.2.1 Встановлення Vue CLI проекту

Навряд чи можна собі уявити сучасну web-розробку, не використовуючи інструментів командного рядка (CLI). Вони значно полегшують та прискорюють процес розробки шляхом скорочення кількості виконуваних повторюваних і трудомістких завдань. Без використання такого інструменту, розробнику прийдеться вручну налаштовувати проект, тестувати та оптимізувати код з відстеженням залежностей.

Саме тому всі сучасні Frontend-фреймворки пропонують власні інструмети для роботи в командному рядку, і Vue.js – не виняток. Мало того, з виходом останньої версії, Vue CLI зробив крок вперед, в порівнянні з іншими фреймворками. Теперь він не тільки дуже потужний і гнучкий, а також являється повноцінним GUI (Graphical User Interface), тобто справжнім графічним інтерфейсом.

Створення нових проектів на Vue.js тепер набагато простіше, ніж було до цього.

Vue CLI являє собою набір інструментів для швидкого прототипування, легкого створення нових додатків та ефективного управління проектами на Vue.js.

Він складається з трьох основних інструментів:

- CLI – це глобально встановлений npm-пакет, що забезпечує основний функціонал через команду “vue” в консолі. Це дозволяє легко створити новий проект, або просто швидко зробити прототип початкових ідей. Якщо потрібно зробити більш конкретний візуальний контроль над проектами, то можна відкрити графічну, як описувалось раніше, GUI-версію;

- сервіс CLI (CLI Service) – залежність для розробки (бінарний файл vue-cli-service), який встановлено локально на кожен проект, за допомогою CLI. Він дозволяє розробляти проект, зібрати його в продакшен, а також перевірити конфігурацію проекту всередині webpack;
- модулі CLI (CLI Plugins) – це npm-пакети, що надають додаткові можливості для проекту. Модулі можна додати в будь-який час процесу розробки.

У web-додатку Task Manager було використано дану технологію у зв’язку з економією часу, оскільки Vue CLI прискорює та полегшує розробку проекту на Vue.js, завдяки великої кількості функціональних можливостей:

- архітектура на основі плагінів. Vue CLI повністю працює на плагінах, що робить його дуже гнучким і розширюваним. Можна вибрати, які з вбудованих плагінів потрібні під час процесу створення нового проекту. Але все не обмежено тільки цим, адже можема додати будь-яку кількість плагінів в будь-який момент після створення проекту;
- Vue CLI повністю налаштовується, він являється розширюваним і обновлюваним інструментом;
- набір офіційних встановлених плагінів, який об'єднує професійні інструменти екосистеми Frontend. Для розробки знадобились: Babel, ESLint та, власне, сам TypeScript – як мова розробки;
- один за замовчуванням шаблонів, який ми можемо змінити відповідно до потрібних потреб під час створення проекту або після цього;
- не потрібно використовувати витяг залежностей (eject). На відміну від CLI-інструментів React і Angular ми можемо безпечно перевірити і налаштувати конфігурацію webpack нашого проекту в будь-який час після створення без необхідності вилучення залежностей додатки і перемикання на ручний спосіб управління його конфігурації;
- миттєве створення прототипів без необхідності в будь-якої конфігурації;

- режим використання сучасних можливостей. Це означає, що можена збирати наш додаток для сучасних браузерів, але з автоматичною підтримкою для старих (крос-браузерність);
- повномасштабний графічний інтерфейс для створення, оновлення та управління складними проектами без будь-яких труднощів;
- API для призначеного для користувача інтерфейсу плагінів. Vue UI надає API для плагінів, який можна використовувати для додавання власних функціональних можливостей до GUI-версії версії командного рядка;
- багато корисних плагінів від спільноти.

Для створення потрібного нам проекту потрібна лиш одна стрічка у консолі Git Bash, див. рис. 4.4.

```
$ vue create task-manager
```

Рисунок 4.4 – Створення проекту Vue CLI

Далі було обрано пресет (шаблон) налаштувань, до яких входили: TypeScript, Babel, ESLint та SASS, Vue-Router та Vuex.

4.2.2 Структура Vue Router

Vue Router – офіційна бібліотека Vue.js, яка була використана в ролі помічника при розробці SPA-додатку.

При створенні SPA-додатків за допомогою Vue.js ця бібліотека просто життєво необхідна, адже саме вона допомагає будувати різні маршрути, використовуючи тільки один HTML-файл, замість багатьох, як в багатосторінкових додатків.

Додаючи Vue Router в проект, було компоновано додаток з окремих компонентів. Простіше кажучи, Vue Router порівнює створені компоненти з маршрутами, які вказуються у файлі налаштування та відображає потрібний компонент за потрібним маршрутом.

Також при розробці використовувався глобальний об’єкт router, для отримання потрібних даних по мрашруту.

В табл. 4.1 показано маршрути, які використовуються в додатку Task Manager.

Таблиця 4.1 – Маршрути в Task Manager

Назва	Маршрут
Login	/login
SignUp	/sign-up
HomePage	/
Profile	/profile
TaskList	/task-list
Task	/task/:id або /task у випадках редагування/додавання задачі відповідно
Project	/project/:id або /project у випадках редагування/додавання задачі відповідно
ForgetPassword	/forget-password

4.2.3 Ініціалізація Axios з HTTP клієнтом

Одне з фундаментальних завдань інтерфейсного додатку - це взаємодія з серверами за допомогою протоколу HTTP. JavaScript може надсилати мережеві запити на сервер і завантажувати нову інформацію за потреби, не перезавантажуючи сторінку.

Термін такої операції - асинхронний JavaScript та XML (AJAX) , який використовує об'єкт XMLHttpRequest для зв'язку з серверами. Він може надсилати та отримувати інформацію у різних форматах, включаючи JSON, XML, HTML та текстові файли.

В додатку було використано один зі способів як це зробити, а саме, за допомогою Axios.

Axios – це бібліотека JavaScript, або ж в даному випадку бібліотека для Vue.js, яка використовується для надсилання HTTP-запитів з Node.Js або XMLHttpRequests.

Основними особливостями Axios є:

- можливість використання для перехоплення HTTP-запитів та відповідей;
- автоматичне трансформування запитів та відповідей;
- захист на стороні клієнта XSRF (Cross-site request forgery);
- вбудована підтримка прогресу завантаження;
- можливість скасувати запити.

На рис 4.5 показана ініціалізація Axios в додатку для можливості використання розроблених API.

```
import axios from "axios";

export default axios.create({
  baseURL: "http://localhost:8080/api",
  headers: {
    "Content-type": "application/json"
  }
});
```

Рисунок 4.5 – Ініціалізація Axios

5 ТЕСТУВАННЯ

Тестування веб-додатків - це комплекс послуг, який може включати в себе різні види тестування програмного забезпечення.

Основна мета будь-якого тестування, в тому числі і тестування веб-додатків, - виявити всі помилки в програмному забезпеченні і розробити рекомендації щодо їх запобігання в майбутньому.

В процесі тестування було використано декілька його типів:

1. Функціональне тестування, або ж процес оцінки поведінка додатку. Цей тип тестування дозволяє визначити, чи всі розроблені функції поводяться так, як потрібно. Було виконано з використанням раніше підготовленим тестовим сценарієм.

2. Usability тестування (User Experience) – перевірка комфортного використання додатку для користувача, тобто наскільки легко знайти необхідний функціонал, виконати бажані дії.

Оскільки основною з переваг web-додатку Task Manager є його простота, то цей пункт тестування є одним з найважливіших і складається з наступних кроків:

- навігаційне тестування сайту. Чи всі сторінки, кнопки і поля на них, зрозумілі у використанні, доступ до головної сторінки і меню з усіх інших сторінок можливий, навігація проста і інтуїтивно зрозуміла;
- тестування контенту. Відсутність граматичних або орфографічних помилок, контент інформативний і структурований, зображення і заголовки мають відповідні розміри і розміщені правильно;
- зручність використання. Чи зрозуміла структура веб-додатку, яке враження справляє і чи є зайві компоненти на сторінках;
- тестування UI (User Interface). Відповідність стандартам графічних інтерфейсів і елементів дизайну, правильність локалізованих версій, тестування з різними розширеннями, на смартфонах і планшетах (адаптивність).

Для тестування адаптивності, тестування відображення інтерфейсу було проведено на наступних розширеннях екрану (px):

- 1920x1080;
- 1366x768;
- 1024x768;
- 800x600;
- 375x812;
- 320x568.

3. Тестування сумісності – процес оцінки поведінки додатку в різних браузерах, операційних системах.

Коректна робота залежить від типу браузера. Щоб перевірити, що компоненти відображаться на різних браузерах коректно, їх функціонал відпрацьовує в заданих браузерах була зроблена перевірка крос-браузерності, яка була проведена на наступних браузерах:

- Internet Explorer 11;
- Chrome 29;
- Firefox 20;
- Edge 12.

Також тестування додатку проведено на різних операційних системах для уникнення помилок на різних пристроях. Таке тестування було проведено на наступних операційних системах:

- Windows;
- MacOS;
- Linux.

На мобільних пристроях не тільки може не правильно відображатись інтерфейсна частина, але й невірно відпрацьовувати функціонал, тому також було виконано перевірку функціональності на мобільних простроях.

4. Тестування продуктивності – комплекс перевірок, спрямований на визначення лімітів продуктивності додатку.

5. Тестування безпеки – знаходження всіх місць, в яких могли припуститись помилки. Web-додаток та web-сервер нерозривно пов’язані, то тестування безпеки проводилось в обох випадках. Тестуючи захист додатку, було проведено пошук вразливих місць для атаки на користувачів. Тестуючи захист web-серверу, було проведено пошук вразливих місць для атаки на сервер і його інфрастуктуру.

ВИСНОВКИ

В сучасному світі все частіше почали з’являтися інструменти, які дозволяються полегшувати життя та пришвидшувати дії, які відбуваються протягом тих чи інших проміжків часу. Одним з таких є робочий процес, в якому організованість та продуктивність є основами успіху.

Для досягнення поставленої мети було розроблено web-додаток Task Manager, що є одним з вищеописаних інструментів. Він дуже простий у використанні для створення проектів та задач для кожного працівника компанії.

Одною з найголовніших переваг додатку є легкість. Не потрібно витрачати багато часу на те, щоб розібратись у функціоналі. Достатньо зареєструватись, створити потрібні проекти та прив’язати до них задачі, в яких, в свою чергу, поставити відповідального.

Шляхом зменшення часу на розподілення задач, робочий процес стає більш продуктивним. Економлячи на кожній задачі хоча б 10 хвилин, в результаті можна досягти вагомих успіхів.

Додаток реалізовано за допомогою найсучасніших технологій, в основу яких входили: Vue.js, TypeScript, Node.js та MySQL. З їх допомогою, додаток вдалось розробити максимально швидко. Також використання таких технологій дозволяє набагато легше розширювати функціонал додатку. Можливо після появи нових ідей, або ж після відгуків користувачів.

ЛІТЕРАТУРА

1. Asana [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://habr.com/en/post/151162/
2. Erik Hanchett, Benjamin Listwon, Vue.js in Action. – Manning Publications Co., 2019.. – 24с.
3. Jira Software [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://www.atlassian.com/software/jira
4. Karl Wiegers, Joy Beatty, Software Requirements - Third Edition – Microsoft Press - Redmond, Washington - 2013. — 150-192 с.
5. Mike Cantelon, Marc Harter, T.J. Holowaychuk, Nathan Rajlick, Node.js in Action, - Maning Publications - 2017. – 11-22 с.
6. Nathan Rozentals, Mastering TypeScript – Packt Publishing, 2015. – 28-40 с.
7. Node.js [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://nodejs.org/
8. Shabbir Challawala, Jaydip Lakhatariya, Chintan Mehta, Kandarp Patel - MySQL 8 for Big Data – Packt Publishingt Ltd. - 2017. – 226 с.
9. Vue CLI docs [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://cli.vuejs.org/guide/
10. Vue Router [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://router.vuejs.org/
11. Vue.js. Режим доступу: https://vuejs.org/v2/guide/
12. В чем приемущества Node.js? [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://artjoker.ua/ru/blog/v-chem-preimushchestva-nodejs/.
13. Таск менеджер [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: https://biz30.timedoctor.com/task-manager/
14. Тестирование веб-проектов: основные этапы и советы [Електронний ресурс]: [Інтернет-портал]. – Електронні дані. – Режим доступу: http://qalight.ua/baza-znaniy/testirovanie-veb-proektov-osnovnye-etapy-i-sovety/

Додаток А

Головний HTML файл

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>Task Manager</title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but msb doesn't work properly without JavaScript enabled. P
lease enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Додаток Б

Налаштування серверу

server.js

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));

// parse requests of content-type - application/json
app.use(bodyParser.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome" });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Додаток В

Налаштування Vue Router

router.js

```
import Vue from 'vue';
import Router from 'vue-router';
Vue.use(Router);
export default new Router({
    mode: 'history',
    routes: [
        {
            path: '/',
            name: 'home-page',
            component: () => import('@/pages/HomePage.vue'),
        },
        {
            path: '/login',
            name: 'login',
            component: () => import('@/pages/Login.vue'),
        },
        {
            path: '/sign-up',
            name: 'sign-up',
            component: () => import('@/pages/SignUp.vue'),
        },
        {
            path: '/task/:id?',
            name: 'task',
            component: () => import('@/pages/Task.vue'),
        },
        {
            path: '/profile',
            name: 'profile',
            component: () => import('@/pages/Profile.vue'),
        },
        {
            path: '/project/:id?',
            name: 'project',
            component: () => import('@/pages/Project.vue'),
        },
        {
            path: '/forget-password',
            name: 'forget-password',
            component: () => import('@/pages/ForgetPassword.vue'),
        },
    ],
});
```

Додаток Г

Конфігураційні файли клієнту

App.vue

```
<template>
    <div id="app">
        <router-view />
    </div>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';

@Component
export default class App extends Vue { }
</script>

<style lang="scss">
@import url("https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500;600
;700&display=swap");

#app {
    font-family: "Montserrat", Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    color: #2c3e50;
    line-height: 1.5;
}
</style>
```

main.ts

```
import '@babel/polyfill'
import 'mutationobserver-shim'
import Vue from 'vue'
import './plugins/bootstrap-vue'
import App from './App.vue'
import router from './router'
import store from './store'
import BootstrapVue from 'bootstrap-vue';
import 'bootstrap-vue/dist/bootstrap-vue.css';
import axios from 'axios';
import VueAxios from 'vue-axios';
import VueMask from 'v-mask';
```

```
Vue.use(BootstrapVue);
Vue.use(VueAxios, axios);
Vue.use(VueMask);

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

shims-vue.d.ts

```
declare module '*.vue' {
  import Vue from 'vue';
  export default Vue;
}

declare module 'vue-carousel';
declare module 'gsap';
declare module 'vue-custom-scrollbar';
```

shims-tsx.d.ts

```
import Vue, { VNode } from 'vue';

declare global {
  namespace JSX {
    // tslint:disable no-empty-interface
    interface Element extends VNode {}
    // tslint:disable no-empty-interface
    interface ElementClass extends Vue {}
    interface IntrinsicElements {
      [elem: string]: any;
    }
  }
}
```

vue.config.js

```
module.exports = {
  publicPath: process.env.NODE_ENV === 'production'
    ? '/task-manager/'
    : '/',
  filenameHashing: false,
```

```
  css: {
    loaderOptions: {
      sass: {
        prependData: `@import "@/assets/scss/main.scss";`
      }
    }
  }
}
```

tslint.json

```
{
  "defaultSeverity": "warning",
  "extends": [
    "tslint:recommended"
  ],

  "linterOptions": {
    "include": [
      "src//*.ts",
      "src//*.vue"
    ],
    "exclude": [
      "node_modules/**"
    ]
  },

  "rules": {
    "quotemark": [true, "single"],
    "indent": [true, "spaces", 4],
    "interface-name": false,
    "ordered-imports": false,
    "object-literal-sort-keys": false,
    "no-consecutive-blank-lines": false,
    "member-access": false,
    "forin": false,
    "no-empty": false,
    "prefer-for-of": false,
    "whitespace": [
      false,
      "check-branch",
      "check-decl",
      "check-operator",
      "check-separator",
      "check-type"
    ]
  }
}
```

store.ts

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.Store({
    state: {
        tasks: null,
        profileInfo: null,
        projects: null,
        taskTypes: null,
        taskPriorities: null,
        taskStatuses: null,
        usersList: null,
        task: null,
        userInfo: null,
        serverUrl: 'https://localhost:8080',
    },
    getters: {
        getTasks(state: any) {
            return state.tasks;
        },
        getServerUrl(state: any) {
            return state.serverUrl;
        },
        getProfileInfo(state: any) {
            return state.profileInfo;
        },
        getProjects(state: any) {
            return state.projects;
        },
        getUserInfo(state: any) {
            return state.userInfo;
        },
    },
    mutations: {
        setTasks(state: any, polyload: any) {
            state.tasks = polyload;
        },
        setServerUrl(state: any, polyload: any) {
            state.serverUrl = polyload;
        },
        setProfileInfo(state: any, polyload: any) {
            state.profileInfo = polyload;
        },
        setProjects(state: any, polyload: any) {
            state.projects = polyload;
        },
        setTaskStatuses(state: any, polyload: any) {
```

```
            state.taskStatuses = polyload;
        },
        setPriorities(state: any, polyload: any) {
            state.taskPriorities = polyload;
        },
        setUsersList(state: any, polyload: any) {
            state.usersList = polyload;
        },
        setTask(state: any, polyload: any) {
            state.task = polyload;
        },
        setUserInfo(state: any, polyload: any) {
            state.userInfo = polyload;
        },
    },
    actions: {
        async addNewProject(context: any, polyload: any) {
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            // tslint:disable-next-line:max-line-length
            await settings(context.getters.getServerUrl + '/api/addNewProject')
                .catch((error) => {
                    throw error;
                });
        },
        async getProjectByUserId(context: any, polyload: any) {
            return new Promise((resolve, reject) => {
                axios.get(context.getters.getServerUrl +
                    '/api/project?userId=' + polyload.userId)
                    .then((response) => {
                        context.commit('setProjects', response.data);

                        resolve(response);
                    }).catch((error) => {
                        reject(error);
                        throw error;
                    });
            });
        },
        async getTasks(context: any, polyload: any) {
            return new Promise((resolve, reject) => {
                axios.get(context.getters.getServerUrl +
                    '/api/task')
                    .then((response) => {
                        context.commit('setTasks', response.data);
                        resolve(response);
                    }).catch((error) => {
```

```
                reject(error);
                throw error;
            });
    });
},
async getTaskTypes(context: any, polyload: any) {
    return new Promise((resolve, reject) => {
        axios.get(context.getters.getServerUrl +
            '/api/taskTypes')
            .then((response) => {
                context.commit('setTaskTypes', response.data);
                resolve(response);
            }).catch((error) => {
                reject(error);
                throw error;
            });
    });
},
async getCurrentUser(context: any, polyload: any) {
    return new Promise((resolve, reject) => {
        axios.get(context.getters.getServerUrl +
            '/api/currentUser')
            .then((response) => {
                context.commit('setUserInfo', response.data);
                resolve(response);
            }).catch((error) => {
                reject(error);
                throw error;
            });
    });
},
async getUserList(context: any, polyload: any) {
    return new Promise((resolve, reject) => {
        axios.get(context.getters.getServerUrl +
            '/api/userListInCompany')
            .then((response) => {
                context.commit('setUsersList', response.data);
                resolve(response);
            }).catch((error) => {
                reject(error);
                throw error;
            });
    });
},
async getTaskById(context: any, polyload: any) {
    return new Promise((resolve, reject) => {
        axios.get(context.getters.getServerUrl +
            '/api/taskById?id=' + polyload.id)
            .then((response) => {
                context.commit('setTask', response.data);
                resolve(response);
```

```
                }).catch((error) => {
                    reject(error);
                    throw error;
                });
        });
    },
    async addNewTask(context: any, polyload: any) {
        const settings = axios.create({
            baseURL: '',
            method: 'post',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' },
            data: polyload.data,
        });
        await settings(context.getters.getServerUrl + '/api/addNewTask')
            .catch((error) => {
                throw error;
            });
    },
    async login(context: any, polyload: any) {
        const settings = axios.create({
            baseURL: '',
            method: 'post',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' },
            data: polyload.data,
        });
        await settings(context.getters.getServerUrl + '/api/login')
            .catch((error) => {
                throw error;
            });
    },
    async logout(context: any, polyload: any) {
        const settings = axios.create({
            baseURL: '',
            method: 'post',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' },
        });
        await settings(context.getters.getServerUrl + '/api/logout')
            .catch((error) => {
                throw error;
            });
    },
    async registration(context: any, polyload: any) {
        const settings = axios.create({
            baseURL: '',
            method: 'post',
            headers: { 'Content-Type': 'application/json;charset=UTF-8' },
            data: polyload.data,
        });
        await settings(context.getters.getServerUrl + '/api/registration')
            .catch((error) => {
                throw error;
```

```
            });
        },
        async editTask(context: any, polyload: any) {
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            await settings(context.getters.getServerUrl + '/api/editTask')
                .catch((error) => {
                    throw error;
                });
        },
        async editUser(context: any, polyload: any) {
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            await settings(context.getters.getServerUrl + '/api/editUser')
                .catch((error) => {
                    throw error;
                });
        },
        async deleteProject(context: any, polyload: any) {
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            await settings(context.getters.getServerUrl + '/api/deleteProject')
                .catch((error) => {
                    throw error;
                });
        },
        async deleteTask(context: any, polyload: any) {
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            await settings(context.getters.getServerUrl + '/api/deleteTask')
                .catch((error) => {
                    throw error;
                });
        },
        async deleteUser(context: any, polyload: any) {
```

```
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            await settings(context.getters.getServerUrl + '/api/deleteUser')
                .catch((error) => {
                    throw error;
                });
        },
        async addCompany(context: any, polyload: any) {
            const settings = axios.create({
                baseURL: '',
                method: 'post',
                headers: { 'Content-Type': 'application/json;charset=UTF-8' },
                data: polyload.data,
            });
            await settings(context.getters.getServerUrl + '/api/addCompany')
                .catch((error) => {
                    throw error;
                });
        },
    },
});
```

ДЕМОНСТАРЦІНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-додатку «Task manager» для постановки задач на проекті мовою TypeScript

Виконав студент 5 курсу
групи ППЗ-52
Гусак С.Р.
Керівник роботи
Шевченко С.М.

Київ – 2021

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Впровадження і використання Task Manager було корисним і актуальним раніше. Але в умовах карантину, масового переходу на формат віддаленої роботи, їх застосування стає життєво необхідним. Адже набагато зручніше ставити і коригувати завдання, контролювати їх виконання, керувати командою в єдиному інтерфейсі, ніж окремо спілкуватися з кожним учасником проекту.

- **Мета роботи** – забезпечення чіткого контролю графіку робітників компанії. Чітке уявлення про стан робочого процесу та процес розробки продукту як для робітників, так і для інших членів команди. Розподілення графіку з найменшими витратами часу.
- **Об'єкт дослідження** – програмне забезпечення для постановки задач на проектах.
- **Предмет дослідження** – інструмент для управління проектами, web-додаток, за допомогою якого можна в кілька кліків роздавати завдання всім своїм робітникам.

2

АНАЛОГИ

	Переваги	Недоліки
Asana	• відносна простота; • прикріплення файлів з хмарних сервісів.	• відсутність підтримки української та російської мов; • платне використання; • відсутність тайм-трекера.
Jira	• високий рівень кастомізації; • пріоритетність задач; • інтеграція зі сторонніми сервісами.	• платне використання; • складність використання із-за великої кількості функціоналу.
Basecamp 3	• наявність додатків для ПК, Android та IOS; • інтуітивно зрозумілий інтерфейс.	• дуже висока ціна для використання; • немає цілодобової підтримки; • відсутність підтримки української та російських мов.

3

ВИМОГИ ДОДАТКУ

- *Вимоги зовнішнього інтерфейсу:* крос-платформеність, крос-браузерність, адаптивність інтерфейсу.
- *Основні функціональні вимоги:* реєстрація, авторизація, відновлення паролю, реєстрація компанії, керування проектами, керування задачами, фільтрація списків та редагування профілю.
- *Основні нефункціональні вимоги:* usability, продуктивність та надійність.
- *Зовнішні вимоги:* безпека даних, конфіденційність.

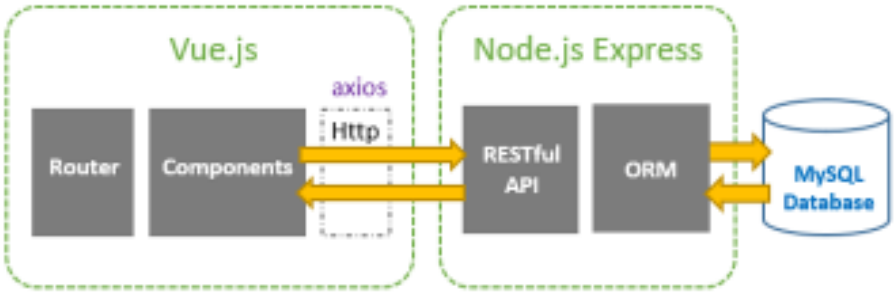
4

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

- *В якості редактору коду використано* Visual Studio Code.
- *Для вирішення інтерфейсних задач використана зв'язка* TypeScript + Vue.Js + SCSS + HTML;
- *В ролі серверної мови програмування використана* Node.js;
- *Для управління базою даних використана СУБД* MySQL.

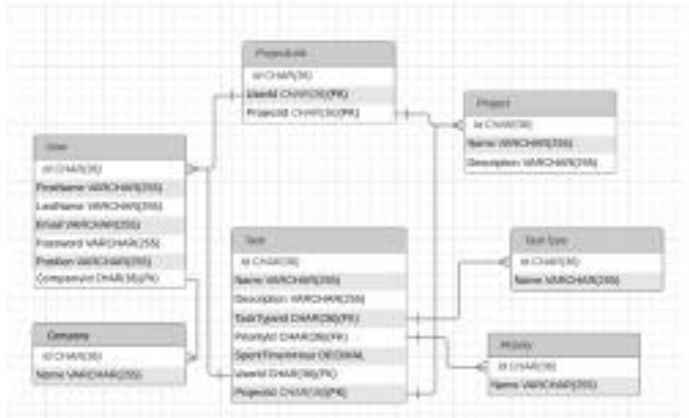
5

АРХІТЕКТУРА ДОДАТКУ



- Node.js Express експортує REST API та взаємодіє з базою даних MySQL за допомогою Sequelize ORM.
- Vue Client надсилає HTTP-запити та отримує HTTP-відповіді, використовуючи Axios.

6



Entity Relationship Diagram у web-додатку Task Manager в даній версії складається з 7 таблиць в базі даних.

7

ВИСНОВКИ

1. Для досягнення поставленої мети було розроблено web-додаток Task Manager, що є одним з вищеописаних інструментів. Він дуже простий у використанні для створення проектів та задач для кожного працівника компанії.
2. Одною з найголовніших переваг додатку є легкість. Не потрібно витрачати багато часу на те, щоб розібратись у функціоналі. Достатньо зареєструватись, створити потрібні проекти та прив'язати до них задачі, в яких, в свою чергу, поставити відповідального.
3. Шляхом зменшення часу на розподілення задач, робочий процес стає більш продуктивним. Економлячи на кожній задачі хоча б 10 хвилин, в результаті можна досягти вагомих успіхів.
4. Використання сучасних технологій при розробці, не тільки пришвидшило процес дослідження, але це ще й спрощує подальшу розробку, яка можлива після появи нових ідей, або після відгуку користувачів.

8