Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка програмного забезпечення керування проектами інформаційних системи»

Студента 2 курсу, 6-м групи, спеціальності 121 «Інженерія програмного забезпечення», спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Бородая Микити Андрійовича

Науковий керівник кандидат технічних наук, доцент _____ підпис керівника — Цензура Микола Олександрович

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Криворучко Олена Володимирівна

КИЇВ – 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Бородаю Микиті Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка програмного забезпечення керування проектами інформаційних системи»
Затверджена наказом ректора від "13" грудня 2019 р. № 4304
2. Строк здачі студентом закінченої проекту 01 грудня 2020
3. Цільова установка та вихідні дані до проекту
Мета проекту проаналізувати найбільш ефективні методи та засоби управління проектами інформаційних систем і розробити програмне рішення на основі результатів аналізу.
Об'єкт дослідження методи та засоби керування проектами інформаційних систем.
Предмет дослідження розробка настільного додатку керування проектами за допомогою Electron.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ДІЯЛЬНОСТІ КЕРУВАННЯ ПРОЕКТАМИ

1.1 ВИЗНАЧЕННЯ КЕРУВАННЯ ПРОЕКТАМИ

1.2 МЕТОДОЛОГІЇ КЕРУВАННЯ ПРОЕКТАМИ

1.3 ТЕХНІЧНЕ ЗАВДАННЯ ДО РОЗРОБКИ ВЕБ-ДОДАТКУ

1.4 ВИСНОВКИ ДО РОЗДІЛУ

РОЗДІЛ 2. АНАЛІЗ ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 РОЗРОБКА НАСТІЛЬНИХ ДОДАТКІВ ЗА ДОПОМОГОЮ ВЕБ-ІНСТРУМЕНТІВ

2.2 АРХІТЕКТУРА ELECTRON

2.3 ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ

2.4 ВИСНОВКИ ДО РОЗДІЛУ

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 ПРОЕКТУВАННЯ БАЗИ ДАНИХ

3.2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

3.3 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

3.4 ВИСНОВКИ ДО РОЗДІЛУ

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Дослідження діяльності керування проектами*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Аналіз інструментів розробки*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Програмна реалізація системи*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*08.12.2020-09.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту _____ Цензура М.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми _____ Криворучко О.В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент _____ Бородай М.А.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту

_____

(підпис, дата)

Відмітка про попередній захист _____Цензура М.О._____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента _____Бородай М.А._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Криворучко О. В._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження проект призначений аналізу методів та засобів управління проектами інформаційних систем та розробці настільного додатку за допомогою фреймворку Electron.

В результаті аналізу визначено вагомі переваги та недоліки деяких методів управління проектами, на основі на аналізу обрано найбільш ефективні методи для управління проектами інформаційних систем.

Розробка серверної частини складається з проектування архітектури та створення доступного API-серверу за допомогою бібліотеки Express. Для розробки клієнтської частини застосовано фреймворк Electron та бібліотека для розробки інтерфейсів React. Проектування бази даних виконано за вказівкам з технічного завдання.

Ключові слова: Electron, настільний додаток, керування проектами.

ABSTRACT

According to the purpose of research project is intended for the analysis of methods and means of project management of information systems and development of the desktop application with Electron framework.

As a result of the analysis defied weighty advantages and disadvantages of project management methods.

The development of the server part consists of designing the architecture and creating an accessible API server using Express library. Electron framework and React library are used for client development. Database is designed according to the instructions in the specification.

Keywords: Electron, desktop application, project management,

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

СУБД – систему управління базами даних.

JSON – текстовий формат обміну даними.

ЗМІСТ

ВСТУП

Система управління проектами має важливу цінність в нинішньому світі розробки інформаційних систем. За допомогою даного рішення робота над проектом розподіляється на завдання для виконання певних цілей. Для кожного завдання виділяється кількість ресурсів які залежать від пріоритету і термінів здійснення.

В даний час системою управління проектами користуються виключно всі відомі організації і компанії які мають відділи по розробці, або на пряму спеціалізуються в даній області. Таким чином плануючи розробку цілого проекту, можна заощадити ресурси, що є одним з головних чинників в нинішньому світі розробки програмного забезпечення.

У світі існує багато різних готових рішень систем управління проектами, більшість з яких включають в себе ефективні методи управління проектами, такі як Agile, Scrum, Kanban та інші.

Одні з найбільш розвинених таких програмних рішень – Jira, Notion, Trello.

Актуальність. Застосування систем управління проектами інформаційних систем є однією з найбільш головних частин в розробці програмного забезпечення. Як великі корпорації, так і маленькі стартапи використовують дане рішення для організації і планування своїх проектів.

Об’єктом дослідження є методи та засоби керування проектами інформаційних систем.

Предметом дослідження є розробка настільного додатку керування проектами за допомогою Electron.

Метою випускного кваліфікаційного проекту є проаналізувати найбільш ефективні методи та засоби управління проектами інформаційних систем і розробити програмне рішення на основі результатів аналізу.

Завдання дослідження:

- Визначити вимоги та рекомендації до виконання проекту;
- Проаналізувати методи та засоби управління проектами інформаційних систем;
- Розглянути готові програмні рішення систем керування проектами;
- Спроектувати та розробити серверне забезпечення;
- Спроектувати таблиці бази даних;
- Спроектувати та розробити клієнтську частину.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ДІЯЛЬНОСТІ КЕРУВАННЯ ПРОЕКТАМИ

1.1 Визначення керування проектами

Проект – це певний процес для досягнення певних цілей і вирішення конкретного бізнес-завдання.

Отже, управління проектами – це діяльність, спрямована на досягнення поставлених завдань, реалізацію певних планів, використовуючи наявні ресурси – час, капітал, людей.

В основі управління проектами лежить планування – короткострокове або на більш тривалий період. У бізнес-процесах планування ґрунтується на певних методиках планування: в залежності від пріоритету завдань і термінів їх виконання.

Управління проектами - це вирішення ряду невеликих окремих завдань на різних етапах проекту. Шляхом вирішення більш дрібних дій можна наближатися до поставленої мети.

Тобто, управління проектами – це постійний перехід від простого до складного, і трансформація одного великого завдання в більш прості заходи, що складаються з шаблонних процедур. Головне - це закріпити окремого виконавця для вирішення кожного невеликого завдання, який повинен виконати цю окрему дію за конкретний проміжок часу.

Разом, можна виділити ряд певних ознак проекту, які відрізняють його від інших видів діяльності:

- будь-який проект спрямований на досягнення конкретних цілей;

- проект включає в себе координоване виконання взаємопов'язаних дій;
- проект має обмежену протяжність у часі, з певним початком і кінцем;
- кожен проект певною мірою неповторний і унікальний.

Що включає в себе управління проектами

У складну на перший погляд систему управління проектами входить ряд послідовних дій:

- визначення і формування вимог до проекту;
- постановка максимально чітких і зрозумілих цілей;
- установка і реалізація комунікації між задіяними в проекті сторонами;
- урівноваження проектних обмежень: зокрема бюджету, ресурсів, ризиків, дедлайнів, якості;
- спілкування з командою, облік їх потреб / побажань / очікувань і корекція існуючих планів відповідно до отриманим матеріалом.

Всі ці дії розподіляються на окремі процеси рис.1.1. (Процеси керування проектами): ініціація проекту, планування, виконання та контроль, завершення. Саме ретельне планування, організація завдань і проектних складових, забезпечення необхідними ресурсами і контроль дієвості обраної стратегії – це і є те головне, що входить в управління проектами з перспективою досягнення поставленої мети.

- Ініціація (тобто старт проекту) є знайомство з проектом. Визначається його суть і цілі, формується відповідна під нього команда.
- Планування – найважливіша частина в управлінні проектом. Складність в тому, що під час цього етапу ретельно прописуються

всі дії, які повинна здійснити команда для досягнення заданої мети. Для цього проект спочатку розділяється на частини і набір дрібних завдань. Створюється певний «графік робіт», в якому прописуються дедлайни для кожної з задач. Також опрацьовується список необхідних ресурсів. При цьому планування включає в себе періодичне коригування, адже в процесі роботи постійно з'являються нові нюанси і під задачі;

- Виконання і контроль. Цей етап слід чергувати з попереднім. В ідеальній системі управління проектом все виглядає так: поставили завдання, виконали, проконтролювали, внесли в план необхідні корективи, поставили таку задачу і так далі.
- Завершення проекту. На цьому етапі робиться контрольна перевірка виконаної роботи і обов'язково зберігаються вихідні дані, задіяні інструкції і регламенти. Це потрібно для того, щоб навіть нова людина в команді змогла розібратися, що і як робили до нього.

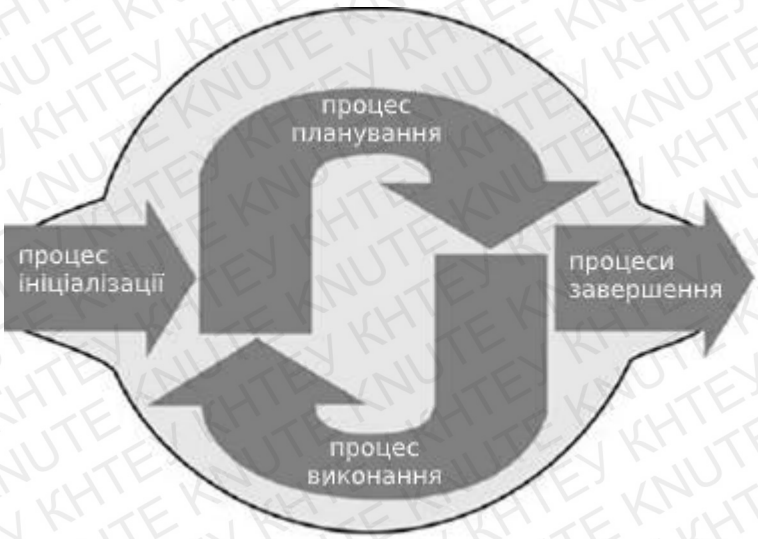

Рис. 1.1. Процеси керування проектами

1.2 Методології керування проектами

Проект – це унікальний захід, що не піддається стандартизації. Однак процеси управління проектами піддаються стандартизації і документи, які формалізують ці процеси, отримали назву методології управління проектами. Причому деякі методології управління проектами застосовні для всіх типів проектів в різних областях. Інші ж, навпаки, підходять тільки для управління конкретними типами проекту. Так, для сфери дорожнього будівництва найбільш придатною буде одна проектна методологія, в той час як для проекту з розробки програмного забезпечення – інша.

Методологія Waterfall – сама «стара» з усіх. Вперше вона була викладена американським вченим в галузі інформатики Уінстоном Уокером Ройсом в 1970 році у відповідь на потребу управління все більш ускладняються процесом розробки програмного забезпечення. З тих пір вона набула широкого поширення, особливо в сфері програмного забезпечення.

Каскадна модель характеризується послідовністю. Крім цього, вона в значній мірі орієнтована на вимоги.

Методологія Waterfall ділиться на три окремих етапи. Спочатку необхідно зібрати і проаналізувати вимоги, потім розробити рішення (і підхід), впровадити рішення і виправити проблеми, якщо вони з'явилися.

Кожен етап цього процесу автономний. Щоб перейти до наступного, необхідно завершити попередній етап.

Agile – це ще одна методологія управління с акцентом на розробці програмного забезпечення. З'явилася вона як результат незастосовності методології Waterfall в рамках складних проектів.

Хоча ідеї, властиві Agile, вже давно використовуються в сфері розробки ПЗ, формально методологія з'явилася лише в 2001 році, коли кілька представників з IT випустили Agile-маніфест.

Agile повністю протилежна методології Waterfall за підходом і ідеології. Сама назва з англійської мови перекладається як «Гнучкий», а це значить, що в управлінні використовується швидкий і гнучкий підхід.

Методологія швидше характеризується невеликими циклічними змінами, які впроваджують у відповідь на зміну вимог.

Scrum - це не повнофункціональна методологія управління проектами. Це скоріше підхід до методології Agile з акцентом на командах проекту, спринтах і щоденних зборах.

Незважаючи на те що Scrum запозичує принципи і процеси з Agile, до цього підходу властиві свої методи і тактики управління проектами.

В рамках підходу Scrum в центрі проекту - команда. Тому передбачається, що команда характеризується самоорганізацією і самоврядуванням.

1.3 Технічне завдання до розробки веб-додатку

1. Загальні вимоги

Необхідно створити простий та зрозумілий за інтерфейсом настільний додаток, який надає можливості керування проектами інформаційних систем.

1.1 Вимоги до інструментів розробки

1.1.1. Інструменти для розробки серверної частини

- мова програмування TypeScript;
- програмна платформа Node та фреймворк Express;
- бібліотека для взаємодії з базою даних TypeORM.

1.1.2. . Інструменти для розробки клієнтської частини

- мова програмування TypeScript;
- бібліотека для розробки інтерфейсів React;
- біблітека стилізації styled-components;
- webpack – інструмент для складання модульних файлів JavaScript;
- babel – компілятор сучасного JavaScript-коду для підтримки старих браузерів.

2. Вимоги до проектування таблиць бази даних.

2.1. Таблиця проектів.

- name – назва;
- url – посилання;
- description – опис;
- category – категорія.

2.2. Таблиця користувачів.

- name – назва;
- email – електронна пошта;
- avatarUrl – посилання на зображення.

2.3. Таблиця задач проекту.

- title – назва;
- type – тип;
- status – статус виконання;
- priority – пріоритет;
- description – опис;
- estimate – термін виконання.

2.4. Таблиця коментарів задач.

- body – зміст.

2. Вимоги до розробки серверного забезпечення.

2.1. Вимоги до структури.

Серверне забезпечення має дотримуватися послідовної структури, а саме:

- models – моделі бази даних;
- controllers – функції які використовуються в роутері;
- routes – функції для обробки REST-запитів.

2.2. Вимоги до реалізації функціональності авторизації.

- застосування криптографічного шифрування паролю користувачів;
- генерація ключу доступу для валідації сесії на клієнті;
- ключ доступу має включати в себе мета-дані.

2.3. Вимоги до захисту обміну даних.

Для організації обміну даних між сервером та браузером, повинен використовуватися протокол HTTP та його розширення HTTPS для безпечного обміну даних.

3. Вимоги до розробки клієнтської частини.

3.1. Вимоги до оформлення сторінок.

Кожна сторінка, включає в себе заголовок, опис і найголовніше певний функціонал.

3.1.1. Вимоги до змісту сторінок створення задач.

Форма заповнення з наступними полями:

- поле назви;
- поле опису;
- поле пріоритету;
- поле призначення персоналу;
- поле термінів.

3.2. Вимоги до оформлення інтерфейсу.

Візуалізація повинна забезпечувати зручний для користувача інтерфейс, який відповідає наступним вимогам:

- наявність локалізованого (україномовного) інтерфейсу користувача;
- меню навігації.

1.4 Висновки до розділу

Досліджено діяльність управління проектами, а також методології для ефективного планування та виконання завдань проекту.

Проаналізовано відомі методології управління проектами, такі як Agile, Scrum і Kanban. Кожна методології є ефективною, але для цього потрібно враховувати структуру команди і проекту.

Складено технічне завдання по розробці програмного рішення даного проекту, вказані важливі аспекти щодо структури серверної і клієнтської частини.

РОЗДІЛ 2. АНАЛІЗ ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Розробка настільних додатків за допомогою веб-інструментів

Веб-технології розвиваються дуже швидко, настільки швидко, що навіть для розробки настільних додатків використовують мову програмування JavaScript.

Головною невід'ємною частиною розробки програмного забезпечення є хостинг для проектів заснований на системі контролю версій – GitHub, за допомогою якого і поширюється модель з відкритим вихідним кодом. Крім того, організація GitHub розробила JavaScript фреймворк Electron, який дозволяє розробляти рідні графічні додатки для настільних операційних систем використовуючи веб-технології. Фреймворк включає в себе Node та бібліотеку Chromium.

Electron дає можливість використовувати вже знайомі веб-технології, замість того, щоб вивчати нову мову програмування, або графічний фреймворк. Таким чином кількість розробників настільних додатків різко зросли.

Розробка на Electron має таку ж структуру як і створення веб-сайтів, а саме наявність back-end серверу, бази-даних, і графічної оболонки.

Back-end сервер використовується для виконання складних завдань, обробки великої кількості даних, а також для виконання запитів на зчитування і запису інформації до бази даних, на цьому можливості не обмежуються. Реалізована підтримка виконання запитів до інших

серверів, наприклад до API-сервера або хмарного сховища.

Структура і взаємодія з базами даних нічим не відрізняється від звичайних настільних додатків, за винятком того, що при веб-розробці в зв'язці з NodeJS часто використовують документ орієнтовану СУБД MongoDB. Така висока популярність описується вже знайомим терміном для веб-розробників JSON. Така СУБД класифікується як NoSQL і використовує JSON-подібні документи і схеми бази даних. Однак для більш серйозних проектів, в яких є багато пов'язаних між собою даних, має сенс використовувати реляційну базу даних, наприклад PostgreSQL або MySQL.

Для створення графічного інтерфейсу, все набагато простіше, так як не потрібно вивчати ніяких нових графічних фреймворків. Для відтворення програми використовується браузерна технологія Chromium, що дає нам можливість створювати інтерфейси за допомогою мови розмітки HTML і таблиці стилів CSS. Так само не варто забувати, що Chromium використовує JavaScript движок V8, який постійно оновлюється, надаючи новий функціонал до мови програмування JavaScript.

Для клієнтської розробки є можливість застосовувати будь-які бібліотеки і фреймворки, починаючи від допоміжних бібліотек типу Babel, закінчуючи улюбленими бібліотеками для розробки інтерфейсів, таких як React.

ElectronJS має свою технологію виконання коду під час розробки зі змінами у реальному часі. Також надає можливість бандлінгу всіх наших пакетів у виконувані файли для операційних систем Windows, Linux та MacOS.

2.2 Архітектура Electron

У Electron підтримується декілька процесів, один з яких запускає скрипт «main» – є основним процесом. Скрипт, який виконується у основному процесі може відображати GUI шляхом створення веб-сторінок. У додатках Electron завжди є один головний процес, але не більше.

Так як Electron використовує Chromium для відображення веб-сторінок, то використовується мульти-процесорна архітектура Chromium. Кожна веб-сторінка Electron виконується у власному процесі, які називають процесами візуалізації.

У браузерах, веб-сторінки зазвичай виконуються в ізольованому середовищі і їм не відкривається доступ до рідних ресурсів. Проте користувачі Electron, мають право використовувати API Node на веб-сторінках, тим самим дозволяючи взаємодіяти на низькому рівні операційної системи.

Відмінності між основними процесами і процесами візуалізації

Основний процес створює веб-сторінки шляхом створення екземплярів BrowserWindow. Кожен екземпляр BrowserWindow запускає веб-сторінку у процесі візуалізації. Коли цей екземпляр знищується, відповідний процес візуалізації також припиняється.

Основний процес керує всіма веб-сторінками та процесами їх візуалізації. Кожен процес візуалізації ізольований і піклується лише про активну веб-сторінку.

Виклик DOM API на веб-сторінках заборонений, оскільки цей процес є досить небезпечний і може викликати витік ресурсів. Якщо необхідно виконувати DOM операції на веб-сторінці, процес візуалізації

веб-сторінки повинен спілкуватися з основним процесом для запиту виконання цих операцій.

Використання API Electron

Electron має доступний ряд API-інтерфейсів, що підтримують розробку настільних додатків у обох процесах – основному процесі і процесі візуалізації.

Всі API-інтерфейси призначаються типам процесів. Багато з них можуть використовуватися лише з основного процесу, деякі тільки з процесу візуалізації, а деякі – з обох процесів.

Наприклад, якщо вікно у Electron створюється з використанням класу BrowserWindow, то він доступний тільки у основному процесі.

Electron має модуль remote, який надає можливість викликати основний процес для виконання завдань.

Використання API Node.js

Electron надає доступ до Node з обох типів процесів. Це має два важливих наслідки:

1. Усі API, доступні в Node також доступні і у Electron.
2. Node модулі можна використовувати разом з Electron. В даний час NPM пропонує найбільше у світі сховище бібліотек – можливість використовувати добре підтримувані і протестовані рішення, які раніше застосовувалися лише на серверних додатках.

Існує одна важлива обмовка: рідні Node модулі (тобто модулі, які вимагають компіляції власного коду, перш ніж вони можуть бути використані) повинні бути скомпільовані для використання з Electron.

Макети і CLI

Розробка в Electron різноманітна – немає одного істинного шляху; для розробки, складання, упаковки або випуску програми. Додаткові особливості для Electron, як для збірки, так і для часу виконання, зазвичай можна знайти в NPM в окремих пакетах, що дозволяє розробникам як створити додаток, так і побудувати конвеєр, в якому вони потребують.

Цей рівень модульності і розширюваності гарантує, що всі розробники, які працюють з Electron, як великі, так і малі за розміром групи, ніколи не обмежені тим, що вони можуть або не можуть робити в будь-який час протягом свого життєвого циклу розробки. Проте, для багатьох розробників один з керованих спільнотою шаблонів або інструментів командного рядка може значно спростити компіляцію, упаковку і випуск програми.

Шаблон є тільки відправною точкою - це, так би мовити, полотно, з якого створюється свій додаток. Вони зазвичай приходять у вигляді сховища, можна клонувати і налаштовувати контент.

З іншого боку, інструмент командного рядка продовжує підтримувати вас протягом всієї розробки і випуску. Вони більш корисні і підтримувані, але забезпечують дотримання рекомендацій про те, як код повинен бути структурований і побудований.

Electron Forge об’єднує існуючі (і добре підтримувані) інструменти збірки в єдиний пакет. Forge поставляється з готовим до використання шаблоном, використовуючи Webpack як бандлер. Включає в себе приклад конфігурації TypeScript і надає два конфігураційних файли. Він використовує ті ж основні модулі, які використовуються великим співтовариством Electron (наприклад, electron-packager). Зміни, що вносяться замовниками Electron (наприклад, Slack), також приносять користь користувачам Forge:

- electron-builder – повноцінне рішення для упаковки і створення готового до користування послугою Electron, яке фокусується на інтегрованому досвіді.
- electron-builder додає одну єдину залежність, орієнтовану на простоту, і управляє всіма додатковими вимогами зсередини. Electron-builder замінює функції і модулі, які використовуються Electron (такими як автоматичне оновлення) до призначених для користувача.
- electron-react-boilerplate - шаблон для застосування Electron в зв'язці з бібліотекою React, використовує вбудований electron-builder.

Налагодження додатків

Всякий раз, коли додаток Electron не поводиться так, як потрібно, інструментарій засобів налагодження може допомогти знайти помилки у кодуванні, вузькі місця продуктивності або можливості оптимізації.

Найбільш повним інструментом для налагодження окремих процесів є набір інструментів розробника Chromium. Він доступний для всіх процесів, включаючи екземпляри таких об'єктів як BrowserWindow, BrowserView, і WebView.

Налагодження основного процесу трохи складніше, оскільки не можливо відкрити для нього інструменти розробника. Інструменти Chromium Developer Tools можуть використовуватися для налагодження основного процесу Electron завдяки співпраці між Chromium і Node.

Щоб поширювати додаток з Electron, потрібно упакувати і зробити його ребрендинг. Найпростіший спосіб зробити це – використовувати один з наступних інструментів упаковки:

- electron-forge
- electron-builder
- electron-packager.

Ці інструменти подбають про всі кроки, які потрібно зробити для того, щоб в кінцевому підсумку отримати готовий до поширення додаток Electron, такі як упаковка додатку, ребрендинг файлу що виконується, установка правильних значків і створення інсталяторів.

Безпека додатку розробленого на Electron

Коли ви працюєте в середовищі Electron, важливо розуміти, що це не веб-браузер.

Electron дозволяє створювати функціонально розвинені додатки для настільних комп'ютерів за допомогою веб-технологій. JavaScript має доступ до файлової системи, та призначена для створення скриптів. Завдяки цим можливостям можна створювати високоякісні рідні додатки, але це так само множить ризики збільшення разом з додатками повноважень коду.

Показ довільного змісту від недостовірних джерел тягне за собою ризики безпеки, які Electron не призначений для виправлення. До відома, найбільш популярні додатки Electron (Atom, Slack, Visual Studio Code, etc) призначені для локальної роботи (або з довіреними віддаленими вузлами, без Node інтеграції). Якщо додаток виконує код з онлайн джерел, обов'язково потрібно забезпечити безпеку коду.

Процес виконання додатку

1. Клієнт запускає додаток Electron, як правило, з бажаної платформи, наприклад Ubuntu на базі Windows / MacOS / Linux.
2. Додаток робить запит у вікно, використовуючи основний (Main) процес.
3. Main Process робить запит на запуск вікна разом з renderer.js (Renderer process).

4. Вікно входу буде запущено і приєднано до основного процесу, доступного для операцій.
5. Користувач надає інформацію про початок введення інформації, шляхом обробки натискання кнопки у процесі візуалізації.
6. Основний процес отримає відповідну інформацію через слухача подій та запустить
7. Вікно додатку відкривається на основі запиту за допомогою процесу Renderer.
8. Цей цикл продовжується для всієї програми до тих пір, поки не буде вирішена певна умова використання.

2.3 Інструменти для розробки серверної та клієнтської частини

Бібліотека React – використовується для розробки простих і складних інтерфейсів, впроваджує віртуальне DOM-дерево (Virtual DOM), яке порівнює елементи з поточним DOM-деревом і якщо є відмінності між елементами, відповідно оновлює їх. Таким чином зменшуючи навантаження на браузер користувача.

Node – JavaScript платформа заснована на основі рушію V8. Часто використовується веб-розробниками у зв'язку з простотою і вже знайомими принципами мови програмування JavaScript. Має переваги, такі як запуск сервера на різних операційних системах, велику підтримку з боку спільноти.

Найчастіше в розробці серверних програмних рішень у зв'язці з Node використовується фреймворк Express. Переваги використання даного фреймворку, це легка і зрозуміла організація структури коду, можливість обробки шляхів (routing). Таким чином серверне програмне

забезпечення виглядає як API-сервіс, до якого виконуються запити, обробляються і повертають результат до браузера користувача.

PostgreSQL – надійна об'єктно-реляційна система управління базами даних, має величезний вибір між типами даних і функціями. Дає можливість використовувати JSON-дані в таблицях, створювати двомірні масиви та нові типи даних.

2.4 Висновки до розділу

В даному розділі було розглянуто створення настільних додатків за допомогою фреймворку Electron, який надає можливість використовувати веб-інструменти для реалізації клієнт частини.

Проаналізовано архітектуру і роботу фреймворку Electron, головна перевага даного фреймворку – це JavaScript двигун V8, який є самим оптимізованим і включає в себе всі останні можливості по специфікації EcmaScript.

Для розробки серверної частини була вибрана мова програмування JavaScript і платформа Node. Серверна частина в даному проекті відіграє роль API сервера, який обробляє запити з клієнта, для цього існує фреймворк Express, з яким розробка подібних серверів стає набагато простіше.

Система управління базами даних PostgreSQL є одним з найбільш підходящих рішень для даного проекту, так як включає в себе набір функцій для роботи з JSON. Для взаємодії СУБД з сервером була обрана бібліотеки pg і TypeORM - надає можливість типізувати моделі за допомогою мови програмування TypeScript.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Проектування бази даних

Для проектування бази даних потрібно враховувати зазначені вказівки в технічному завданні.

Виходячи з технічного завдання потрібні наступні таблиці, рис. 3.1. (Фізична модель БД):

- User – таблиця користувачів, має зв'язок багато-до-одного з таблицею «Project», один-до-багатьох з таблицею «Comment» і багато-до-багатьох з таблицею «Issue»;
- Project – таблиця проектів, має зв'язок один-до-багатьох з таблицями «User» і «Issue»;
- Issue – таблиця завдань проекту, має зв'язок багато-до-одного з таблицею «Project» і зв'язок багато-до-багатьох з таблицею «User»;
- Comment – таблиця коментарів, має зв'язок багато-до-одного з таблицею «User» і «Issue».

Процес ініціалізації та створення бази даних виконується в допоміжному інструменті для PostgreSQL – pgAdmin v4. Або в разі впровадження бази даних на хостинг можна скористатися командним рядком для взаємодії з базою даних. Для цього потрібно зайти в папку з встановленим PostgreSQL і запустити psql, який запросить пароль користувача СУБД.

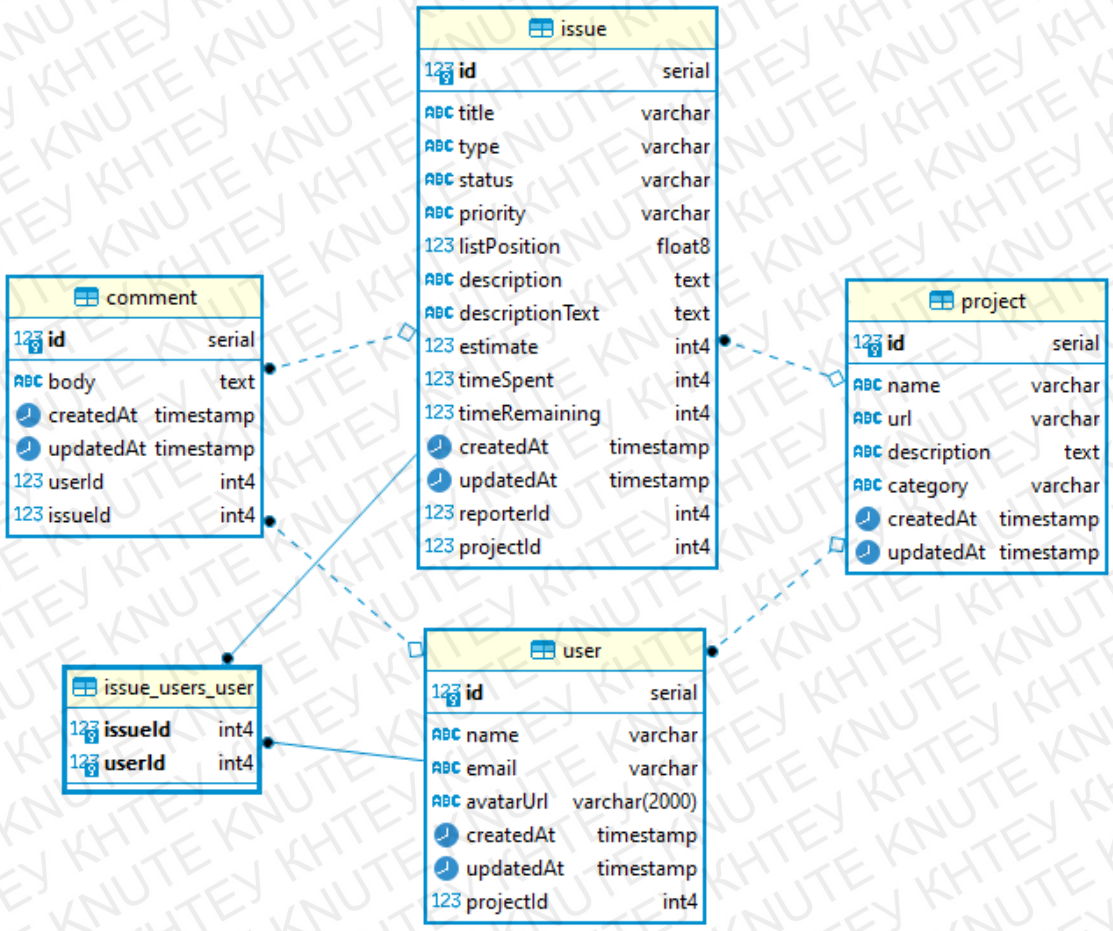

Рис. 3.1. Фізична модель БД

3.2 Розробка серверної частини

Для розробки серверної частини використовується мова програмування TypeScript, платформа Node і фреймворк Express. Головним завданням є створити швидке забезпечення для отримання і обробки запитів з клієнта.

Для взаємодії з базою даних використовується бібліотека pg і TypeORM.

Для ініціалізації проекту потрібно завантажити і встановити платформу Node з офіційного сайту, бажано встановлювати LTS версію, так як останні не завжди правильно працюють з деякими бібліотеками.

Наступним етапом є встановлення всіх необхідних бібліотек і фреймворк Express за допомогою пакетного менеджера npm, який встановлюється разом з Node.

Установка бібліотек зазвичай починається з допоміжних бібліотек для роботи над проектом, а саме:

- ts-node і typescript - для роботи з TypeScript на платформі Node;
- eslint - для підсвічування синтаксису;
- prettier - для форматування коду;
- nodemon - для безперервної роботи з сервером на стадії розробки;
- cross-env - для виконання npm скриптів в будь-якому середовищі розробки.

З основних бібліотек:

- express - простий і гнучкий фреймворк для розробки веб та мобільних додатків, або ж звичайних API-серверів;
- pg - драйвер для взаємодії з СУБД PostgreSQL;
- typeorm - бібліотека для взаємодії з базою даних, використовується в зв'язці з TypeScript для написання типів моделей;
- jsonwebtoken - для генерації захищеного токена авторизації користувача.

Всі встановлені бібліотеки додаються в файл package.json на рис. 3.2. (Файл package.json – список встановлених бібліотек), який містить інформацію про проект, скрипти виконання і два розділи під бібліотеки:

- dependencies – містять бібліотеки які компілюються в основний код;
- devDependencies – допоміжні бібліотеки, які використовуються тільки на стадії розробки.

```
"dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "jsonwebtoken": "^8.5.1",
    "pg": "^8.5.1",
    "typeorm": "^0.2.29"
},
"devDependencies": {
    "@types/cors": "^2.8.6",
    "@types/express": "^4.17.2",
    "@types/jsonwebtoken": "^8.3.5",
    "@types/node": "^12.12.11",
    "@typescript-eslint/eslint-plugin": "^2.7.0",
    "@typescript-eslint/parser": "^2.7.0",
    "cross-env": "^6.0.3",
    "eslint": "^6.1.0",
    "eslint-config-airbnb-base": "^14.0.0",
    "eslint-config-prettier": "^6.7.0",
    "eslint-plugin-import": "^2.18.2",
    "eslint-plugin-prettier": "^3.1.1",
    "nodemon": "^2.0.0",
    "prettier": "^1.19.1",
    "ts-node": "^8.5.2",
    "tsconfig-paths": "^3.9.0",
    "typescript": "^3.7.2"
},
```

Рис. 3.2. Файл package.json – список встановлених бібліотек

Застосовуючи встановлені бібліотеки створюється вхідний файл в якому відбувається ініціалізації серверу та підключення до бази даних на рис. 3.3. (Файл «index.ts»).

```
import express from 'express';
import cors from 'cors';
import { handleError } from 'middleware/errors';
import { RouteNotFoundError } from 'errors';
import createDatabaseConnection from 'database/createConnection';

const establishDatabaseConnection = async (): Promise<void> => {
  try {
    await createDatabaseConnection();
  } catch (error) {
    console.error(error);
  }
};

const initializeExpress = (): void => {
  const app = express();

  app.use(cors());
  app.use(express.json());
  app.use(express.urlencoded());

  app.use((req, _res, next) => next(new RouteNotFoundError(req.originalUrl)));
  app.use(handleError);

  app.listen(process.env.PORT || 3000);
};

const initializeApp = async (): Promise<void> => {
  await establishDatabaseConnection();
  initializeExpress();
};

initializeApp();
```

Рис. 3.3. Файл «index.ts»

Для створення підключення до бази даних використовується функція «createConnection» бібліотеки TypeORM (див. додаток А), яка приймає наступний об'єкт конфігурації:

- type – тип СУБД;
- host – хост на якому розташована СУБД;
- port – порт СУБД;
- username і password - ім'я та пароль користувача СУБД;
- database – назва створеної бази даних;
- entities – моделі таблиць.

Дану функцію бажано винести в окремий файл і передати об'єкт конфігурації за допомогою бібліотеки dotenv на рис. 3.4. (Код функції створення підключення до БД «createDatabaseConnection.ts»

```
import { createConnection, Connection } from 'typeorm';
import * as entities from 'entities';

const createDatabaseConnection = (): Promise<Connection> => {
  return createConnection({
    type: 'postgres',
    host: process.env.DB_HOST,
    port: Number(process.env.DB_PORT),
    username: process.env.DB_USERNAME,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_DATABASE,
    entities: Object.values(entities),
    synchronize: true,
  });
}

export default createDatabaseConnection;
```

Рис. 3.4. Код функції створення підключення до БД «createDatabaseConnection.ts»

Entities – це моделі таблиць які реалізують колонки, зв'язку та валідацію вхідних даних за допомогою TypeORM. При створенні даних моделей потрібно визначати колонки і їх типи даних, які описані в розділі проектування бази даних.

З огляду на вказівки в технічному завданні потрібно створити чотири моделі «User», «Project», «Issue» і «Comment», враховуючи їх зв'язки і типи полів, приклад на рис. 3.5. (Код моделі проекту «Project.ts»).

```
@Entity()
class Project extends BaseEntity {
  static validations = {
    name: [is.required(), is.maxLength(100)],
    url: is.url(),
    category: [is.required(), is.oneOf(Object.values(ProjectCategory))],
  };

  @PrimaryGeneratedColumn()
  id: number;

  @Column('varchar')
  name: string;

  @Column('varchar', { nullable: true })
  url: string | null;

  @Column('text', { nullable: true })
  description: string | null;

  @Column('varchar')
  category: ProjectCategory;

  @CreateDateColumn({ type: 'timestamp' })
  createdAt: Date;

  @UpdateDateColumn({ type: 'timestamp' })
  updatedAt: Date;

  @OneToMany(
    () => Issue,
    issue => issue.project,
  )
  issues: Issue[];

  @OneToMany(
    () => User,
    user => user.project,
  )
  users: User[];
}
```

Рис. 3.5. Код моделі проекту «Project.ts»

За допомогою TypeScript декораторів та TypeORM були створені такі поля:

- id – поле номера яке може бути тільки числом, має первинний ключ;
- name – строкове поле назви;

- url – строкове поле посилання, «nullable» позначає що це поле є не об'язковим;
- description – текстове поле опису;
- category – строкове поле категорії;
- createdAt – поле з відміткою дати створення;
- updatedAt – поле з відміткою дати оновлення;

За допомогою декоратора «@OneToMany» створено зв'язок один-к-багатьом з таблицями «Issue» і «User».

Створивши всі потрібні частини для взаємодії з базами даних залишилося розробити обробники запитів і контролери. Обробники запитів потрібні для взаємодії клієнта з сервером, а контролери в свою чергу виконують певні функції для отримання даних і відправки результатів на клієнт.

Контролери є асинхронними функціями, подають запит на інформацію з бази даних, і тільки після отримання цієї інформації відправляється результат на клієнт, приклад на рис. 3.6. (Код контролеру коментарів «Comment.ts»).

```
export const create = catchErrors(async (req, res) => {
  const comment = await createEntity(Comment, req.body);
  res.respond({ comment });
});

export const update = catchErrors(async (req, res) => {
  const comment = await updateEntity(Comment, req.params.commentId, req.body);
  res.respond({ comment });
});

export const remove = catchErrors(async (req, res) => {
  const comment = await deleteEntity(Comment, req.params.commentId);
  res.respond({ comment });
});
```

Рис.3.6. Код контролеру коментарів «Comment.ts»

Дані функції виконуються в обробниках запитів і приймають два параметра «request» і «response».

- request – містить інформацію зазначену клієнтів при виконанні запиту;
- response – містить набір методом для відправки результат на клієнт.

Для створення обробників запитів використовуються методи бібліотеки Express:

- get – для обробки GET запитів на відправку даних;
- post – для обробки POST запитів на отримання даних;
- put – для обробки PUT запитів на оновлення даних;
- delete – для обробки DELETE запитів на видалення даних.

Виходячи з вимог в даному проекті потрібно створити обробники запитів для проектів, завдань, коментарів і авторизації, приклад на рис. 3.7. (Код обробників запитів коментарів «Comment.ts»).

```
app.post('/comments', comments.create);
app.put('/comments/:commentId', comments.update);
app.delete('/comments/:commentId', comments.remove);

app.get('/issues', issues.getProjectIssues);
app.get('/issues/:issueId', issues.getIssueWithUsersAndComments);
app.post('/issues', issues.create);
app.put('/issues/:issueId', issues.update);
app.delete('/issues/:issueId', issues.remove);

app.get('/project', projects.getProjectWithUsersAndIssues);
app.put('/project', projects.update);

app.get('/currentUser', users.getCurrentUser);
```

Рис. 3.7. Код обробників запитів коментарів «Comment.ts»

Другим параметром даної функції є сам контролер, який спрацьовує якщо виконується запит до одного з оброблювачів.

3.3 Розробка клієнтської частини

Для початку роботи над додатком необхідно встановити фреймворк Electron. Після установки даного фреймворку створюється вхідний файл «app.js» (див. додаток Б), який містить надбудови для запуску веб-додатків у вигляді настільних додатків. Цього достатньо для роботи настільного додатку в середовищі розробки.

Наступним етапом є розробка інтерфейсу за допомогою бібліотеки React. Для початку необхідно створити вхідний файл рис. 3.8. (Вхідний файл «index.jsx») для ініціалізації додатку, який містить розбиту на компоненти реалізацію всього інтерфейсу.

```
const App = () => (
  <Fragment>
    <NormalizeStyles />
    <BaseStyles />
    <Toast />
    <Routes />
  </Fragment>
);

ReactDOM.render(<App />, document.getElementById('root'));
```

Рис. 3.8. Вхідний файл «index.jsx»

Після ініціалізації програми можна перейти до проектування та розробки головної сторінки проекту рис. 3.9. (Інтерфейс сторінки проекту), яка містить інформацію про проект і можливість взаємодіяти з завданнями проекту.

При завантаженні даної сторінки створюється запит на сервер для отримання інформації та завдань вказаного проекту (див. додаток В).

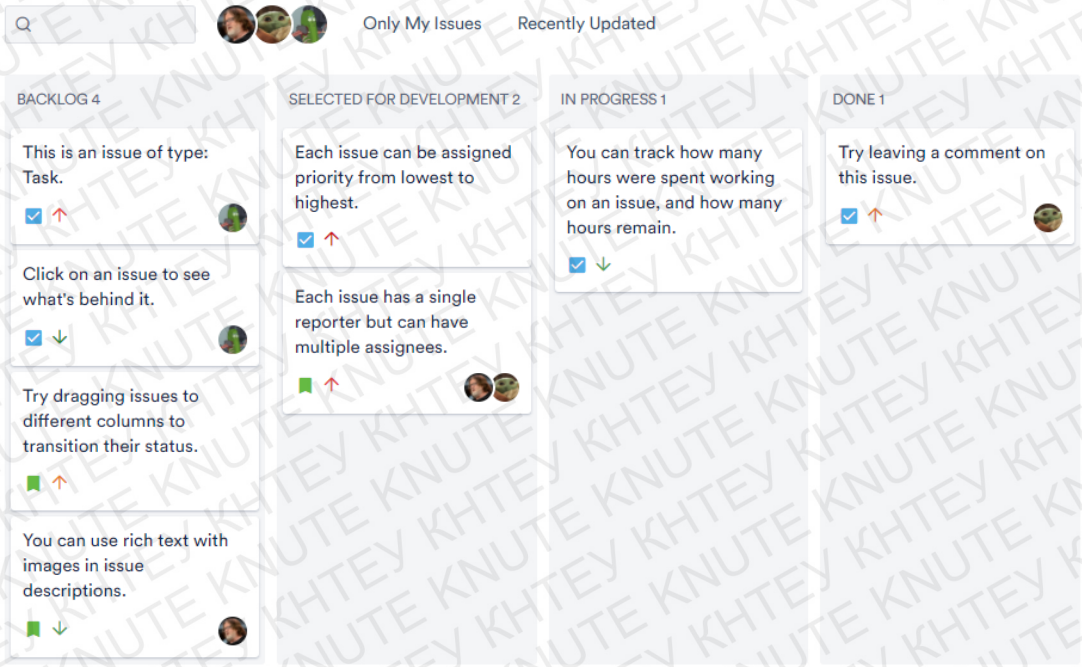

Рис. 3.9. Інтерфейс сторінки проекту

Для відображення завдань проекту необхідно надати можливість додавати ці завдання, для цього використовується модальне вікно, яке містить наступні поля для заповнення рис. 3.10. (Інтерфейс створення задачі):

- тип завдання - поле надає вибір з таких типів як Task, Bug і Story;
- короткий опис;
- повний опис – поле з можливістю редагування тексту;
- доповідач – поле для вказівки користувача, який створив завдання;
- призначення – поле для призначення користувачів на дане завдання;
- пріоритет – поле для вказівки пріоритету даного завдання.

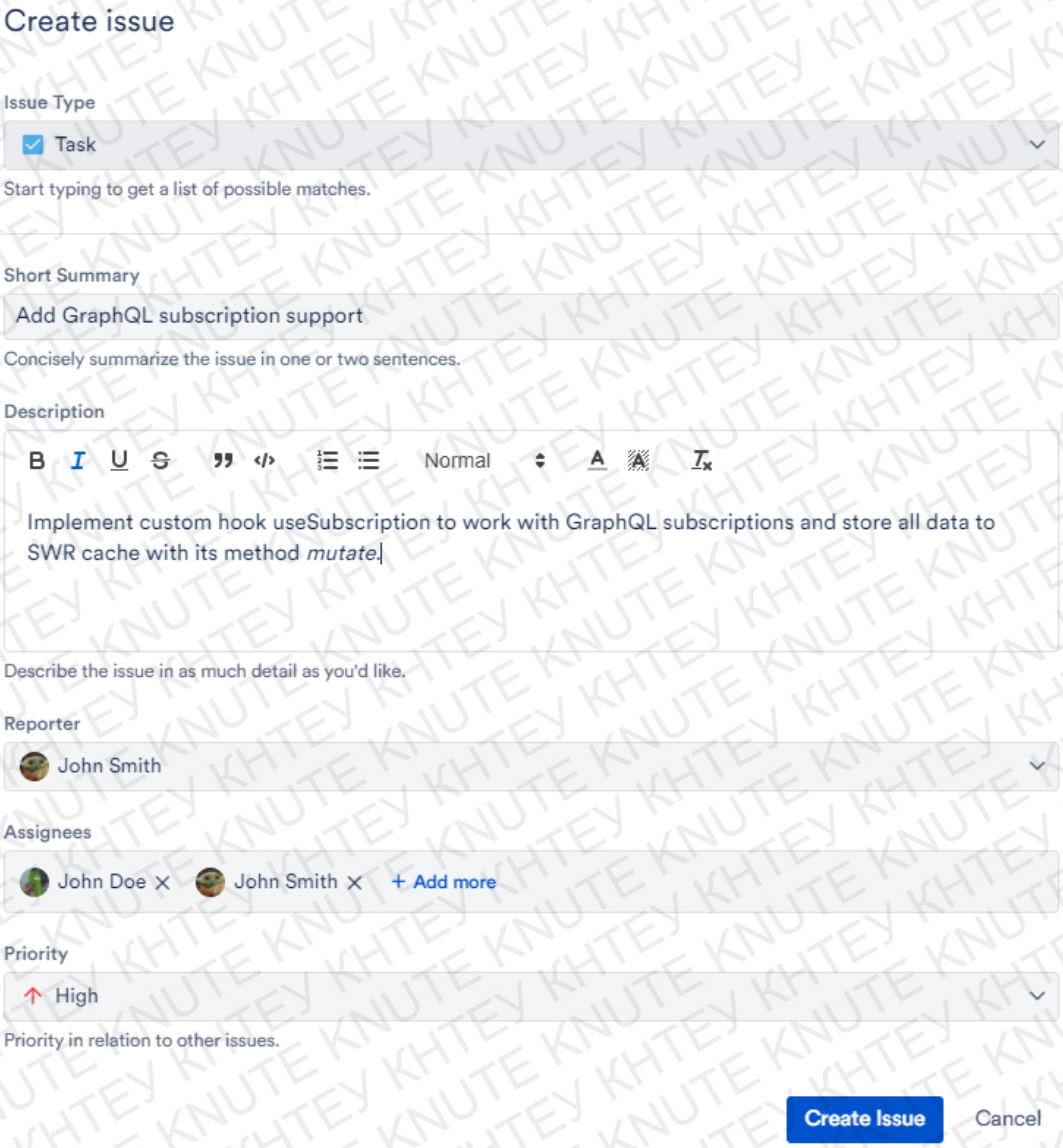

Рис. 3.10. Інтерфейс створення задачі

Для перегляду і комунікації по певній задачі створено модальне вікно, яке містить в собі інформацію про завдання, обговорення з командою і можливість вказати терміни виконання рис. 3.11. (Інтерфейс вікна задачі).

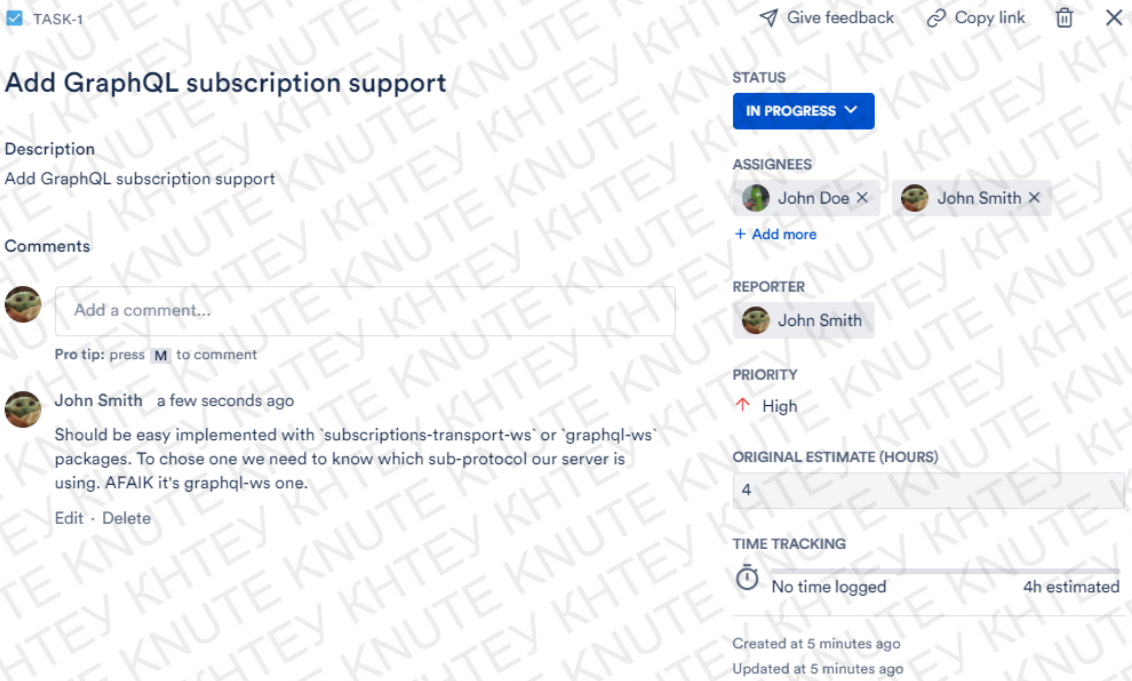

Рис. 3.11. Інтерфейс вікна задачі

3.4. Висновки до розділу

В даному розділі було спроектовано і розроблено клієнтську і серверну частину настільного додатку і структуру бази даних за допомогою сучасних засобів розробки, таких як: Electron, React, NodeJS, Express і PostgreSQL.

Серверна частина розроблена з архітектури Route-Controller-Service, яка ідеально підходить для простих API-серверів. Використовуючи TypeORM створені моделі таблиць БД, також завдяки TypeScript, написані типи для кожної моделі.

Клієнт частина розроблена як звичайний веб-додаток за допомогою бібліотеки React, застосовуючи всі ті ж підходи та інструменти, що і в веб-розробці. Настільний додатки є результатом впровадження фреймворка

Electron, який має можливість відкривати веб-додатки всередині вікна поточної операційної системи.

Таблиці бази даних спроектовані за стандартними принципами реляційної бази даних.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході виконання випускного кваліфікаційного проекту було розглянуто проектування системи керування проектами. Було проаналізовано методи та засоби керування проектами.

Проаналізовано засоби розробки настільних додатків за допомогою інструментів для веб-розробки. Розглянуто фреймворк Electron, який надає можливості впровадження веб-засобів до настільних додатків.

Спроектовано та розроблено структуру настільного додатку дотримуючись принципів простого і зрозумілого інтерфейсу. Клієнтську частину розроблено, як односторінковий додаток за допомогою бібліотеки для розробки інтерфейсів React.

Серверне забезпечення розроблено за архітектурою Model-Controller-Router, за допомогою якої поділяється логіка на три окремі рівня, які не залежать один від одного.

У подальшому розвитку проекту планується додати можливість створювати свої проекти, додавати позначки до задач проекту, проектувати та впроваджувати більше методів для керування проектами інформаційних систем та розробити мобільний додаток застосовуючи вже розроблені програмні рішення. БД та серверне забезпечення може використовуватися яким завгодно клієнтом, тому головною задачею буде розробити клієнтську частину за допомогою React Native або Flutter.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Видання іноземною мовою:

1. Simpson K. You Don’t Know JS: Async & Performance / Kyle Simpson. – O’Reilly Media, Inc, 2015 – 247 p.
2. Banks. A. Learning React. Functional Web Development with React and Redux / Alex Banks, Eve Porcello. – O’Reilly Media, 2017 – 350 p.
3. Brown E. Web Development with Node and Express, 2nd Edition / Ethan Brown – O’Reilly Media, 2019 – 409 p.

Електронні ресурси:

4. Документація бібліотеки для розробки інтерфейсів React [Електронний ресурс]. – [Режим доступу]: https://reactjs.org/docs.
5. Документація фреймворку Electron [Електронний ресурс]. – [Режим доступу]: https://www.electronjs.org/docs.

ДОДАТКИ

Додаток А

Код файлу створення підключення до БД createConnection.ts

```
import { createConnection, Connection } from 'typeorm';
import * as entities from 'entities';

const createDatabaseConnection = (): Promise<Connection> => {
  return createConnection({
    type: 'postgres',
    host: process.env.DB_HOST,
    port: Number(process.env.DB_PORT),
    username: process.env.DB_USERNAME,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_DATABASE,
    entities: Object.values(entities),
    synchronize: true,
  });
}

export default createDatabaseConnection;
```

Додаток Б

Код фалу ініціалізації Electron app.js

```
const {app, BrowserWindow} = require('electron');

let mainWindow;

app.on('window-all-closed', function() {
 if (process.platform != 'darwin')
   app.quit();
});

app.on('ready', function() {
 // Create the browser window.
 mainWindow = new BrowserWindow({
       width: 800,
       height: 600,
       frame:false,
       transparent: true
 });

 mainWindow.setIgnoreMouseEvents(true)
 mainWindow.loadURL('file://' + __dirname + '/index.html');

 mainWindow.on('closed', function() {
  mainWindow = null;
 });
});
```

Додаток В

Код файлу сторінки проекту project.ts

```
const Project = () => {
  const match = useRouteMatch();

  const [{ data, error, setLocalData }, fetchProject] = useApi.get('/project');

  if (!data) return <PageLoader />;
  if (error) return <PageError />;

  const { project } = data;

  const updateLocalProjectIssues = (issueId, updatedFields) => {
    setLocalData(currentData => ({
      project: {
        ...currentData.project,
        issues: updateArrayItemById(currentData.project.issues, issueId, updatedFields),
      },
    }));
  };

  return (
    <ProjectPage>
      <Route
        path={`${match.path}/board`}
        render={() => (
          <Board
            project={project}
            fetchProject={fetchProject}
            updateLocalProjectIssues={updateLocalProjectIssues}
          />
        )}
      />
      {match.isExact && <Redirect to={`${match.url}/board`} />}
    </ProjectPage>
  );
};
```