

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

МЕТОДІВ ВІДСТЕЖЕННЯ ВЕРСІЙ

АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ

RESEACRH AND SOFTWARE IMPLEMENTATION OF
METHODS FOR TRACKING THE HARDWARE VERSIONS

Виконав: студент(ка) 2 курсу, групи КНТ-129м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Кикоть В.І.

(прізвище та ініціали)

Керівник Колпакова Т.О.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (КОД І НАЙМЕНУВАННЯ)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Кикотя Владислава Ігоровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів відстеження версій апаратного забезпечення. Research and Software Implementation of Methods for Tracking the Hardware Versions
- керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
- затверджені наказом закладу вищої освіти від “03” листопада 2020 року № 301
2. Строк подання студентом проєкту (роботи) 5 грудня 2020 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдання дослідження. 2. Матеріали та методи. 3. Проектування програмного застосунку. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми 6. Організаційно-економічна частина. 7. Охорона праці та безпека у надзвичайних ситуаціях. 8. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Колпакова Т.О., доцент		
Організаціо-економічна частина	Пожуєва Т.О., професор		
Охорона праці та цивільна безпека	Коробко О.В., ст. викладач		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “ 2 ” вересня 2020 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області.	1 тиждень	Розділ 1
2	Постановка завдання.	2 тиждень	ТЗ
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	3–4 тижні	Розділ 2
4	Проектування програмного застосунку.	5 тиждень	Розділ 3
5	Основні рішення щодо реалізації компонентів системи	6 тиждень	Розділ 4
6	Розробка програмного коду програми.	7–8 тижні	Текст програми
7	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5-6
8	Розробка організаційно-економічної частини	10 тиждень	Розділ 7
9	Розробка заходів з охорони праці та безпека у надзвичайних ситуаціях.	11 тиждень	Розділ 8
10	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
11	Нормоконтроль та рецензування.	14–15 тижні	
12	Захист дипломної роботи магістра.	16 тиждень	

Студент(ка)

(підпис)

Кикоть В.І.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Колпакова Т.О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
127 с., 12 табл., 36 рис., 3 дод., 42 джерела.

ПОРІВНЯННЯ TEKCTY, TYPESCRIPT, WEBASSEMBLY, NODEJS,
NESTJS, REACT, REDUX, MVC, FLUX.

Об'єкт дослідження: дослідження та програмна реалізація методів відстеження апаратного забезпечення.

Предмет дослідження: програмний застосунок для відстеження версій апаратного забезпечення.

Мета роботи: розробка програмного застосунку для відстеження версій апаратного забезпечення та удосконалення життєвого циклу апаратного забезпечення шляхом впровадження таск-менеджменту, а також удосконалення алгоритму порівняння текстових фрагментів шляхом оптимізації під поточну платформу.

Матеріали та методи: для успішної реалізації поставленої мети роботи використовуються найкращі методи вирішення питання та сучасний підхід імплементації рішень шляхом оптимізації під поточну платформу.

Технічні засоби: для дослідження та програмної реалізації буде використовуватися ноутбук MacBook Pro 15 2019.

Результати. Створено програмний вебзастосунок, що складається з серверної та клієнтської частин та дозволяє створювати проєкти апаратного забезпечення, команди розробників, компоненти проєкту, додавати розробників до компоненту та відстежувати версії специфікації.

Висновки. Розроблено програмний застосунок для відстеження версій апаратного забезпечення, який дозволяє порівнювати текстові фрагменти за допомогою оптимізованого під поточну платформу алгоритму Myers.

Галузь використання – підприємства, що займаються розробкою апаратного забезпечення.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 127 pages, 36 figures, 12 tables, 3 appendixes, 42 sources.

TEXT COMPARISON, TYPESCRIPT, WEBASSEMBLY, NODEJS, NESTJS, REACT, REDUX, MVC, FLUX.

Object of study is a research and software implementation of methods for tracking the hardware versions.

Subject of study is a software application for tracking the hardware versions.

The purpose of the work is to create a software application for tracking the hardware versions and improving the life cycle of hardware by adding the task management system and improving the algorithm of text comparison by optimization under the current platform are used.

Materials and methods: for successful realization of the purpose of work the best methods of the decision of a question and the modern approach of implementation of decisions by optimization under the current platform are used.

Tools: the MacBook Pro 15 2019 laptop.

Results. the software WEB-application was created, consisting of server and client parts and allows to create hardware projects, development teams, project components, add developers to the component and track versions of the specification.

Conclusions. A software application for tracking the hardware versions has been developed, which allows to compare text fragments using the Myers algorithm optimized for the current platform.

The field of use is enterprises engaged in hardware development.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	9
Вступ	10
1 Аналіз предметної області та постановка завдання	11
1.1 Концепція систем управління життєвим циклом апаратного забезпечення та менеджменту задач	11
1.2 Аналіз життєвого циклу апаратного забезпечення	13
1.3 Огляд методів версійності та збереження історії даних	17
1.4 Огляд методів порівняння текстових фрагментів	20
1.5 Постановка завдання дослідження.....	23
1.6 Висновок за розділом 1	24
2 Матеріали та методи.....	25
2.1 Впровадження таск-менеджменту до моделі життєвого циклу апаратного забезпечення.....	25
2.2 Інтеграція та оптимізація алгоритму порівняння текстових фрагментів Myers	26
2.3 Висновки за розділом 2	33
3 Проєктування програмного застосунку.....	34
3.1 Вибір стеку технологій серверної частини	34
3.2 Проєктування архітектури серверної частини.....	40
3.3 Вибір стеку технологій клієнтської частини.....	42
3.4 Проєктування архітектури	45
3.5 Висновки за розділом 3	46
4 Основні рішення щодо реалізації компонентів системи.....	47
4.1 Основні модулі бази даних	47

4.2 Основні модулі серверної частини.....	50
4.3 Основні модулі клієнтської частини.....	52
4.4 Висновки до розділу 4.....	54
5 Експлуатація, тестування та експериментальне дослідження програми	55
5.1 Опис застосування програми.....	55
5.2 Інструкція з експлуатації програми	56
5.3 Методика та результати тестування	62
5.4 Експериментальні дослідження та отримані результати.....	63
5.5 Аналіз, узагальнення та обґрунтування достовірності отриманих результатів.....	64
5.6 Висновки за розділом 5	65
6 Організаційно-економічна частина.....	66
6.1 Опис ідеї проєкту.....	66
6.2 Аналіз ринкових можливостей запуску реалізації ідеї	67
6.3 SWOT-аналіз	69
6.4 Цільові групи потенційних споживачів.....	70
6.5 Планування дослідження та розробки програмного продукту	70
6.6 Визначення витрат на розробку	73
6.7 Розрахування капітальних вкладень	82
6.8 Оцінка економічної ефективності	84
7 Охорона праці та безпека у надзвичайних ситуаціях.....	86
7.1 Аналіз потенційних небезпек	86
7.2 Заходи із забезпечення безпеки.....	87
7.3 Заходи з забезпечення виробничої санітарії та гігієни праці	89
7.4 Заходи з забезпечення безпеки у надзвичайних ситуаціях.....	94

Висновки.....	99
Перелік джерел посилання.....	101
Додаток А Технічне завдання.....	105
Додаток Б Текст програми.....	111
Додаток В Слайди презентації	121

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
IDE	– Integrated Development Environment;
JS	– JavaScript;
LCS	– Longest Common Subsequence;
MVC	– Model-View-Controller;
ORM	– Object-relational mapping;
PLM	– Product Lifecycle Management;
SCD	– Slowly Changing Dimensions;
SQL	– Structured query language;
TS	– TypeScript;
UI	– User Interface;
UX	– User Experience;
БД	– база даних;
ПЗ	– програмне забезпечення;
СКБД	– система керування базами даних.

ВСТУП

Сьогодні виробництво апаратного забезпечення займає важливу роль у розвитку технологій та економіки. З ростом потреби апаратного забезпечення ростуть і вимоги до нього, а отже і складність його виробництва. Для планування, розробки та ведення документації апаратного забезпечення використовуються різні програмні застосунки, що надають той чи інший функціонал. Використання таких систем на виробництві дозволяє сформувати команду розробників, розподілити між ними задачі, контролювати, аналізувати, а також формалізувати процес розробки апаратного забезпечення, що в свою чергу підвищує ефективність виробництва та підвищує конкурентоспроможність підприємства.

Зазвичай такі програмні застосунки складаються з таких частин:

- база даних (БД) – сховище даних у якому закладена бізнес логіка програмного застосунку;
- серверна частина – частина застосунку, що відповідає за обробку даних та їхній зв'язок з БД та клієнтською частиною;
- клієнтська частина – частина програмного застосунку, що відповідає за надання інтерфейсу користувачеві для взаємодії з даними та бізнес логікою програмного застосунку.

За мету магістерської дипломної роботи було обрано розробку програмного застосунку, що удосконалює життєвий цикл апаратного забезпечення шляхом додавання таск-менеджменту, дозволяє створювати проєкт апаратного забезпечення, формувати команду, створювати компоненти апарату та залучати учасників команди до його розробки, а також вести специфікацію компонентів з можливістю перегляду минулих версій та їхнього порівняння з поточною версією за допомогою оптимізованого під поточно платформу алгоритму порівняння текстових фрагментів та наглядною демонстрацією результату його роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Концепція систем управління життєвим циклом апаратного забезпечення та менеджменту задач

1.1.1 Системи управління життєвим циклом апаратного забезпечення

Управління життєвим циклом апаратного забезпечення (product lifecycle management, PLM) – сукупність процесів, які виконуються від моменту виявлення потреб суспільства в певній продукції до моменту задоволення цих потреб і утилізації продукту [1].

Дана концепція направлена на управління всією інформацією про виріб і зв'язаних з ним процесах, протягом всього його життєвого циклу.

Управління процесами життєвого циклу апаратного забезпечення є складним завданням, для її вирішення застосовують цілий набір автоматизованих систем управління. Саме тому, важливими функціями технології PLM є формування інформаційного простору, єдиного для різних автоматизованих систем, що використовуються на підприємстві, об'єднання методик і засобів інформаційної підтримки виробів протягом усіх стадій життєвого циклу. Концепція управління життєвим циклом апаратного забезпечення має на увазі забезпечення взаємодії не тільки засобів автоматизації різних виробників на одному підприємстві, а й забезпечення взаємодії різних автоматизованих систем багатьох підприємств [1].

Можна виділити наступні ключові переваги використання систем управління життєвого циклу апаратного забезпечення:

- скорочення термінів проєктування і виробництва;
- скорочення кількості помилок;
- посилення контролю за якістю;
- скорочення витрат;
- супровід інтелектуальної власності підприємства.

Застосування концепції PLM у складному багатоопераційному виробництві, характерному для підприємств, що створюють апаратне забезпечення, стає необхідною умовою підвищення конкурентоспроможності підприємства. Ця концепція дозволяє відстежувати кожну партію, а при необхідності кожен екземпляр, продукції на всіх етапах життєвого циклу апаратного забезпечення, за рахунок чого підвищується якість продукції, що випускається і ступінь задоволеності замовника [1].

1.1.2 Системи управління задачами

Система управління задачами – це інструмент для управління проектами, які значно спрощують та оптимізують процес розробки програмного чи апаратного забезпечення.

Дана концепція направлена на задоволення наступних критеріїв:

- спрощення постановки задач;
- зручність планування стеку задач;
- можливість створення структури компанії та її відділів;
- розмежування ієрархії працівників та відділів;
- прозорий робочий процес;
- повнота картини про прогрес та результати роботи над задачами.

Однією з найбільш поширених можливостей даної концепції є можливість планування подій і управління завданнями. Вимоги можуть відрізнятися в залежності від того, як використовується інструмент. Найбільш поширеними є [2]:

- планування різних подій, що залежать одна від одної;
- планування розкладу роботи співробітників і управління ресурсами;
- розрахунок часу, необхідного на рішення кожної з задач;
- сортування завдань залежно від термінів їх завершення;
- управління декількома проектами одночасно.

Програмне забезпечення для управління проєктами надає велику кількість необхідної інформації, такої як [2]:

- список завдань для співробітників та інформацію про розподіл ресурсів;
- огляд інформації про терміни виконання завдань;
- ранні попередження про можливі ризики, що пов'язані з проєктом;
- інформація про робоче навантаження;
- інформація про хід проєкту, показники і їх прогнозування.

Застосування концепції системи управління задачами в розробці будь-якого програмного чи апаратного забезпечення є необхідністю, оскільки правильно організований процес розробки ставить перед працівниками чіткі цілі та завдання, що в свою чергу підвищують швидкість та якість виробництва, а також, якість комунікації в середині команди, що також зменшує ризики непорозумінь в ході виконання взаємопов'язаних задач.

1.2 Аналіз життєвого циклу апаратного забезпечення

Під PLM-системою розуміється не один продукт, а сукупність різних програмних продуктів, в тому числі від різних розробників та постачальників. Хоча потрібно зауважити, що іноді підприємства витрачають ресурси на створення власних програмних продуктів, що заточені під специфіку виробництва, та компонують в собі функціонал декількох незалежних програмних продуктів.

Системи життєвого циклу апаратного забезпечення повинні вирішувати такі задачі, як створення інженерних даних, так і управління цими даними. Система повинна мати змогу обмінюватися даними з системою управління проєктами, а також, при необхідності, з інформаційними системами замовника, або з іншими підприємствами, що зв'язані з даним підприємством.

Основні стадії життєвого циклу апаратного забезпечення: проєктування, виробництво, реалізація, технічна експлуатація, утилізація. В залежності від

складності виробництва та апаратного забезпечення, при більш детальному розгляді виділяють від 11 до 13 стадій життєвого циклу апаратного забезпечення [1]:

- маркетинг і вивчення ринку;
- проектування і розробка продукту;
- планування і підготовка виробництва;
- закупівля матеріалів та комплектуючих;
- виробництво та надання послуг;
- упаковка і зберігання;
- реалізація;
- установка (монтаж) і введення в експлуатацію;
- технічна підтримка та обслуговування;
- сервісне обслуговування (технічне обслуговування, ремонт та експлуатація), діяльність або експлуатація;
- утилізація та переробка.

Нижче розглянуто основні життєві цикли апаратного забезпечення, що присутні на будь-якому такому виробництві.

1.2.1 Проектування

Ця стадія передбачає проведення таких етапів, як [3]:

- дослідження та обґрунтування розробки – стадія життєвого циклу продукції від виникнення задуму до обґрунтування можливості й доцільності створення виробів і матеріалів;
- розроблення продукції – стадія життєвого циклу продукції, яка полягає у зміні стану продукції – від формулювання вимог технічного завдання щодо виконання дослідно-конструкторських чи науково-дослідних робіт на створення (модернізацію) продукції до втілення їх у нових (модернізованих) дослідних зразках, у нових (модифікованих) матеріалах;

– підготовлення виробництва продукції – сукупність заходів, що забезпечують готовність підприємства до освоєння виробництва продукції необхідної якості та у визначеному обсязі випуску і включає конструкторське підготовлення виробництва, а також технологічне підготовлення виробництва.

Конструкторське підготовлення виробництва – це процес створення комплекту конструкторської документації, необхідної для виготовлення та експлуатації виробу (виробів) [3].

Технологічне підготовлення виробництва – це сукупність заходів, яка охоплює проєктування технологічних процесів виробництва, вибір та розміщення устаткування, визначення технологічного оснащення, розроблення методів технічного контролю, нормування матеріально-технічних витрат і забезпечує випуск продукції потрібного рівня якості за встановлених термінів та обсягів випуску [3].

1.2.2 Виготовлення продукції

Виробництво продукції – процес організації та здійснення виготовлення продукції [3].

Стадію виробництва продукції укрупнено можна розділити на дві підстадії:

- підготовку промисловості до виробництва нового (вдосконаленого) виробу і його освоєння;
- власне виготовлення виробу як продукту праці.

У серійному (масовому) виробництві виріб проходить ряд послідовних етапів: економічне освоєння (доведення технологічних процесів, конструкторської документації, адаптація всіх служб тощо); стабільне (стале) виготовлення продукції; модернізація і вдосконалення споживчих властивостей продукції; зняття з виробництва.

1.2.3 Реалізація продукції

Реалізація продукції – постачання продукції, яка оплачена покупцем або на яку є згода покупця щодо оплати в майбутньому поставленої йому продукції, що включає підготовку до транспортування, транспортування, зберігання, монтаж і підготовку до експлуатації [3].

1.2.4 Експлуатація продукції

Експлуатація виробу – стадія життєвого циклу продукції, на якій реалізують, підтримують та відновляють якість виробу [3].

Стадія експлуатації виробу складається з низки етапів, серед яких власне експлуатація (застосування), включаючи обслуговування, ремонт, відновлення, тобто експлуатацію продукту з метою вияву його корисного ефекту, заради якого продукцію було створено [3].

На етапі власне експлуатації ставиться мета високоефективного використання продукції у споживача. Її реалізують шляхом розробки експлуатаційної документації і науково обґрунтованих норм витрачання запасних частин, забезпечення необхідного рівня надійності продукції [3].

1.2.4 Утилізація продукції

Утилізація продукції – комплекс технологічних операцій, спрямованих на використання як вторинної сировини продукції, що втратила свої корисні властивості. Ця стадія, також, передбачає зняття з виробництва; демонтаж та утилізацію ліквідованих знарядь праці (предметів споживання) [3].

1.3 Огляд методів версійності та збереження історії даних

При розробці баз даних (БД) для систем управління життєвим циклом апаратного забезпечення, зазвичай потрібно забезпечити підтримку версійності і збереження історії об'єктів. Наприклад, у апаратної частини можуть змінитись вимоги до технічних параметрів, що в свою чергу, може вплинути закладений час для розробки даної деталі. В багатомірному моделюванні це називається повільна зміна вимірювання (slowly changing dimension, SCD), тобто вимірювання не ключових атрибутів, які мають тенденцію змінюватися з часом. Всього існує 6 основних типів (методів) SCD, які визначають, як історія змін може бути відображена на моделі даних.

1.3.1 Тип 0

Нульовий тип (SCD0) являє собою пасивний метод, оскільки передбачається, що після першого потрапляння в базу даних, значення атрибутів такого типу ніколи більше не будуть змінюватися. Прикладом даного методу може бути серійний номер деталі.

Цей метод практично ніким не використовується, оскільки він не підтримує версійності. Даний тип використовується як нульова точка відліку для методології SCD.

1.3.1 Тип 1

Перший тип (SCD1) використовує звичайний перезапис старих даних новими. В чистому виді даний метод також не підтримує версійності і використовується лише там, де історія не потрібна, оскільки вона повністю втрачається, тобто після оновлення даних неможливо відстежити ланцюжок змін.

Хоча, потрібно зауважити, що в деяких системах керування базами даних (СКБД) для даного типу існує можливість додати обмежену підтримку версійності засобами самої СКБД або відстеженням змін через тригери. Прикладом даного функціоналу є функція Flashback query в СКБД Oracle.

Плюсами даного методу є [4]:

- відсутність надмірності даних;
- дуже проста структура даних.

Серед мінусів даного методу можна виділити відсутність збереження історії даних [4].

1.3.3 Тип 2

Другий тип (SCD2) полягає у створенні нових записів до таблиці для кожної окремої версії с додаванням поля, що містить у собі ключовий атрибут даної версії. Наприклад кожний новий запис може містити номер версії та інтервал часу, протягом якого дані запису можна вважати актуальними.

Плюсами даного методу є [4]:

- збереження повної і необмеженої історії версій;
- зручний та простий спосіб доступу до даних необхідного періоду.

Серед мінусів даного методу можна виділити те, що він[4]:

- створює надмірність або створення додаткових таблиць для збереження змінюваних атрибутів вимірювання;
- ускладнює структуру моделі даних або додає надмірність у випадках, коли аналітиці потрібно узгодження даних в таблиці фактів з конкретними версіями і при цьому даний факт може бути не узгодженим з поточною для даного факту версією історії даних. Наприклад, у виробництва змінилась адреса, а доставку деталей потрібно виконати по старим значенням.

1.3.4 Тип 3

Третій тип (SCD3) полягає у додаванні нових полів, що містять у собі значення попередніх атрибутів для підтримки історії даних. При додаванні нових даних, попередні дані перезаписуються поточними значеннями [4]-[5].

Даний тип використовується нечасто і потрібний тільки в тих ситуаціях, коли необхідно відслідковувати зміни конкретних параметрів [5].

Плюсами даного методу є [4]:

- невеликий об'єм даних;
- простий і швидкий доступ до історії даних.

Серед мінусів даного методу – обмежена історія даних [4].

1.3.5 Тип 4

Четвертий тип (SCD4) полягає у збереженні історії даних у окремій таблиці. У цьому методі основна таблиця завжди перезаписується поточними даними з перенесенням попередніх до іншої таблиці. Зазвичай даний тип використовується для аудиту змін або створення архівних таблиць [4].

Плюсами даного методу є швидка робота з поточною версією [4].

Серед мінусів даного методу можна виділити розділ однією сутності на різні таблиці [4].

1.3.6 Тип 5 (1 + 4)

П'ятий тип (SCD5) базується на міні-виміру 4 типу шляхом вбудовуванням ключа міні-вимірювання “поточного профілю” в базовому вимірюванні, яке перезаписується як атрибут 1 типу. Цей підхід названий п'ятим типом, оскільки він базується на четвертому і першому типах. Даний метод дозволяє отримати доступ до призначених в даний час значенням атрибутів міні-

вимірювання разом з іншими значеннями базового вимірювання без зв'язування через таблицю фактів [5].

1.3.7 Гібридний / Тип 6 (1 + 2 + 3)

Шостий тип (SCD6) був придуманий Ральфом Кімболом як комбінація вищезазначених методів і призначений для ситуацій, котрі дані методи не враховують, або для більшої зручності роботи з даними. Даний метод полягає у внесенні додаткової надмірності: береться за основу 2 тип, додається сурогатний атрибут для альтернативного огляду версій (3 тип), і перезаписуються одна, або всі попередні версії (тип 1).

1.4 Огляд методів порівняння текстових фрагментів

Для візуалізації змін та прогресу робочого процесу систем управління життєвим циклом апаратного забезпечення, що забезпечують підтримку версійності і збереження історії об'єктів, найкращим рішенням є використання функціоналу, що дозволяє порівнювати тестові фрагменти між собою.

Двома основними алгоритмами для порівняння текстових фрагментів являються Hunt-Szymanski алгоритм, також відомий як Hunt-McIlroy алгоритм, та Myers diff алгоритм, що також відомий під назвою An $O(ND)$ Difference Algorithm.

Дані алгоритми схожі між собою, оскільки суть роботи їх в тому, що вони знаходять найкоротші шляхи через граф, що відображає простір редагування між двома рядками тексту або текстовими файлами. Під простором редагування тут розуміється кількість вставлених чи видалених елементів текстових фрагментів, необхідних для перетворення одної послідовності в іншу. Хоча, не зважаючи на схожий принцип дані алгоритми мають і відмінності, що впливають на швидкість роботи даних методів [6].

1.4.1 Hunt-McIlroy алгоритм

Даний алгоритм знаходить збіги у всьому рядку чи файлі і індексує їх за допомогою так званих k -кандидатами, де k – це довжина найдовшої загальної послідовності (Longest Common Subsequence, LCS). LCS поступово доповнюється шляхами пошуку збігів, які потрапляють в правильні ординати, при цьому кожний шлях запам'ятовується [7].

Проблема даного підходу полягає в тому, він виконує більше роботи, ніж необхідно. Дана реалізація в гіршому випадку має такі часові та просторові складності, що визначаються за формулою 1.1, чи за формулою 1.2 [7].

$$O(MN), \quad (1.1)$$

$$O(MN \log M), \quad (1.2)$$

де M – кількість рядків у першому файлі;

N – кількість рядків у другому файлі.

1.4.2 An $O(ND)$ Difference алгоритм

Даний алгоритм являє собою реалізацію “розділяй та володарюй” парадигми розробки алгоритмів. Цей метод працює шляхом рекурсивного знаходження головного збігу двох послідовностей з найменшим скриптом редагування (трансформування рядка А в рядок Б) [8]. Як тільки алгоритм знаходить цей центральний збіг, то він зберігає тільки цей збіг, і дві послідовності: попередню та ту, що йде після знайденого збігу. Далі, знову виконується рекурсивне порівняння, доки є що порівнювати. Пошук центрального збігу виконується шляхом порівняння кінців послідовностей, доки це можливо, після чого скрипт редагування інкрементується. Далі відбувається сканування кожної найдовшої позиції, що була, а також перевіряється,

наскільки далі порівняння може виконуватися. Якщо знайдений збіг зустрічається зі збігом іншого кінця, то вважається, що алгоритм знайшов центральний збіг [6].

Перевагою даного алгоритму є те, що він займає простір у пам'яті, який можна визначити за формулою 1.3, а кількість операцій за формулою 1.4 [6].

$$O(M + N), \quad (1.3)$$

$$O(ND), \quad (1.4)$$

де N – це сума рядків A та B ;

D – це розмір мінімального сценарію редагування для даних послідовностей.

1.4.3 Головна перевага алгоритму Myers над алгоритмом Hunt-McIlroy

Алгоритм Myers в основному має перевагу над алгоритмом Hunt-McIlroy, оскільки він не запам'ятовує кожний шлях під час виконання, а також не потребує етапу передбачення, щоб визначити, куди йому потрібно рухатися. Все це дозволяє даному методу сконцентруватися тільки на найдаліших позиціях, котрі він зміг досягнути, завдяки скрипту редагування з найменшою складністю. Ця “найменша складність” гарантує, що знайдений результат являється LCS, що означає відсутність потреби запам'ятовувати шлях, по котрому алгоритм пройшов, щоб знайти збіг, що в свою чергу потребує використання меншого простору пам'яті [7]-[8].

Завдяки тому, що алгоритм An $O(ND)$ Difference не обчислює всі збіги наперед, чим і уникає їхнє збереження у пам'яті, дає йому певні переваги як в часі виконання, так і в просторі пам'яті, що є необхідними для виконання алгоритму [6].

1.5 Постановка завдання дослідження

На основі проробленої роботи з аналізу концепції систем управління життєвого циклу апаратного забезпечення, менеджменту задача, а також огляду методів версійності, збереження історії даних та методів порівняння текстових фрагментів, можна зробити висновок, що для програмної реалізації відстеження версій апаратного забезпечення потрібно удосконалити модель життєвого циклу апаратного забезпечення за допомогою впровадження таск-менеджменту, реалізувати систему версійності та збереження даних, виходячи з результату огляду, що було виконано в розділі 1.3, а також реалізувати функціонал порівняння текстових фрагментів з використанням алгоритму $An\ O(ND)$ Difference, та його удосконалення шляхом оптимізації під поточну платформу.

З цього можна зробити висновок, що програмна реалізація повинна мати наступний функціонал:

- можливість створення облікового запису працівника виробництва;
- створення проєкту з можливістю редагування його опису;
- можливість швидкої зміни поточного статусу проєкту;
- створення компонентів проєкту з можливістю редагування його опису;
- можливість швидкої зміни поточного статусу компонента, та часу, що було витрачено на його проєктування;
- збереження версій специфікації компонента та надання доступу до будь-якої за вимогою;
- можливість порівняти тестові фрагменти поточної версії специфікації з будь-якою попередньою версією;
- можливість залучання нових співробітників до розробки проєкту, тим самим формуючи команду;
- можливість розподілу компонентів проєкту між його учасниками, тим самим управляти задачами на проєкті.

1.6 Висновок за розділом 1

В даному розділі було виконано аналіз предметної області, а саме: розглянуто концепцію систем управління життєвим циклом та менеджменту задач, зроблено аналіз життєвого циклу апаратного забезпечення, розглянуто методи версійності та збереження історії даних, а також розглянуто методи порівняння текстових фрагментів. На основі проведеного аналізу було поставлено завдання дослідження та сформовано вимоги до програмного забезпечення (ПЗ).

2 МАТЕРІАЛИ ТА МЕТОДИ

У першому розділі було проаналізовано концепції систем управління життєвим циклом, менеджменту задач, а також основні методи версійності та порівняння текстових фрагментів. Даний аналіз показав, що всі ці концепції та методи можуть бути використані для створення системи відстеження версій апаратного забезпечення.

Виходячи з цього, можна зробити висновок, що компонування концепцій систем управління життєвим циклом та менеджменту задач, інтеграція алгоритму порівняння текстових фрагментів є ключовими моментами в реалізації системи.

У другому розділі розглянуто удосконалення моделі життєвого циклу апаратного забезпечення за допомогою впровадження елементів таск-менеджменту, а також удосконалення алгоритму Myers для порівняння текстових фрагментів шляхом оптимізації під поточно платформу.

2.1 Впровадження таск-менеджменту до моделі життєвого циклу апаратного забезпечення

Як уже було сказано у першому розділі, системи управління життєвим циклом апаратного забезпечення можу компонувати у собі декілька програмних застосунків. Дана методика використовується для модифікації та удосконалення робочого процесу та швидкості виробництва. Тому, було вирішено удосконалити модель життєвого циклу за допомогою інтеграції у нього таск-менеджменту.

Для управління життєвого циклу є доцільним впровадження наступного функціоналу:

- створення проєкту та його редагування;
- відображення загальної інформації проєкту;
- керування статусом проєкту, що і є відображенням фази життєвого циклу розроблюваного апаратного забезпечення;

Для розширеної управлінням задач системи важливим є наступний функціонал:

- створення команди шляхом додаванням персоналу до проєкту;
- створення компонентів проєкту та їхнє редагування;
- керування статусом компоненту, що є і прямим індикатором його поточного стану;
- залучення членів проєкту до розробки компонентів, що є делегуванням та розподілом обов’язків.

Розширення життєвого циклу апаратного забезпечення за допомогою такск-менеджменту має такі переваги як:

- формалізація процесу розробки;
- прискорення процесу розробки;
- формування команди та розподіл обов’язків між її учасниками;
- ведення документації проєкту та його компонентів, а також збереження у сих даних у одному місці;
- відстеження готовності переходу проєкту на наступну фазу, шляхом аналізу поточного статусу компонентів;
- аналітика та побудова діаграм, основою яких є історія зміни статусів компонентів.

2.2 Інтеграція та оптимізація алгоритму порівняння текстових фрагментів Myers

Поточною платформою для розробки клієнтської частини програмної реалізації системи управління життєвим циклом апаратного забезпечення є Web. Виходячи з цього, інтегрувати алгоритм Myers можна двома шляхами:

- використовуючи JavaScript/TypeScript;
- використовуючи Web Assembly.

Детальніше про ці технології написано у 3 розділі.

2.2.1 JavaScript імплементація алгоритму Myers

Класична реалізація алгоритму Myers на мові JavaScript програмування JavaScript не враховує сучасних методів роботи движків JS.

Перш за все, для збереження скрипта редагування та найдальших точок діагоналей використовується об'єкт типу масив (рисунок 2.1). Дане рішення зумовлено тим, що індекси масиву можуть містити від'ємні значення, оскільки дана імплементація циклу, що проходить по графу, доки не буде досягнута найдальша точка (рисунок 2.2).

```

63     const m = a.length;
64     const n = b.length;
65     const delta = n - m;
66
67     // Edit scripts, for each diagonal.
68     const es = {}; // { ..., [-1]: inserted, [0]: deleted, ...}
69     // Furthest points, the furthest y we can get on each diagonal.
70     const fp = {};
71

```

Рисунок 2.1 – Структура базової реалізації збереження скрипта редагування та найдальших точок діагоналей

```

112 // Traverse the graph until we reach the end of the longer string.
113 do {
114     // Updates furthest points and edit scripts for diagonals below delta.
115     for ( k = -p; k < delta; k++ ) {
116         fp[ k ] = snake( k );
117     }
118
119     // Updates furthest points and edit scripts for diagonals above delta.
120     for ( k = delta + p; k > delta; k-- ) {
121         fp[ k ] = snake( k );
122     }
123
124     // Updates furthest point and edit script for the delta diagonal.
125     // note that the delta diagonal is the one which goes through the sink (m, n).
126     fp[ delta ] = snake( delta );
127
128     p++;
129 } while ( fp[ delta ] !== n );

```

Рисунок 2.2 – Цикл проходження діагоналей базової реалізації

На сьогодні, дана реалізація збереження даних не є оптимально, з сучасною оптимізацією звичайних масивів та з появою компіляції “на льоту” (Just-in-time compilation, JIT). Справа в тому, що в разі використання динамічної

структури об'єкту є неможливим використання методу оптимізації вбудованого кешування (Inline Caching).

Метою вбудованого кешування є прискорення прив'язки методів під час виконання за рахунок запам'ятовування результатів попереднього пошуку методу безпосередньо на стороні виклику. Вбудоване кешування особливо корисно для мов з динамічною типізацією, де велика частина, якщо не вся прив'язка методів відбувається під час виконання, і де часто можна використовувати таблиці віртуальних методів [9]. Іншими словами, у разі використання статичної структури об'єкту, JIT-компіляція запам'ятовує структуру викликів об'єкту при проході циклом, та виконує операції використовуючи на JS інтерпретатор, а оптимізовану версію машинного коду, що в деяких випадках дає пришвидшує процес виконання операції в 7 разів [10].

Іншою причиною недоцільності використання масиву типу об'єкт для збереження даних в випадку алгоритму Myers є те, що звичайні масиви, зараз є набагато більше оптимізовані для збереження та використання даних.

По-перше, при роботі з масивами движок JS їх його для подальшого оптимізованого виконання операцій. Наразі JS движок V8 від Google має наступні типи масивів [11]:

- малі цілі числа (Small Integers) – [1, 2, 3];
- дійсне число з подвійною точністю (Double) – [1.9, 2, 3.5];
- звичайний елемент з динамічною типізацією (Regular) – [1, "x", 3];

По-друге, наявність даних у масиві також приводить його до двох станів, що мають вплив на оптимізацію роботи з даними масиву. Даними станами є:

- запакований масив (Packed Array), тобто такий масив, у якого всі комірки з даними є заповненими;
- масив з дірками (Holey Array), тобто такий масив, у якого одна чи більше комірок є порожніми.

Відповідно до цього у масиву формуються 6 типів даних, що вистроюють собою ієрархію оптимізації, і перехід з однієї стадії на іншу можливий тільки у

одному напрямку. На рисунку 2.3 зображено 6 типів масиву та процес переходу одного типу на інший [11].

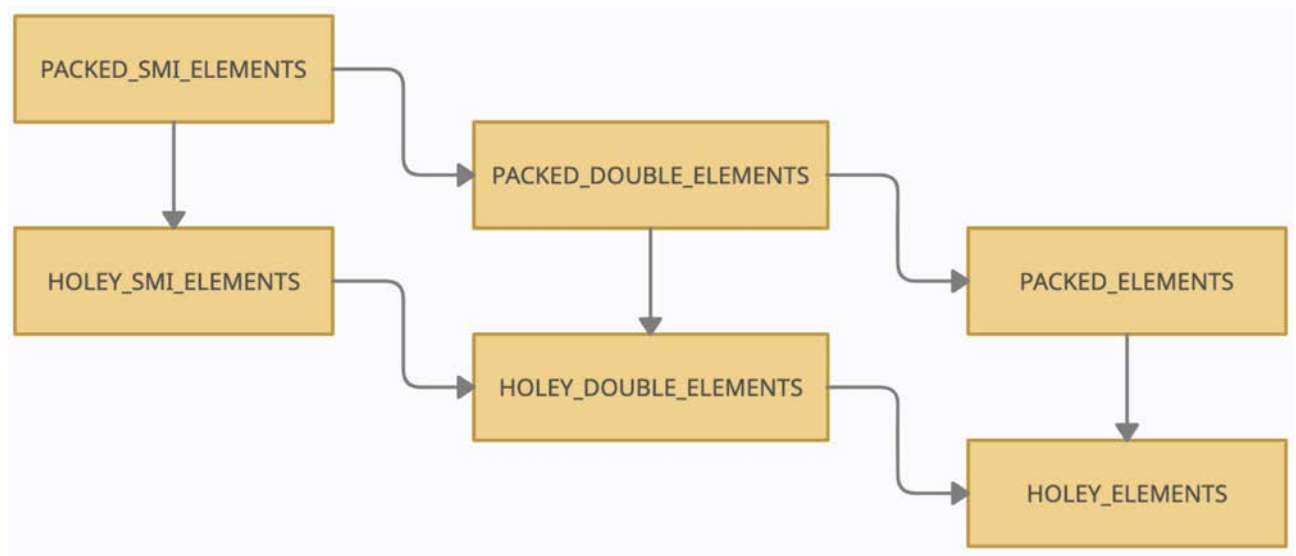


Рисунок 2.3 – Схема переходів типів масиву

JavaScript движок V8 має різні набори оптимізації для операцій залежно від виду елементів масиву (`Array.prototype.forEach`, `Array.prototype.map` тощо).

Створити звичайний масив в JS можна за допомогою конструктор класу `Array`, чи динамічно.

Створення масив через конструктор класу `Array` відбувається наступними чином – `new Array(n)`. Даний метод має наступні переваги:

- движок JS попередньо розподіляє простір в пам'яті для n елементів;
- оптимізоване створення та додавання нових елементів до масиву.

Недоліками даного методу є те, що створений масив являється типу `Holey`, а також операції типу `Array.prototype.forEach`, `Array.prototype.map` працюють повільніше.

Динамічне створення масиву відбувається наступними чином – `const array = []; array.push('x')`. Даний метод має наступні переваги:

- створений масив являється типу `Packed`;
- операції типу `Array.prototype.forEach`, `Array.prototype.map` є оптимізованими та працюють швидше.

Недоліками даного методу є те, що движок JS попередньо не розподіляє простір в пам'яті для n елементів, а також створення та додавання нових елементів до масиву є не оптимізованим і є повільнішим в порівнянні з використанням конструктора класу `Array`.

Таким чином, беручи до уваги методи роботи і оптимізації сучасних JS движків, було вирішено використати для збереження даних звичайний масив, який створено за допомогою конструктору класу `Array` (рисунок 2.4), оскільки в даній реалізації алгоритму не використовується цикл типу `Array.prototype.forEach`, тому для алгоритму є важливим створення та додавання нових елементів до масиву.

```
30      const m = a.length;
31      const n = b.length;
32
33      const delta = n - m;
34      const offset = m + 1;
35      const size = m + n + 3;
36
37      // Edit scripts, for each diagonal.
38      const es = new Array(size);
39      // Furthest points, the furthest y we can get on each diagonal.
40      const fp = new Array(size);
```

Рисунок 2.4 – Оптимізована структура збереження скрипта редагування та найдальших точок діагоналей

Для роботи зі звичайним масивом необхідно переписати цикл для проходу по графу (рисунок 2.5), оскільки базова імплементація алгоритму передбачає використання від'ємних чисел у якості індексу елементу, що є не можливим з використанням звичайного масиву.

```

79      let p = -1;
80
81      // Traverse the graph until we reach the end of the longer string.
82      do {
83          ++p
84          // Updates furthest points and edit scripts for diagonals below delta.
85          for (let k = -p; k <= delta - 1; ++k) {
86              fp[k + offset] = snake( k );
87          }
88
89          // Updates furthest points and edit scripts for diagonals above delta.
90          for (let k = delta + p; k >= delta + 1; --k ) {
91              fp[k + offset] = snake(k);
92          }
93
94          // Updates furthest point and edit script for the delta diagonal.
95          // note that the delta diagonal is the one which goes through the sink (m, n).
96          fp[delta + offset] = snake(delta);
97      } while ( fp[delta + offset] !== n );

```

Рисунок 2.5 – Цикл проходу діагоналей оптимізованої реалізації алгоритму

Також до модуля з кодом алгоритму було додано метод з використанням мемоізації. Даний метод оптимізації кешує вхідні параметри та результат виконання функції, тому наступний раз коли функція буде викликана з тими самими параметрами, метод пропустить фазу обчислення та відразу поверне результат. Для реалізації мемоізації було використано бібліотеку `lodash.memoize`, оскільки вона добра справляється з поставленим завданням та надає можливість регулювати розмір кешу.

Щоб перевірити результат оптимізації було проведено тест продуктивності (Benchmark test) [12]. В якості вхідних параметрів для тесту було використано строку А довжиною 2233 символи, та строку Б довжиною 2697 символів. Основа для цих строк є однаковою за виключенням випадкових змін та вставок. Результат benchmark-тестування зображено на рисунку 2.6.

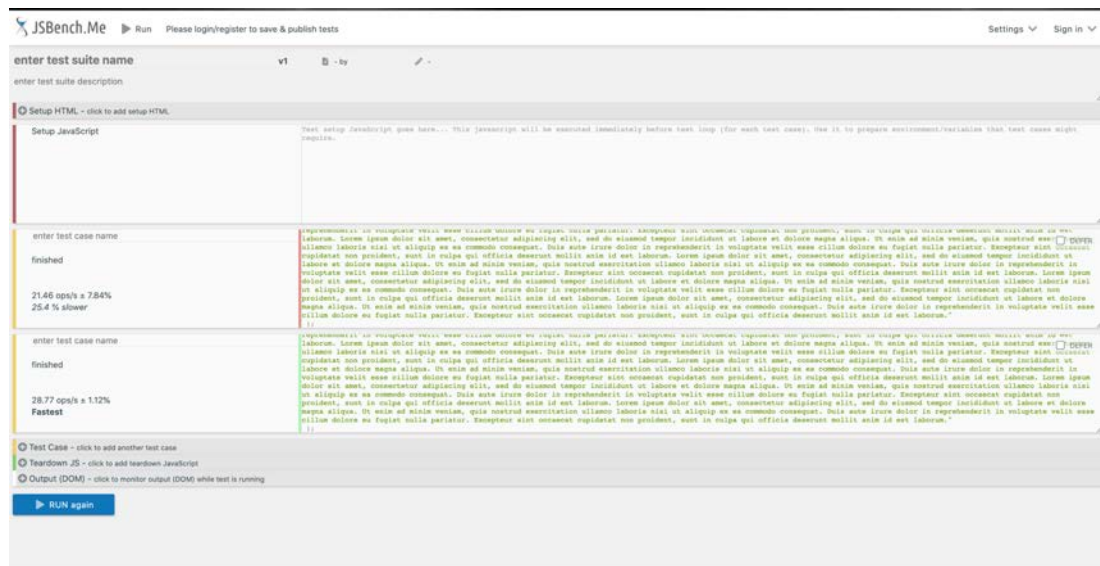


Рисунок 2.6 – Benchmark-тестування базової та оптимізованої версії алгоритму Myers

З результатів benchmark-тестування зробити висновок, що оптимізована версія алгоритму Myers є швидшою на 25.4% ніж його базова імплементація на JS, а також є стабільнішою, оскільки межі в яких швидкість алгоритму коливається є $\pm 1.12\%$, в той час як базова імплементація має межі коливання в $\pm 7.84\%$.

Повна оптимізована версія коду алгоритму наведена у додатку Б.

2.2.2 Web Assembly імплементація алгоритму Myers

Останнім найдієвішим способом оптимізації для Web може бути Web Assembly, оскільки дана технологія дає можливість реалізацію алгоритму на C, C++ та Rust скомпілювати в машинний код та запустити у JavaScript середовищі.

Було вирішено переписати оптимізовану версію алгоритму Myers на C++ та запустити скомпільовану в Web Assembly версію коду в браузері.

Після реалізації алгоритму на C++ та його запуску з використанням Web Assembly, було зроблено порівняння результату роботи алгоритмі, у ході якого було визначено, що Web Assembly версія алгоритму працює повільніше в декілька разів. Для тестування було використано звичайно консоль проєкту,

оскільки сервіс для benchmark-тестування [12], не надає можливості запускати код на Web Assembly.

Дана різниця у продуктивності пояснюється тим, що для поточної версії Web Assembly 1.0, є досить важким процес транспортування даних в такій кількості, що і уповільнює процес роботи.

Код імплементації алгоритму Myers на C++ наведено у додатку Б.

2.3 Висновки за розділом 2

У другому розділі було проаналізовано процес удосконалення моделі життєвого циклу апаратного забезпечення за допомогою впровадження елементів task-менеджменту, а також було розглянуто та випробувано способи удосконалення алгоритму Myers для порівняння текстових фрагментів шляхом оптимізації під поточно платформу, після чого було зроблено benchmark-тестування для виявлення найкращої реалізації алгоритму.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

В результаті аналізу концепції систем управління життєвим циклом апаратного забезпечення та менеджменту задач, огляду методів версійності, а також огляду методів порівняння текстових фрагментів, було обрано архітектуру та стек технологій, що дозволяють зберігати історію даних найефективнішим шляхом, реалізувати інтерфейс та логіку робочого процесу для ефективної взаємодії користувача та системи, а також без проблем інтегрувати і різними методами оптимізувати алгоритм порівняння текстових фрагментів, з можливістю наглядної демонстрації результату виконання алгоритму.

3.1 Вибір стеку технологій серверної частини

3.1.1 База даних та інструменти взаємодії з даними

При виборі стеку технологій для серверної частини, у першу чергу, слід звернути увагу на базу даних (БД) та інструменти, що дозволяють ефективно з нею взаємодіяти.

Сьогодні, двома найефективнішими типами баз даних є:

- реляційні бази даних, що часто називаються Structured Query Language (SQL) базами даних;
- документно-орієнтовані бази даних, що також відомі під назвою NoSQL (non SQL) бази даних.

Ці підходи по організації даних мають свої плюси, мінуси та деякі особливості, що і впливають на кінцеве рішення, при виборі бази даних для системи.

Реляційні бази даних – це така база даних, що заснована на реляційній моделі даних. Слово “реляційний” походить від англійського слова relation (відношення). Для роботи з реляційними БД застосовують реляційні СКБД. Інакше кажучи, реляційна база даних – це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеня [13].

Реляційна база даних є сукупністю елементів даних, організованих у вигляді набору формально описаних таблиць, з яких дані можуть бути доступними або повторно зібрані багатьма різними способами без необхідності реорганізації таблиць бази даних [13].

Використання реляційних БД було запропоноване Едгаром Коддом в 1970 році [13].

Реляційна структура даних дає високу гарантію запобігання утворенню аномалій даних, оптимальну роботу з даними, що взаємопов'язані та знаходяться в різних таблицях, а також хороший компроміс між швидкістю записом та отримання даних.

Документо-орієнтована система керування базами даних (англ. document-oriented database) – система керування базами даних, спеціально призначена для зберігання ієрархічних структур даних (документів) і зазвичай реалізована за допомогою підходу NoSQL. В основі документо-орієнтованих СКБД лежать документні сховища (англ. document store), котрі мають структуру дерева (іноді лісу). Структура дерева починається з кореневого вузла і може містити кілька внутрішніх і листових вузлів. Листові вузли містять дані, які при додаванні документа заносяться в індекси, що дозволяє навіть при досить складній структурі знаходити місце (шлях) шуканих даних. API для пошуку дозволяє знаходити за запитом документи та частини документів. На відміну від сховищ типу ключ-значення, вибірка за запитом до документного сховища може містити частини великої кількості документів без повного завантаження цих документів в оперативну пам'ять [14].

Документи можуть бути організовані (згруповані) в колекції. Їх можна вважати віддаленим аналогом таблиць реляційних СКБД, але колекції можуть містити інші колекції. Хоча документи колекції можуть бути довільними, для ефективнішого індексування краще об'єднувати у колекцію документи зі схожою структурою [14]-[15].

Документо-орієнтована база даних дає можливість організовувати структуру даних таким чином, що вона може бути динамічно змінена, а також

добре оптимізованою під запис, або збереження даних, що іноді може викликати такий побічний ефект, як аномалії даних.

Розроблювана система відстеження версій апаратного забезпечення повинна мати:

- чітку структуру даних, що має досить низьку вірогідність зміни, чи розширення новими полями;
- зберігати велику кількість даних, що взаємозалежні між собою;
- збережені дані повинні бути захищені від аномалій при запису та оновленні;
- у співвідношенні частоти запису та отримання даних, остання має значну перевагу.

Виходячи з потреб, що були описані вище, можна зробити висновок, що реляційна база даних найкраще підходить для задоволення потреб розроблюваного програмного застосунку.

Для роботи з такою базою даних, найкраще підходить СКБД під назвою PostgreSQL, оскільки є потреба у частому пошуку даних, що взаємопов'язані між собою, та розкидані по різних таблицях, для оптимізації та запобігання аномалії даних. PostgreSQL дуже швидко працює з даними, добре масштабується та надає велику кількість функціоналу, для ефективного запису, пошуку та агрегації даних.

Для швидкості розробки та ефективної роботи з PostgreSQL, гарним рішенням є використовувати об'єктно-реляційне відображення (Object-relational mapping, ORM). Найкращим варіантом є TypeORM, оскільки, вона дуже ефективно працює з PostgreSQL, є типізованою, має велику кількість готових SQL рішень та дає можливість писати чисті SQL запити, дозволяє інтеграцію сідів (дефолтних даних, що додаються до БД за допомогою скрипта), а також автоматично генерує міграції на основі оновлених сутностей, що описують моделі даних та способи їхньої взаємодії між собою.

Для швидкого та автоматизованого розгортання бази даних на будь-якому сервері було вирішено обрати технологію Docker.

Docker – це набір продуктів платформи як послуги (англ. Platform as a Service, PaaS), які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, які називаються контейнерами. Контейнери ізольовані один від одного і об'єднують власне програмне забезпечення, бібліотеки та файли конфігурації; вони можуть спілкуватися між собою за чітко визначеними каналами. Усі контейнери запускаються одним ядром операційної системи і тому використовують менше ресурсів, ніж віртуальні машини [16]. Простіше кажучи, Docker надає можливість створити ізольований Linux-контейнер у якому знаходиться база даних або проєкт. Як саме та з якими конфігураціями буде створено контейнер та його вміст, визначає файл конфігурацій, у якому описуються метадані від порту бази даних до бібліотек, що повинні бути встановлені при першому запуску скрипта. Будь-які дані, що знаходяться в проєкті можна зв'язати з директорією на сервері, та в реальному часі обмінюватися даними в обидві сторони. Створений контейнер з усіма даними можна експортувати до сервісу Docker Hub, після чого він може бути надісланим до інших серверів.

3.1.2 Компоненти серверна частини

Оскільки система відстеження версій апаратного забезпечення повинна працювати з великою кількістю даних, мати можливість працювати з високим навантаженням мережі у випадку використання системи на великому виробництві, а також підтримувати масштабування та найновіші технології, було вирішено обрати NodeJS платформу для реалізації серверної частини програмного застосунку.

NodeJS – це програмна платформа з відкритим кодом в основі якої лежить движок V8, що трансліює JavaScript (JS) в машинний код [17]. Суть цієї платформи в тому, що вона призначена для виконання високопродуктивних мережових застосунків, що написані на мові JS, або її розширені TypeScript (TS), детальніше про який написано в розділа 2.3. Дана платформа також підтримує

інтеграцію Web Assembly (WS), котра дозволяє запустити код написаний на C, C++ та Rust у JavaScript середовищі. Обробка великої кількості запитів можлива завдяки тому, що цикл подій (Event loop) працює таким чином, що ввід та вивід даних не блокують потік, що і дозволяє обробляти велику кількість запитів одночасно. Також NodeJS має в своєму арсеналі бібліотеку libuv, яка написана на мові програмування C, та бере на себе роботу з файлами, та високонавантаженими процесами. Всі ці фактори є суттєвими для коректної роботи системи у разі великого навантаження.

Працювати з чистим NodeJS досить незручно, оскільки для дуже багато речей реалізовувати з нуля, наприклад об'єднання частин запиту, що лежать у буфері. Для прискорення розробки, а також запобігання рутинних задач, з NodeJS зазвичай використовують один з багатьох фреймворків, що були створені для цієї платформи. Одним з найпопулярніших та найсучасніших таких рішень є NestJS.

NestJS – це прогресивний фреймворк для NodeJS, що призначений для створення ефективних, надійних та масштабованих серверних застосунків [18].

Даний фреймворк надає строгу структуру проєкту, що являє собою модульну систему, кожний з яких може повністю не залежати від інших, але при потребі, може бути легко інтегрований у будь-який інший, при дотриманні строгої послідовності дій, що є принципом роботи всієї модульної системи фреймворку. Також NestJS мотивує розробників використовувати TypeScript для розробки, що в свою чергу ще сильніше покращує надійність модульної системи, та можливість безпечно та ефективно масштабувати проєкт у майбутньому.

Великою перевагою даного фреймворка над іншими є те, що він має реалізацію усіх інструментів, які є необхідними при розробці серверного застосунку, а саме: обробку різноманітних помилок та їхнє відловлювання у різних частинах системи, авторизація та менеджмент рівнів доступу до даних, генерація схем для REST, чи GraphQL API, тощо. Якщо команда розробників NestJS не підтримує якусь з бібліотек, що мають ключове значення в системі, то

ця бібліотека, зазвичай з часом заточується під фреймворк розробниками самої бібліотеки.

NestJS також добре підходить для реалізації патерну розділу даних застосунку – “Модель-Представлення-Контролер” (англ. Model-view-controller, MVC), який детальніше розглянуто у пункті 3.2.

Всі вище зазначені фактори слугують причиною, за якою даний фреймворк є найкращим рішенням для розроблюваної системи. Останнім ключовим фактором, що вплинув на рішення використовувати NestJS, можна зазначити те, що TypeORM, котру було обрано у розділі 2.1.1, для взаємодії з СКБД PostgreSQL, підтримує повну інтеграцію з фреймворком, а також надає можливість реалізації шаблон репозиторію, що повністю інкапсулює логіку взаємодії з базою даних, що дає можливість використовувати один і той самий інтерфейс, для взаємодії з різними СКБД.

В якості інтерфейсу для взаємодії з клієнтською частиною було обрано RESTful API, оскільки масштаби та логіка розроблюваного застосунку повністю задовольняються даним рішенням, а відсутність мобільного клієнта робить не обов’язковим використання GraphQL API, для оптимізації мережі та споживання батареї мобільних девайсів.

Для описання RESTful API інтерфейсів, документування, а також створення адміністративної панелі, було вирішено використати Swagger. Даний інструмент надає можливість автоматично генерувати документацію до API, тестові кейси, а також можливість виконувати запити до бази даних з використанням авторизації. На рисунку 3.1 зображено адміністративну панель, що була згенерована за допомогою Swagger.

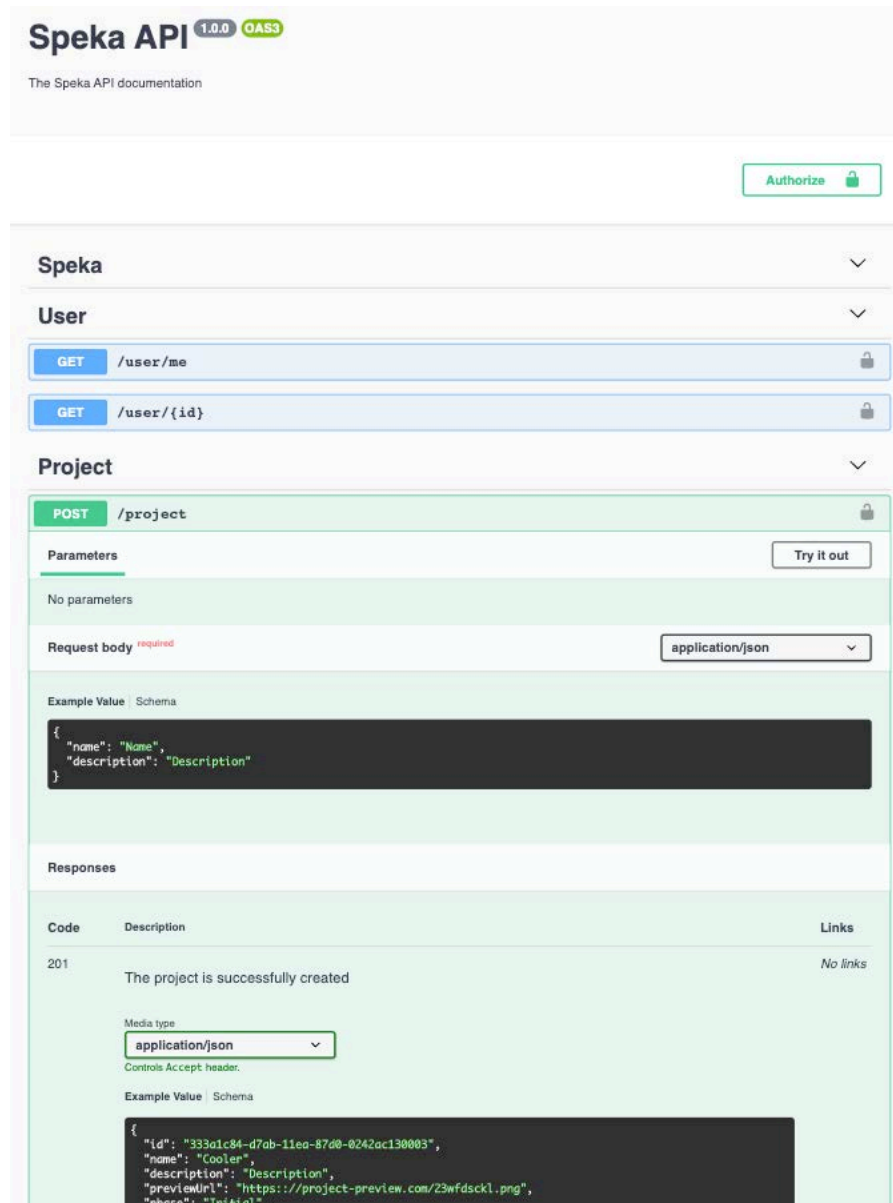


Рисунок 3.1 – Адміністративна панель Swagger

3.2 Проектування архітектури серверної частини

Обраний стек технологій дозволяє використовувати патерн для розділу даних застосунку під назвою “Модель-Представлення-Контролер”. MVC – це шаблон дизайну програмного забезпечення, який зазвичай використовується для розробки користувацьких інтерфейсів, що розділяє відповідну логіку програми на три взаємопов'язані елементи. Це робиться для відокремлення внутрішнього подання інформації від способів подання та прийняття інформації користувачем [19]. Головною цінністю даного підходу є те, що кожний

компонент являє собою інкапсульовану систему, що з'єднана з іншими компонентами за допомогою інтерфейсу, а це значить, що кожний елемент цієї системи може бути з легкістю заміненим. Обмін даними між елементами виконується у певній послідовності, що дозволяє відстежувати потік даних, та з легкістю відлагоджувати код та знаходити уразливі місця системи.

MVC шаблон має наступні компоненти [19]:

- модель (model) – це центральний компонент системи, який являє собою динамічну структуру даних застосунку, незалежну від інтерфейсу користувача, що безпосередньо управляє даними, логікою та правилами застосунку;
- представлення (view) – це будь-яке подання інформації системи, що може мати будь-який вигляд. Ключовим моментом є те, що представлення не є місцем для обробки даних;
- контролер (controller) – це рівень який приймає дані та оброблює їх для подальшого транспортування до моделі або представлення.

На рисунку 3.2 зображено принцип взаємодії елементів шаблону Model-View-Controller [20].



Рисунок 3.2 – Схема взаємодії елементів шаблону MVC

В розроблюваному програмному застосунку клієнтська частина виступає в ролі представлення, що за допомогою REST API зв'язана з методами серверної

частини, що являє собою контролер, який за допомогою модулів NestJS та TypeORM, робить SQL запити до бази даних. БД, що використовує в якості СКБД PostgreSQL, надає інтерфейс для взаємодії контролера з даними та виступає у ролі моделі застосунку, де і зберігаються всі дані та бізнес логіка системи.

Обраний стек та шаблон проєктування надає можливість розміщувати всі частини програмного застосунку на одному сервері з можливістю вертикального та горизонтального масштабування.

Вертикальне масштабування полягає збільшенні потужності сервера шляхом додавання ОЗУ та ядер ЦП.

Горизонтальне масштабування полягає у розміщенні системи на різних серверах, а також у додаванні нових сутностей для розподілу навантаження та підвищенні відмовостійкості системи в цілому.

3.3 Вибір стеку технологій клієнтської частини

Оскільки клієнтська частина являє собою веб застосунок, то найкращим рішенням для її реалізації є найпопулярніша бібліотека на сьогодні – React.

React – це JavaScript бібліотека, що призначена для створення односторінкового застосунку (Single Page Application, SPA), чи застосунку, рендеринг якого відбувається на сервері (Server Side Rendering, SSR) [21]. Дана бібліотека є дуже продуктивною, оскільки вона використовує технологію віртуальну об'єктну модель документа (Virtual Document Object Model, Virtual DOM), що являє собою легковісне віртуальне представлення справжнього DOM. Бібліотека за допомогою добре оптимізованих алгоритмів перевіряє зміни у Virtual DOM, а в разі їхнього знаходження змінює тільки відповідний вузол у справжньому DOM. Даний підхід є дуже продуктивним методом відображення веб застосунку [22].

Дана бібліотека заснована на Flux архітектурі, детальніше про яку написано у наступному розділі. Також React дозволяє будувати власну структуру

проєкту, та використовувати різноманітні інструменти для менеджменту даних в середині застосунку, як власні, так і рішення представлені іншими розробниками.

Найпопулярнішим та одним з найефективнішим рішенням для організації менеджменту та збереження даних в середині додатку є бібліотека Redux, що слідує всім принципам Flux архітектури, та надає розробнику інструментарій для відслідковування змін у застосунку, а також можливість моніторингу усіх дій, що призводять до зміни стану даних системи на часовій лінії, з можливістю переглянути послідовну зміни, та регулювати швидкість їхніх виконання [22]. На рисунку 3.3 зображено розширення Redux DevTools, за допомогою якого можливо виконувати дії, що було описано вище.

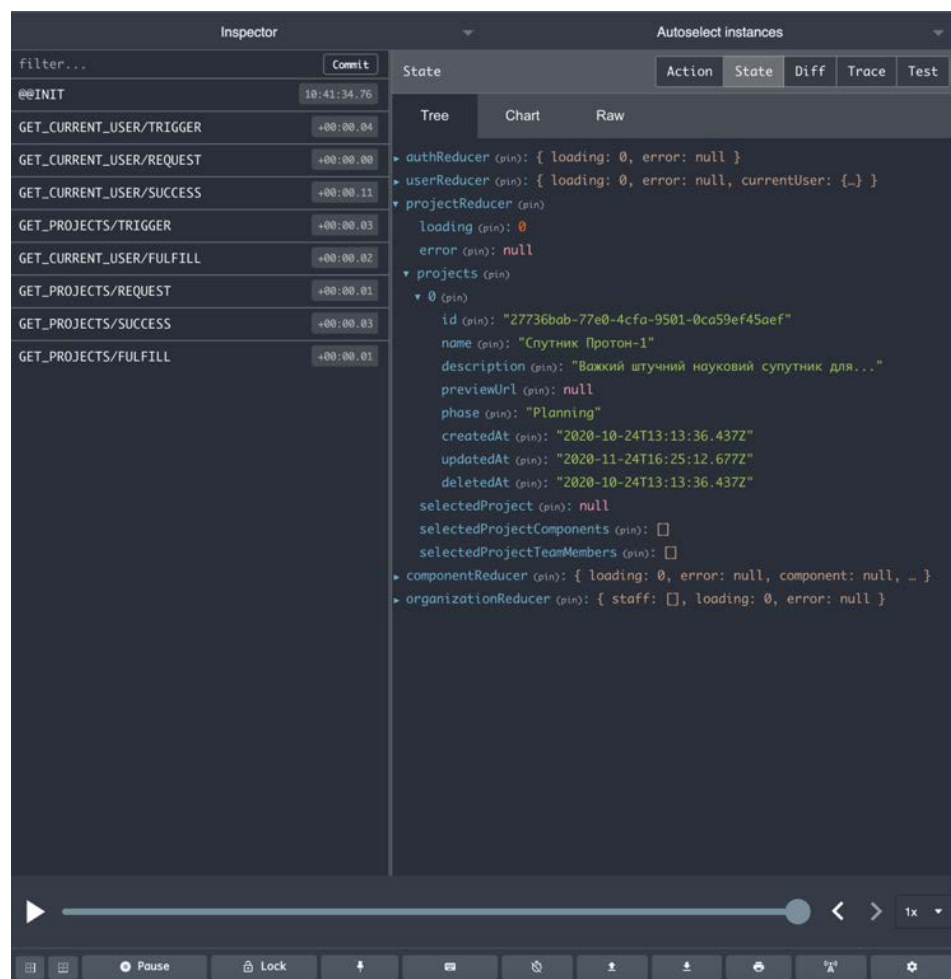


Рисунок 3.3 – Redux DevTools

Також бібліотека React дозволяє використовувати разом з собою розширення для мови програмування JavaScript під назвою TypeScript. Для більшого розуміння, потрібно зазначити, що:

а) JavaScript – це мультипарадигмова мова програмування, що підтримує об'єктно-орієнтований, імперативний і функціональний стилі програмування, а також є реалізацією стандарту ECMAScript (стандарт ECMA-262). JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерях як мова сценаріїв для додання інтерактивності веб-сторінок [23];

б) TypeScript – це мова програмування, розроблена та підтримувана корпорацією Майкрософт. Це строгий синтаксичний набір JavaScript з додаванням статичної типізації до мови. TypeScript призначений для розробки великих додатків та перекомпіляції в JavaScript. Оскільки TypeScript є надмножиною JavaScript, існуючі програми JavaScript також є дійсними програмами TypeScript [24]. Головними перевагами TS над JS є те, що він додає типізацію, підтримку повноцінних класів, а також підключення модулів. Саме тому дану мову програмування було обрано для розроблення серверної та клієнтської частин системи.

Для ведення специфікації та відображення відмінностей між її версіями було обрано фреймворк під назвою CKEditor 5, що являє собою “що бачиш, те і отримаєш” (What You See Is What You Get, WYSIWYG) редактор. Даний фреймворк має широкий спектр можливостей редагування тексту, що є важливим для акцентування уваги на окремих моментах специфікації. Також, CKEditor 5 надає інструментарій, для того, щоб наглядно та зрозуміло відобразити результат роботи алгоритму порівняння текстових фрагментів користувачеві.

3.4 Проектування архітектури

Оскільки для написання клієнтської частини було обрано бібліотеку React, а для менеджменту та збереження даних – Redux, то найкращим вибором для побудови архітектури фронтенду є Flux-архітектура.

Flux-архітектура – це архітектурний підхід або набір шаблонів програмування для побудови призначеного для користувача інтерфейсу веб-додатків, що поєднується з реактивним програмуванням і побудований на односпрямованих потоках даних [25]. Тобто, це означає, що весь потік даних застосунку проходить за одним і тим самим принципом, що дозволяє з легкістю відстежувати кожну зміну стану додатку та більш швидко та ефективно відлагоджувати застосунок.

Flux-архітектура складається з таких слів [25]:

- представлення (view) – компонент, що відповідає за відображення інформації користувачеві у будь-якому виді та надання інтерфейсу для взаємодії з застосунком;
- дія (action) – подія що викликається представленням та слугує тригером запиту до сервера (виклику API);
- диспетчер (dispatcher) – призначений для передачі отриманих даних з відповіді API виклику до сховища даних;
- сховище (store) – являє собою місце, де зберігаються дані застосунку.

Представлення завжди підписане на оновлення сховища, тому кожна зміна даних веде до того, що підписані компоненти представлення моментально отримують найактуальніший стан сховища.

На рисунку 3.4 схематично зображено потік даних у Flux-архітектурі [26].



Рисунок 3.4 – Потік даних у Flux-архітектурі

3.5 Висновки за розділом 3

У третьому розділі було виконано і обґрунтовано вибір стеку технологій для серверної та клієнтської частини системи відстеження версій апаратного забезпечення. На основі обраного стеку технологій було обрано та обґрунтовано MVC архітектуру для серверної частини, а також Flux-архітектури для клієнтської частини.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

На основі аналізу вимог до системи відстеження версій апаратного забезпечення, а також обраного стеку та архітектури програмного застосунку, можна чітко виділити три основні компоненти системи:

- база даних;
- серверний застосунок;
- клієнтський застосунок.

У даному розділі розглянуто основні модулі розроблюваного програмного застосунку.

4.1 Основні модулі бази даних

Оскільки для сховища даних системи було обрано реляційну базу даних, то для збереження даних та представлення моделей використовуються таблиці, що мають зв'язки між собою які були закладені бізнес логікою системи відстеження версій апаратного забезпечення. На рисунку 4.1 зображено схему бази даних розроблюваного програмного застосунку.



Рисунок 4.1 – Схема бази даних

Проаналізувавши схему бази даних, можна зробити висновок, що БД складається з таких основних чотирьох модулів:

- організація (organization);
- користувач (user);
- проєкт (project);
- компонент (component).

Дані модулі зв'язані між собою та складаються з основних та допоміжних таблиць.

4.1.1 Модуль організації

Даний модуль складається з однієї таблиці `organization`, яка містить інформацію про організацію в якій використовується розроблений програмний застосунок. Дана таблиця є одною з найголовніших у системі, оскільки вона зв'язана зв'язком один до багатьох з такими сутностями як користувач та проєкт.

4.1.2 Модуль користувача

Модуль користувача складається з однієї основної таблиці `user`, що містить основну інформацію про користувача, та двома допоміжними:

- `userDetails`, яка зв'язана з основною таблицею зв'язком один-до-одного, для збереження даних, що викликаються не так часто та не мають великого впливу на бізнес логіку системи;
- `role`, яка зв'язана з основною таблицею зв'язком один-до-багатьох та зберігає у собі роль користувача у системі для подальшого регулювання рівнів доступу до функціоналу та даних.

4.1.3 Модуль проєкту

Даний модуль складається з основної таблиці `project`, що містить основну інформацію про проєкт, та таблиці `projectTeamMember`, що слугує проміжною таблицею у зв'язку багато-до-багатьох між таблицею `project` та `user`. Даний зв'язок є важливим, оскільки у одного користувача може бути багато проєктів, а у одного проєкту може бути багато користувачів, що формують команду.

4.1.4 Модуль компоненту

Модуль компоненту складається з основної таблиці `component`, яка містить основну інформацію про компонент, та допоміжних таблиць:

– `componentDetails`, що має зв'язок багато-до-одного з основною таблицею, та слугує для збереження версій специфікації компоненту. Дана таблиця була створена на основі другого (SCD2) та четвертого (SCD4) типів версійності даних з модифікацією. Головними відмінностями є те замість того, щоб зберігати поточну версію в основній таблиці, вона відразу зберігається в допоміжній таблиці, що значно оптимізує процес оновлення даних, оскільки не потрібно перезаписувати старі дані та переносити їх у допоміжну таблицю. Також, виходячи з того, що таблиця містить поле дати створення (`createdAt`), це дає можливість визначити проміжок часу, на якому кожна з версій була поточною, що і задовольняє один з постулатів 2 типу версійності;

– `componentAssignee`, що слугує проміжною таблицею у зв'язку багато-до-багатьох між таблицею `component` та `user`. Даний зв'язок є важливим, оскільки у одного користувача може бути багато компонентів, які на нього зазначені, а у одного компоненту може бути багато користувачів, що і формує систему делегації та таск-менеджменту.

4.2 Основні модулі серверної частини

Модулі серверної частини та їхні складові зумовлені обраним стеком технологій та розробленою схемою бази даних, що керується бізнес логікою системи. На рисунку 4.2 зображено файлову структуру серверної частини застосунку.

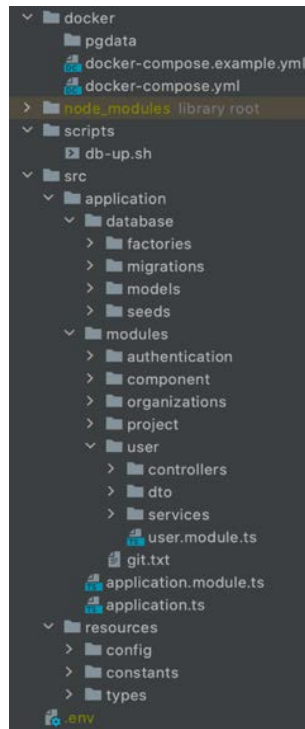


Рисунок 4.2 – Файлова структура серверної частини

Головними файлами застосунку є `application.ts` в який є вхідною точкою в програму, та `application.module.ts` до якого підключаються усі внутрішні модулі системи, а також зовнішні, такі як модулі TypeORM, Swagger, тощо.

Далі можна побачити, що у папці `modules` містяться усі внутрішні модулі системи, основу для яких стала схема бази даних та її бізнес логіка. Кожний з модулів складається з таких частин:

- головний файл з розширенням `*.module.ts`, в якому відбувається підключення інших файлів модулю, а також експортуються зовнішні сервіси, що беруть участь у роботі модулю;
- папка `services` містить файли, в яких відбуваються виклики до бази даних та описується логіка роботи з ними;
- папка `controllers` містить файли, в яких налаштовуються RESTful API, Swagger схеми та виклики до сервісів, що реалізовані в папці `services`;
- папка `dto` (від Data Transfer Object) містить інтерфейси, що описують дані для їхнього передачі між підсистемами.

Папка `database` відповідає за зв'язок з базою даних та в цілому заснована на функціоналі `TypeORM`. Кожна з папок має своє стратегічне призначення:

- папка `models` містить моделі даних, що засновані на схемі БД;
- папка `migrations` містить SQL міграції, які генеруються та запускаються за допомогою скриптів, що перевіряють зміни моделі даних та поточний стан бази даних;
- папки `seeds` та `factories` містять дефолтні дані та логіку їхнього додавання до бази даних за допомогою скриптів.

У папці `docker` містяться конфігурації для створення та функціонування бази даних в `Docker` контейнері.

У папці `scripts` знаходиться файл `db-up.sh`, що являє собою скрипт для запуску ініціалізації `Docker` контейнеру, бази даних, міграцій та запису дефолтних даних до БД.

У папці `resources` знаходяться константи, типи, інтерфейси та файли з конфігураціями, що використовуються у різних модулях застосунку.

Файл `.env` містить змінні, що є валідними для конкретного середовища виконання програми.

Код частини вище описаних файлів наведено у додатку Б.

4.3 Основні модулі клієнтської частини

Модулі клієнтської частини та їхні складові зумовлені обраним стеком технологій, бізнес логікою системи та розробленим інтерфейсом серверної частини. На рисунку 4.3 зображено файлову структуру клієнтської частини застосунку.

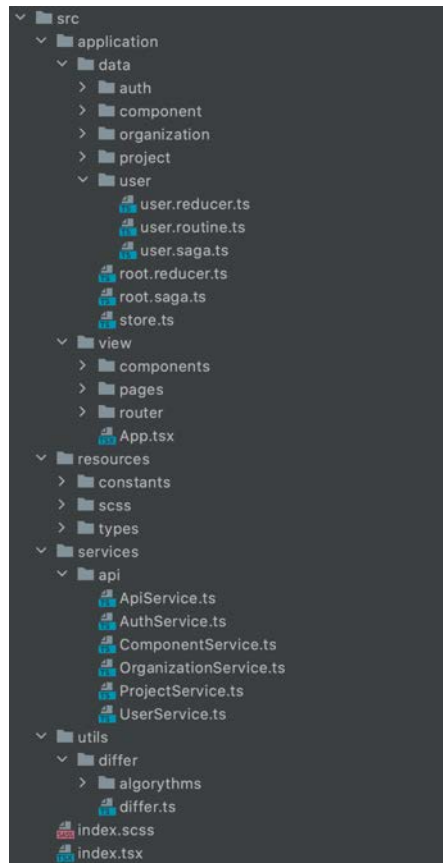


Рисунок 4.3 – Файлова структура клієнтської частини

Головними файлами клієнтської частини є `index.tsx`, оскільки він є вхідною точкою в застосунок, а також в ньому бібліотека React знаходить вузол у DOM для генерації всього застосунку, та файл `App.tsx`, в якому відбувається підключення основних провайдерів клієнтської частини, таких як Redux.

Папка `view` відповідає за UI застосунку та складається з таких папок:

- `pages`, що містить контейнери в яких відбувається робота з логікою та трансфер даних до компонентів, що займаються відображенням отриманих даних;
- `components`, що містить усі UI компоненти застосунку. Як правило ці компоненти не працюють з логікою, а займаються виключно відображенням даних;
- `router`, що являє собою роутинг систему застосунку.

Папка `data` відповідає за менеджмент даних та сховища застосунку. Дана папка містить наступні ключові компоненти:

- файли *.routine.ts, що містять в собі менеджмент дій (action), та трансферних даних (payloads), для роботи з redux;

- файли *.saga.ts, що містять функції, в яких за допомогою бібліотеки redux-saga відбувається робота з зовнішніми даними (side effects), такими як виклики до серверної частини;

- файли *.reducer.ts, що містять чисті функції (pure functions), що відловлюють дії (actions) та записують дані до загального сховища даних застосунку;

- файл store.ts, в якому відбувається налаштування сховища застосунку.

Папка services містить сервіси для викликів серверної частини за допомогою RESTful API.

У папці resources знаходяться константи, типи, інтерфейси та зміни стилів.

У папці utils знаходиться модуль для порівняння тестових фрагментів.

Код частини вище описаних файлів наведено у додатку Б.

4.4 Висновки до розділу 4

В даному розділі було наведено опис основних модулів бази даних, клієнтської та серверної частин розробленого програмного застосунку, основу реалізацією яких стала бізнес логіка та обраний стек технологій.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис застосування програми

Розроблений програмний застосунок призначений для відстеження версій апаратного забезпечення з можливістю відстежувати життєвий цикл апаратного забезпечення, статуси окремих компонентів, а також формування команди розробників. Даний застосунок можна використовувати на підприємствах, що розробляють апаратне забезпечення.

Для вирішення поставлених задач було реалізовано наступний функціонал:

- створення проєкту апаратного забезпечення;
- зміна фази життєвого циклу проєкта;
- створення компоненту проєкту;
- зміна статусу виконання проєкту;
- створення команди та розподілення компонентів між ними;
- можливість порівняти минулі версії специфікації з поточною.

Для забезпечення версійності схему бази даних було спроектовано згідно бізнес логіки застосунку та модифікації другого (SCD2) та 4 (SCD4) типів версіонування та збереження історії даних.

Для порівняння тестових фрагментів було використано оптимізований під поточну платформу алгоритм Myers.

Для коректної роботи програмного застосунку, необхідно мати комп'ютер з наступними технічними характеристиками:

- операційна система Windows 7 та вище, macOS X Yosemite 10.10 та вище, Ubuntu 14.04 та вище;
- процесор Pentium 4 з тактовою частотою 1,3 ГГц та вище;
- обсяг оперативної пам'яті 1,5 ГБ та більше;
- 2 ГБ та більше вільного місця на жорсткому диску;

- доступ до інтернету для зв'язку з сервером;
- підключення до локальної мережі у випадку використання локального серверу.

5.2 Інструкція з експлуатації програми

5.2.1 Звернення до програмного застосунку

Для початку роботи з клієнтською частиною програмного застосунку потрібно ввести посилання на розгорнуту релізну версію в адресну строку браузера. Після чого перед користувачами відкриється сторінка авторизації яку потрібно пройти для отримання доступу для взаємодії з інтерфейсом застосунку.

Для запуску серверної частини застосунку необхідно виконати скрипт `db-up.sh`, котрий розгорне базу даних та виконає всі міграції якщо це не було зроблено раніше, а також запустить серверну частину застосунку.

5.2.2 Початкові та вихідні дані

Вхідними даними є:

- введені користувачем дані авторизації;
- введені користувачем дані для створення та редагування проєкту;
- введені користувачем дані для створення та редагування компоненту проєкту.

Вихідними даними застосунку є дані, що зберігаються в базі даних.

5.2.3 Виконання програмного застосунку

Робота з застосунком розпочинається з авторизації користувача у системі (рис. 5.1).

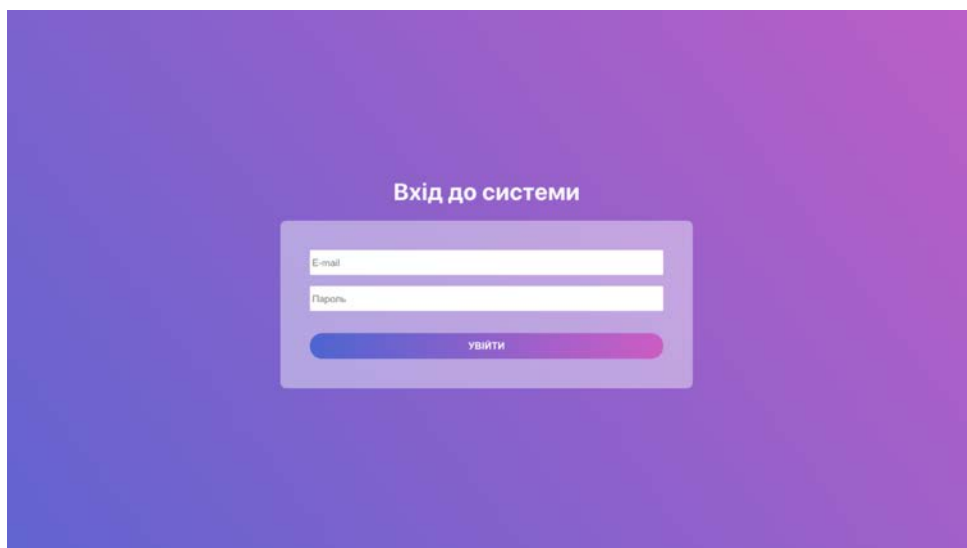


Рисунок 5.1 – Сторінка авторизації

Після авторизації, перед користувачем відкривається сторінка з усіма проєктами, на котрих він є учасником (рис. 5.2).

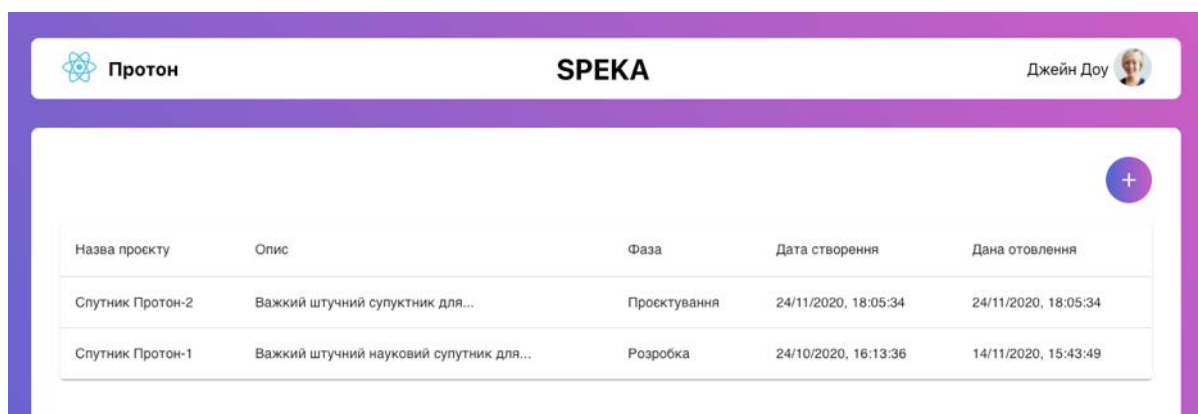


Рисунок 5.2 – Сторінка з проєктами користувача

Користувач має можливість створити новий проєкт, ввівши його назву та опис у відповідному модальному вікні (рис. 5.3).

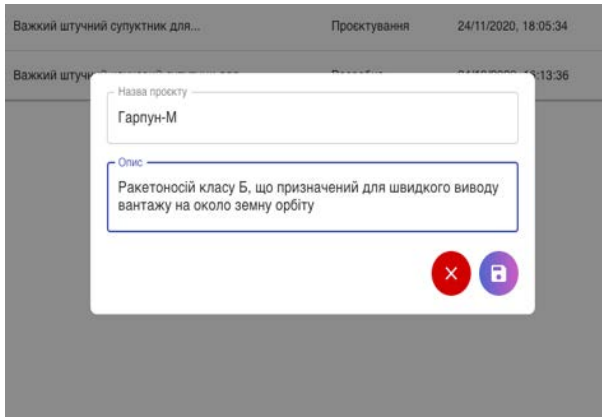


Рисунок 5.3 – Модальне вікно створення проєкту

На сторінці з інформацією про проєкт (рис. 5.4), є інтерфейс для управління проєктом, а також таблиця з компонентами.

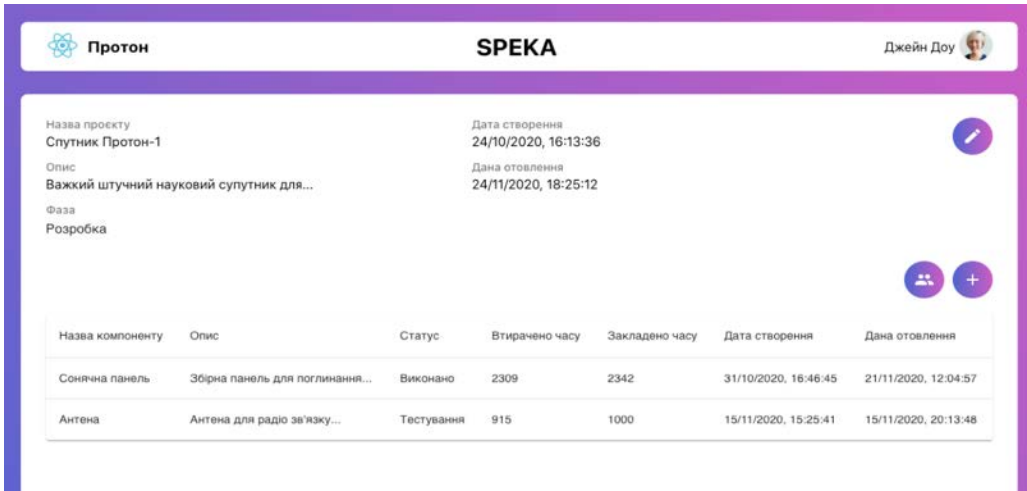


Рисунок 5.4 – Сторінка проєкту

Користувач може змінювати опис проєкту, а також поточно фазу (рис. 5.5).

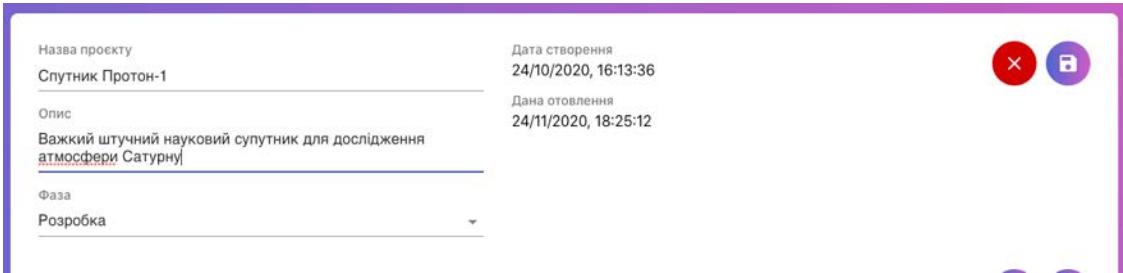


Рисунок 5.5 – Редагування проєкту

Користувач також має можливість переглянути список учасників команди проєкту (рис. 5.6), а також додати нових за потребою (рис. 5.7).

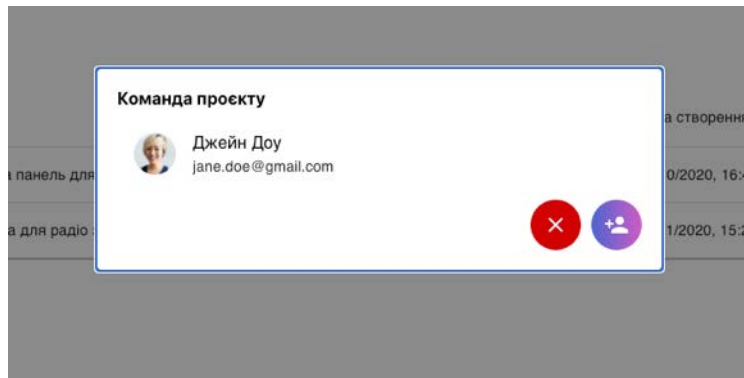


Рисунок 5.6 – Модальне вікно учасників команди проєкту

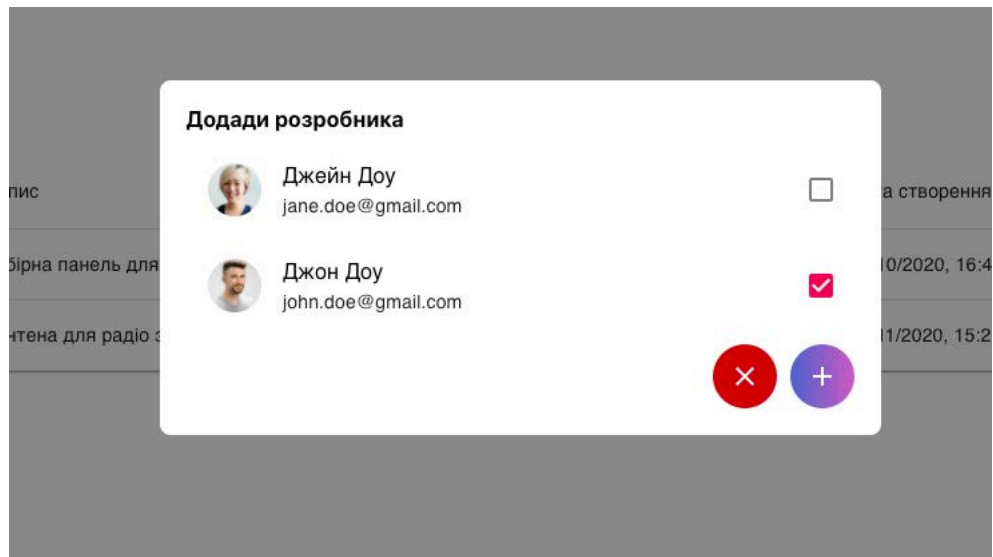


Рисунок 5.7 – Модальне вікно додавання розробника до проєкту

Кожний учасник команди проєкту має можливість створити новий компонент (рис. 5.9).

Рисунок 5.9 – Сторінка створення компоненту

Детальну інформацію про компоненту можна переглянути на його сторінці (рис. 5.10). При потребі, користувач може змінювати інформацію та поточний статус компоненту.

Рисунок 5.10 – Сторінка компоненту

Порівняння поточної версії та минулої відбувається за допомогою алгоритму Myers, що порівнює два тестових фрагментів, різниця між якими явно демонструється за допомогою WYSISWYG редактору CKEditor 5 (рис. 5.11).

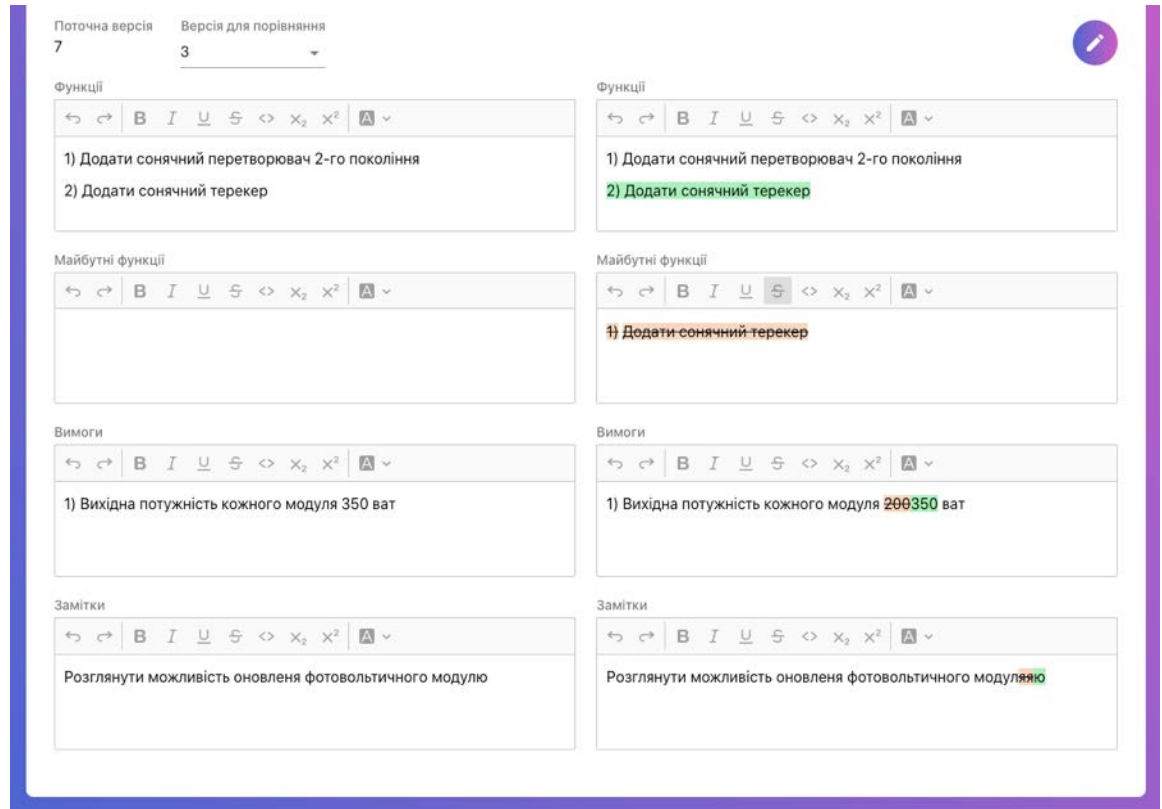


Рисунок 5.11 – Порівняння поточної та минулої версії специфікації

Щоб вийти з системи, користувачеві необхідно натиснути на аватар в верхній панелі застосунку де він і зможе обрати функцію завершення авторизованої сесії. Для завершення роботи з застосунком, користувачеві достатньо закрити вкладу в браузері або перейти на інше посилання.

5.2.4 Повідомлення оператору

Для створення правильного та ефективного UX програмний застосунок використовує повідомлення, що надають інформацію користувачеві про поточний стан системи на наступні дії.

На екрані авторизації, в разі ведення некоректного email або паролю, користувачеві буде виведено наступне повідомлення “Введений email або пароль є неправильними”.

У разі закінчення сесії та неможливості її відновлення, користувач отримає повідомлення “Сесія закінчена, для продовження роботи, будь ласка, увійдіть знову”.

У випадку надсилання некоректних даних до серверу, користувач отримує повідомлення з інформацією про невалідний параметр виклику.

У разі пошуку неіснуючого проєкту чи компоненту, користувач отримує повідомлення про те, що даний проєкт чи компонент не існує в базі даних.

Якщо помилка трапляється на сервері, то користувачеві відображається наступне повідомлення “На сервері трапився внутрішній збій”.

Всі ці повідомлення носять інформаційний характер, з наданням інструкцій для подальших дій, як у випадку з закінченою сесією.

Інтуїтивно зрозумілий інтерфейс та валідація полів введення інформації, надають користувачеві представлення роботи з програмним застосунком не потребуючи додаткових підказок від системи.

5.3 Методика та результати тестування

Для тестування було використано підхід, при якому перевіряється надійність програмного застосунку. Це робиться для знаходження виняткових ситуацій, що можуть відбутися під час виконання застосунку. В ході тестування було виявлено слабкі сторони застосунку, та надано інструкції оператору для подальших дій при виникненні виняткових ситуацій, що описано у розділі 5.2.4. Також було створено систему, яка відловлює помилки та забезпечує надійне функціонування застосунку.

Стабільно роботу з розробленим програмним застосунком забезпечує оброблення виняткових ситуацій, що є можливим з використанням `try`, `catch`, `finally` блоків мови програмування TypeScript. Якщо помилка трапляється у блоці

try, то блок catch моментально відловлює помилку та надає інформацію про збій, що далі використовується для обробки виняткової ситуації. Блок finally завжди виконується після завершення блоку try або catch. В розробленому застосунку блок finally використовується для зупинення процесу завантаження. Використання блоків try, catch, finally є важливим, оскільки саме вони запобігають повному краху скрипта програми.

На серверній частині застосунку також використовуються можливості фреймворку NestJS та бібліотеки class-validator, що виконують валідацію даних, відловлюють помилки які не були перехоплені за допомогою блоків try, catch, finally, а також надають інформацію, про збій, котра може бути використана розробником або відображена користувачеві.

На клієнтській частині застосунку також використовуються можливості бібліотеки React, яка надає такі методи як componentDidCatch та getDeliveredErrorFromProps. Дані методи являються частиною життєвого циклу компоненту, та надають можливість інкапсулювати помилку у даному компоненті без будь-якого впливу на батьківські компоненти застосунку.

5.4 Експериментальні дослідження та отримані результати

Дослідження покращеного алгоритму Myers відбувалося за допомогою benchmark-тестування, головними критеріями якого є час виконання оптимізованої та звичайної версій алгоритму, а також стабільність роботи методів.

На рисунку 2.6 зображено результати виконання benchmark-тестування. Згідно з отриманих результатів benchmark-тестування можна зробити висновок, що оптимізована версія алгоритму Myers є швидшою на 25.4% ніж його базова імплементація на JS, а також є стабільнішою, оскільки межі в яких швидкість алгоритму коливається є $\pm 1.12\%$, в той час як базова імплементація має межі коливання в $\pm 7.84\%$.

Також типізація алгоритму за допомогою TypeScript, а також мемоізація, значно підвищують легкість інтеграції алгоритму у система, та оптимальне використання ресурсів, навіть у тих випадках коли підключення алгоритму до системи виконано не самим оптимальним способом.

На рисунку 5.11 наглядно зображено результат виконання оптимізованого алгоритму Myers.

5.5 Аналіз, узагальнення та обґрунтування достовірності отриманих результатів

Ретельне тестування алгоритму Myers та розробленого застосунку для відстеження версій апаратного забезпечення показали, що розроблена система є надійною та добре оптимізованою, оскільки бере до уваги особливості роботи движків JS для найоптимальнішого використання їхніх можливостей.

У порівнянні з аналогами, розроблений застосунок надає удосконалений життєвий цикл апаратного забезпечення за допомогою інтегрованої системи таск-менеджменту, що в свою чергу дає користувачеві більше можливостей для контролю процесу розробки, а також підвищує швидкість та продуктивність роботи команди.

Реалізована система версійності та порівняння тесових фрагментів забезпечує легку роботу з специфікаціями компонентів апаратного забезпечення, а також зменшує кількість помилок, що можуть виникнути у ході її складання.

Запропоноване рішення комбінує у собі сучасний підхід до організації робочого процесу, а також сучасні технологічні рішення, що підвищують надійність та продуктивність системи.

У перспективі, у розробленому додатку можна покращити засоби взаємодії між учасниками команди шляхом додавання сповіщень, а також коментарів до специфікації. Для покращення оптимізованої версії алгоритму Myers, можна буде використати Web Assembly, після того, як розробники даної технології виправлять проблему з транспортуванням даних між Web Assembly та

JavaScript. Дане рішення дозволить досягти швидкості виконання алгоритму на рівні швидкості виконання його реалізації на C++.

5.6 Висновки за розділом 5

У 5 розділі було виконано опис розробленого застосунку, розроблено інструкцію з експлуатації програми, а також проведено тестування та експериментальні дослідження. У ході аналізу отриманих результатів було встановлено, що розроблений застосунок надає комбінацію функціоналу та технічних рішень, що значно покращують процес розробки апаратного забезпечення у порівнянні з аналогами системи.

6 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

Метою магістерської дипломної роботи є створення програмного дослідження та програмна реалізація методів відстеження версій апаратного забезпечення.

6.1 Опис ідеї проєкту

В таблиці 6.1 наведено опис ідеї проєкту.

Таблиця 6.1 – Опис ідеї

Зміст ідеї	Напрямок застосування	Вигоди для користувача
Система відстеження версій апаратного забезпечення з удосконаленням життєвого циклу апаратного забезпечення шляхом впровадження таск-менеджменту, версіонування специфікації, а також впровадження методу порівняння текстових фрагментів за допомогою оптимізованого під поточну платформу алгоритму Myers.	Державне промислове виробництво	Компонування процесу життєвого циклу апаратного забезпечення та таск-менеджменту надає можливість створювати команду розробників та розподіляти між ними обов'язки на проєкті. Загальний статус проєкту та статуси компонентів дають можливість створення аналітичної бази, аналізувати процес розробки та вносити до нього зміни. Можливість порівнювати переглядати минулі версії специфікації та порівнювати їх з поточною за допомогою алгоритму порівняння текстових фрагментів та візуалізації результату, підвищує швидкість, ефективність та зменшує кількість помилок при створенні специфікації компонентів
	Приватні підприємства	
	Фізична особа-підприємець	

Основую даного проєкту являється шаблон програмування назвою “Модель-Представлення-Контролер” (Model-View-Controller, MVC). В якості “Моделі” використовується реляційна база даних з використанням СКБД PostgreSQL. В якості “Представлення” використовується Web-застосунок написаний з використанням бібліотеки React та мови програмування TypeScript, що дозволяє з легкістю впровадити та оптимізувати алгоритм Myers для порівняння текстових фрагментів. Для реалізації “Контролеру” використовується серверний застосунок створений на платформі NodeJS з використанням фреймворку NestJS та мови програмування TypeScript.

Даний підхід гарантує швидку та в міру витратну розробки системи, а також обраний стек технологій гарантує, що на ринку праці завжди є висококваліфіковані спеціалісти, що здатні розроблювати дану систему, та надавати консультації. Розроблювати проєкт можуть як і вузькоспеціалізовані Front-end та Back-end розробники, так і один Full-stack розробник.

6.2 Аналіз ринкових можливостей запуску реалізації ідеї

В таблиці 6.2 наведено попередню характеристику потенційного ринку

Таблиця 6.2 – Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Головні конкуренти	Головними конкурентами є компанії, що займаються розробкою систем контролю життєвого циклу апаратного забезпечення та таск-менеджменту
2	Динаміка ринку	В зв'язку з кризою викликаною пандемією коронавірусу, деякі підприємства зменшують обсяги виробництва, що в свою чергу викликає спад

Продовження таблиці 6.2

3	Наявність обмежень для входу	Відсутні. Конкуренти не надають можливості компонування функціоналу життєвого циклу та таск-менеджменту з можливістю версіювання специфікації
4	Специфічні вимоги до стандартизації та специфікації	Відсутні. Інтуїтивно зрозумілий інтерфейс повністю задовольняє стандартні вимоги

В таблиці 6.3 наведено потенційні групи клієнтів, їх характеристики, а також сформульовано орієнтований перелік вимог до товару для кожної з груп.

Таблиця 6.3 – Попередня характеристика потенційних клієнтів

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп	Вимоги споживачів (користувачів)
1	Потреба у описі проекту та його компонентів з можливістю зміни поточної фази та статусу з цілю моніторингу прогресу та аналітики	Державне промислове виробництво	Використання застосунку для загального опису проекту та зміни фаз та статусу процесу розробки	Інтуїтивно зрозумілий інтерфейс та можливість швидко змінювати фази виконання проекту та поточні статуси компонентів проекту
2	Потреба у формуванні команди, розподілу відповідальності між її учасниками	Приватні підприємства	Використання застосунку з цілю управління командою та таск-менеджментом	Можливість менеджменту команди та делегація компонентів серед учасників

Продовження таблиці 6.3

3	Потреба у веденні специфікації компонентів з можливістю переглядати попередні версії та порівнювати їх з поточно	Фізична особа-підприємець	Використання програмного застосунку з цілю ведення документації, з можливістю перегляду версій та їх порівняння для швидкої передачі продукту чи додавання нових розробників	Інтуїтивно зрозумілий інтерфейс для швидкої інтеграції нових працівників у робочій процес, ведення документації компонентів та можливість порівняти минулі версії з поточною
---	--	---------------------------	--	--

6.3 SWOT-аналіз

У таблиці 6.4 наведено SWOT-аналіз середовища реалізації інноваційного проєкту.

Таблиця 6.4 – SWOT аналіз

Сильні сторони (S)	Слабкі сторони (W)
Інноваційність, оптимізація алгоритму Myers, широкий спектр використання	Витрати на підтримку системи, залежність від сторонніх бібліотек
Можливості (O)	Загрози (T)
Впровадження інноваційної системи в виробництво, вихід на міжнародний ринок, розширення функціоналу	Зловживання можливістю збереження версій специфікації, що веде до надлишку даних, поява конкурентів

6.4 Цільові групи потенційних споживачів

У таблиці 6.5 наведено результат вибору потенційних груп споживачів.

Таблиця 6.5 – Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Державне промислове виробництво	Низька	Середній	Майже відсутня	Важка
2	Приватні підприємства	Висока	Високий	Низька	Середня
3	Фізична особа-підприємець	Середня	Середній	Низька	Легка
Які цільові групи обрано: 2, 3					

6.5 Планування дослідження та розробки програмного продукту

Весь процес розробки програмного забезпечення ділиться на етапи. Для кожного етапу вказується трудомісткість, кількість виконавців і тривалість робіт. У розробці беруть участь програміст протягом 1,6 місяця та консультант протягом 0,25 місяця. Розробка розпочинається 2 вересня і повинна бути виконана до 28 жовтня 2020 року. Результати планування та етапи наведено у таблиці 6.6 [27].

Таблиця 6.6– Етапи розробки програмного забезпечення

Найменування робіт	Трудомісткість		Виконавці	Тривалість, днів
	люд.- днів	% до підсумку		
1 Аналіз предметної області	2	3,57	програміст	2
2 Аналіз методів збереження історії даних	2	3,57	програміст	2
3 Аналіз методів порівняння текстових фрагментів	2	3,57	програміст	2
4 Визначення вимог для програмного застосунку	3	5,36	програміст, консультант	2 1
5 Проєктування архітектури	5	8,93	програміст, консультант	3 2
6 Створення програмного коду	33	58,93	програміст	33
7 Тестування та налагодження програмного застосунку	5	8,93	програміст, консультант	3 2
8 Складання програмної документації	4	7,14	програміст, консультант	2 2
Разом	56	100		56

Виходячи з отриманих даних таблиці 6.6, створюємо стрічковий графік планування розробки продукту (табл. 6.7), де сірий колір – консультант, чорний – програміст.

Таблиця 6.7 – Стрічковий графік планування розробки продукту

Найменування робіт	Календарний період, дні							
	02.09-08.09	09.09-15.09	16.09-22.09	23.09-29.09	30.09-06.10	07.10-13.10	14.10-20.10	21.10-28.10
1 Аналіз предметної області								
2 Аналіз методів збереження історії даних								
3 Аналіз методів порівняння текстових фрагментів								
4 Визначення вимог для програмного застосунку								
5 Проектування архітектури								
6 Створення програмного коду								
7 Тестування та налагодження програмного застосунку								
8 Складання програмної документації								

6.6 Визначення витрат на розробку

Для визначення витрат на розробку програмного застосунку складається калькуляція кошторисної вартості робіт, яка включає наступні пункти [27]:

- основна заробітна плата;
- додаткова заробітна плата;
- відрахування на єдиний соціальний внесок;
- матеріали і комплектуючі вироби;
- витрати на спеціальне обладнання;
- накладні витрати.

Для розрахунку заробітної слід взяти до уваги, що час, витрачений на розробку програмного застосунку складає у кількості 56 робочих днів.

Тривалість робіт визначають за формулою 6.1.

$$T_{ц} = \frac{Q}{R}, \quad (6.1)$$

де $T_{ц}$ – кількість днів;

Q – трудомісткість;

R – кількість робітників (люд.).

6.6.1 Розрахунок основної заробітної плати

У таблиці 6.8 проводиться розрахунок основної заробітної плати.

Таблиця 6.8 – Розрахунок основної заробітної плати

Посада виконавця	Чисельність, чол.	Місячний оклад, грн	Кількість місяців	Сума ЗП, грн
Програміст	1	12 000	1,6	19 200
Консультант	1	15 000	0,25	3 750

Продовження таблиці 6.8

Підсумок	2			22 950
----------	---	--	--	--------

6.6.2 Розрахунок додаткової заробітної плати

Додаткова заробітна плата приймається рівною 10% від основної заробітної плати працівника і розраховується за формулою 6.2.

$$ЗП_{\text{дод}} = ЗП_{\text{осн}} \times 0,1. \quad (6.2)$$

Після підстановки величини основної заробітної плати в дану формулу, отримується:

$$ЗП_{\text{дод}} = 22\,950 \times 0,1 = 2\,295 \text{ грн.}$$

6.6.3 Відрахування на єдиний соціальний внесок

Відрахування на єдиний соціальний внесок становлять 22% і беруться від основної та додаткової заробітної плати. Вони розраховуються за формулою 6.3 [27]:

$$ВІД = (ЗП_{\text{осн}} + ЗП_{\text{дод}}) \times 0,22, \quad (6.3)$$

де $ЗП_{\text{осн}}$ – основна заробітна плата (грн.);

$ЗП_{\text{дод}}$ – додаткова заробітна плата (грн.).

$$ВІД = (22\,950 + 2\,295) \times 0,22 = 5\,553,9 \text{ грн.}$$

Додаткові соціальні відрахування становлять 5 553,9 грн.

6.6.4 Визначення витрат на матеріал

Витрати на матеріали розраховуються за формулою 6.4.

$$M = \sum_{i=1}^n (\text{Ц}_i N_i (1 + K_{\text{т.з.}}) - \text{Ц}_{\text{іо}} N_{\text{іо}}), \quad (6.4)$$

де M – витрати на матеріали, покупні напівфабрикати та комплектуючі (грн.);

$K_{\text{т.з.}}$ – коефіцієнт, що враховує транспортно–заготівельні витрати, $K_{\text{т.з.}} = 0,05$;

Ц_i – ціна i -го найменування чи комплектуючого (грн.);

N_i – потреба в i -му матеріалі, напівфабрикаті, комплектуючих;

$\text{Ц}_{\text{іо}}$ – ціна зворотних відходів i -го найменування матеріалу (грн.), $\text{Ц}_{\text{іо}} = 0$;

$N_{\text{іо}}$ – кількість зворотних відходів i -го найменування, $N_{\text{іо}} = 0$;

n – кількість найменувань матеріалів, напівфабрикатів та комплектуючих.

У таблиці 6.8 наведено витрати на матеріали для розробки.

Таблиця 6.8 – Витрати на матеріали

Найменування, вид, тип, марка	Одиниця виміру	Ціна за одиницю вим., грн.	Кількість од. вим.	Сума витрат, грн.
Набір канцелярії Dream CZ000501	Набір	250	2	500
Картридж для лазерного принтера Cannon 054	шт.	1864	1	1 865
Папір формату A4 Ballet Universal	Листів	0,186	500	93
Всього				2 458

Розраховуємо остаточні витрати на матеріали:

$$M = (1 + 0,05) \times (500 + 1865 + 93) = 2\,580,9 \text{ грн.}$$

Разом витрати на матеріали складають 2 580,9 грн.

6.6.5 Витрати на ПК

У статтю витрат на ПК входять витрати на придбання, транспортування, монтаж і налагодження нестандартного обладнання. Також необхідно розрахувати витрати на оплату машинного часу ЕОМ (амортизаційні витрати на обслуговування та вартість спожитої електроенергії).

Амортизаційні відрахування обчислюються за формулою 6.5.

$$A = \Phi_B \frac{H_a}{100}, \quad (6.5)$$

де Φ_B – балансова вартість обчислювальної техніки (грн.);

H_a – норма амортизаційних відрахувань на повне відновлення обчислювальної техніки (%).

Балансова вартість ноутбуку Asus X509JP-EJ068 (90NB0RG2-M01020), що використовується становить 19 500 грн. Одержуємо:

$$A = 19\,500 \times 0,25 = 4\,875 \text{ грн.}$$

Амортизаційні відрахування становлять 4 875 грн.

Для розрахунку вартості експлуатації обладнання використаємо формулу 7.6.

$$C_e = N_n \Phi_{\text{еф}} K_{\text{зч}} K_{\text{зп}} C_e, \quad (6.6)$$

де N_n – номінальна потужність ЕОМ, кВт;

$\Phi_{\text{еф}}$ – річний ефективний фонд часу роботи ЕОМ;

$K_{\text{зм}}$ – середній коефіцієнт завантаження за часом;

$K_{\text{зп}}$ – коефіцієнт завантаження за потужністю;

C_e – ціна одного кВт*год. Електроенергії, грн./(кВт*год.).

Номинальна потужність ЕОМ становить 0,2 кВт. Річний ефективний фонд часу роботи ЕОМ (ноутбуку працівника) становить 1800 годин. Середні коефіцієнти завантаження за часом і за потужністю становлять 0,6 і 0,9. Ціна однієї кіловат-години електроенергії становить 1,68 грн.

$$C_{el} = 0,2 \times 1800 \times 0,9 \times 0,6 \times 1,68 = 326,6 \text{ грн.}$$

Заробітна плата обслуговуючого персоналу розраховується за формулою 6.7.

$$ЗП_{обсл} = ФЗР_p (1 + K_{від}) \frac{t_{обсл}}{Ф_{эф. обсл}}, \quad (6.7)$$

де $ФЗР_p$ – річний фонд заробітної плати (основної та додаткової) обслуговуючих робіт (грн.);

$K_{від}$ – коефіцієнт відрахувань на соціальне страхування і до інших фондів (%);

$t_{обсл}$ – час протягом року, необхідний на технічне обслуговування ЕОМ, (ч/рік);

$Ф_{эф. обсл.}$ – річний ефективний фонд часу обслуговуючого персоналу.

Місячна заробітна плата обслуговуючого персоналу становить 5000 грн., а річний фонд заробітної плати становить 55 000 грн.

Річний ефективний фонд робочого часу обслуговуючого ПК робітника становить 1800 годин/рік.

На обслуговування одного ПК витрачається по 1 годині на місяць, що на рік становить 12 години. З чого виходить, що:

$$ЗП_{обсл} = 55\,000 \times (1 + 0.22) \times \frac{12}{1800} = 447,33 \text{ грн.}$$

Стаття “Поточний ремонт обладнання” приймається рівною 3% від балансової вартості обладнання і складає 585.

Стаття “Інші витрати” приймається рівною п'яти відсоткам від суми всіх попередніх статей витрат на утримання та експлуатацію устаткування. Сума всіх попередніх статей дорівнює 6 233,93 грн., 5% від суми становлять 311,7 грн [27].

Кошторис витрат на утримання та експлуатацію обладнання описано у таблиці 6.9.

Таблиця 6.9 – Кошторис витрат

Найменування статей витрат	Сума, грн.
Амортизація обладнання	4 875
Експлуатація ПК (окрім витрат на поточний ремонт)	326,6
Заробітна плата основна та додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	447,33
Поточний ремонт обладнання	585
Інші витрати	311,7
Разом	6 545,63

Витрати на оплату машинного часу ЕОМ для написання й відлагодження програмного застосунку вираховується за формулою 6.8.

$$C_{mc} = P_{екс} t_{мч} , \quad (6.8)$$

де C_{mc} – витрати на сплату машинного часу (грн.);

$P_{екс}$ – експлуатаційні витрати на один час машинного часу ЕОМ;

$t_{мо}$ – машинний час ЕОМ для написання та відлагодження програмного застосунку.

$$P_{екс} = \frac{6\,545,63}{1800} = 3,64 \text{ грн./машино-час.}$$

ЕОМ експлуатується 56 днів в одну зміну, що становить у сумі 448 годин. Таким чином витрати на сплату машинного часу складуть:

$$C_{mc} = 3,64 \times 448 = 1630,72 \text{ грн.}$$

6.6.6 Витрати на сервер

У якості сервера використовується Сервер ARTLINE Business T17 (T17v09) балансової вартості 19 000 грн. Підставивши у формулу 6.5 значення балансової вартості отримаємо:

$$A = 19\,000 \times 0,25 = 4\,750 \text{ грн.}$$

Амортизаційні відрахування становлять 4 750 грн.

За формулою (6.6) річний фонд часу роботи ЕОМ (сервера) становить 8760 годин. Середні коефіцієнти завантаження за часом і за потужністю становлять 0,6 і 0,9. Ціна однієї кіловат-години електроенергії становить 1,68 грн.

$$C_{e2} = 0,2 \times 8760 \times 0,9 \times 0,6 \times 1,68 = 1\,589,41 \text{ грн.}$$

Заробітна плата обслуговуючого персоналу розраховується за формулою (7.7).

Місячна заробітна плата обслуговуючого персоналу становить 5000 грн., а річний фонд заробітної плати становить 55 000 грн.

Річний ефективний фонд робочого часу обслуговуючого ПК робітника становить 1800 годин/рік.

На обслуговування одного ПК витрачається по 1 годині на місяць, що на рік становить 12 години. З чого виходить, що:

$$ЗП_{\text{обсл}} = 55\,000 \times (1 + 0.22) \times \frac{12}{1800} = 447,33 \text{ грн.}$$

Стаття “Поточний ремонт обладнання” приймається рівною 3% від балансової вартості обладнання і складає 570.

Стаття “Інші витрати” приймається рівною п'яти відсоткам від суми всіх попередніх статей витрат на утримання та експлуатацію устаткування. Сума всіх попередніх статей дорівнює 7 356,74 грн., 5% від суми становлять 367,84 грн.

Кошторис витрат на утримання та експлуатацію обладнання описано у таблиці 6.10.

Таблиця 6.10 – Кошторис витрат

Найменування статей витрат	Сума, грн.
Амортизація обладнання	4 750
Експлуатація ПК (окрім витрат на поточний ремонт)	1589,41
Заробітна плата основна та додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	447,33
Поточний ремонт обладнання	570
Інші витрати	367,84
Разом	7 724,58

Для розробки ЕС також експлуатується сервер протягом 56 днів. Таким чином витрати на сплату машинного часу становлять:

$$C_{\text{мс}} = \frac{56}{365} \times 7\,724,58 = 1\,185,14 \text{ грн.}$$

6.6.7 Накладні витрати

Накладні витрати – це кошти, що витрачаються на загальне управління, загальногосподарські потреби (заробітна плата апарату управління, канцелярія тощо), а також витрати на експлуатація та утримання будівель [27].

Накладні витрати включають у вартість розробки програми непрямым шляхом – у відсотках до основної заробітної плати розробників. В даному випадку накладні витрати становлять 40% до основної заробітної (22 950 грн.) плати розробників, що складає 9 180 грн [27].

Результати визначення витрат на розробку програми приведені в таблиці 6.11.

Таблиця 6.11 – Вартість розробки програми

Стаття витрат	Сума (грн.)	Питома вага до підсумку (%)
Основна заробітна плата	22 950	50,58
Додаткова заробітна плата	2 295	5,06
ЄСВ	5 553,9	12,24
Вартість матеріалів	2 580,9	5,69
Витрати на ПК	1 630,72	3,59
Витрати на сервер	1 185,14	2,61
Накладні витрати	9 180	20,23
Разом	45 375,66	100,00

6.6.8 Розрахунок експлуатаційних витрат

Витрати, пов'язані з експлуатацією програмного застосунку протягом року, можна визначити за формулою 6.9.

$$И = P_{\text{екс. ктс}} \times T_{\text{ктс}} + \frac{S_{\text{рп}}}{n \times T_{\text{с}}}, \quad (6.9)$$

де $P_{\text{екс. ктс}}$ – експлуатаційні витрати, що припадають на 1 ч машинного часу КТС споживача програмного продукту, грн / машино–год;

$T_{\text{ктс}}$ – сумарний машинний час КТС протягом року, потрібний даному споживачеві програми для задач, що вирішуються за її допомогою, машино–год;

$S_{\text{рп}}$ – сумарні витрати на розробку програми, грн .;

n – кількість споживачів даної програми;

$T_{\text{с}}$ – термін служби даної програми до її морального зносу, років.

$$И = 7\,724,58 + \frac{45\,375,66}{1 \times 10} = 12\,262,15 \text{ грн.}$$

6.6.9 Розрахунок витрат на адміністратора

Місячна заробітна плата адміністратора становить 6000 грн., звідси знаходимо, що річний фонд заробітної плати адміністратора становить [27]:

$$ЗП_{\text{адм}} = 6000 \times 12 \times (1 + 0,22) = 87\,840 \text{ грн.}$$

6.7 Розрахування капітальних вкладень

Обсяг капітальних вкладень споживача програмного застосунку K , пов'язаних з його розробкою / купівлею / впровадженням, визначають за формулою 6.10.

$$K = K_{\text{пп}} + K_{\text{пф}}, \quad (6.10)$$

де $K_{\text{пп}}$ – виробничі витрати, грн .;

$K_{пф}$ – капітальні вкладення в виробничі фонди, необхідні для впровадження програми, грн.

Виробничі витрати, що включають витрати пов'язані з розробкою і налагодженням програмного застосунку, математичного та інформаційного забезпечення, проєктуванням системи, знаходяться за формулою 6.11.

$$K_{пл} = \frac{S_{рп}}{n} + K_{пр пл}, \quad (6.11)$$

де $S_{рп}$ – сумарні витрати на розробку або придбання програмного продукту, грн;

n – кількість споживачів даної програми;

$K_{пр пл}$ – інші виробничі витрати, грн.

Капітальні вкладення в виробничі фонди включають витрати на придбання, установку, монтаж і налагодження комплексу технічних засобів (КТЗ) з урахуванням можливого використання КТЗ як для вирішення завдань для яких призначена розробляється програма, так і для завдань, що вирішуються за іншими програмними засобами [27].

Питомі капітальні вкладення в виробничі фонди, що припадають на частку даного програмного продукту, можна визначити за формулою 6.12.

$$K_{пф} = \frac{K_{ктс} \times T_{ктс}}{\Phi_{еф.ктс}}, \quad (6.12)$$

де $K_{ктс}$ – витрати на придбання, установку, монтаж і налагодження КТЗ, грн. ;

$T_{ктс}$ – машинний час КТЗ, потрібне даному споживачеві програми для завдань, що вирішуються за допомогою даної програми, машино–годин / рік;

$\Phi_{еф.ктс}$ – річний ефективний фонд часу роботи КТЗ, машино–год / рік.

$$K = \frac{45\,375,66}{1} + \frac{19\,000 \cdot 8\,760}{8\,760} = 64\,375,66 \text{ грн.}$$

6.8 Оцінка економічної ефективності

Метою створення даного програмного застосунку є отримання прибутку, залучення інвесторів та клієнтів, покращення процесу створення апаратного забезпечення та його документування.

Мінімальна ціна підписки на подібні програмні застосунки коштує 2000 грн. на місяць. Виходячи з цього було вирішено продати 100 підписок вартістю 1700 гривень.

Прибутком від продаж можна вважати кошти, що були отримані від підписок користувачів за виключенням експлуатаційних витрат, а також заробітної плати адміністратору [27]. Роль адміністратора полягає у підтримці зв'язку з клієнтами, а також у вирішенні технічних питань.

Прибуток можна розрахувати за формулою 6.13.

$$\Pi = \Pi_{\text{ост. від продажу}} - 3\Pi_{\text{адм}} - \text{И} \quad (6.13)$$

$$\Pi = 1700 \times 100 - 87\,840 - 12\,262,15 = 69\,897,85 \text{ грн.}$$

Річну економію можна розрахувати за формулою 6.14.

$$E_p = \Pi - E_n^a \times K, \quad (6.14)$$

де E_p – річна економія від впровадження системи, грн.;

Π – річний прибуток від впровадженої системи;

E_n^a – нормативний коефіцієнт ефективності капітальних вкладень в засоби автоматизації;

K – обсяг капітальних витрат, пов'язаних зі створенням ПП.

$$E_p = 69\,897,85 - 0,15 \times 64\,375,66 = 60\,241,5 \text{ грн.}$$

Коефіцієнт ефективності вкладених коштів в програмний застосунок E і термін їх окупності $T_{ок}$ розраховують за формулами 6.15 та 6.16.

$$E = \frac{\Pi}{K} \quad (6.15)$$

$$T_{ок} = \frac{K}{\Pi} \quad (6.16)$$

Виходячи з цього, за формулою 6.15 знаходимо:

$$E = \frac{69\,897,85}{64\,376,6} = 1,1.$$

За формулою 6.16 знаходимо:

$$T_{ок} = \frac{64\,376,6}{69\,897,85} = 0,92 \text{ року.}$$

Виходячи з цього, можна зробити висновок, що розроблений програмний застосунок є економічно доцільним, оскільки його окупність складає 0,92 року, що є значно меншим терміном за допустиме значення часу окупності продукту, що в свою чергу становить 6,67 року.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Аналіз потенційних небезпек

Для роботи з функціоналом програмного застосунку, що було розроблено, користувач повинен використовувати персональний комп'ютер. В залежності від комплектації, користувач може використовувати периферійні пристрої та інші електронні засоби.

Для планування та організації заходів безпеки було проведено аналіз потенційних небезпек з подальшим визначенням потенційно небезпечних виробничих чинників, пошук причин їх появи та з'ясування можливих наслідків впливу на людину [28].

До небезпек, що являються потенційними при роботі у в офісах чи кабінетах слід віднести наступні групи [28]:

- потенційні небезпеки фізичного характеру;
- потенційні небезпеки психологічного характеру;
- потенційні небезпеки санітарно-гігієнічного характеру;
- потенційні небезпеки, пов'язані з порушенням правил пожежної безпеки;
- потенційні небезпеки, пов'язані з проявом наслідків надзвичайних ситуацій.

Для забезпечення заходів безпеки умов праці було детально розглянуто кожну з потенційних небезпек, що вказано вище.

Потенційні небезпеки фізичного характеру пов'язані з можливим механічним травмуванням, причиною якого є недотримання вимог ергономіки, зокрема неправильне планування робочого простору та незадовільний стан оснащення офісу та інші фактори, вірогідність бути ураженим електричним струмом у разі несправності комп'ютерної та офісної техніки, або недотриманні правил електробезпеки, а також негативні наслідки на опорно-руховий апарат робітника, який працює тривалий період часу у малорухливому стані [28].

Потенційні небезпеки психологічного характеру пов'язані з розумовим та емоційним вигоранням робітника, що є наслідком перенавантаження працівника завданнями з досить вузькими часовими проміжками для виконання, праця в понадурочний час, відсутність перерв під час робочого дня, а також незадоволення організацією робочого місця, що і викликає зниження працездатності, втому та підвищене психофізичне навантаження [28].

Потенційні небезпеки санітарно-гігієнічного характеру пов'язані з негативним впливом електромагнітних полів та іонізуючого випромінювання, що створюються офісним обладнанням, незадовільним станом повітряного середовища, негативним впливом недостатнього або підвищеного освітлення робочої зони, підвищеним рівнем шуму та вібрацій робочого середовища тощо [28].

Потенційні небезпеки, пов'язані з порушенням правил пожежної безпеки, що є наслідками недотримання правил пожежної безпеки, коротке замикання, займання електричного обладнання, причиною якого є несправність або низька якість техніки.

Потенційні небезпеки, пов'язані з проявом наслідків надзвичайних ситуацій, що є наслідками неправильних дій персоналу в умовах надзвичайних ситуацій, таких як пожежа, стихійне лихо тощо [28].

7.2 Заходи із забезпечення безпеки

З цілю запобігання виникненню небезпек фізичного, психологічного, санітарно-гігієнічних, а також небезпек пов'язаних порушенням правил пожежної безпеки та з проявами наслідків надзвичайних ситуацій, було розглянуто та проведено аналіз потенційних небезпек з посиланням на нормативно-правові документи. Проведений аналіз документів надав можливість спланувати та провести низку заходів, що є необхідними для забезпечення техніки безпеки.

Для планування заходів з забезпечення техніки безпеки, було переглянуто та проаналізовано наступні нормативно правові документи:

- ПУЕ-2017 “Правила улаштування електроустановок”;
- ДСанПіН 3.3.2.007-98 “Державні санітарні правила и норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин”;
- ДСН 3.3.6.042-99 “Державні санітарні норми параметрів мікроклімату”;
- НАПБ Б.03.002-2007 “Норми визначення категорій приміщень, будинків та зовнішніх установок по вибухопожежної та пожежної безпеки”;
- НПАОП 0.00-1.28-10 “Правила охорони праці під час експлуатації електронно-обчислювальних машин”;
- ДСН 3.3.6.037-99 “Санітарні норми виробничого шуму, ультразвуку та інфразвуку”;
- НПАОП 0.00-4.12-05 “Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці”.

Заходи з забезпечення техніки безпеки діляться на організаційні та технічні [28].

Згідно до стандарту [29], ПК підключено до мережі змінного струму з напругою 220В та частотою 50Гц. Приміщення в якому знаходяться ПК відносяться до категорії “Приміщення без підвищеної небезпеки”. При суворому дотриманні правил стандарту [29], ураження електричним струмом є можливим тільки у випадку несправності електричного прибору, або пошкодженні кабелів живлення.

Згідно до стандарту [29], для захисту персоналу від ураження електрострумом, захисне заземлення приміщень виконано таким чином, що опір контуру захисного заземлення електрообладнання приміщення з ПК не перевищує 4 Ом.

Щоб зменшити негативний вплив ПК на організм робітників, при організації робочих місць працівників було враховано вимоги стандарту [30].

Для запобігання погіршення самопочуття та зниження працездатності, а також зменшення ризиків отримання профзахворювань працівниками, було створено такі офісні умови у приміщенні, що відповідають стандарту [31].

Приміщення для офісу було обрано виходячи з того, що воно не повинно знаходитись поряд, над або під вибухонебезпечними приміщеннями категорії А та Б, згідно до стандарту [32]. На вибір приміщення також вплинули норми стандарту [33], згідно з якими приміщення не повинно межувати з приміщеннями у яких рівень шуму та вібрацій перевищує допустиму норму. Важливо зазначити, що обране приміщення для офісу не знаходиться в підвалі чи цокольному поверсі, оскільки це порушує правила, що описані в стандарті [34].

Всі вище перелічені заходи є технічними. Для запобігання можливих травмувань та нещасних випадків, з у сими новими працівниками, а також у період роботи проводяться інструктажі, атестації та навчання згідно до стандарту [35].

Для запобігання психофізіологічні небезпеки та профілактики профзахворювань, що можуть бути отримані внаслідок тривалої сидячої роботи та роботи з дисплеєм монітору проведено наступні заходи:

- працівникам дозволено витрачати 10 хвилин кожної робочої години на те щоб зробити гімнастику та дати очам час на відпочинок;
- встановлено перекладину та масажери для спини;
- створено зону відпочинку з диваном та м'якими кріслами;
- створено ігрову зону з рухливими іграми;
- регулярно проводяться заходи в середині компанії, що направлені на відпочинок та групову взаємодію між її учасниками.

7.3 Заходи з забезпечення виробничої санітарії та гігієни праці

Для забезпечення нормальних умов праці, робоче місце працівника було розміщено та укомплектовано згідно до стандарту [30], що передбачає гігієнічні

норми, що є необхідними при комплектації приміщень обладнаних ПК з ВДТ. Комплектація робочого місця відповідає сучасним ергономічним вимог [36]:

- робочий стіл має конструкцію, яка забезпечує оптимальне розміщення робочого обладнання на поверхні столу;
- висота робочої поверхні столу складає 725 мм;
- робочий стіл забезпечує простір для ніг шириною – не менше 500 мм, висотою не менше 600 мм, а також глибиною на рівні простягнутої ноги та на рівні – не менше 650 мм;
- підставка для ніг, що використовується на робочому місці, має глибину не менше 400 мм та ширину не менше 300 мм, а також регулюється по висоті в межах 150 мм та за кутом нахилу опорної поверхні підставки до 20 градусів;
- конструкція крісла забезпечує підтримку раціональної робочої пози під час роботи з ВДТ і ПЕОМ, а також дозволяє змінювати позу з метою зниження статичного напруження м'язів шийно-плечової області і спини для попередження розвитку втоми;
- елементи крісла є напівм'якими та використовують покриття, яке є неслизьким, не електризується, повітронепроникне, а також забезпечує легке очищення від забруднення.

ВДТ та інші комплектуючі ПК розміщені на робочій поверхні столу таким чином, що відстань від очей користувача до екрану дисплея має становити 500-700 мм. Кут зору становить від 10 до 20 град. Кут між верхнім краєм дисплея і рівнем очей користувача має становити не менше 10 град. Екран дисплею розташовано таким чином, що він є перпендикулярним до лінії зору користувача [36].

Дозволена модифікація робочого місця, з попередньою перевіркою ергономічних вимог, якщо фізичні особливості працівника вимагають іншої комплектації та конструкції.

Для забезпечення оптимального рівня освітлення робочого приміщення використовується стандарт [30], згідно з яким мінімальна освітленість

встановлюється в залежності від розряду виконуваних зорових робіт. Для четвертого розряду зорових робіт вона складає від 300 до 500лк [37].

Отже, для забезпечення оптимального рівня освітлення було перевірено освітленість робочого місця працівника за ПК на відповідність розряду виконуваних зорових робіт. Дані вимірювання показали, що природній рівень освітленості поверхні на якій розташований ПК робітника, складає 200лк за освітленості цієї ж поверхні під відкритим небом в 20000лк, з відси можна зробити висновок, що КПО становить 1%, відповідно отримана цифра не відповідає нормативному КПО [37].

Для штучного освітлення приміщення було встановлено люмінесцентні лампи.

Площа робочого приміщення становить 25м^2 . Ширина приміщення і довжина приміщення становлять 5м, висота – 3м. Саме для даного приміщення і було проведено розрахунок штучного освітлення [37].

Щоб визначити потрібну кількість світильників, для забезпечення рівень освітленості, що відповідає нормативам, було визначено світловий потік, що падає на робочу поверхню за формулою 7.1 [37].

$$F = \frac{E \times K \times S \times Z}{\eta}, \quad (7.1)$$

де F – світловий потік, що розраховується, лм;

E – нормована мінімальна освітленість, лк; $E = 300$ лк;

S – площа освітлюваного приміщення (у нашому випадку $S = 20\text{м}^2$);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1,1... 1,2, в даному випадку $Z = 1,1$);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в даному випадку $K = 1,5$);

η – коефіцієнт використання світлового потоку, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп, і обчислюється в долях одиниці та залежить від характеристик світильника, розмірів приміщення, забарвлення стін, стелі та підлоги, що характеризуються коефіцієнтами відбиття від стін ($\rho_{\text{ст}}$), стелі ($\rho_{\text{стелі}}$) та підлоги ($\rho_{\text{п}}$)), значення коефіцієнтів дорівнюють $\rho_{\text{ст}} = 50\%$, $\rho_{\text{стелі}} = 50\%$, $\rho_{\text{п}} = 10\%$.

Щоб знайти значення η , необхідно розрахувати індекс приміщення, який можна розрахувати за формулою 7.2 [37].

$$I = \frac{S}{h(A+B)}, \quad (7.2)$$

де S – площа приміщення, що становить 25м^2 ;

h – розрахункова висота підвісу, що становить $2,9\text{м}$;

A – ширина приміщення, що становить 5м ;

B – довжина приміщення, що становить 5м .

Підставивши значення до формули, отримуємо:

$$I = \frac{25}{2,9 \times (5 + 5)} = 0,86.$$

За допомогою індексу приміщення I , знаходимо коефіцієнт використання світлового потоку за значеннями, що знаходяться в таблиці рисунка 7.1. За таблицею видно, що $\eta = 0,24$ [38].



Коефіцієнт відбиття								Коефіцієнт приміщення. I
Стеля	70%				50%		30%	
Стіни	50%		30%		50%	30%	10%	
Підлога	30%	10%	30%	10%	10%			
Коефіцієнт використання світлового потоку	0,26	0,25	0,20	0,19	0,17	0,13	0,06	0,5
	0,3	0,28	0,24	0,23	0,2	0,16	0,08	0,6
	0,34	0,32	0,28	0,27	0,22	0,19	0,10	0,7
	0,38	0,36	0,31	0,30	0,24	0,21	0,11	0,8
	0,40	0,38	0,34	0,33	0,26	0,23	0,12	0,9
	0,43	0,41	0,37	0,35	0,28	0,25	0,13	1,0
	0,46	0,43	0,39	0,37	0,30	0,26	0,14	1,1
	0,48	0,46	0,42	0,40	0,32	0,28	0,15	1,25
	0,54	0,49	0,47	0,44	0,34	0,31	0,17	1,5
	0,57	0,52	0,51	0,47	0,36	0,33	0,18	1,75
	0,60	0,54	0,54	0,50	0,38	0,35	0,19	2,0
	0,62	0,56	0,57	0,52	0,39	0,37	0,20	2,25
	0,64	0,58	0,59	0,54	0,40	0,38	0,21	2,5
	0,68	0,60	0,63	0,57	0,42	0,40	0,22	3,0
	0,70	0,62	0,66	0,59	0,43	0,41	0,23	3,5
	0,72	0,64	0,64	0,61	0,45	0,42	0,24	4,0
	0,75	0,66	0,72	0,64	0,46	0,44	0,25	5,0

Рисунок 7.1 – Коефіцієнт використання світлового потоку світильника встановленого на стелі

Підставляємо отримані значення до формули 7.1 і отримуємо:

$$F = \frac{300 \times 1,5 \times 25 \times 1,1}{0,24} = 51\,562,5 \text{ лм.}$$

Для освітлення приміщення використовуються люмінесцентні лампи типу ЛБ 40-1, світловий потік яких ($F_{\text{л}}$) становить 4320 лм. Необхідну кількість ламп для світильників було розраховано за формулою 7.3 [37].

$$N = \frac{F}{F_{\text{л}}}, \quad (7.3)$$

де N – кількість ламп, що визначається;

F – сумарний світловий потік, $F = 51\,562,5$ лм;

$F_{\text{л}}$ – світловий потік лампи, $F_{\text{л}} = 4320$ дм.

Підставивши значення до формули знаходимо:

$$N = \frac{51\,562,5}{4320} \approx 12.$$

В робочому приміщенні використовуються світильники типу ЛПП. Кожний світильник укомплектований двома лампами ЛБ 40-1. Звідси і було прийнято рішення використовувати 6 світильників з 12 працюючими лампами для оптимального освітлення кімнати.

Щоб забезпечити оптимальний рівень параметрів повітряного середовища виробничого приміщення використовується стандарт [31], до вимог яких відповідно і було приведено параметри повітряного середовища виробничого приміщення (табл. 7.1) [28].

Таблиця 7.1 – Параметри повітряного виробничого середовища кімнати

Період року	Температура, °C	Відносна вологість, %	Швидкість руху, м/с
Холодний	22 – 24	40 – 60	0,1
Теплий	23 – 25	40 – 60	0,1 – 0,2

7.4 Заходи з забезпечення безпеки у надзвичайних ситуаціях

7.4.1 Заходи з пожежної безпеки

В робочому приміщенні розробки програмного застосунку джерелами займання вогню можуть стати електричні складові ПК, периферійні пристрої, офісна техніка, кухонна техніка, кондиціонери, обігрівачі та інша допоміжна техніка. Для розміщення техніки та комплектування робочих місць використовується офісні так кухонні меблі. Перенавантаження мережі живлення, коротке замикання, іскра чи людський чинник, можуть призвести до займання горючих матеріалів, таких як меблі, офісна канцелярія тощо.

Для запобігання пожежонебезпечних ситуацій було проведено планування та подальше впровадження заходів пожежної безпеки згідно з вимогами стандарту [39].

Оскільки робоче приміщення є обладнаним ПК з ВДТ та іншою офісною технікою ще перебуває під напругою, то згідно з до стандарту [39] є вірогідність займання пожеж класу – А (горіння твердих речовин), а також класу Е (горіння електроустановок, що перебувають під напругою).

При плануванні та впровадженні заходів пожежної безпеки було враховано, що робоче приміщення, згідно з 6 розділом та таблицею 6.1 стандарту [40] слід віднести до категорії Д (знижено пожежонебезпечна).

Для виявлення пожежної небезпеки у виробничому приміщенні було встановлено систему пожежного захисту ДПС-038, що реагує на зміну температури, виявляє димові ознаки, та добре працює у приміщенні з площею 30м^2 , а оскільки площа робочого приміщення складає 25м^2 , то даний датчик повністю відповідає потребам офісу.

Для пожежогасіння встановлено вуглекислотні вогнегасники типу ВВК-3,5 у кількості двох штук на відстані не більше 10 м у робочому приміщенні площа якого становить 25м^2 . Дане рішення прийнятого з того, що даний вогнегасник призначений для гасіння електроустановок, що знаходяться під напругою, містить заряд вогнегасної сили 3кг. і більше, на 20м^2 , що повністю задовольняє потреби поточної площі приміщення, а також стандарту [39].

Для запобігання пожеж а приміщенні обладнаному ПК з ВДТ та іншою офісною технікою, на силовому обладнанні встановлено захисні пристрої, що реагують на зміну напруги в системі, та вимикають електроживлення з цілю запобігання короткого замикання.

З організаційних заходів запобігання пожеж, з у сими новими працівниками, а також у період роботи проводяться інструктажі з пожежної безпеки. У приміщенні на видних місцях закріплені плани евакуації.

У випадку пожежної небезпеки передбачено виходи з двох сторін приміщення. Призначені для евакуації коридори та проходи є вільними, мають

невелику довжину за мінімальною кількістю поворотів, та є достатньо широкими. Оскільки робоче приміщення знаходиться на першому поверсі, то пороги або ступені на всьому шляху евакуації є відсутніми.

7.4.2 Заходи з цивільного захисту. Укриття населення у захисних спорудах цивільного захисту

У зв'язку з подіями на сході України питання укриття населення у захисних спорудах цивільного захисту є дуже важливими особливо у районах, що знаходяться поряд з зоною проведення бойових дій [41].

Укриття населення у захисних спорудах є основним та найефективнішим способом захисту населення від вражаючих факторів факторів, що виникають у наслідок надзвичайних ситуацій, воєнних дій, терористичних атак. Даний спосіб захисту населення полягає у застосуванні системи захисних споруд, що відповідають характеру загрози та вимогам захисту категорій населення [41]-[42].

В залежності від захисних властивостей характеру загроз, захисні споруди цивільного захисту поділяються на [41]:

- сховище, що являє собою герметичну споруду для захисту людей, в якій протягом певного часу створюються умови, що виключають вплив на них небезпечних факторів, які виникають внаслідок надзвичайної ситуації, воєнних (бойових) дій та терористичних актів;
- протирадіаційне укриття, що являє собою негерметичну споруду для захисту населення. В даному типі захисних споруд створюються умови, що виключають вплив іонізуючих променів на людей у разі радіоактивного забруднення навколишнього середовища;
- протирадіаційне укриття, що являє собою захисну споруду цивільного захисту, яка зводиться з використанням спеціальних конструкцій за короткий проміжок часу, призначенням якої є захист населення від дії засобів ураження в особливий період.

Під впливом деяких факторів небезпеки, причиною яких являються надзвичайні ситуації, що трапляються у мирний час, а також дії засобів ураження в особливий період, для захисту населення також можуть використовуватися споруди подвійного призначення, а також найпростіші укриття [41].

Споруда подвійного призначення – це наземна або підземна споруда, що може бути використана за основним функціональним призначенням і для захисту населення [41].

Найпростіше укриття – це фортифікаційна споруда, цокольне або підвальне приміщення, що знижує комбіноване ураження людей від небезпечних наслідків надзвичайних ситуацій, а також від дії засобів ураження в особливий період [41].

При повітряній небезпеці, мінометному вогню або артобстрілу у якості найпростіших укриттів також можна використовувати відкриті земляні щілини.

Якщо у якості найпростішого укриття використовується підвальне або напівпідвальне приміщення, то у разі ущільнення дверних прорізів можна досягти зменшення опромінювання у 50-100 разів, що дорівнює показнику типового протирадіаційного укриття [41].

Саме тому, з початком бойових дій на сході нашої держави, органами місцевої влади активно фінансується та проводиться створення та комплектація найпростіших укриттів, шляхом приведення підвальних приміщень житлових будинків у належний стан, а також забезпеченням запасами води, продуктів, медикаментів та приладами освітлення.

Також необхідно зазначити, що до захисних споруд застосовуються певні вимоги, що повинні бути дотримані [42]:

- захисні споруди повинні забезпечувати захист населення від усіх вражаючих факторів;
- будуватися на ділянках місцевості, які не затоплюються;
- мати входи та виходи з таким самим ступенем захисту, що і основне приміщення сховища;
- повинні бути присутні аварійні виходи на випадок обвалу;

- мати вільні підходи, де не повинно бути вибухонебезпечних матеріалів або матеріалів, що при горінні виділяють велику кількість диму;
- основні приміщення повинні мати висоту не менше ніж 2,2 метри та рівень підлоги, що є вищим менше ніж на 20 сантиметрів від рівня ґрунтових вод;
- фільтровентиляційне обладнання сховища повинно очищувати повітря від усіх шкідливих домішок і забезпечувати подачу чистого повітря в межах установлених норм.

Захисні споруди можуть використовуватися у мирний час для потреб народного господарства, але за умови, що експлуатація захисної споруди не призводить до порушення її захисних властивостей, а також переведення приміщення у режим укриття під час надзвичайної ситуації відбувається за мінімально короткий термін [42].

Захисні споруди цивільного захисту повинні забезпечувати створення необхідних санітарно-гігієнічних умов для людей. Основними показниками цих умов є вміст вуглекислого газу, температура та вологість повітря. Повітря має містити не більше 1% вуглекислого газу (гранично допустима концентрація дорівнює 3%), мати відносну вологість не більше ніж 70% (гранично допустима вологість дорівнює 80%) та температуру повітря не вище 23°C (гранично допустимою температурою + 31°C) [42].

ВИСНОВКИ

В результаті виконання дипломної роботи магістра було проведено аналіз концепцій життєвого циклу апаратного забезпечення та менеджменту задач, розглянуто методи версійності та збереження історії даних, а також розглянуто алгоритми порівняння текстових фрагментів. Виходячи з результатів проведеного аналізу було сформовано вимоги до розроблюваного програмного застосунку, удосконалено життєвий цикл апаратного забезпечення шляхом додавання таск-менеджменту, обрано стек технологій та алгоритм для порівняння текстових фрагментів.

В ході оптимізації обраного алгоритму Myers було переписано структуру даних збереження процесу виконання алгоритму, змінено логіку циклу проходження по скрипту редагування алгоритму, додано мемоізацію для кешування результатів порівняння невеликих строк, а також додано типізацію за допомогою мови програмування TypeScript. Було проведено benchmark-тестування оптимізованої та базової реалізації версій алгоритму, з результатів якого можна зробити висновок, що оптимізована версія алгоритму Myers є швидшою на 25.4% ніж його базова імплементація на JS, а також є стабільнішою, оскільки межі в яких швидкість алгоритму коливається є $\pm 1.12\%$, в той час як базова імплементація має межі коливання в $\pm 7.84\%$. Було також реалізовано C++ версію алгоритму, та за допомогою технології Web Assembly запущено в JavaScript середовищі. Тестування показало, що дана реалізація в деяких випадках є повільнішою у декілька разів за оптимізовану версію на TS. Причиною такої різниці стала нездатність поточної версії Web Assembly швидко транспортувати строкові дані.

Відповідно до вимог версійності було створено реляційну базу даних з використанням СКБД PostgreSQL.

Серверна частина застосунку реалізована на мові програмування TS та на платформі NodeJS з використанням фреймворку NestJS та шаблону проєктування MVC.

Клієнтську частину застосунку реалізовано на мові програмування TS та за допомогою бібліотеки React. В якості шаблону архітектури було обрано Flux-архітектуру.

Розроблений програмний застосунок дозволяє створювати проєкт апаратного забезпечення, формувати команду розробників проєкту, створювати компоненти, додавати розробників до компоненту та зберігати історію специфікації з можливістю порівняння минулої версії з поточною версією специфікації та наглядної демонстрації результати роботи алгоритму користувачеві.

Виконано опис призначення та роботи розробленого програмного забезпечення, а також надано приклад роботи з програмним застосунком.

Розроблений програмний застосунок повністю задовольняє вимоги, які було сформовано за результатами аналізу проблеми. Тестування програмного застосунку виявило вразливості застосунку та випадки у яких слід відображати повідомлення про виняткову ситуацію оператору. Розроблений UI/UX є інтуїтивно зрозумілим, отже додаткові підказки для роботи з програмним застосунком не потребуються.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. PLM – Управління життєвим циклом продукту [Електрон. ресурс]. – Режим доступу : <https://www.it.ua/knowledge-base/technology-innovation/product-lifecycle-management-plm/>.
2. Система управління проектами [Електрон. ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Система_управління_проектами/.
3. Життєвий цикл продукції [Електрон. ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Життєвий_цикл_продукції/.
4. Версионность и история данных [Электрон. ресурс]. – Режим доступу : <https://habr.com/ru/post/101544/>.
5. Kimball, R. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling / R. Kimball, M. Ross – NY : Wiley, 2013. – 419 p.
6. Myers diff algorithm vs Hunt-McIlroy algorithm [Electronic resource]. – Access mode : <https://stackoverflow.com/questions/42635889/myers-diff-algorithm-vs-hunt-mcilroy-algorithm/>.
7. Hunt-Szymanski algorithm [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Hunt-Szymanski_algorithm#k-candidates/.
8. Myers, W. An O(ND) Difference Algorithm and Its Variations / W. Myers – AZ : University of Arizona, 1986. – 15 p.
9. Встроенное кеширование [Электрон. ресурс]. – Режим доступу : https://ru.qaz.wiki/wiki/Inline_caching/.
10. How JavaScript works [Electronic resource]. – Access mode : <https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tips-on-how-to-write-optimized-code-ac089e62b12e/>.
11. JavaScript Engine Internals [Electronic resource]. – Access mode : https://www.youtube.com/watch?v=-lt6a9kbc_k/.
12. JSBench.Me [Electronic resource]. – Access mode : <https://jsbench.me/>.
13. Реляційна база даних [Електрон. ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Реляційна_база_даних/.

14. Документно-орієнтована систем керування БД [Електрон. ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Документно-орієнтована_система_керування_базами_даних/.
15. Tiwari, S. Professional NoSQL / S. Tiwari – NY : Wiley, 2011. – 384 p.
16. Matthias, K. Docker: Up & Running: Shipping ReLIable Containers in Production / K. Matthias – Sebastopol : O'Reilly Media, 2018. – 352 p.
17. Syed, B. Beginning Node.js / B. Syed – New Yourk : Apress, 2014. – 308 p.
18. NestJS [Electronic resource]. – Access mode : <https://nestjs.com/>.
19. Model-View-Controller [Electronic resource]. – Access mode : <https://en.wikipedia.org/wiki/Model-view-controller/>.
20. MVC Concept Diagram [Elcetronic resource]. – Access mode : https://www.researchgate.net/figure/Smalltalk80-MVC-pattern-View-and-Controller-work-as-a-pair-allowing-the-user-to-interact_fig2_269303611/.
21. React [Electronic resource]. – Access mode : <https://reactjs.org/>.
22. Бэнкс А. React и Redux: функциональная веб-разработка / А. Бэнкс, Е. Порселло – Санкт-Петербург : Питер, 2018. – 329 с.
23. White. A Major JavaScript Engines / A. White – Indianapolis : Wiley, 2009. – 975 p.
24. TypeScript [Electronic resource]. – Access mode : <https://en.wikipedia.org/wiki/TypeScript/>.
25. Flux-архитектура [Электрон. ресурс]. – Режим доступа : <https://ru.wikipedia.org/wiki/Flux-архитектура/>.
26. Flux-architecture [Electronic resource]. – Access mode : <https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/>.
27. Методичні вказівки по економічному обґрунтуванню дипломних проєктів для студентів спеціальностей 8.05010301 «Програмне забезпечення автоматизованих систем» та 8.05010201 «Інформаційні технології

проектування» для усіх форм навчання / Укл.: Остапенко В.В., Сердюк Є.М.– Запоріжжя: ЗНТУ, 2015.– 22 с.

28. Методичні вказівки до виконання розділу “Охорона праці та безпеки у надзвичайних ситуаціях” у дипломних роботах студентів ступеню “Магістрів” та освітньою програмою: “Інженерія програмного забезпечення спеціальності 121: Інженерія програмного забезпечення” / Укл.: Коробко О.В. – Запоріжжя: НУ “Запорізька політехніка”, 2020. – 28 с.

29. Правила улаштування електроустановок : ПУЕ-2017. – [Чинний від 2017-07-21]. – К.: Міненерговугілля України, 2017. – 617 с. – (Національний стандарт України).

30. Державні санітарні правила и норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин : ДСанПіН 3.3.2.007-98. – [Чинний від 1998-12-10]. – К.: Міністерство охорони здоров’я України, 1998. – 9 с. – (Національний стандарт України).

31. Державні санітарні норми параметрів мікроклімату : ДСН 3.3.6.042-99. – [Чинний від 1999-12-01]. – К.: Міністерство охорони здоров’я України, 1999. – 10 с. – (Національний стандарт України).

32. Норми визначення категорій приміщень, будинків та зовнішніх установок по вибухопожежної та пожежної безпеки : НАПБ Б.03.002-2007. – [Чинний від 2007-12-03]. – К.: МНС України, 2007. – 9 с. – (Національний стандарт України).

33. Санітарні норми виробничого шуму, ультразвуку та інфразвуку : ДСН 3.3.6.037-99. – [Чинний від 1999-12-01]. – К.: Міністерство охорони здоров’я України, 1999. – 34 с. – (Національний стандарт України).

34. Правила охорони праці під час експлуатації електронно-обчислювальних машин : НПАОП 0.00-1.28-10. – [Чинний від 2010-03-26]. – К.: Державний комітет України з промислової безпеки, охорони праці та гірничого нагляду, 2010. – 8 с. – (Національний стандарт України).

35. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці : НПАОП 0.00-4.12-05. – [Чинний від 2007-11-16].

– К.: Державний комітет України з промислової безпеки, охорони праці та гірничого нагляду, 2007. – 6 с. – (Національний стандарт України).

36. Ергономічні вимоги [Електрон. ресурс]. – Режим доступу : <https://www.gpp.in.ua/robota/ergonomichni-vimogi-dlya-organizatsiji-robochogomistsya.html/>.

37. Штучне освітлення [Електрон. ресурс]. – Режим доступу : https://cpo.stu.cn.ua/Oksana/rozrah_rozd_OP_DP_bak_spec_mag/80.html/.

38. Коэффициента использования светового потока [Электрон. ресурс]. – Режим доступа : <https://stroyday.ru/kalkulyatory/elektrohozyajstvo-kalkulyatory/kalkulyatory-rascheta-osveshennosti-pomeshheniya.html/>.

39. Правила пожежної безпеки України : НАПБА.01.001-2014. – [Чинний від 2014-12-30]. – К.: Міністерство внутрішніх справ України, 2014. – 86 с. – (Національний стандарт України).

40. Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою : ДСТУ Б В.1.1-36:2016. – [Чинний від 2017-01-01]. – К.: Міненергобуд України, 2017. – 86 с. – (Національний стандарт України).

41. Укриття населення у захисних спорудах цивільного захисту [Електрон. ресурс]. – Режим доступу : https://protocol.ua/ua/kodeks_tsivilnogo_zahistu_uk_vid_02_10_2012_5403_vistatt_ua_32/.

42. Укриття населення в захисних спорудах [Електрон. ресурс]. – Режим доступу : <http://www.toretsk-vca.gov.ua/downloads/gz/soorujeniya/16.pdf/>.

ДОДАТОК А
Технічне завдання

Вступ

Ціллю розроблюваного програмного застосунку являється відстеження версій апаратного забезпечення.

A.1 Підстави для розробки

Підставою для розробки є завдання на дипломну роботу на тему “Дослідження та програмна реалізація методів відстеження версій апаратного забезпечення”, затверджене наказом № 301 від 03 листопада 2020 р. у НУ “Запорізька політехніка”.

A.2 Призначення розробки

Функціональним призначенням програмного застосунку являється надання користувачу можливості створювати проєкт апаратного забезпечення, формувати команду, створювати компоненти апарату та залучати учасників команди до його розробки, а також вести специфікацію компонентів з можливістю перегляду минулих версій та їхнього порівняння з поточною версією за допомогою оптимізованого під поточно платформу алгоритму порівняння текстових фрагментів та наглядною демонстрацією результату його роботи.

A.3 Вимоги до програми або програмного продукту

A.3.1 Вимоги до функціональних характеристик

Програма повинна дозволяти виконувати наступні функції:

- створення, редагування та можливість зміни поточного статусу проєкту;
- додавання нових членів команди до проєкту;

- створення, редагування та можливість зміни поточного статусу компоненту проєкту;
- призначати розробників компоненту;
- переглядати будь-яку версію специфікації компоненту з можливістю порівняння з поточною за допомогою алгоритму порівняння текстових фрагментів Myers

Вхідними даними є:

- введені користувачем дані авторизації;
- введені користувачем дані для створення та редагування проєкту;
- введені користувачем дані для створення та редагування компоненту проєкту.

Вихідними даними програми є інформація, збережена до бази даних.

A.3.2 Вимоги до надійності

Програмний застосунок повинен коректно функціонувати та забезпечувати збереження введених користувачами даних.

Стійке функціонування системи має бути забезпечене обробленням наступних ситуацій:

- введені некоректні дані користувача під час авторизації;
- помилка при створенні проєкту чи компоненту;
- помилка додаванні користувача до проєкту чи компоненту;
- помилка отриманні даних про проєкт чи компонент;
- помилка при порівнянні версій специфікації компоненту;
- внутрішня помилка серверної частини.

A.3.3 Умови експлуатації

Клієнтська частина являє собою WEB-застосунок, а отже середовищем виконання є браузер і всі умови використання повністю залежать від розробника

браузеру. Питанням обслуговування займається служба підтримки браузеру. Серверна частина та база даних застосунку розгорнуті на віддаленому сервері, підтримкою працездатності якого займається адміністратор.

Сирцевий код програмного застосунку може бути записаний на компакт-диск, жорсткий диск, тверди й накопичувач, Flash-накопичувач чи віддалений репозиторій.

A.3.4 Вимоги до складу та параметрів технічних засобів

Для коректної роботи необхідна наявність наступних технічних засобів:

- операційна система Windows 7 та вище, macOS X Yosemite 10.10 та вище, Ubuntu 14.04 та вище;
- процесор Pentium 4 з тактовою частотою 1,3 ГГц та вище;
- обсяг оперативної пам'яті 1,5 ГБ та більше;
- 2ГБ та більше вільного місця на жорсткому диску;
- доступ до інтернету для зв'язку з сервером;
- підключення до локальної мережі у випадку використання локального серверу.

A.3.6 Вимоги до маркування та упакування

Програмний застосунок повинен поширюватися на носії чи репозиторій з назвою, описом призначення програмного застосунку, вимог та функціоналу, що надається користувачеві.

A.3.7 Вимоги до транспортування та збереження

Вимоги до транспортування і зберігання відповідають вимогам віддалених репозиторіїв та зовнішніх накопичувачів.

A.4 Стадії та етапи розробки

Розробка повинна бути проведена в таких стадіях:

- написання технічного завдання;
- проєктування програмного застосунку;
- написання програмного коду;
- тестування розробленого застосунку;
- написання документації;
- впровадження.

На стадії написання технічного завдання потрібно виконати наступні завдання:

- постановка завдання;
- визначення технічних вимог застосунку;
- визначення вимог до розроблюваного ПЗ;
- визначення термінів необхідних для розробки застосунку на складання документації до нього;
- вибір стеку технологій;
- затвердження технічного завдання.

На стадії проєктування виконується створення архітектури ПЗ.

На стадії написання програмного коду необхідно виконати роботу з програмування ПЗ та його розміщенням.

На стадії тестування виконується пошук вразливостей застосунку і тестування функціоналу відповідно до опису та вимог, що було поставлено. У разі виявлення виняткової ситуації складається звіт, що передається розробнику. Після виправлення помилки відбувається контрольне тестування функціоналу. Після виправлення всіх помилок, що було виявлено в ході тестування, розпочинається стадія складання програмної документації.

На стадії написання документації до ПЗ виконується опис розробленого застосунку у відповідних документах. Після опису документації розпочинається

процес впровадження програмного застосунку у виробництво та передача клієнту.

A.5 Порядок контролю та приймання

Дипломна робота магістра допускається до захисту після підписання економічної частини та частини з охорони праці, перевірки керівником та нормоконтролером, затверджується завідувачем кафедри і захищається в присутності членів екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б.1 Клієнтський застосунок

Б.1.1 Текст файлу diff.ts

```

export enum EActions {
    EQUAL = "equal",
    INSERT = "insert",
    DELETE = "delete",
}

export const diff = (a: string, b: string): EActions[] => {
    const aLength = a.length;
    const bLength = b.length;

    // Temporary action type statics.
    let _insert: EActions;
    let _delete: EActions;

    // Swapped the arrays to use the shorter one as the first one.
    if ( bLength < aLength ) {
        const tmp = a;

        a = b;
        b = tmp;

        // We swap the action types as well.
        _insert = EActions.DELETE;
        _delete = EActions.INSERT;
    } else {
        _insert = EActions.INSERT;
        _delete = EActions.DELETE;
    }

    const m = a.length;
    const n = b.length;

    const delta = n - m;
    const offset = m + 1;
    const size = m + n + 3;

    // Edit scripts, for each diagonal.
    const es = new Array(size);
    // Furthest points, the furthest y we can get on each diagonal.
    const fp = new Array(size);

    function snake(k: number) {
        // We use -1 as an alternative below to handle initial values ( instead
of filling the fp with -1 first ).
        // Furthest points (y) on the diagonal below k.
        const y1 = (fp[k - 1 + offset] !== undefined ? fp[k - 1 + offset] : -
1 ) + 1;
        // Furthest points (y) on the diagonal above k.
        const y2 = fp[k + 1 + offset] !== undefined ? fp[k + 1 + offset] : -
1;
        // The way we should go to get further.
        const dir = y1 > y2 ? -1 : 1;

        // Clone previous changes array (if any).
        if (es[k + dir + offset]) {
            es[k + offset] = es[k + dir + offset].slice(0);
        }

        // Create changes array.
    }

```

```

    if (!es[k + offset]) {
      es[k + offset] = [];
    }

    // Push the action.
    es[k + offset].push(y1 > y2 ? _insert : _delete);

    // Set the beginning coordinates.
    let y = Math.max(y1, y2);
    let x = y - k;

    // Traverse the diagonal as long as the values match.
    while (x < m && y < n && a[x] == b[y]) {
      x++;
      y++;
      // Push no change action.
      es[k + offset].push(EActions.EQUAL);
    }

    return y;
  }

  let p = -1;

  // Traverse the graph until we reach the end of the longer string.
  do {
    ++p
    // Updates furthest points and edit scripts for diagonals below delta.
    for (let k = -p; k <= delta - 1; ++k) {
      fp[k + offset] = snake( k );
    }

    // Updates furthest points and edit scripts for diagonals above delta.
    for (let k = delta + p; k >= delta + 1; --k ) {
      fp[k + offset] = snake(k);
    }

    // Updates furthest point and edit script for the delta diagonal.
    // note that the delta diagonal is the one which goes through the sink
    (m, n).
      fp[delta + offset] = snake(delta);
    } while ( fp[delta + offset] !== n );

    // Return the final list of edit changes.
    // We remove the first item that represents the action for the injected
    nulls.
    return es[delta + offset].slice(1);
  };

```

Б.1.2 Текст файла App.ts

```

import React, { Component, ReactNode } from "react";
import { Provider } from "react-redux";

import { Router } from "../router";
import { configureStore } from "../data/store";

const store = configureStore();

export class App extends Component {
  render(): ReactNode {
    return (

```

```

        <Provider store={store}>Router /></Provider>
    );
}
}

```

Б.1.3 Текст файла store.ts

```

import createSagaMiddleware from "redux-saga";
import { applyMiddleware, createStore, Store } from "redux";
import { composeWithDevTools } from "redux-devtools-
extension/developmentOnly";

import { rootSaga } from "../root.saga";
import { rootReducer } from "../root.reducer";

const sagaMiddleware = createSagaMiddleware();

export const configureStore = (): Store => {
    const middleware = [sagaMiddleware];
    const enhancers = [applyMiddleware(...middleware)];

    const store = createStore(rootReducer,
    composeWithDevTools(...enhancers));
    sagaMiddleware.run(rootSaga);

    return store;
};

```

Б.1.4 Текст файла ApiService.ts

```

// @ts-ignore
import { cookies } from "brownies";
import axios, { AxiosInstance, AxiosRequestConfig } from "axios";

import { BASE_URL } from "../../resources/constants/urls";

export class ApiService {
    axiosInstance: AxiosInstance;

    constructor(config: AxiosRequestConfig = {}) {
        this.axiosInstance = axios.create({
            baseURL: BASE_URL,
            headers: { Accept: "application/json" },

            ...config,
        });

        this.axiosInstance.interceptors.request.use((config:
        AxiosRequestConfig) => {
            const { accessToken } = cookies;

```

```

    if (accessToken) {
        config.headers.Authorization = `Bearer ${accessToken}`;
    }

    return config;
});

this.axiosInstance.interceptors.response.use(
    (response) => {
        if (response.data) {
            return response.data;
        }

        return response;
    },
    (error) => {
        const { message } = error?.response?.data;

        if (message) {
            throw new Error(message);
        }

        throw error;
    },
);
}
}

```

Б.1.5 Текст файла Diff.h

```

#include <string>
#include <vector>

typedef unsigned int uint;

class Diff {
public:
    static uint snake (int k, int p, int pp, int m, int n, std::string
a,    std::string    b,    std::string    _insert,    std::string    _delete,
std::vector<std::vector<std::string>> &es, uint offset);
    static std::string compare(std::string a, std::string b);
};

```

Б.1.5 Текст файла Diff.cpp

```

#include <string>
#include <vector>
#include <sstream>

#include "Diff2.h"

typedef unsigned int uint;

#define onp_swap(type, val, a, b) type val=a;a=b;b=val;
#define onp_max(a, b) ((a) > (b) ? (a) : (b))
#define onp_seq_len(seq) strlen(seq)

uint Diff2::snake (int k, int p, int pp, int m, int n, char* a, char *b,
std::string _insert, std::string _delete, std::vector<std::vector<std::string>>
&es, uint offset) {
    int y = onp_max(p, pp);
    int x = y - k;

    int dir = p > pp ? -1 : 1;

    if (!es[k + dir + offset].empty()) {
        es[k + offset] = es[k + dir + offset];
    }

    es[k + offset].push_back(p > pp ? _insert : _delete);

    while (x < m && y < n && a[x] == b[y]) {
        ++x;
        ++y;

        es[k + offset].emplace_back("equal");
    }

    return y;
}

std::string Diff2::compare(char *a, char *b) {
    int a_length = onp_seq_len(a);
    int b_length = onp_seq_len(b);

    std::string _insert, _delete;

```

```

if (b_length < a_length) {
    std::swap(a, b);

    _insert = "delete";
    _delete = "insert";
} else {
    _insert = "insert";
    _delete = "delete";
}

const int m = onp_seq_len(a);
const int n = onp_seq_len(b);
const int delta = n - m;

uint offset = m + 1;
uint size = m + n + 3;

std::vector<std::vector<std::string>> es(size);
int *fp = (int *)malloc(sizeof(int) * size);

int p = -1;

for (int i = 0; i < size; i++) {
    fp[i] = -1;
}

do {
    ++p;
    for (int k = -p; k <= delta - 1; ++k) {
        fp[k + offset] = snake(k, fp[k - 1 + offset] + 1, fp[k + 1 +
offset], m, n, a, b, _insert, _delete, es, offset);
    }
    for (int k = delta + p; k >= delta + 1; --k) {
        fp[k + offset] = snake(k, fp[k - 1 + offset] + 1, fp[k + 1 +
offset], m, n, a, b, _insert, _delete, es, offset);
    }
    fp[delta + offset] = snake(delta, fp[delta - 1 + offset]+1, fp[delta
+ 1 + offset], m, n, a, b, _insert, _delete, es, offset);
} while (fp[delta + offset] != n);

std::vector<std::string> resultVector = es[delta + offset];

if (resultVector.empty()) {

```

```

        return "";
    }

    resultVector.erase(resultVector.begin());

    std::ostringstream vts;

    std::copy(resultVector.begin(),      resultVector.end()      -1,
std::ostream_iterator<std::string>(vts, ", "));
    vts << resultVector.back();

    return vts.str();
}

```

Б.2 Серверний застосунок

Б.2.1 Текст файлу application.ts

```

import { NestFactory, Reflector } from "@nestjs/core";
import { SwaggerModule, DocumentBuilder } from "@nestjs/swagger";
import { ClassSerializerInterceptor, ValidationPipe } from
"@nestjs/common";

import { ApplicationModule } from "../application.module";

async function bootstrap() {
    const app = await NestFactory.create(ApplicationModule);

    const options = new DocumentBuilder()
        .setTitle("Speka API")
        .setDescription("The Speka API documentation")
        .setVersion("1.0.0")
        .addTag("Speka")
        .addBearerAuth({ type: "http", scheme: "bearer", bearerFormat:
"Bearer" }, "Access token")
        .build();

    const document = SwaggerModule.createDocument(app, options);
    SwaggerModule.setup("api", app, document);

    app.enableCors();
    app.useGlobalPipes(new ValidationPipe());
    app.useGlobalInterceptors(new
ClassSerializerInterceptor(app.get(Reflector)));
}

```

```
    await app.listen(process.env.PORT);
  }
}
```

```
bootstrap();
```

Б.2.2 Текст файла `typeorm.config.ts`

```
import dotenv from "dotenv";
import { ConnectionOptions } from "typeorm";

dotenv.config();

const ormconfig: ConnectionOptions & { seeds: string[], factories: string[]
} = {
  type: "postgres",
  host: process.env.DB_HOST,
  port: Number(process.env.DB_PORT),
  username: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  entities: ["dist/application/database/models/**/*.entity{.ts,.js}"],
  migrations: ["dist/application/database/migrations/**/*.js"],
  subscribers: ["dist/application/database/subscribers/**/*.js"],
  seeds: ["dist/application/database/seeds/**/*.seed.js"],
  factories: ["dist/application/database/factories/**/*.factory.js"],
  cli: {
    entitiesDir: "src/application/database/entities",
    migrationsDir: "src/application/database/migrations",
    subscribersDir: "src/application/database/subscribers",
  },
  synchronize: false,
  logging: false,
};

export = ormconfig;
```

Б.2.3 Текст файла `component-details.entity.ts`

```
import { ApiProperty } from "@nestjs/swagger";
import { Entity, Column, PrimaryGeneratedColumn, CreateDateColumn,
ManyToOne } from "typeorm";

import { Component } from "../component.entity";
```

```

@Entity("componentDetails")
export class ComponentDetails {
  @PrimaryGeneratedColumn("uuid")
  @ApiProperty({example: "333a1c84-d7ab-11ea-87d0-0242ac130003",
description: "Component details uuid"})
  id: string;

  @Column({ default: "" })
  @ApiProperty({ example: "Some features", description: "Features of the
component" })
  features: string;

  @Column({ default: "" })
  @ApiProperty({ example: "Some future features", description: "Future
features of the component" })
  futureFeatures: string;

  @Column({ default: "" })
  @ApiProperty({ example: "Some notes", description: "Notes of the
component" })
  notes: string;

  @Column({ default: "" })
  @ApiProperty({ example: "Some requirements", description: "Requirements
of the component" })
  requirements: string;

  @Column()
  @ApiProperty({ example: 1, description: "The version of component
details" })
  version: number;

  @CreateDateColumn({ type: "timestamp with time zone", default: () =>
"CURRENT_TIMESTAMP" })
  @ApiProperty({ example: 1596696377370, description: "The creation time
of the component details" })
  createdAt: number;

  @ManyToOne(type => Component, component => component.details)
  @ApiProperty({ description: "Component" })
  component: () => Component;
}

```

ДОДАТОК В
Слайди презентації

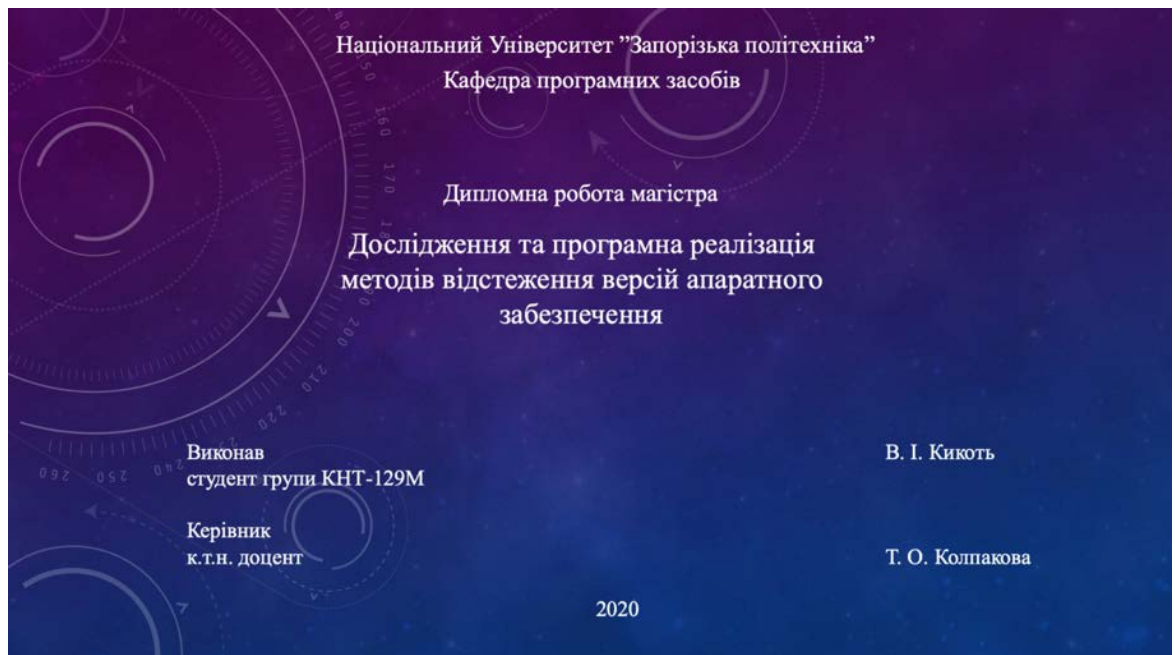


Рисунок В.1 – Слайд №1

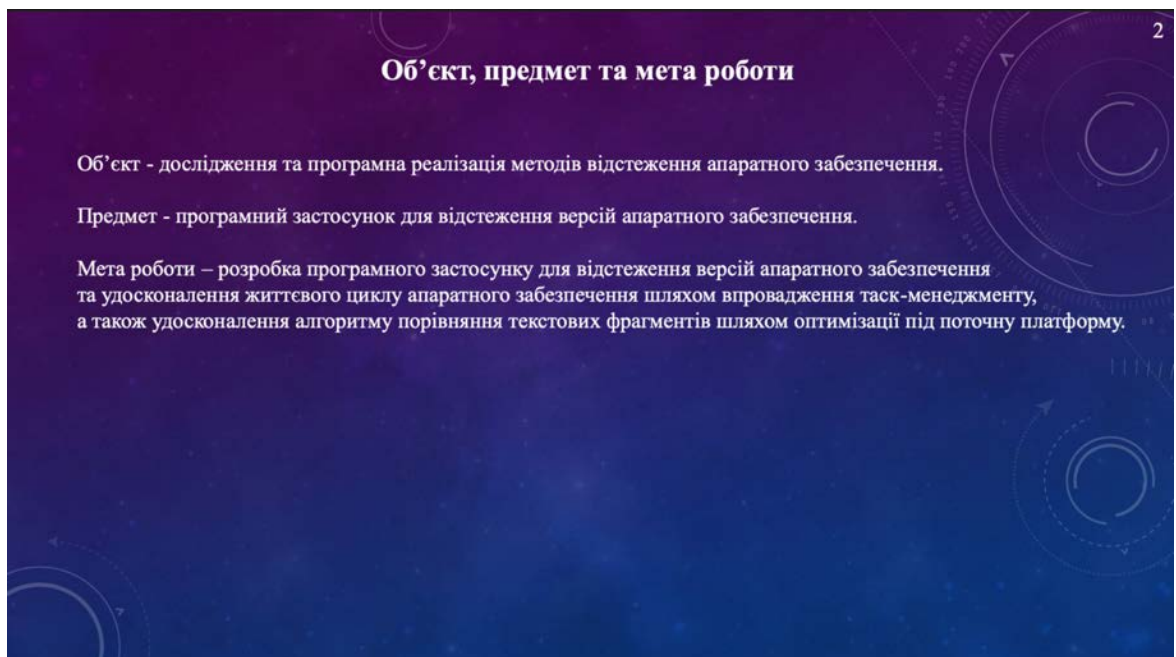


Рисунок В.2 – Слайд №2

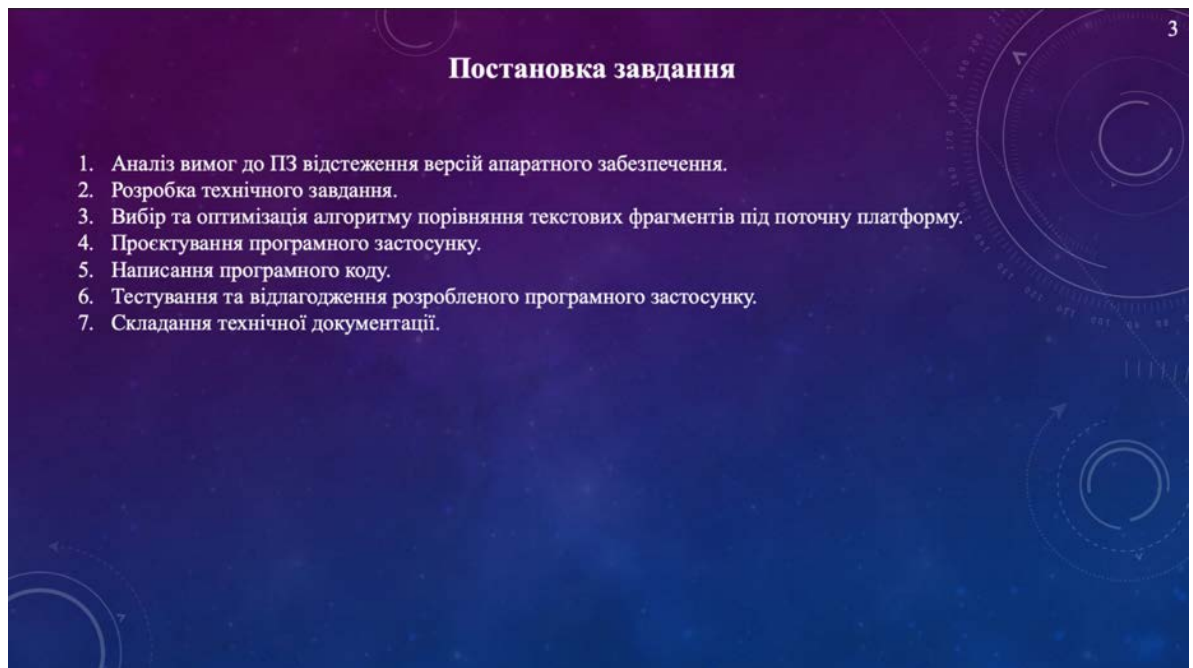


Рисунок В.3 – Слайд №3

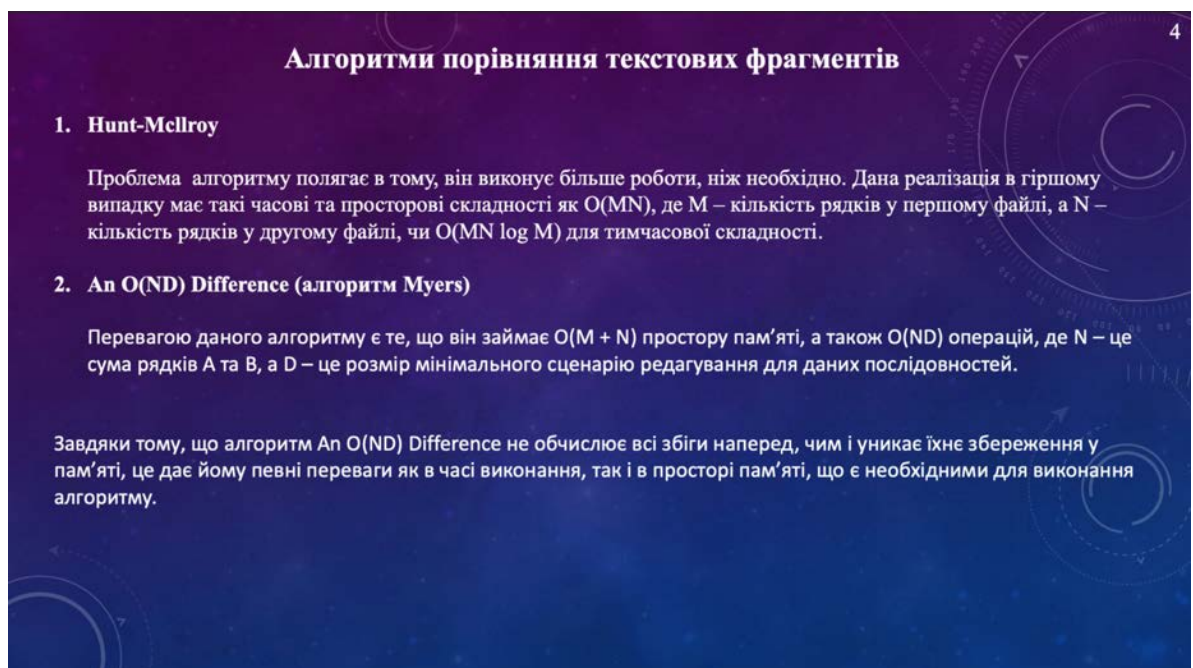


Рисунок В.4 – Слайд №4

Модифікація алгоритму Myers

5

Етапи модифікації алгоритму Myers:

1. Зміна структури даних збереження скрипта редагування та найдальших точок діагоналей з об'єкта типу масив на звичайний масив, для оптимізації процесів движка JavaScript та JIT-компіляції (рисунок 1).
2. Визначення розміру масиву скрипта редагування та найдальших точок діагоналей, для попереднього розподілу пам'яті для елементів та оптимізації створення і додавання нових елементів.
3. Модифікація циклу проходження по строкам, щоб запобігти від'ємним числам в якості індексів.
4. Додавання TypeScript для типізації коду алгоритму.
5. Мемоізації результату виконання алгоритму для невеликих строк.
6. Benchmark-тестування оптимізованої версії та базової версії алгоритму (рисунок 2).
7. Імплементация оптимізованої версії алгоритму на C++ та запуск розробленого коду у JavaScript середовищі з використанням WebAssembly.

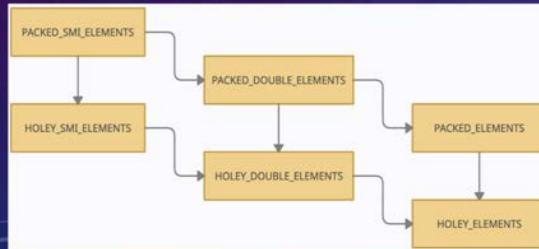


Рисунок 1 – Типізація масиву движком JavaScript

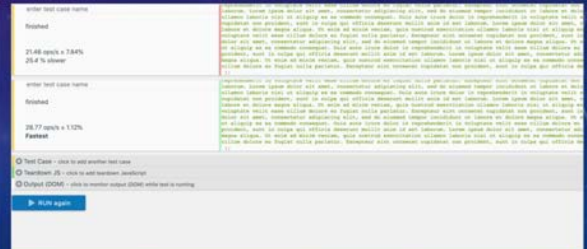


Рисунок 2 – Benchmark-тестування оптимізованої та базової версії алгоритму

Рисунок В.5 – Слайд №5

Проектування серверної частини

6

Вибір NodeJS фреймворку		
	NestJS	Express
Сумісність з TypeScript	Дуже висока	Середня
Строга структура	Присутня	Відсутня
Архітектура	MVC	MVC
CI автоматизація	Присутня	Відсутня
Підтримка розробником	Дуже висока	Висока
Підтримка спільнотою	Дуже висока	Висока
Використання декораторів	Дуже висока	Низька
Наявність заточених під фреймворк бібліотек	Дуже висока	Середня
Обробка виняткових ситуацій	Дуже висока	Середня
Документованість	Дуже висока	Висока



Рисунок 3 – MVC архітектура

В якості архітектури використано шаблон Model-View-Controller (рис. 3).

Рисунок В.6 – Слайд №6

Проектування клієнтської частини

Вибір інструменту для розробки front-end частини		
	React	Angular
Сумісність з TypeScript	Висока	Дуже висока
Строга структура	Відсутня	Присутня
Архітектура	Flux	MVC
СІ автоматизація	Присутня	Присутня
Підтримка розробником	Дуже висока	Висока
Підтримка спільнотою	Дуже висока	Середня
Тип реалізації	Бібліотека	Фреймворк
Наявність сумісних бібліотек	Дуже висока	Середня
Сторони інструменти розробки	Присутні	Відсутні
Надійність	Висока	Середня
Документованість	Дуже висока	Висока



Рисунок 4 – Flux-архітектура

В якості архітектури використано шаблон Flux (рис. 4).

Рисунок В.7 – Слайд №7

Схема бази даних

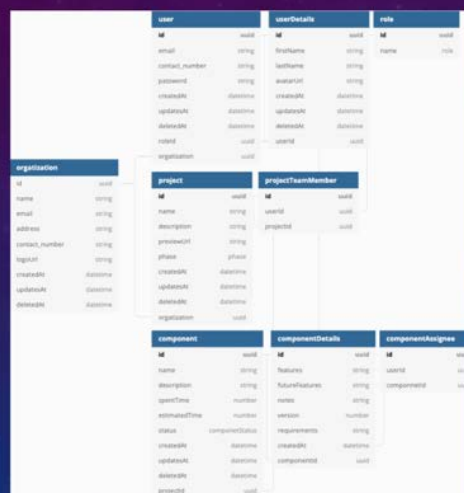


Рисунок 5 – Схема бази даних

Рисунок В.8 – Слайд №8

Схема функціонування програми

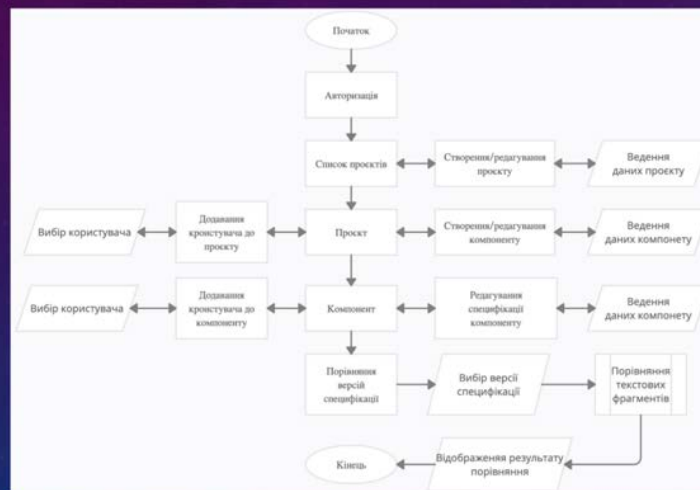


Рисунок 6 – Схема функціонування програми

Рисунок В.9 – Слайд №9

Інтерфейс застосунку

Вхід до системи

Е-мэйл:

Пароль:

увійти

Рисунок 7 – Сторінка авторизації

Протон SPEKA Дієн Дю

Назва проєкту	Опис	Фаза	Дата створення	Дата отримання
Супутник Протон-2	Важкий штучний супутник для...	Проектирування	24/11/2020, 18:05:34	24/11/2020, 18:05:34
Супутник Протон-1	Важкий штучний науковий супутник для...	Розробка	24/10/2020, 16:13:36	14/11/2020, 15:43:49

Рисунок 8 – Сторінка списку проєктів

Важкий штучний супутник для... Проектирування 24/11/2020, 18:05:34

Важкий штучний науковий супутник для... Розробка 24/10/2020, 16:13:36

Назва проєкту
Гарпун-М

Опис
Ракетноосій класу Б, що призначений для швидкого виводу вантажу на орбіту

Створення проєкту

Рисунок 9 – Модальне вікно створення проєкту

Протон SPEKA Дієн Дю

Назва проєкту	Опис	Дата створення	Дата отримання
Супутник Протон-1	Важкий штучний науковий супутник для...	24/10/2020, 16:13:36	24/11/2020, 18:25:12
Фаза	Розробка		

Назва компоненту	Опис	Статус	Витрачено часу	Закладено часу	Дата створення	Дата отримання
Спеціальна панель	30-ти панель для підключення...	Висновок	2308	2342	21/10/2020, 18:48:45	21/11/2020, 12:04:57
Антенна	Антенна для радіо зв'язку...	Тестування	915	1000	15/11/2020, 19:25:41	15/11/2020, 20:13:48

Рисунок 10 – Сторінка проєкту

Рисунок В.10 – Слайд №10

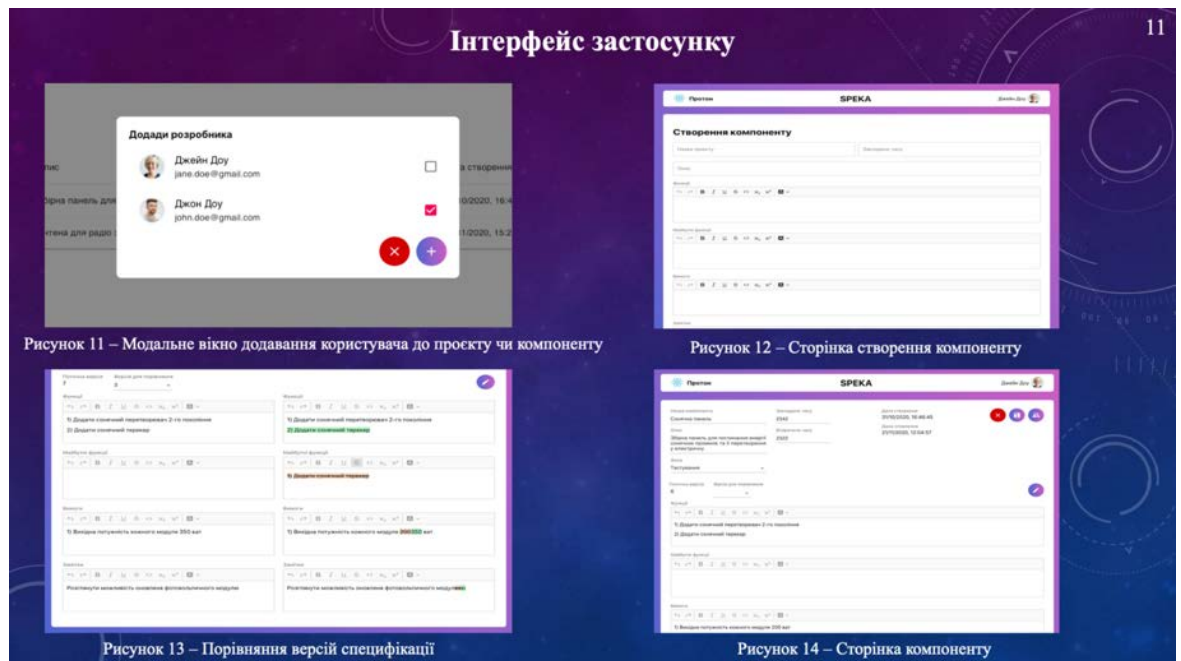


Рисунок В.11 – Слайд №11

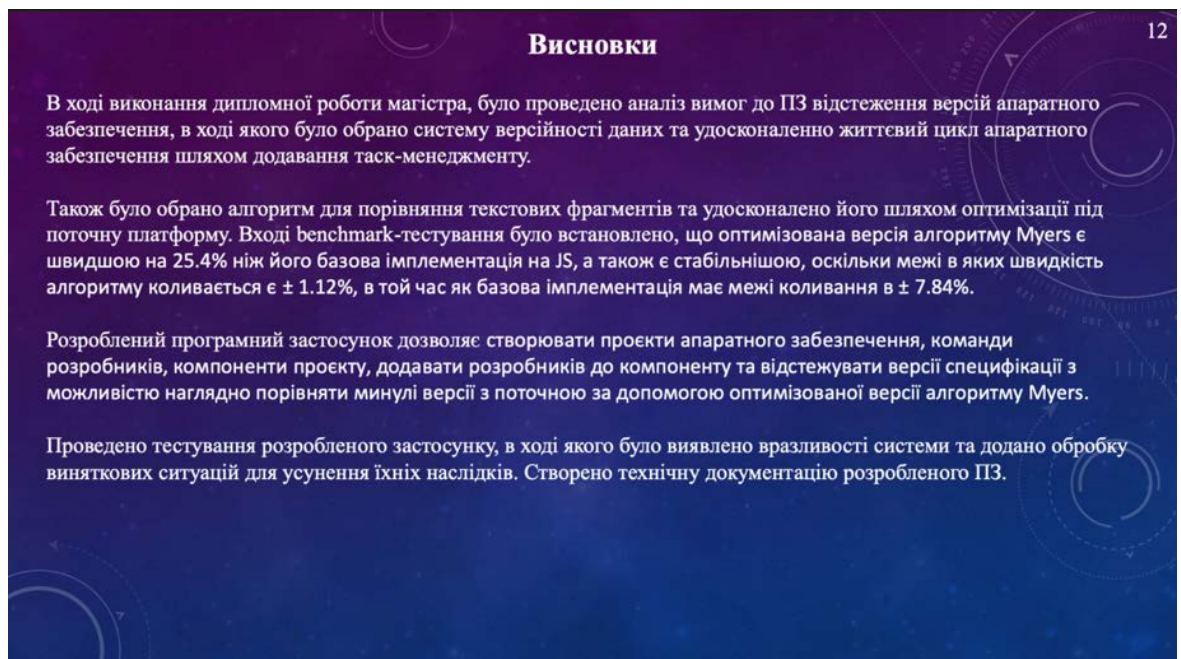


Рисунок В.12 – Слайд №12