МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Центр післядипломної освіти
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів розробки веб-орієнтованих сервісів

Виконав:
студент 2 курсу групи ІПЗздм-21-1
Атмар А.Е
(прізвище, ініціали)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми Освітньо-наукова
Керівник проф. Шубін І.Ю
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ Дудар З.В.

2023 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

Кафедра Програмної Інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми Освітньо-наукова

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____»_____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Атмара Атала Емала

1. Тема роботи «Дослідження методів розробки веб-орієнтованих сервісів» затверджена наказом по університету від «03» квітня 2023 р. № 83Стз

2. Термін подання студентом роботи до екзаменаційної комісії «22» травня 2023р.

3. Вихідні дані до роботи методи розробки веб-орієнтованих систем, середовище об'єктно-орієнтованого проектування

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі, постановка задачі, методи розробки веб-орієнтованих систем, архітектура та проєктування програмної системи, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення. Javascript, NodeJS, React

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	03 квітня 2023 р.	Виконано
2	Огляд існуючих методів	10 квітня 2023	Виконано
3	Розробка алгоритмів, проектування та розробка ПЗ	15 квітня 2023	Виконано
4	Підготовка пояснювальної записки	20 квітня 2023 р.	Виконано
5	Спецчастина	28 квітня 2023 р.	Виконано
6	Підготовка презентації та доповіді	03 травня 2023 р.	Виконано
7	Попередній захист	12 травня 2023 р.	Виконано
8	Нормоконтроль, рецензування	15 травня 2023	Виконано
9	Занесення роботи в електронний архів	19 травня 2023	Виконано
10	Допуск до захисту в зав. кафедри	11 травня 2023 р.	Виконано

Дата видачі завдання «23» _січня_ 2023 р.

Завдання прийняв до виконання_____ Атмар Атал

Керівник, проф. _____ Ігор ШУБІН

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 72 с., 10 рис., 13 джер.

АНАЛІЗ МЕТОДІВ РОЗРОБКИ ВЕБ-СЕРВІСІВ, ВЕБ-ОРІЄНТОВАНІ СЕРВІСИ: РОЗРОБКА ТА ВПРОВАДЖЕННЯ, ВИБІР ОПТИМАЛЬНОГО МЕТОДУ РОЗРОБКИ ВЕБ-СЕРВІСУ, ЕФЕКТИВНІ МЕТОДИ РОЗРОБКИ ВЕБ-СЕРВІСІВ.

Об'єкт дослідження — розробка веб-інтегрованого середовища розробки для Javascript з використанням Javascript.

Метою роботи є планування розробки веб-інтегрованого середовища розробки на JavaScript для мови програмування JavaScript, яке забезпечує можливість компіляції коду та завантаження зовнішніх бібліотек. Методи розробки базуються на таких технологіях, як Javascript, NodeJS, React.

У результаті роботи було досліджено методи створення веб-інтегрованого середовища розробки на JavaScript. Було проведено аналіз та порівняння існуючих методів та концепцій, які використовуються в даній галузі. Планується створення веб-інтегрованого середовища розробки на мові програмування JavaScript з попередньою назвою «JSWave».

ANALYSIS OF WEB SERVICE DEVELOPMENT METHODS, CHOOSING THE OPTIMAL METHOD OF WEB SERVICE DEVELOPMENT, EFFECTIVE METHODS OF WEB SERVICE DEVELOPMENT, WEB-ORIENTED SERVICES: DEVELOPMENT AND IMPLEMENTATION.

The object of the research is the development of a web-integrated development environment for JavaScript using JavaScript. The aim of the work is to plan the development of a web-integrated development environment in JavaScript for the JavaScript programming language, which provides the ability to compile code and load external libraries. Development methods are based on technologies such as JavaScript, NodeJS, React.

As a result of the work, methods for creating a web-integrated development environment in JavaScript were studied. Analysis and comparison of existing methods and concepts used in this field were carried out. It is planned to create a web-integrated development environment in the JavaScript programming language with the preliminary name "JSWave".

Умови публікації пояснювальної записки

Я, Атмар Атал Емал
(прізвище, ім'я, по батькові)

студент групи ІПЗздм-21-1 здобувач вищої освіти на другому (магістерському) рівні

кафедра програмної інженерії,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему

Дослідження методів розробки веб-орієнтованих сервісів
(назва роботи)

що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У зв'язку зі зростанням інтернету все більше програм розробляються як веб-додатки, які працюють у браузері. І хоча інтегровані середовища розробки (IDE) є складним технічним програмним забезпеченням, чия функціональність ще не повністю відтворена в веб-додатках, існує безліч веб-орієнтованих рішень, що реалізують деякі їх аспекти. Для різних мов програмування існує безліч веб-сайтів, які надають текстовий редактор та компілятор. Для мови JavaScript існує безліч бібліотек для розширення можливостей редагування тексту в браузері, що додають такі важливі функції редакторів коду, як підсвічування синтаксису та автодоповнення.

Однією з головних проблем, які вирішує веб-інтегроване середовище розробки, є полегшення процесу розробки веб-додатків. Завдяки використанню веб-технологій та можливості працювати в браузері, веб-середовища розробки дозволяють розробляти програми з будь-якого пристрою, де є доступ до інтернету. Це може бути як персональний комп'ютер, так і мобільний пристрій.

Однією з головних переваг веб-середовищ є можливість спільної роботи над проектами в реальному часі. Декілька розробників можуть працювати над одним проектом, бачити зміни один одного та коментувати їх. Це дозволяє суттєво прискорити процес розробки та скоротити час, що витрачається на взаємодію між розробниками.

Крім того, веб-середовища розробки часто надають безліч інструментів та функцій, які значно спрощують процес розробки. Наприклад, підсвічування синтаксису, автодоповнення коду, можливість швидкого пошуку помилок та багато іншого. Це дозволяє скоротити час, що витрачається на написання коду, і зробити його більш читабельним та зрозумілим для інших розробників.

Одна з головних практичних користей веб-інтегрованих середовищ розробки (IDE) для JavaScript полягає в тому, що вони полегшують та прискорюють процес розробки веб-додатків [1].

Веб-IDE дозволяють розробникам створювати, тестувати, налагоджувати та деплоїти веб-програми з одного місця, без необхідності перемикання між різними інструментами та програмами.

Таким чином, веб-інтегровані середовища розробки є дуже корисним інструментом для розробки веб-додатків. Вони дозволяють швидко та ефективно створювати програми, прискорюють процес розробки та дозволяють працювати в команді над одним проектом.

Однак існуючі веб-орієнтовані IDE часто не є універсальними і підтримують лише найпопулярніші мови програмування. Для мови JavaScript вибір веб-орієнтованих IDE є досить обмеженим, як і їх функціональність. У роботі буде представлений розділ, присвячений аналізу існуючих рішень. Також деякі веб-IDE можуть мати недостатньо інтуїтивно зрозумілий інтерфейс, що ускладнює роботу з ними, деякі веб-IDE можуть бути повністю залежними від інтернету, що обмежує їх використання в офлайн-середовищі.

Для вирішення цієї проблеми в рамках проекту було створено спеціальне програмне забезпечення з наступними функціями: виділення синтаксису введеного коду, можливість завантаження зовнішніх бібліотек для їх використання, автоматичне доповнення коду на основі введених змінних та методів.

Для реалізації цього було використано JavaScript для клієнтської частини середовища розробки та HTTP-сервера, а також Javascript був використаний для компіляції введеного коду та аналізу завантажених бібліотек [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

В цьому розділі порівняємо різні середовища веб-розробки. Існує кілька типів середовищ розробки JavaScript[3]: настільні, веб-інтегровані та хмарні. Настільні середовища розробки забезпечують локальний доступ до функціональності та інструментів, що може бути корисним для більш складних проектів, але потребує додаткових ресурсів на стороні користувача. Веб-інтегровані середовища розробки, такі як JSFiddle та JS Bin (рисунок 1.1), надають доступ до інструментів через веб-браузер, що забезпечує простіший доступ для користувачів, що працюють на різних пристроях.

Рисунок 1.1 – Інтерфейс JS Bin

Хмарні середовища розробки, такі як Cloud9 (рисунок 1.2) та Codeanywhere, надають функціональність веб-інтегрованих середовищ розробки, але також дозволяють зберігати та працювати з проектами у хмарі.

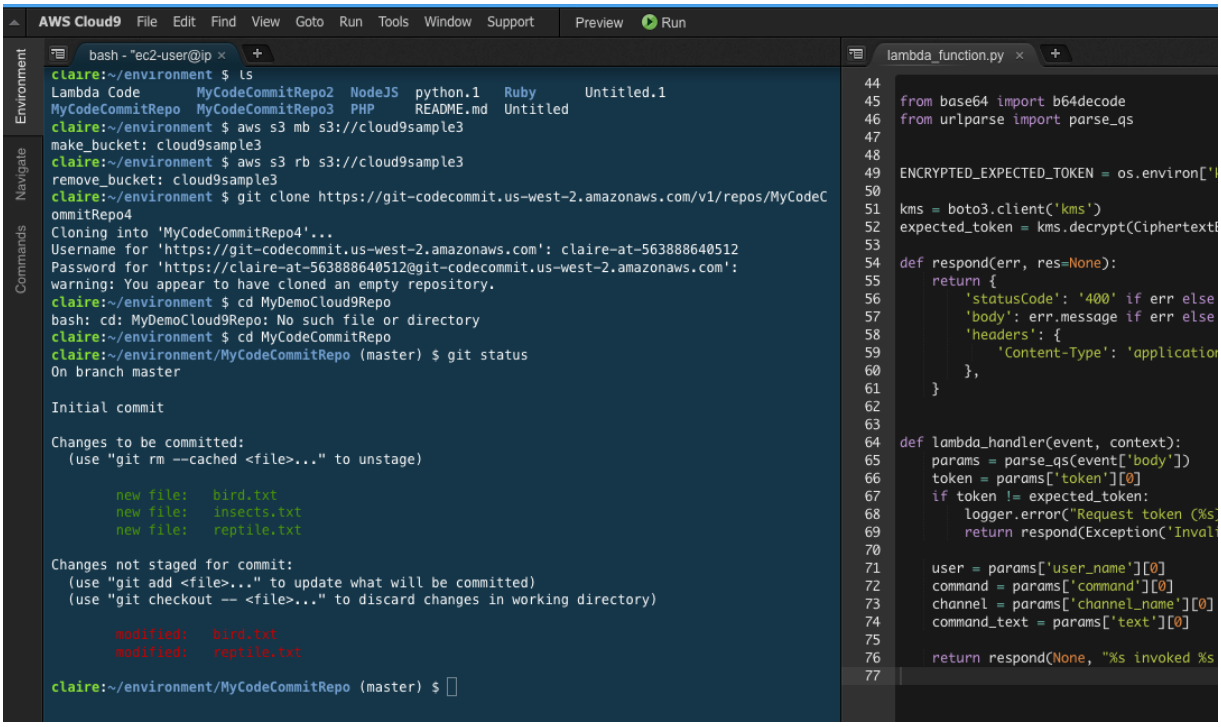

Рисунок 1.2 – Головний інтерфейс Cloud9

Веб-інтегровані середовища розробки можуть бути особливо корисні для розробників-початківців, які ще не хочуть або не можуть використовувати більш складні настільні середовища розробки. Вони також можуть бути корисні для розробників, які працюють з безліччю різних пристроїв та операційних систем, оскільки вони можуть забезпечити більш єдиний інтерфейс та досвід роботи. Більше того, веб-інтегровані середовища розробки можуть забезпечити швидкий доступ до документації, онлайн-курсів та інших ресурсів, що може бути корисним для навчання та підвищення кваліфікації.

Загалом, веб-інтегровані середовища розробки JavaScript можуть бути хорошим вибором для багатьох розробників, особливо для початківців, які ще не готові використовувати більш складні середовища розробки. Вони також можуть забезпечити більш єдиний досвід роботи та швидкий доступ до ресурсів.

Крім того, веб-середовище розробки має ще низку інших переваг. Зокрема, вона забезпечує можливість роботи над проектом у колаборативному режимі. Усі учасники проекту можуть працювати з одним і тим самим кодом і бачити зміни, які вносяться іншими учасниками, що полегшує процес спільної розробки. Тут буде

доречно порівняння з VSCode. Обидва середовища розробки мають можливість колаборативної розробки. Однак, веб-середовище розробки має деякі переваги порівняно з локальним середовищем, таким як VSCode.

По-перше, веб-середовище розробки забезпечує більш просту інтеграцію з іншими інструментами та сервісами, що знижує поріг входження для нових користувачів та полегшує зміну середовища розробки.

По-друге, веб-середовище розробки дозволяє працювати з проектом з будь-якого пристрою, який має доступ до Інтернету. Таким чином, це дозволяє розробникам працювати з проектом з будь-якого місця та в будь-який час.

По-третє, веб-середовище розробки забезпечує більш просту настройку для спільної розробки проекту з колегами. Вам не потрібно налаштовувати додаткове програмне забезпечення на кожному пристрої користувача, що робить процес спільної роботи над проектом більш зручним та швидким.

Також веб-середовища розробки дозволяє легко інтегруватися з іншими інструментами, які використовуються в процесі розробки. Наприклад, можна легко підключати зовнішні бібліотеки або використовувати різні плагіни та розширення для покращення функціональності середовища розробки.

Однією з головних переваг веб-середовища розробки є його доступність. Користувачі можуть працювати з проектом із будь-якого місця, де є доступ до інтернету. Це зручно для тих, хто працює в команді з різних географічних точок або для тих, хто воліє працювати віддалено.

Крім того, веб-середовище розробки може бути більш економічно ефективним. На відміну від локальних середовищ розробки, які можуть вимагати додаткових витрат на обладнання, ліцензії програмного забезпечення і т.д., веб-середовище розробки може бути доступним за підпискою, що дозволяє знизити витрати на початковий вхід в індустрію.

В цілому, веб-середовище розробки надає безліч переваг, таких як доступність, легка інтеграція з іншими інструментами, можливість колаборативної роботи та економічна ефективність. Це робить її привабливою для широкого кола розробників та дозволяє заощадити час та сили у процесі розробки.

1.2 Виявлення проблем та актуалізація рішень

Веб-середовище розробки JavaScript може вирішити ряд проблем, пов'язаних з розробкою на даній мові програмування. Наприклад:

– проблеми із залежностями: веб-середовища розробки може надати інструменти для керування залежностями, такі як завантаження та встановлення бібліотек, а також сповіщення про доступні оновлення.

– проблеми з тестуванням: веб-середовище розробки може надати інструменти для автоматичного тестування коду, що спростить та прискорить процес розробки.

– проблеми з налагодженням: веб-середовище розробки може надати інструменти для налагодження коду JavaScript, включаючи відстеження помилок, точки зупинки і т.д.

– проблеми з продуктивністю: веб-середовище розробки може допомогти оптимізувати код для покращення продуктивності програм на JavaScript.

– проблеми із спільною роботою: веб-середовище розробки може надати інструменти для спільної роботи над проектами, такі як системи контролю версій, обмін файлами тощо.

– проблеми з безпекою: веб-середовище розробки може надати інструменти для забезпечення безпеки програм на JavaScript, включаючи перевірку на вразливості, шифрування і т.д.

Крім того, веб-середовища розробки може допомогти вирішити такі проблеми:

– втрата даних: багато середовищ розробки не надають можливість автоматичного збереження змін у режимі реального часу, що може призвести до втрати даних у разі збою системи або непередбаченої помилки. Веб-середовище розробки може вирішити цю проблему, надаючи можливість автоматичного збереження даних на віддаленому сервері.

– необхідність переносити налаштування між різними комп'ютерами: у разі використання настільного середовища розробки, розробнику може знадобитися

щоразу налаштовувати середовище на новому комп'ютері. Веб-середовища розробки може вирішити цю проблему, надаючи можливість доступу до налаштувань та конфігурацій через віддалений сервер.

– необхідність оновлення програмного забезпечення: багато настільних середовищ розробки потребують регулярних оновлень для покращення функціональності та виправлення помилок. Веб-середовище може вирішити цю проблему, забезпечуючи автоматичне оновлення програмного забезпечення на сервері, без необхідності вручну оновлювати програмне забезпечення на кожному комп'ютері.

– необхідність спілкування та співробітництва: веб-середовище розробки може допомогти вирішити проблему співпраці та комунікації, надаючи можливість працювати над проектом спільно та обмінюватися інформацією через віддалений сервер. Це може підвищити ефективність роботи та прискорити процес розробки.

– необхідність швидкого та зручного доступу до проектів: веб-середовище розробки може вирішити проблему необхідності швидкого та зручного доступу до проектів, надаючи можливість працювати з проектами через інтернет-браузер на будь-якому пристрої з доступом до інтернету. Це може бути особливо корисним у разі роботи з кількома проектами або у разі потреби працювати з віддалених місць.

1.3 Постановка задачі

Постановка задачі розробки системи полягає у створенні веб-інтегрованого середовища розробки на JavaScript, яке надаватиме розробникам зручний інтерфейс для написання, налагодження, тестування та компіляції коду, а також можливість завантаження зовнішніх бібліотек (рисунок 1.3).

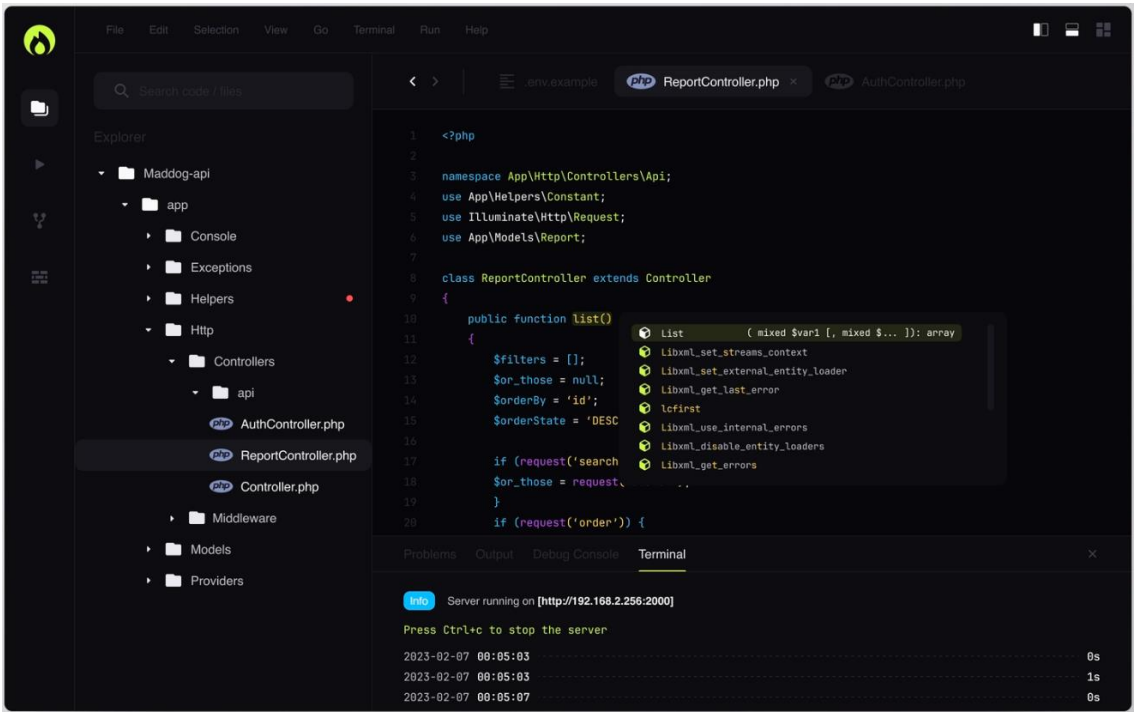

Рисунок 1.3 – Зручний дизайн веб-інтегрованого середовища розробки

Система повинна підтримувати всі основні можливості, необхідні для розробки на мові програмування JavaScript, такі як підсвічування синтаксису, автодоповнення, інтегровану консоль, можливість управління проектами та залежностями, а також можливість запуску тестів та профілювання коду.

Метою проекту є створення засобу розробки на JavaScript, який буде легко доступний, швидко завантажуватись і просто використовуватись. Воно має бути дружнім до користувача та надавати безліч можливостей для підвищення продуктивності, зручності та надійності веб-додатків. Основне завдання розробників – створення універсальної платформи, яка зможе використовуватись для розробки як простих, так і складних програм.

1.3.1 Основні функції системи

– редагування та відлагодження коду: ця функція дозволяє розробникам редагувати та відлагоджувати код безпосередньо з веб-інтерфейсу.

- віддалене редагування: функція віддаленого редагування дозволяє розробникам працювати з кодом з будь-якого пристрою, що підключений до мережі Інтернет.
- можливість підключення до проекту різних бібліотек і фреймворків.
- автодоповнення коду (autocomplete) для полегшення написання коду.
- можливість швидкого переходу до визначення функції або змінної.
- можливість працювати над проектом в колаборативному режимі, що полегшує спільну розробку коду з кількома учасниками проекту.

1.3.2 Допущення та залежності

Існують такі допущення:

– великий обсяг коду та складність веб-інструментів можуть призвести до недостатньої продуктивності. Необхідно ретельно працювати над оптимізацією та забезпечити ефективність інструментів для максимальної продуктивності розробників.

– недостатня або неповна документація може призвести до складнощів у використанні веб-інструментів. Розробники повинні забезпечити детальну документацію для забезпечення легкої розуміння та використання інструментів.

– інструменти зазвичай залежать від браузерів та їх можливостей. Розробникам потрібно враховувати різні версії браузерів та їх особливості, щоб забезпечити підтримку для якомога більшої кількості користувачів.

– підтримка веб-інтегрованого середовища розробки є важливим фактором при його використанні. Розробники повинні бути в змозі швидко отримати відповіді на свої запитання та проблеми.

1.3.3 Релізи

Щодо функцій на вищому рівні, кожен наступний реліз буде розширювати наявну програмну систему, не змінюючи основний концепт

В початковому релізу повинен бути присутній наступні функції:

– базовий інтерфейс користувача, авторизація та аутентифікація;
– створення та видалення проектів
– робота з файлами та текстами

Реліз 2:

– підтримка мов програмування: JavaScript, TypeScript, CSS, HTML
– візуальний редактор коду з можливістю автодоповнення та форматування написаного розробником коду.
– підтримка відладки коду з використанням вбудованого у програмну систему, консольного логування.
– інтеграція з Git для контролю версій проектів.

Реліз 3:

– підтримка додаткових фреймворків та бібліотек
– підтримка плагінів
– менеджер пакетів

Реліз 4:

– рефакторинг коду та оптимізація продуктивності
– підтримка тестування коду

1.3.4 Бізнес - потреби та пріоритетність

Збільшення продуктивності розробників: веб-орієнтована IDE може прискорити процес розробки, дозволяючи розробникам швидше писати та налагоджувати код, а також легко підключати та використовувати бібліотеки та інструменти.

Зменшення витрат: використання веб-орієнтованої IDE може знизити вартість розробки за рахунок використання безкоштовних або недорогих інструментів, а також зменшити витрати на навчання співробітників, оскільки веб-орієнтована IDE може бути інтуїтивнішою і простішою у використанні.

Поліпшення якості коду: веб-орієнтована IDE може пропонувати різні функції, такі як автодоповнення, перевірку синтаксису та статичний аналіз, які допомагають розробникам писати якісніший та безпечніший код.

Поліпшення командної роботи: веб-орієнтована IDE може надавати можливість спільної роботи над проектами та контролю версій.

1.3.5 Середовище оточення

Проект веб-інтегрованого середовища розробки для Javascript буде написаний з використанням HTML5, CSS та Javascript. Для стилізації будуть використані CSS фреймворки, такі як Bootstrap [4] або Materialize. Для розробки функціональності клієнтської частини використовуватимемо фреймворки та бібліотеки Javascript, такі як React [5] або Vue.js.

Back-end буде реалізований за допомогою Node.js [6] та його фреймворку, такого як Express або Koa. Для бази даних буде використана NoSQL база даних, наприклад, MongoDB. Для забезпечення безпеки даних будуть використовуватися методи шифрування та аутентифікації, такі як JSON Web Tokens .

Наше середовище розробки буде забезпечувати можливість відладки та тестування коду, а також інтегруватися з різними зовнішніми сервісами та бібліотеками. Ми будемо стежити за найновішими технологіями та стандартами веб-розробки, щоб забезпечити оптимальну продуктивність та швидкість роботи нашого середовища.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Етапи розробки

Першим етапом розробки прототипу є складання вимог до програмної системи та створення концептуального дизайну інтерфейсу користувача.

Після цього можна переходити до розробки базових функцій системи та їх інтеграції з інтерфейсом користувача. На цьому етапі можна використовувати ітеративний підхід, додавши нові функції та вдосконалюючи існуючі з кожною ітерацією.

Для забезпечення успішного першого етапу розробки прототипу необхідно чітко визначити цілі та завдання проекту, а також зробити аналіз ринку та конкурентів. Важливо також мати якісну команду розробників, яка зможе вирішувати технічні проблеми та ефективно співпрацювати між собою.

На другому етапі розробки прототипу системи можна зосередитися на детальному вивченні потреб користувачів та визначенні функціональності програмної системи. Крім того, на цьому етапі можна вивчити можливості інтеграції з іншими інструментами та сервісами, що можуть збільшити функціональність системи та покращити її ефективність. Важливим етапом є визначення архітектури системи та вибір технологій, які будуть використовуватися для розробки системи.

Крім того, на другому етапі можна провести дослідження та тестування, щоб переконатися у працездатності та ефективності системи, а також врахувати можливі проблеми з безпекою даних та розглянути можливість застосування методів шифрування даних.

Нарешті, на другому етапі можна створити прототип системи та провести тестування його роботи, отримати відгуки користувачів та врахувати їхні рекомендації щодо подальшого розвитку системи.

2.2 Вимоги

Вимоги до проекту - це опис обов'язкових характеристик, функцій та інших параметрів, які повинен мати проект для досягнення мети. Ці вимоги можуть бути розподілені на дві основні категорії: функціональні та нефункціональні.

2.2.1 Функціональні вимоги

У нашому проекті середовища розробки для Javascript з використанням Javascript можуть бути такі функціональні вимоги:

– редагування та створення файлів: можливість створювати, відкривати, редагувати, зберігати та видаляти файли всередині середовища розробки.

– підтримка компіляції вихідного коду: можливість компілювати JavaScript код та виконувати його на сервері чи клієнті.

– підсвічування синтаксису: автоматичне підсвічування синтаксису коду для полегшення читання та написання коду.

– автодоповнення та пропозиції коду: автоматичне доповнення коду під час введення та пропозиція можливих варіантів коду.

– рефакторинг коду: можливість перейменування змінних, функцій, класів і методів, отримання методів, зміни типів та багато іншого без необхідності ручної зміни коду.

– налагодження коду: можливість налагодження коду на рівні рядка для виявлення та виправлення помилок.

– зберігання налаштувань: середовище повинно зберігати налаштування користувача, такі як розмір шрифту, кольорову схему, підключені бібліотеки тощо.

– робота з бібліотеками та фреймворками: середовище повинно підтримувати підключення бібліотек та фреймворків JavaScript для полегшення розробки.

– завантаження та виконання коду: користувач повинен мати змогу

завантажувати та виконувати код JavaScript в середовищі, в тому числі віддаленому сервері.

2.2.2 Нефункціональні вимоги

Вимоги, що не стосуються безпосередньо функціональності системи, але визначають характеристики її роботи та ефективності. Основні нефункціональні вимоги можуть включати:

– продуктивність: система повинна забезпечувати швидкодію під час редагування, компіляції та виконання коду;

– надійність: система має бути стабільною та надійною в роботі;

– легкість у використанні: система повинна бути інтуїтивно зрозумілою та простою у використанні для початківців та досвідчених розробників;

– розширюваність: можливість додавання додаткових плагінів та розширень для додаткової функціональності;

– підтримка різних браузерів: система повинна підтримувати роботу в різних браузерах без сумісності;

– документація та підтримка: наявність детальної документації та підтримки користувачів.

2.3 Інтерфейс користувача

Нашу систему веб-інтегрованого середовища розробки для Javascript має бути легкою та зручною у використанні, тому інтерфейс користувача повинен бути розроблений з урахуванням цих вимог. Головна сторінка має містити відображення переваг нашого продукту, що приверне увагу користувачів та забезпечить ефективну рекламу нашої системи. Кнопки реєстрації та авторизації повинні бути доступні на головній сторінці, щоб забезпечити швидкий доступ користувачам до функцій системи.

Після авторизації, користувач повинен мати доступ до основних функцій системи, а інтерфейс має бути зрозумілим та простим для навігації. Основний інтерфейс користувача повинен містити панель з інструментами, що містять основні функції системи, такі як створення нового проекту, відкриття та редагування існуючих проектів, пошук функцій, відладку та тестування коду.

Інтерфейс користувача повинен також підтримувати різні типи файлів та форматів даних, для чого можуть бути розміщені додаткові панелі з інструментами для конвертації файлів та відображення даних у зручному для користувача форматі. Необхідно також забезпечити адаптивність інтерфейсу для різних розмірів екрану та пристроїв, щоб користувачам було зручно користуватися нашою системою.

Важливо, щоб інтерфейс користувача був адаптивним і міг пристосовуватися до різних розмірів екранів і пристроїв. Це дозволить користувачам зручно використовувати систему з будь-якого пристрою з доступом до Інтернету, включаючи комп'ютери, планшети та смартфони. Інтерфейс повинен бути досить простим та ергономічним, щоб без додаткових зусиль користувачі могли легко отримати доступ до всіх функцій системи. Для забезпечення привабливого зовнішнього вигляду та зручності використання потрібне використання чітких та лаконічних символів та підказок. Наша система має бути зручною та простою у використанні, щоб користувачі могли розробляти програми на Javascript швидко та ефективно [7].

Вимоги до продуктивності програмної системи включають в себе:

– швидкість роботи: програмна система повинна бути достатньо швидкою для задоволення потреб користувачів (швидкість роботи вимірюється в мілісекундах або секундах і залежить від кількості даних, які оброблює система, а також від алгоритмів, які використовуються для обробки цих даних);

– продуктивність: програмна система повинна бути достатньо продуктивною для обробки великих обсягів даних, які можуть збільшуватись в майбутньому (продуктивність вимірюється в кількості операцій, які система може виконувати за одиницю часу);

– надійність: програмна система повинна бути достатньо надійною для

запобігання виникнення помилок або аварій, що можуть призвести до втрати даних або недоступності сервісу для користувачів (надійність вимірюється в часі, протягом якого система може працювати без збоїв);

– масштабованість: програмна система повинна бути достатньо масштабованою для забезпечення роботи з великою кількістю користувачів і обробки великих обсягів даних (масштабованість вимірюється в кількості користувачів, яких може обслуговувати система, і в обсягу даних, який може обробляти система).

Вимоги щодо сумісності програмної системи включають у себе необхідність забезпечення роботи системи на різних операційних системах та веб-браузерах, щоб користувачі могли використовувати її на будь-яких пристроях.

Вимоги щодо технічної підтримки програмної системи включають у себе можливість оновлення та виправлення помилок для забезпечення безперебійної та ефективної роботи.

Нарешті, інтерфейс повинен бути безпечним та надійним. Ми повинні забезпечити захист користувачів та їхніх даних від несанкціонованого доступу та зловживань. Тому, ми маємо включити в інтерфейс системи заходи безпеки, такі як шифрування даних, двофакторна аутентифікація та перевірка на наявність потенційних загроз безпеці перед реєстрацією нових користувачів. Усі ці аспекти повинні бути враховані при розробці інтерфейсу користувача системи.

Отже, при формуванні вимог до середовища веб-розробки для мови JavaScript необхідно враховувати різноманітні вимоги, такі як функціональні та нефункціональні вимоги, вимоги до продуктивності, сумісності та технічної підтримки.

2.4 Схема взаємодії компонентів

Середовища розробки для Javascript з використанням Javascript, схема взаємодії компонентів демонструє, які компоненти відповідають за різні функції,

такі як редагування коду, його токенізація, рендеринг у форматі HTML і виведення в текстовий редактор, а також для завантаження бібліотек та виконання скомпільованого коду.

Це інтегроване середовище розробки, яке складається з клієнтської та серверної частини. Кожну з них можна поділити на багато компонентів, кожен з яких виконує свою функцію. Нижче наведено схему компонентів для клієнтської частини додатка (рисунок 2.1).

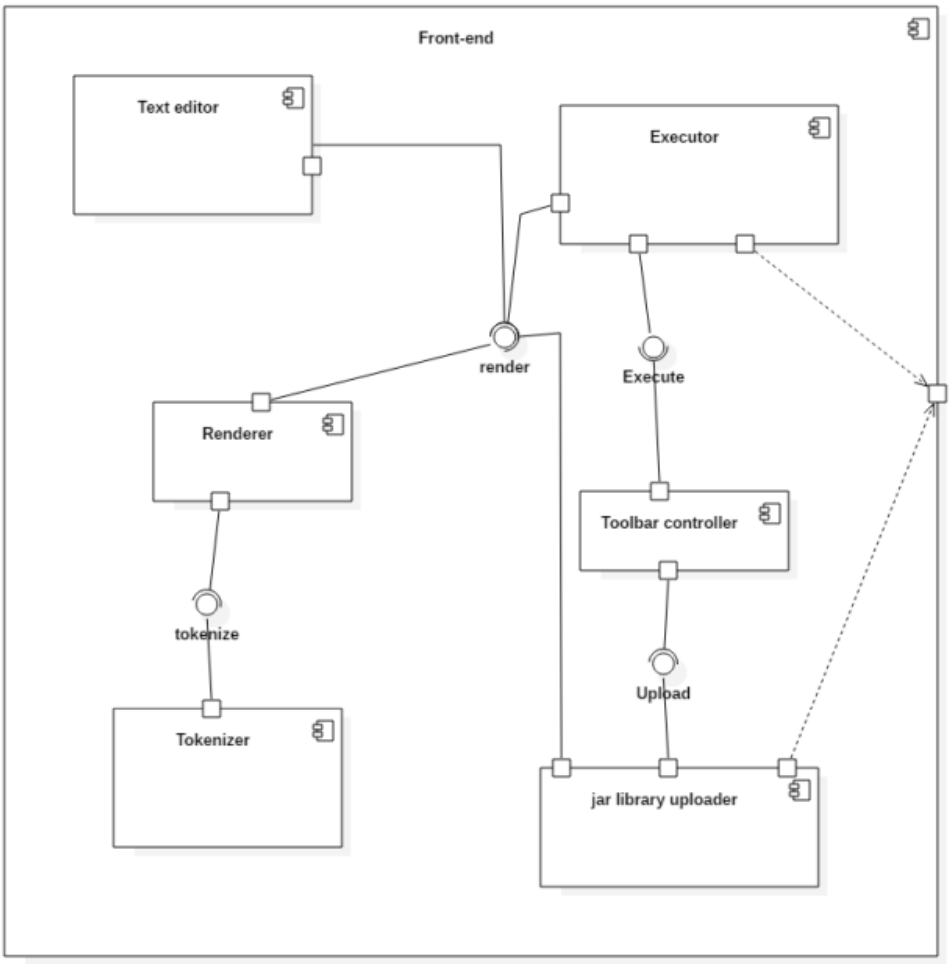

Рисунок 2.1 – Компоненти Front-end

У нашому проекті зображена схема взаємодії компонентів, серед яких можна виділити:

– Text editor - відповідає за редагування коду.

– Tokenizer - відповідає за токенізацію тексту, введеного в редактор коду.

– Renderer - відповідає за відображення тексту у форматі HTML і виведення його в текстовий редактор

– Toolbar controller - дозволяє користувачеві завантажити jar бібліотеку або скомпілювати код за допомогою jar library uploader та Executor відповідно.

2.5 Апаратний інтерфейс

Кінцевою метою проекту є створення онлайн-інструменту для розробки JavaScript-програм, що надає інтуїтивно зрозумілий інтерфейс і зручний доступ до функціоналу для користувачів з різними рівнями досвіду в програмуванні. Для досягнення цієї мети було розроблено архітектурний дизайн програми, заснований на трьох рівнях – клієнтському, серверному та рівні бази даних.

Серверна сторона програми надає API для взаємодії з клієнтською стороною та базою даних. Клієнтська сторона представлена у вигляді веб-інтерфейсу, який забезпечує зручний доступ до функціонала програми та взаємодію із сервером за допомогою API [8]. База даних є сховищем всіх даних, необхідні роботи програми (рисунок 2.2).

Рисунок 2.2 – Використання ORM у програмній системі

Для захисту даних у базі даних було вжито відповідних заходів. Усі дані передаються між клієнтом та сервером у зашифрованому вигляді за допомогою протоколу HTTPS. База даних захищена паролем та має обмежений доступ лише для авторизованих користувачів. У базі даних використовуються механізми хешування паролів для збереження конфіденційності інформації користувачів.

Додатковим заходом захисту даних може бути забезпечення резервного копіювання бази даних з регулярними періодичними оновленнями, щоб у випадку втрати даних або непередбачуваної ситуації забезпечити швидкий і безперебійний відновлення роботи системи. Крім того, можливість контролю доступу до бази даних та моніторингу використання системи можуть допомогти виявити і запобігти можливим загрозам безпеки даних.

Також, важливим елементом захисту даних є регулярне оновлення програмного забезпечення, яке використовується для роботи з базою даних. Оновлення допоможуть усунути можливі уразливості та забезпечити стабільну роботу системи.

Нарешті, важливим елементом захисту даних є навчання користувачів та встановлення правил безпеки.

2.6 Атрибути програмної системи

2.6.1 Надійність

Для забезпечення безпеки в проекті будуть використані базові заходи, такі як захист доступу до бази даних паролем та обмеження доступу лише для авторизованих користувачів. Також, усі дані повинні передаватися між клієнтом та сервером у зашифрованому вигляді за допомогою протоколу HTTPS.

Для забезпечення надійності системи, було використовується високоякісний та надійний код. Перед розгортанням системи було проведено тестування на локальному сервері для виявлення можливих помилок та проблем. Для

автоматизації процесу розгортання використовувалися інструменти, такі як Docker та Kubernetes.

Для забезпечення моніторингу роботи системи та логування подій встановлено систему резервного копіювання даних. Також, була приділена достатня увага безпеці та надійності системи. Однак, необхідно враховувати фактор ризику відсутності команди для розробки та підтримки проекту. Збільшення обсягу роботи може призвести до необхідності збільшення команди для забезпечення надійності та безпеки системи.

2.6.2 Доступність

Щоб забезпечити більшу доступність системи, можна створити додаткові сервери для бізнес-логіки, які будуть працювати в режимі взаємозаміни. Якщо один сервер вийде з ладу або не зможе обробити запит, інший сервер може взяти на себе його роботу. Gateway сервіс можна використовувати як посередника між клієнтом та серверами.

Gateway сервіс отримує запити від клієнтів та перенаправляє їх на вільний сервер бізнес-логіки. Це підвищує доступність та надійність системи, але вимагає додаткових зусиль для розгортання, налагодження та підтримки системи, а також забезпечення її безпеки.

2.6.3 Безпека

Забезпечення безпеки системи – важливий аспект, який необхідно враховувати під час її розробки та експлуатації. Деякі з основних аспектів безпеки, які можна враховувати для забезпечення безпечної та надійної роботи системи, включають такі.

Аутентифікація та авторизація. Система повинна мати механізми автентифікації та авторизації, щоб гарантувати, що лише користувачі з

необхідними правами можуть проникнути у функції та ресурси системи.

Шифрування даних для запобігання несанкціонованому доступу до конфіденційної інформації система повинна використовувати шифрування даних при їх передачі та зберіганні.

Аудит безпеки. У системі мають бути передбачені механізми аудиту безпеки, які гарантують, що пов'язані з безпекою події, аналогічні до спроб отримання несанкціонованого доступу до системи, реєструються.

Захист від уразливостей. Система має бути захищена від уразливостей, які можуть бути використані для отримання несанкціонованого доступу до системи або модифікації даних. Захист від шкідливих програм Система повинна мати механізми захисту від шкідливих програм, таких як зараження та шкідливі програми.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз вимог

Під час роботи візьмемо за основу водоспадну модель життєвого циклу програмного забезпечення. Водоспадна модель є статичною моделлю і підходить до розробки систем лінійно і послідовно, завершуючи одну дію перед іншою.

Проектування веб середовища розробки на JavaScript можна розбити на наступні етапи:

– аналіз вимог: на цьому етапі збираються вимоги користувачів та з'ясовується, які функції та можливості має включати веб середовище;

– проектування архітектури: на цьому етапі вирішується, які технології та інструменти потрібні для розробки, а також яка буде архітектура веб середовища;

– проектування інтерфейсу користувача: на цьому етапі розробляється дизайн інтерфейсу користувача, який має бути зручним та інтуїтивно зрозумілим;

– розробка функціональності: на цьому етапі програмісти розробляють функціональність веб середовища, яка має відповідати вимогам користувачів;

– тестування та налагодження: на цьому етапі веб середовище проходить тестування та налагодження, щоб переконатися в його працездатності та відповідності вимогам користувачів;

– реліз та підтримка: після успішного проходження всіх етапів веб середовище готове до релізу, після релізу потрібно забезпечити підтримку та виправляти помилки, які можуть виникнути у користувачів.

При проектуванні веб-середовища для мови JavaScript необхідно пройти через ряд етапів, таких як аналіз вимог, проектування архітектури, створення бази даних, розробка інтерфейсу користувача, написання програмної логіки, проведення тестування та налагодження, а також розгортання та забезпечення підтримки.

Аналіз вимог є першим етапом проектування програмної системи. На цьому етапі необхідно зрозуміти, які функції та можливості повинна міститисистема, щоб задовольняти потреби користувачів.

Основні вимоги до веб-середовища для мови JavaScript включають такі пункти, як підтримка різних фреймворків та бібліотек, забезпечення безпеки та захисту користувачів та їхніх даних від зловмисних дій, а також швидкість та продуктивність, щоб забезпечити ефективну роботу розробників та користувачів. Для цього система повинна мати розширені можливості редагування та налагодження, підтримувати сумісність з різними браузерами та мати можливість інтеграції з різними плагінами та бібліотеками.

Отже, основні вимоги до нашої веб-середовища для мови JavaScript можуть включати:

– розширені можливості редагування та налагодження: система повинна мати потужні редактори та інструменти для налагодження JavaScript-коду, щоб розробники могли швидко та легко створювати та тестувати програми;

– підтримка різних фреймворків: система повинна бути сумісною з різними фреймворками, такими як React, Angular та Vue, щоб розробники могли використовувати свій улюблений фреймворк та бути більш продуктивними;

– підтримка бібліотек та плагінів: система повинна мати можливість інтеграції з різними бібліотеками та плагінами, щоб розробники могли легко використовувати готові рішення та розширювати функціональність своїх програм;

– підтримка різних браузерів: система повинна бути сумісною з різними браузерами, такими як Chrome, Firefox, Safari та Edge, щоб користувачі могли використовувати наші програми на різних платформах та з різних пристроїв;

– безпека та захист: система повинна мати міцну систему безпеки, щоб захистити користувачів та їхні дані від хакерських атак та зловмисних дій;

– швидкість та продуктивність: система повинна бути швидкою та продуктивною, щоб забезпечити ефективну роботу розробників та користувачів.

Додаткові вимоги до нашої системи включають:

– система повинна бути здатна легко масштабуватись для обробки збільшеного обсягу даних та користувацького навантаження.

– система повинна бути надійною та стійкою до помилок, щоб уникнути втрати даних та забезпечити безперебійну роботу користувачів.

– система повинна бути зручною у використанні та мати зрозумілий та інтуїтивний інтерфейс для користувачів.

3.2 UML проєктування програмної системи

Система має клієнт-серверну архітектуру, що складається з серверної та безлічі клієнтських компонентів. Сервер надає послуги для клієнтських компонентів, на запит яких він відповідає, а також продовжує відслідковувати їхні запити (рисунок 3.1).

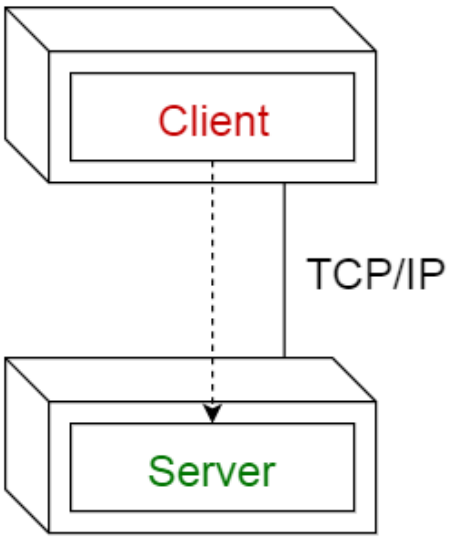

Рисунок 3.1 – Шаблон «Клієнт-сервер»

Клієнт-серверна архітектура є ефективним рішенням для проектування програмних систем, оскільки вона дозволяє забезпечити ефективну взаємодію між компонентами та підвищити масштабованість системи.

Це може бути особливо корисно у випадках, коли необхідно забезпечити доступ до програмної системи для великої кількості користувачів. Додавання додаткових серверних компонентів може забезпечити розподіл навантаження та підвищити продуктивність системи.

Зважаючи на описану вище систему управління проектами, можна скласти

діаграму варіантів використання. Нижче подано приклад діаграми варіантів використання для цієї системи (рисунок 3.2):

Рисунок 3.2 – Діаграма варіантів використання [9]

Опис кожного з варіантів використання:

– зареєструватись: користувач може зареєструватись в системі, введенням своїх особистих даних та інформації про обліковий запис.

– увійти: зареєстрований користувач може увійти в систему, введенням свого імені користувача та пароля.

– створити проект: залогінений користувач може створювати нові проекти, вводячи інформацію про назву та опис проекту.

– переглянути проекти: користувач може переглядати список всіх проектів, до яких він має доступ.

– додати учасника: користувач може додавати інших користувачів до свого проекту.

– переглянути деталі проекту: користувач може переглядати детальну інформацію про конкретний проект.

3.3 Проектування архітектури

На основі наведених вимог архітектура веб-середовища для розробки може мати наступний вигляд:

Система може бути розділена на три основні компоненти:

– проектування архітектури є ключовим етапом у розробці веб-середовища для мови JavaScript, архітектура повинна забезпечити гнучкість, масштабованість та ефективність нашої системи [10];

– один з основних архітектурних підходів для веб-середовища на JavaScript - це підхід клієнт-сервер, у цьому підході клієнтська частина відповідає за інтерфейс користувача, а серверна - за логіку програми та взаємодію з базою даних;

– клієнтська частина може бути розроблена за допомогою бібліотеки React, яка дозволяє створювати складні та динамічні інтерфейси користувача з використанням компонентів, для забезпечення ефективності та взаємодії з сервером можна використовувати бібліотеку Redux;

– серверна частина може бути розроблена з використанням фреймворку Node.js, який забезпечує ефективність та масштабованість за допомогою асинхронного виконання коду, для взаємодії з базою даних можна використовувати MongoDB, яка забезпечує гнучкість та швидкість в операціях з даними;

– для забезпечення безпеки можна використовувати різні методи, такі як хешування паролів, захист від SQL-ін'єкцій та XSS-атак [11].

Представлена нижче діаграма розгортання (рисунок 3.2) ілюструє архітектуру програмної системи з клієнт-серверним підходом. Ця діаграма надає інформацію про платформи та обчислювальні засоби, на яких розгортується

програмна система.

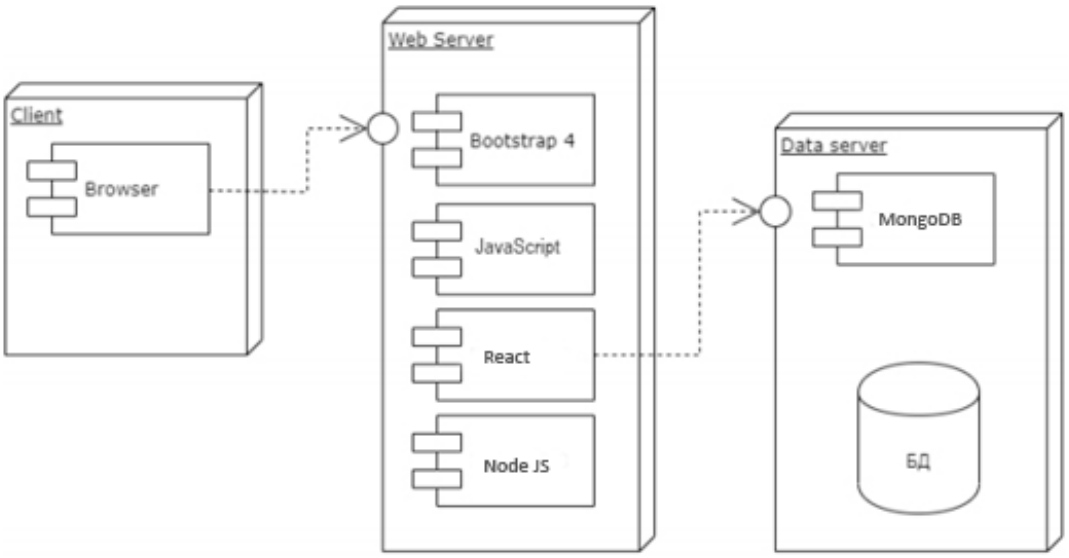

Рисунок 3.2 – Діаграма розгортання програмної системи

Діаграма розгортання повинна детально відображати всі аспекти реалізації програмної системи, тому вона є єдиним інструментом для опису архітектури системи в цілому.

Також, для забезпечення масштабованості та ефективності, можна використовувати різні підходи, такі як кешування даних, реплікацію серверів та балансування навантаження.

Узагальнюючи, проектування архітектури для веб-середовища на мові JavaScript повинно бути добре продуманим і відповідати всім вимогам, які були встановлені на етапі аналізу вимог. Важливо враховувати можливості мови JavaScript та технологій, що використовуються при розробці веб-додатків.

Під час проектування архітектури необхідно визначити ключові компоненти системи, взаємодію між ними та відповідальність кожного компонента. Також необхідно вирішити питання щодо зберігання даних, написання логіки програми та інтерфейсу користувача.

Нарешті, важливо забезпечити можливість розширення системи в майбутньому шляхом розробки модульної архітектури, що дозволить додавати нові функції та можливості без перезапуску системи.

На рисунку 3.3 наведено схем взаємодії апаратного забезпечення.

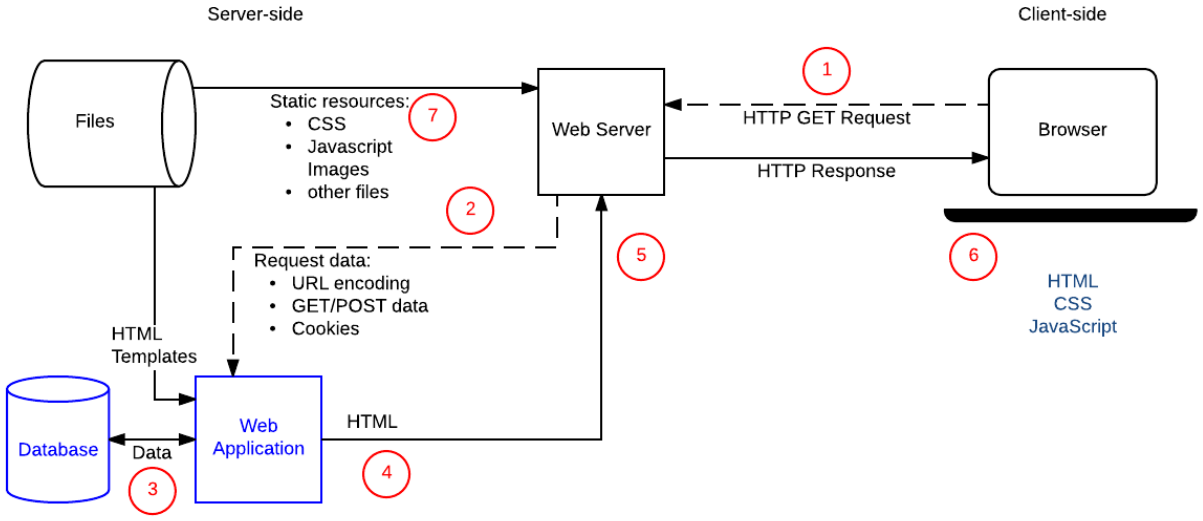

Рисунок 3.3 – Схема взаємодії апаратного забезпечення

Правильно спроектована архітектура системи дозволить забезпечити її стабільність, ефективність та можливість розширення у майбутньому [12].

3.4 Розробка бази даних

База даних має відповідати усі поставленим вище умовам. Для цього необхідно створити необхідні таблиці БД.

Таблиця користувачів (Users collection):

– _id: ObjectId (ідентифікатор користувача);
– name: String (ім'я користувача);
– email: String (адреса електронної пошти користувача);
– password: String (хеш пароля користувача);
– createdAt: Date (дата створення запису користувача);
– updatedAt: Date (дата останнього оновлення запису користувача).

Таблиця проектів (Projects collection):

– _id: ObjectId (ідентифікатор проекту);
– title: String (назва проекту);
– description: String (опис проекту);
– status: String (статус проекту, наприклад, "в роботі", "завершено");
– owner: ObjectId (посилання на користувача, який створив проект);
– createdAt: Date (дата створення проекту);
– updatedAt: Date (дата останнього оновлення проекту).

Таблиця прав доступу (Access Control collection):

– _id: ObjectId (ідентифікатор прав доступу);
– user: ObjectId (посилання на користувача, якому надано доступ);
– project: ObjectId (посилання на проект, до якого надано доступ);
– accessLevel: String (рівень доступу, наприклад, "читання", "редагування").

Таблиця історії (History collection):

– _id: ObjectId (ідентифікатор запису історії);
– project: ObjectId (посилання на проект, в якому було зроблено зміни);
– user: ObjectId (посилання на користувача, який зробив зміни);
– action: String (опис дії, наприклад, "доданий новий файл");
– createdAt: Date (дата та час, коли були зроблені зміни).

Таблиця користувачів дозволяє зберігати інформацію про користувачів, такі як їхні імена, адреси електронної пошти та паролі. Ці дані використовуються для автентифікації користувачів та надання їм доступу до різних функцій веб-програми.

Таблиця проектів зберігає інформацію про проекти, таку як назви, описи та статуси. Вона використовується для управління життєвим циклом проекту, від створення до завершення [13].

Таблиця прав доступу використовується для керування доступом користувачів до проектів. Вона визначає, які дії можуть виконувати користувачі у межах кожного проекту.

Таблиця історії змін використовується для відстеження всіх змін, зроблених у проектах. Ця таблиця зберігає інформацію про всі дії, зроблені користувачами, та може використовуватись для відновлення попередніх версій проектів.

На рисунку 3.2 зображено схему бази даних.

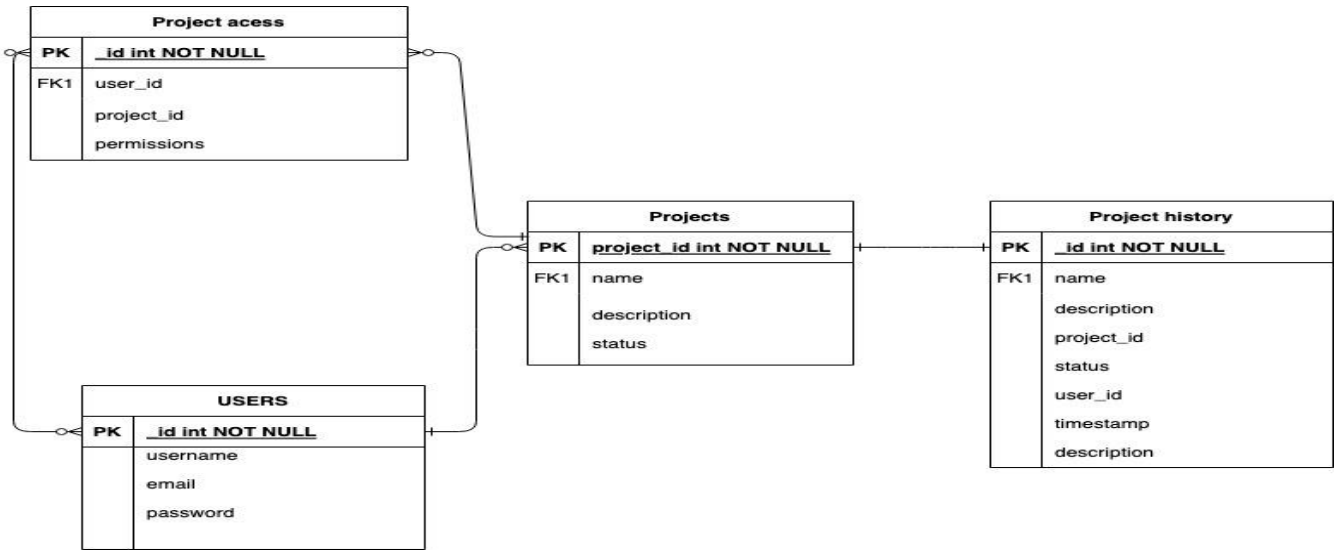

Рисунок 3.4 – Схема бази даних

В проекті ми використовуватимемо MongoDB як базу даних для зберігання інформації про користувачів, їх авторизацію та ролі. MongoDB є документ-орієнтованою базою даних, яка зберігає дані у вигляді документів у форматі BSON (Binary JSON), що дозволяє більш ефективно і швидко зберігати та обробляти великі обсяги даних.

3.5 Приклади алгоритмів та методів

Існує кілька алгоритмів та технік, які можуть бути використані для покращення ефективності та швидкодії додатку. Ось декілька прикладів:

Кешування для зменшення часу відгуку сервера та покращення продуктивності додатку

Використання бібліотеки node-cache для зберігання даних в оперативній пам'яті:

```
const NodeCache = require("node-cache");
const cache = new NodeCache();
// A function that retrieves data from the cache or database if it is not in the cachefunction
getDataFromCacheOrDatabase(key) {
	let data = cache.get(key);
	if (data == undefined) {
// Retrieving data from the database and storing it in the cache
		data = getDataFromDatabase(key);
		cache.set(key, data);
	}
	return data;
}
```

Хешування паролів (Password hashing):

```
const bcrypt = require("bcrypt");

// Hashing the user's password before saving it to the database
const saltRounds = 10;
bcrypt.hash(myPlaintextPassword, saltRounds, function(err, hash) {
// Store the hashed password in the database
});

// Validate the user's password during login
bcrypt.compare(myPlaintextPassword, hash, function(err, result) {
	if (result == true) {
	// Password is correct
	} else {
	// The password is incorrect
	}
});
```

Компресія даних (Data compression):

```
const zlib = require("zlib");
// Compress data before saving to database
const data = "Lorem ipsum dolor sit amet, consectetur adipiscing elit.";
```

```
zlib.deflate(data, function(err, compressedData) {
// Save the compressed data to the database
});

// Unpack the data after getting it from the database
zlib.inflate(compressedData, function(err, data) {
// Data Processing
});
```

Кластеризація даних (Data clustering):

```
// Incoming data
const data = [[1, 2], [2, 4], [5, 6], [5, 7], [6, 8], [7, 9]];
// Set the parameters of the algorithm
const options = {
   k: 2, // number of clusters
}
// Initialize the algorithm
const kmeans = new KMeans(options);
// Run the algorithm
const result = kmeans.cluster(data);
// Output the result
console.log(result);
```

3.6 Розробка інтерфейсу користувача

Для інтерфейсу нашого проекту ми повинні забезпечити простий та інтуїтивно зрозумілий інтерфейс для користувачів, щоб вони могли легко писати та налагоджувати свій код. Ось деякі з можливих функцій, які ми можемо включити до нашого інтерфейсу:

– редактор коду з підсвічуванням синтаксису;
– автодоповнення та підказки;
– можливість завантажувати сторонні бібліотеки;
– панель для відображення помилок та попереджень;

– можливість компілювати та запускати код з інтерфейсу;

– інтерактивна консоль для виведення результатів та введення даних.

Інтерфейс користувача веб-середовища зображена на рисунку 3.5.

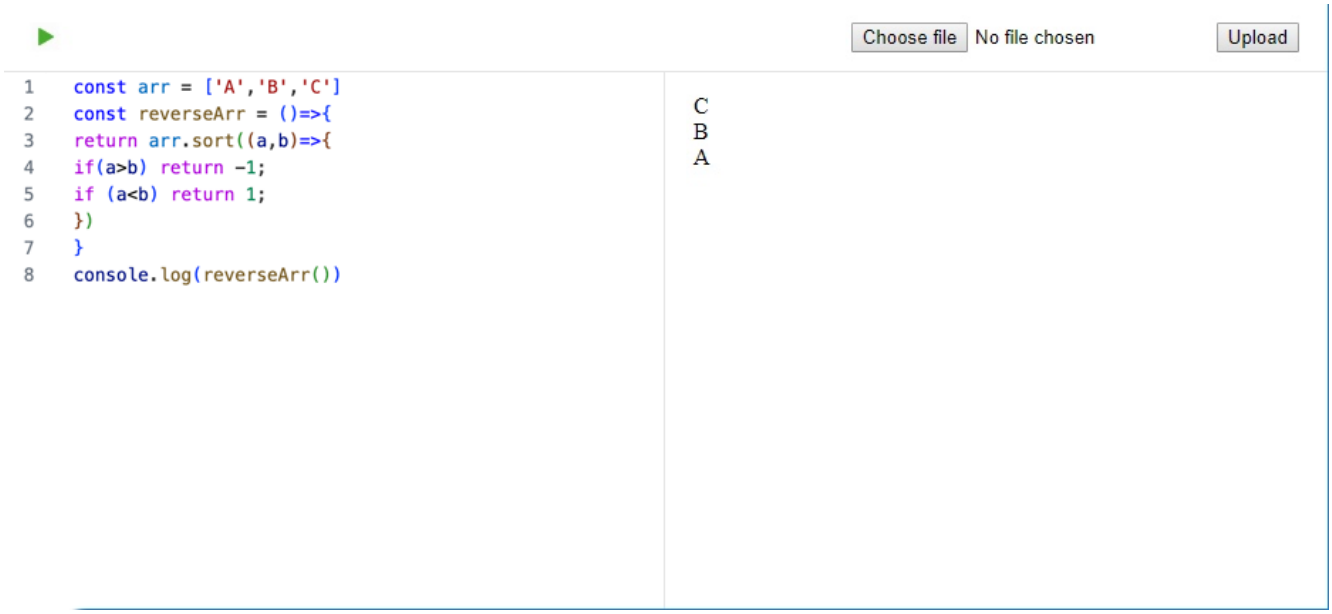

Рисунок 3.5 – Інтерфейс користувача веб-середовища

Ми також маємо переконатися, що наш інтерфейс легко масштабується і може бути легко розширений новими функціями у майбутньому.

Крім того, ми можемо включити можливість зберігати та завантажувати проекти, щоб користувачі могли легко зберігати свій код та працювати з ним у майбутньому. Ми також можемо включити можливість зберігати та завантажувати різні версії коду, щоб користувачі могли легко переходити між різними версіями свого проекту. Для зручності користувачів можна додати можливість встановлювати плагіни та розширення, які розширять можливості нашого інтерфейсу та зроблять його більш корисним та функціональним.

3.7 Розробка логіки програми

Розробка логіки додатку передбачає декілька кроків, в рамках яких будуть створені та втілені наступні програмні блоки:

– на стороні сервера, використовуючи Node.js, створюємо web-додаток, який оброблятиме HTTP-запити, використовуємо фреймворк Express для полегшення цього процесу;

– створюємо маршрути (routes) для сторінки входу та сторінки з редактором коду, а для сторінки входу створюємо HTML-сторінку з формою логіну, де користувач може ввести облікові дані, при надсиланні форми дані надсилаються на сервер для автентифікації;

– для сторінки редактора коду створюємо HTML-сторінку із двома панелями (ліва панель є текстовим редактором, де користувач може писати свій код, а права панель відображає результати коду, що користувач написав на лівій панелі);

– використовуємо фреймворк React для створення компонентів на сторінці, для цього створюємо компоненти для кожної з двох панелей, а також компонент верхньої панелі з кнопкою "зберегти" (коли користувач натискає кнопку, його код зберігається на сервері за допомогою API-запиту);

Для забезпечення швидкості та ефективності системи важливо використовувати підходи, такі як кешування, оптимізація запитів до бази даних та мінімізація завантаження сторінок.

3.8 Тестування та відлагодження

Тестування та налагодження є невід'ємним етапом у розробці нашої програми. До того, як користувачі зможуть почати використовувати її, необхідно пройти повний цикл тестування. Особливу увагу ми приділимо тестуванню навантаження, оскільки алгоритми виявлення потребують значних обчислювальних ресурсів. Для збору та анотування даних для додаткового навчання алгоритму виявлення, ми запросимо волонтерів на етапі відлагодження.

3.8 Розгортання та підтримка

Розгортання та підтримка нашого проекта включає в себе наступні кроки: налаштування сервера та бази даних, розгортання коду на сервері, налаштування конфігурації, розгортання залежностей, підтримка веб-сайту, внесення змін в код, виправлення помилок та надання технічної підтримки.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

У програмній системі веб-середовища розробки для Javascript на мові Javascript, присутні програмні рішення на які варто звернути увагу та ознайомитись, з принципом їх функціонування.

Ось приклад коду, який використовує фреймворк React.js, Node.js, Express.js, MongoDB та Webpack:

```
// server.js

const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const path = require('path');
const app = express();
const port = process.env.PORT || 5000;
// Connect to MongoDB
mongoose.connect('mongodb://localhost/my_database', { useNewUrlParser: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));
// Bodyparser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
// API routes
const items = require('./routes/api/items');
app.use('/api/items', items);
// Serve static assets if in production
if (process.env.NODE_ENV === 'production') {
  // Set static folder
  app.use(express.static('client/build'));
  app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'));
  });
```

```
}

// Start the server
app.listen(port, () => console.log(`Server started on port ${port}`));
```

server.js - це файл, який містить основний серверний код на Node.js та Express.js. У цьому файлі налаштовується з'єднання з базою даних MongoDB, обробляються запити від клієнта та реагується на них відповідним чином. Також в цьому файлі налаштовується сервер на віддачу статичних файлів у випадку, якщо клієнтська частина знаходиться в режимі продакшн.

Далі наведено приклад файлу routes/api/items.js - цей файл містить API-маршрути, які відповідають за обробку запитів до бази даних MongoDB. У цьому файлі описані два маршрути: GET-маршрут для отримання всіх елементів з бази даних та POST-маршрут для створення нового елементу в базі даних.

```
// routes/api/items.js

const express = require('express');
const router = express.Router();

// Item model
const Item = require('../../models/Item');

// @route GET api/items
// @desc Get all items
// @access Public
router.get('/', (req, res) => {
  Item.find()
    .sort({ date: -1 })
    .then(items => res.json(items));
});
// @route POST api/items
// @desc Create an item
// @access Public
```

```
router.post('/', (req, res) => {
  const newItem = new Item({
    name: req.body.name
  });
  newItem.save().then(item => res.json(item));
});
module.exports = router;
```

Останній приклад client/src/App.js - цей файл містить клієнтський код на React.js. У цьому файлі описано класовий компонент App, який містить у собі форму для додавання нового елементу в базу даних та список всіх елементів, які були додані до бази даних. Компонент отримує список елементів з серверу за допомогою AJAX-запиту до API-маршруту GET /api/items та відправляє новий елемент до серверу за допомогою AJAX-запиту до API-маршруту POST /api/items.

```
// client/src/App.js
import React, { Component } from 'react';
import axios from 'axios';
class App extends Component {
  state = {
    items: [],
    name: ''
  };
  componentDidMount() {
    this.getItems();
  }
  getItems = () => {
    axios.get('/api/items').then(res => this.setState({ items:
res.data }));
  };
  onChange = e => {
    this.setState({ [e.target.name]: e.target.value });
  };
  onSubmit = e => {
    e.preventDefault();
```

```
      const newItem = {
        name: this.state.name
      };
      axios.post('/api/items', newItem).then(res => {
        this.setState({ name: '' });
        this.getItems();
      });
    };

    render() {
      const { items, name } = this.state;
      return (
        <div>
          <h1>Items</h1>
          <form onSubmit={this.onSubmit}>
            <input type="text" name="name" value={name}
onChange={this.onChange} />
            <button type="submit">Add Item</button>
          </form>
          <ul>
            {items.map(item => (
              <li key={item._id}>{item.name}</li>
            ))}
          </ul>
        </div>
      );
    }
  }
  export default App;
```

Перевагою цієї реалізації є те, що вона дозволяє створити повноцінний веб-додаток з використанням сучасних технологій та архітектурних підходів. React.js забезпечує зручну та ефективну роботу з клієнтською частиною додатку, Express.js дозволяє швидко та зручно налаштувати серверну частину, MongoDB забезпечує швидкий доступ до даних, а Webpack дозволяє зручно налаштувати та зібрати весь клієнтський код в один файл для продакшн-режим

5 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

При розробці було вирішено скористатися двома методами тестування, які допомогли знизити витрати на тестування, не постраждаючи якість кінцевого програмного забезпечення.

5.1 Підхід 1: Тестуванням займається розробник або менеджер

Даний підхід до тестування програмної системи можна визнати складним для виділення окремо, оскільки насправді він передбачає відсутність професійного тестування. Замість професіонала, який надає об'єктивну оцінку проекту, сам розробник або менеджер відповідають за тестування. Часто такий підхід обумовлений кількома причинами, такими як обмежений бюджет, бажання зекономити кошти, неможливість найняти професійного тестувальника або просто нерозуміння важливості тестування.

Недоліком такого підходу є його вплив на якість тестування. Розробник має "замилені очі" і перевіряє лише ті функції та сценарії, які він сам задумав і реалізував. Він не приділяє достатньо уваги всім сценаріям, які призначені для користувача. Це може стати причиною неузгодженості між очікуваннями користувача та фактичною роботою системи.

Щоб компенсувати втрати якості програмної системи, також можна використовувати комбінацію різних підходів до тестування. Наприклад, можна використовувати самостійне тестування розробниками та менеджерами на ранніх етапах розробки, але при цьому залучати професійного тестувальника на пізніших етапах для оцінки кінцевої якості програмного забезпечення та виявлення глобальних помилок.

5.2 Підхід 2: тестувальника наймають на стадії завершення

Цей підхід полягає у тому, що професійний тестувальник наймається тільки на завершальній стадії розробки або на кожному етапі. Цей метод є типовим для каскадної моделі розробки програмного забезпечення, де проект розділяється на етапи. Проте, цей підхід має свої мінуси, такі як те, що істотні помилки на рівні архітектури можуть бути знайдені занадто пізно, коли проект вже на завершальній стадії, що може призвести до переписування великої кількості коду або переробки проекту з самого початку.

Також, вільне інтуїтивне тестування може призвести до ігнорування ряду альтернативних сценаріїв для користувача. Крім того, зміни, які відбуваються під час розробки, можуть ускладнити розбір баг-репортів, оскільки початкова документація може вже не відповідати кінцевим результатам.

Отже, цей підхід може здатися на перший погляд логічним, але не враховує можливість виявлення глобальних помилок на ранніх етапах розробки.

5.3 Методи тестування програмного забезпечення

Вибір методу тестування програмного забезпечення залежить від багатьох факторів, таких як тип програмного забезпечення, його складність та обсяг, доступні ресурси та знання команди тестувальників.

Метод "чорного ящика" є досить популярним і ефективним методом тестування, коли внутрішня структура програмного забезпечення невідома тестувальникам. Цей метод дозволяє перевірити функціональність програми, що важливо для користувачів та замовників. Крім того, вимоги і специфікація можуть бути усно описані та, незалежно від наявності письмової документації, тестувальник може спиратися на ці описи при складанні тест-кейсів.

Також використання методу "чорного ящика" може допомогти у виявленні неочевидних помилок, що можуть залишатися невиявленими при використанні

інших методів тестування, які базуються на знанні внутрішньої структури програмного забезпечення.

Таким чином, вибір методу "чорного ящика" для тестування програмного забезпечення може бути доцільним з різних поглядів, таких як ефективність, доступність ресурсів, вимог користувачів та замовників, складність програмного забезпечення та знання команди тестувальників.

Основні переваги використання тестування чорним ящиком:

– відсутність необхідності знання деталей роботи програми, що зменшує час та зусилля, необхідні для тестування.
– тестування чорним ящиком дозволяє виявляти помилки та недоліки в роботі програми на основі зовнішніх вимог та очікувань користувачів.
– цей метод тестування дозволяє проводити велику кількість тестів у короткий проміжок часу, що підвищує ефективність тестування та сприяє забезпеченню якості продукту.

Таким чином, ми використовуємо тестування чорним ящиком для забезпечення якості нашої програмної системи, що дозволить нам зосередитися на тестуванні функціональності, яку відчуває користувач, а не на внутрішніх деталях роботи системи.

Крім того, тестування чорним ящиком може бути виконане не тільки на рівні одиниць програмного забезпечення, але й на рівні всієї системи. Це дозволяє виявляти помилки та недоліки взаємодії різних компонентів системи та їх інтеграції.

Нарешті, тестування чорним ящиком може бути більш придатним для випробування систем, що є складними та змінними.

5.4 Тест-кейси

Ключові елементи веб-середовища розробки на JavaScript, які можна протестувати включають:

– функціональність редагування коду: можливість внесення змін до коду, збереження та перевірка правильності результату.
– функціональність збереження проектів: можливість зберігати та відкривати розроблені проекти.
– функціональність відлагодження коду: можливість встановлювати точки зупинки, переглядати значення змінних та інші дії для відлагодження коду.

Нижче наведено приклад тест-кейсів для тестування веб-середовища розробки на JavaScript з використанням методу тестування чорного ящика:

Тестування функціональності редагування коду:

1. Переконатися, що можливо відкрити файл з кодом та внести зміни.
2. Переконатися, що можливо зберегти зміни.
3. Переконатися, що можливо перевірити правильність результату.

Тестування функціональності збереження проектів:

1. Переконатися, що можливо створити новий проект та зберегти його.
2. Переконатися, що можливо відкрити збережений проект.

Тестування функціональності відлагодження коду:

1. Переконатися, що можливо встановити точку зупинки в коді.
2. Переконатися, що можливо переглянути значення змінних.
3. Переконатися, що можливо продовжити виконання коду після точки зупинки.

У кожному тест-кейсі варто вказати деталі тесту, зокрема очікуваний результат, умови проведення тесту, інструкції та коментарі до тесту.

ВИСНОВКИ

Під час виконання нашого проекту був проведений аналіз методів розробки веб-сервісів та вибрали веб середовища для JavaScript.

В результаті проведеного аналізу ми прийняли чотири прогресивні програмні рішення, які відповідають нашим вимогам та можуть забезпечити оптимальну роботу нашого веб-сервісу. Зокрема, ми вирішили використовувати Node.js для реалізації серверної частини нашого веб-сервісу. Для створення інтерфейсу користувача ми обрали React.js. Для зберігання даних ми використовуватимемо MongoDB.

Наша архітектура програмної системи базується на моделі клієнт-сервер, де клієнтська частина буде відповідати за відображення інтерфейсу користувача, а серверна частина - за обробку запитів, збереження даних та передачу відповідей. Ми також обрали REST-архітектуру для забезпечення стандартизованого та ефективного обміну даними між клієнтом та сервером.

За результатами нашої роботи ми можемо зробити висновок, що використання веб-середовищ для JavaScript дозволяє створювати високоякісні веб-сервіси з відповідною швидкодією та безпекою. Добре підібрані технології та архітектура програмної системи забезпечують відмінну роботу веб-сервісу та задовольняють вимоги бізнесу.

Проведена розробка архітектури програмного забезпечення. Під час якої користуючись водоспадною (або каскадною) моделлю побудови багаторівневого процесу розробки сплановані етапи затвердження вимог, проектування, розробки, тестування, розгортання та підтримки продукту.

В процесі роботи розглянут проект з попередньою назвою «JSWave» який допоможе розробникам спростити та прискорити процес створення веб-додатків, забезпечуючи зручний інтерфейс для написання та налагодження коду, автоматичну перевірку помилок та підказки, а також доступ до різних бібліотек та фреймворків для покращення функціональності додатків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. JavaScript: The Definitive Guide / URL: https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/ (дата звернення: 10.03.2023).

2. Eloquent JavaScript / URL: https://eloquentjavascript.net/ (дата звернення: 12.03.2023).

3. MDN Web Docs / URL: https://developer.mozilla.org/en-US/docs/Web (дата звернення: 15.03.2023).

4. Introduction. Bootstrap · The most popular HTML, CSS, and JS library in the world / URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/ (дата звернення: 25.03.2023).

5. React Documentation / URL: https://reactjs.org/docs/getting-started.html (дата звернення: 19.03.2023).

6. "Node.js Documentation / URL: https://nodejs.org/en/docs/ (дата звернення: 16.03.2023).

7. Learning JavaScript Design Patterns Едді Османі. 2012. С 35-75.

8. Node.js Design Patterns Mapio Каскьяро. 2014. С 120-150.

9. UML Distilled: A Brief Guide to the Standard Object Modeling Language Martin Fowler. 2013. С 55-57.

10. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures Zhamak Dehghani, Neal Ford, Rebecca Parsons, Patrick Kua. 2021. С 120-125.

11. SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management Alexandru M. Meier, Martin Kaufmann. 2019. С 78-99.

12. Software Engineering: A Practitioner's Approach Roger S. Pressman. 2014. С 56-65.

13. Perfect Scale Andreas Schulz, Birgit Schulz 2015 С 23-33