Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Програмне забезпечення системи управління локальними та мережевими ресурсами

Виконав: студент IV курсу
групи ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Білоус Сергій Анатолійович
(прізвище та ініціали)

Керівник: к.т.н. доцент кафедри ПЗ Коваленко О.О.
(прізвище та ініціали)

Рецензент: к.т.н., доцент кафедри ОТ Крупельницький Л.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О.Н.
«10» 06 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
к.т.н., професор
Романюк О. Н._____
«12» березня 2024 року

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

Білоусу Сергію Анатолійовичу

1. Тема роботи – Програмне забезпечення системи управління локальними та мережевими ресурсами.

Керівник роботи: Коваленко Олена Олексіївна, к.т.н. доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «11» березня 2024 року № 80.

2. Строк подання студентом роботи 3 червня 2024 року.

3. Вихідні дані до роботи: вхідні дані: метод взаємодії – інтерактивний, елемент рендерингу – рендеринг додатку у веб-переглядачі, мова програмування – Javascript, середовище розробки – VSCode; вихідні дані: Прогресивний веб-додаток (PWA) з можливість нативної роботи на девайсах в онлайн та офлайн режимі, функціями використання мережевих та локальних ресурсів.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка методів й алгоритмів управління файлами на прикладі розробки прогресивного вебзастосунку; розробка програмних модулів додатку; тестування додатку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; демонстрація роботи аналогів, задачі дослідження, наукова новизна, практичне значення, Діаграма діяльності інтерактивних модулів, схема роботи модулю налаштування робочоно місця, опис використаних класів, зразок кешування мережевих файлів, зразок кешування локальних файлів, використання в офлайн режимі, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваленко О.О., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану управління локальними та мережевими ресурсами на прикладі вебзастосунків	14.03.2024 - 31.03.2024	Виконано
2	Розробка моделей, алгоритмів управління на прикладі вебзастосунку релаксації	01.04.2024 - 15.04.2024	Виконано
3	Вибір середовища та мови розробки	05.04.2023 - 10.04.2023	Виконано
4	Розробка прогресивного веб-застосунку для роботи онлайн	16.04.2024- 18.05.2024	Виконано
5	Розробка алгоритмів для кешування мережевих і локальних файлами та роботи в офлайн режимі	19.05.2024 - 24.05.2024	Виконано
6	Тестування програми	25.05.2024 - 28.05.2024	Виконано
7	Оформлення матеріалів до захисту БДР	29.05.2024 - 30.05.2024	Виконано

Студент (підпис) **Білоус С. А.**
(прізвище та ініціали)

Керівник бакалаврської дипломної роботи (підпис) **Коваленко О. О.**
(прізвище та ініціали)

АНОТАЦІЯ

Білоус С. А. Програмне забезпечення системи управління локальними та мережевими ресурсами. бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 56 с. На укр. мові. Бібліогр. : 29 назв; рис. : 12; табл. : 1.

У бакалаврській дипломній роботі проведено детальний аналіз методів і засобів управління локальними та мережевими ресурсами на прикладі розробки прогресивного веб-застосунку із функціоналом забезпечення мережевими та локальними ресурсами та можливістю крос платформного використання в режимі онлайн та офлайн.

Запропоновано методи удосконалення розробки прогресивного веб-застосунку шляхом використання бібліотеки TypeScript, яка є надбудовою над мовою програмування JavaScript, яка додає статичну типізацію. Це дозволяє виявляти широкий спектр програмних помилок, які в JavaScript зустрічаються лише під час виконання. Використання CacheStorage дозволить поєднувати мережеві та локальні файли для персонального налаштування робочого простору і використання функціоналу застосунку в офлайн режимі. Розроблено моделі, алгоритми та програмні модулі кешування даних, поєднання мережевих та локальних файлів для налаштування інтерфейсів застосунку, та інтерфейси для реалізації функціоналу концентрації та релаксації.

Отримані в бакалаврській дипломній роботі результати можна використати для подальшого розвитку технологій для концентрації та боротьби із стресом, з використанням мережевих та локальних ресурсів, в одному застосунку із знайомим та зручним інтерфейсом та в умовах відсутності інтернет з'єднання.

Ключові слова: прогресивний веб-застосунок, онлайн та офлайн, мережеві файли, локальні файли, концентрація, анти стрес, робочий прості, крос платформний.

ABSTRACT

Software for managing local and network resources.. Part 1. Frontend: Bachelor's thesis in the specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2024. 56 p.

In Ukrainian. Bibliography: 20 references; figures: 12; tables: 1.

The bachelor's thesis includes a detailed analysis of methods and tools for developing a progressive web application with functionality to manage network and local resources, as well as cross-platform capabilities for online and offline use.

The proposed methods for improving the development of a progressive web application involve using the TypeScript library, which is a superset of the JavaScript programming language that adds static typing. This allows for the identification of a wide range of programming errors that in JavaScript are only encountered at runtime. The JavaScript library React is used for creating the user interface, addressing the issue of partial content updates in single-page applications. Using CacheStorage enables the combination of network and local files for personalized workspace configuration and utilizing the application's functionality in offline mode. Models, algorithms, and software modules for data caching, combining network and local files to configure application interfaces, and interfaces for implementing concentration and relaxation functionalities have been developed.

The results obtained in the bachelor's thesis can be used for further development of technologies for concentration and stress management, using network and local resources, in a single application with a familiar and user-friendly interface, and in conditions where there is no internet connection.

Keywords: Progressive web application, online and offline functionality, network files, local files, concentration, stress reduction, workspace, cross-platform.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі, де присутні різноманітні фактори відволікання, розсіяності уваги, високого рівня стресу в суспільстві та частими проблемами із інтернет з'єднанням в тому числі пов'язаним із вимкненням світла, актуальність набувають застосунки які надають функціонал концентрації на завданнях, менеджменту рівня стресу із можливістю персонального налаштування необхідного функціоналу та робочого простору і також в режимі офлайн. У цьому контексті, прогресивні веб-застосунки які надають можливість крос платформного використання як в веб-переглядачах так і нативно на мобільних пристроях, набувають актуальності.

Використання технологій, зокрема вебзастосунків, для керування стресом та підвищення концентрації набуває все більшої важливості у нашому сучасному світі. Це пов'язано з тим, що люди все більше віддають перевагу цифровим рішенням для вирішення різних аспектів свого життя, включаючи управління стресом та концентрації на завданнях.

Додатково, обрана тема дослідження враховує необхідність персоналізації робочого простору для кожного користувача. Можливість використання мережевих та локальних файлів дозволяє індивідуалізувати робоче середовище, створюючи комфортні умови для кожного користувача з урахуванням його потреб та уподобань. Це важливо для ефективного управління стресом та концентрації.

Таким чином, обрана тема дослідження поєднує дві актуальні проблеми сучасного життя - стрес та користування веб-технологіями - і пропонує інноваційний підхід до їх розв'язання. Розробка прогресивного веб-застосунку, який дозволить користувачам працювати з файлами як в онлайн, так і в офлайн режимах, та включає в себе інструменти для боротьби із стресом та підвищення концентрації, має потенціал стати важливим інструментом для полегшення щоденного життя людей і підвищення їх якості життя.

Актуальність. Актуальність обраної теми дослідження виразно

відображається у сучасному контексті життя, де стрес стає все більш поширеним явищем. У такому контексті виникає потреба у таких вебзастосунках, оскільки вони можуть забезпечити доступ до необхідних інструментів та ресурсів, де б користувач не знаходився.

Враховуючи ці фактори, дане дослідження пропонує інноваційний підхід, поєднуючи можливості роботи з мережевими та локальними файлами онлайн і офлайн з ефективними інструментами для концентрації та зняття стресу. Це особливо актуально в контексті сучасного швидкого темпу життя, коли люди шукають способи зберегти баланс та ефективно керувати своїм часом та енергією.

Зв'язок роботи з науковими програмами, планами, темами. Виконання дипломної роботи було проведено у відповідності до графіка наукових досліджень, що був розроблений на кафедрі програмного забезпечення.

Мета і завдання дослідження. Метою цього дослідження є розробка програмного забезпечення системи управління локальними та мережевими ресурсами на прикладі прогресивного веб-застосунку, спрямованого на підвищення концентрації та зниження рівня стресу у користувачів за допомогою ефективної роботи з файлами онлайн та офлайн.

Головними завданнями, які необхідно вирішити під час реалізації проєкту, є розробка моделей та алгоритмів управління локальними та мережевими файлами на прикладі створення прогресивного веб-застосунку із зручним користувацьким інтерфейсом, корисними віджетами для концентрації, використанням мережевих та лакальних файлів для налаштування зовнішнього вигляду та інтерфейсів застосунку, доступністю застосунку в онлайн та офлайн режимі.

Задачі дослідження

1. Виконати дослідження та аналіз сфери управління локальними та мережевими ресурсами на прикладі розробки прогресивного веб-застосунку для розуміння предметної області.

2. Виконати детальний порівняльний аналіз популярних онлайн-сервісів для концентрації.
3. Визначити ключові завдання, які потрібно виконати при розробці прогресивного веб-застосунку для концентрації із функціями онлайн та офлайн використання.
4. Виконати проектування модулів фронтенд частини відображення інтерфейсів користувача та функціоналу налаштування застосунку з використанням мережевих та локальних файлів.
5. Розробити алгоритми для модулі для кешування даних та медіа файлів для підтримки роботи застосунку в офлайн режимі.
6. Виконати тестування.

Об’єкт дослідження. Об’єктом дослідження є моделі та алгоритми управління електронними ресурсами на прикладі розробки прогресивного вбезастосунку.

Предмет дослідження – є технології управління локальними та мережевими файлами у вебзастосунках.

Методи дослідження та розробки: метод аналізу для обгрунтування доцільності, вибору інструментів та технологій; метод синтезу – для формування моделей, алгоритмів, макету програмного продукту та інтерфейсу; метод алгоритмізації – для формування основних алгоритмів кешування даних; метод веб-програмування для реалізації програмного продукту з використанням технологій HTML, CSS, JavaScript, TypeScript та бібліотеки React, метод налаштування вебзастосунку для використання.

Новизна отриманих результатів:

Запропоновано метод управління локальними та мережевими файлами, який, на відміну від існуючих дозволить поєднувати мережеві та локальні файли для персонального налаштування робочого простору і використання функціоналу застосунку в офлайн режимі, що дозволить використовувати застосунок в різних умовах доступу до мережі.

Практичне значення розроблений в рамках дослідження прогресивний вебзастосунок може бути використаний в режимах онлайн та офлайн для зниження рівня стресу.

Можливість персоналізації робочого простору дозволяє кожному користувачеві створити оптимальні умови для роботи, враховуючи його індивідуальні потреби та стиль роботи, використовуючи мережеві та локальні ресурси.

Функції прогресивного веб-застосунку надають можливість використання застосунку у веб-переглядачі так і імітуючи роботу нативних застосунків крос платформно в режимі відсутності інтернет з'єднання.

Особистий внесок здобувача. Отримані результати дослідження та розробки представлені автором особисто. Зокрема, у тезах доповіді автор представляє аналіз відомих рішень для створення удосконаленої фронтенд-частини проєкту.

Апробація результатів здійснена на науково-технічній інтернет-конференції «Інформаційне суспільство», Тернопіль 2024.

Публікації. Тези доповідей представлені в збірнику конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ), 2024 р.

Білоус С. А., Коваленко О. О. Аналіз сучасних програмних застосунків для роботи з файлами онлайн та офлайн на прикладі поступового вебзастосунку для концентрації. Інформаційне суспільство.№ 90.Тернопіль 2024.

Структура та обсяг роботи. Пояснювальна записка до бакалаврської дипломної роботи складається з чотирьох розділів.

Пояснювальна записка містить 68 сторінок тексту, 23 рисунка, 1 таблицю, чотири додатки, 28 найменування в списку використаних джерел.

1 АНАЛІЗ РОЗВИТКУ СИСТЕМ УПРАВЛІННЯ ЛОКАЛЬНИМИ ТА МЕРЕЖЕВИМИ РЕСУРСАМИ

1.1. Особливості рішення задач управління ресурсами офлайн та онлайн

Управління локальними та мережевими ресурсами можливо з використанням різних методів та інструментів. Серед них – офлайн та онлайн сховища, кешування та локальне зберігання. Але такі інструменти вимагають спеціальних підходів до безпеки, синхронізації та управління конфліктами [1].

Для розв’язання поставленої задачі можна використовувати різні методи та інструменти. Є кілька ключових аспектів цього аналізу:

Забезпечення можливості роботи з файлами в офлайн-режимі передбачає розробку механізму зберігання даних локально на пристрої користувача. Це може бути реалізовано за допомогою локальної бази даних або файлової системи пристрою. Важливо врахувати відмінності між операційними системами та пристроями користувачів для забезпечення сумісності та оптимальної продуктивності.

Для забезпечення доступу до файлів у режимі онлайн необхідно реалізувати механізм синхронізації з віддаленим сервером або хмарним сховищем. Це може бути виконано за допомогою API відповідного хмарного провайдера або розробленням власного протоколу синхронізації [3]

Щоб забезпечити швидкий доступ до файлів у режимі офлайн, можна використовувати механізми кешування та локального зберігання. Це дозволяє користувачам працювати з файлами, навіть якщо вони тимчасово втратили зв'язок з Інтернетом.

Враховуючи можливість одночасної роботи з файлами у режимі онлайн та офлайн, важливо розробити ефективний механізм управління конфліктами. Це може включати автоматичне об'єднання змін або повідомлення користувача про конфлікт та можливі варіанти вирішення.

Забезпечення конфіденційності та цілісності даних важливо як у режимі онлайн, так і офлайн. Для цього можна використовувати різні методи шифрування даних та механізми автентифікації.

На основі цього аналізу можна розробити декілька варіантів застосунку під певні пристрої із максимальною оптимізацією під певну операційну систему та це вимагатиме комплексне оволодіння значною кількість технологій та значну витрату часу, враховуючи ці параметри вирішено використати крос платформну мову програмування JavaScript та функціонал поступового вебзастосунку із поднанням функціоналу кеш сховища. Це надає можливість розробити комплексний застосунок для концентрації, який буде ефективно працювати як у режимі онлайн, так і офлайн, забезпечуючи користувачам зручний та безпечний доступ до їхніх файлів у будь-який час та в будь-яких умовах.

1.2 Порівняльний аналіз вебзастосунків управління локальними та мережевими ресурсами на прикладі вебзастосунків управління стресом

На сьогоднішній день існує значна кількість вебзастосунків, спрямованих на управління стресом та підвищення продуктивності. Для поставлених завдань важливо проаналізувати можливості роботи застосунків офлайн та онлайн .

LifeAt [4] – Десктопний застосунок пропонує корисні функції релаксації та концентрації, базуючись на індивідуальних потребах користувачів. Він включає інструменти для моніторингу та планування робочих та особистих завдань. Персоналізоване налаштування фону робочого простору та фонової музики використовуючи мережеві файли готових підбірок та плейлистів. Функції створення створених списків завданнь та робочого простору із іншими користувачами застосунку. Моніторингу часу роботи, відпочинку та успішності виконання завдань.

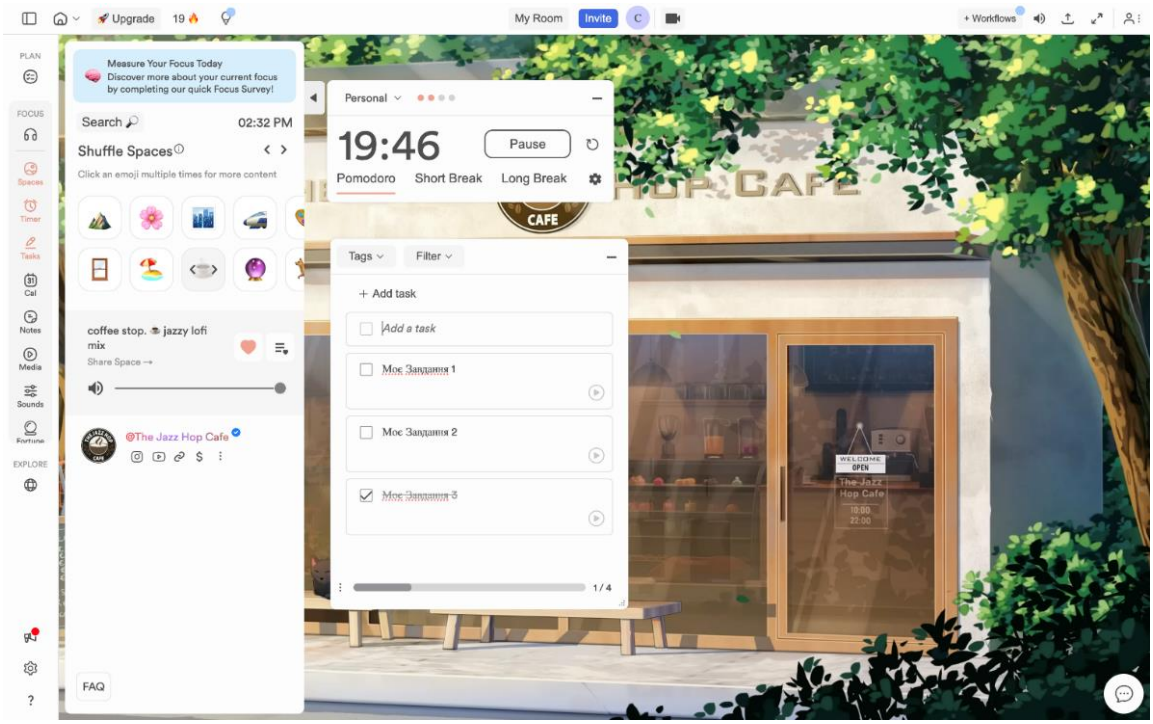


Рисунок 1.2 – Приклад налаштування робочого простору із списком завдань в LifeAt

Headspace [5] – це відомий медитаційний та релаксаційний додаток, який допомагає користувачам знизити рівень стресу, покращити якість сну та підвищити загальний стан психічного благополуччя. Додаток пропонує широкий вибір медитаційних програм та релаксаційних вправ, розроблених професійними психологами та медитаційними експертами. Користувачі можуть вибирати програми відповідно до своїх потреб та цілей, будь то зняття стресу, покращення сну, підвищення концентрації або підтримка психічного здоров'я. Він включає в себе аудіо та відео матеріали для релаксації та медитації.

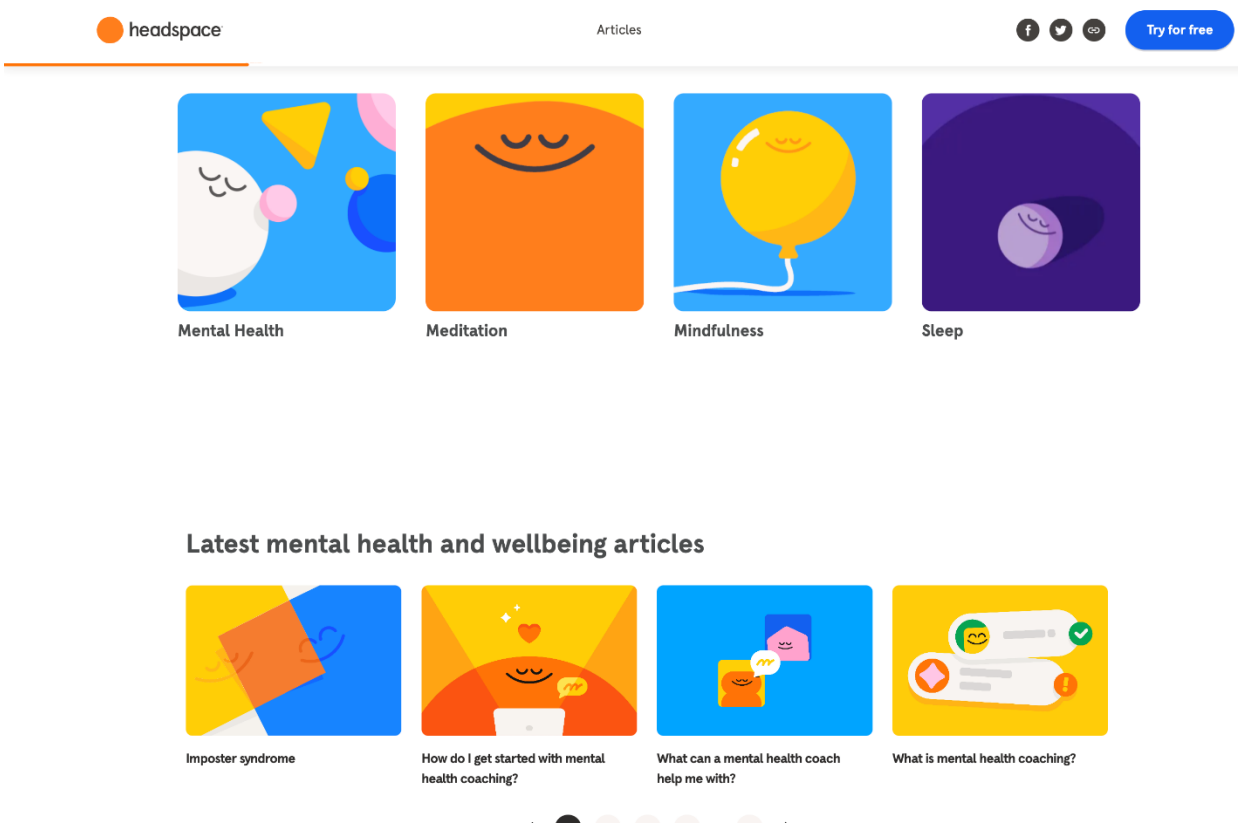


Рисунок 1.3 – Приклад списку корисного функціоналу в Headspace

Noisli – це це веб-додаток для зосередження. За допомогою Noisli [6] можна змішувати та поєднувати різні звуки, щоб створити ідеальне звукове середовище. На даний момент він пропонує 28 високоякісних фонових звуків, які можна відтворювати окремо або комбінувати один з одним. Ви можете регулювати гучність для кожного звуку, щоб налаштувати свої комбінації відповідно до своїх потреб і зробити їх своїми. Окрім створення власних комбо, він також пропонує ретельно підібрані списки відтворення, щоб надихнутися та досліджувати нові звукові пропозиції для різних ситуацій. Надає функцію таймер, який допоможе працювати під час сеансів, щоб запобігти вигорянню, а також текстовий редактор. Надає можливість використання як веб-додатку та як розширення для веб-переглядача.

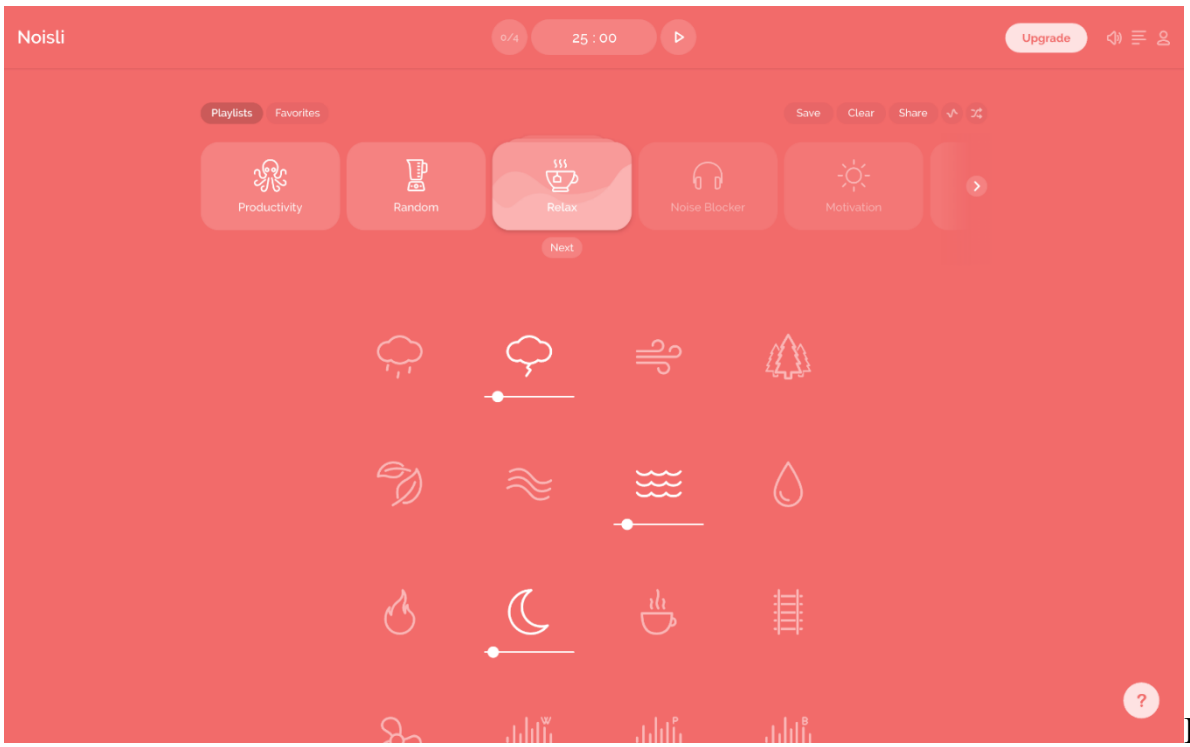


Рисунок 1.4 – Приклад вибору та налаштування фонових звуків в Noisli

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1 – Порівняння аналогів

Критерії	LifeAt	Headspace	Noisli	Власний модуль
Зручність користувацького інтерфейсу	+	+	+	+
Крос платформність	-	-	+	+
Індивідуальна кастомізація	+	-	+	+
Можливість використання мережевих та локальних файлів	-	-	-	+

Продовження таблиці 1.1

Доступність в офлайн режимі	-	-	-	+
Збереження даних за акаунтом користувача	+	+	+	-
Загальна оцінка	50%	34%	70%	85%

Ці застосунки відображають різноманітні підходи до управління стресом та підвищення продуктивності, з використанням мережевих та локальних ресурсів, проте кожен з них має свої переваги та обмеження, які варто врахувати під час розробки прогресивного вебзастосунку.

Відповідно до результатів порівняльної таблиці розробка веб-додатку є актуальною, так як переваги даного модулю є значними для фінального користувача і покриє велику частку необхідного функціоналу у даній категорії використання.

1.3 Постановка задач для прогресивного веб-застосунку з системою управління локальними та мережевими ресурсами

Проаналізувавши функціональність аналогів були прийняті завдання для розробки власного веб-додатку для концентрації із функціоналом використання мережевих та локальних ресурсів:

- визначити спосіб відображення та взаємодії із користувачем;
- розробити об’єктну модель класів;
- розробити базовий функціонал прогресивного веб-застосунку
- розробити необхідні налаштування для кешування та роботи додатку офлайн
- розробити інтерфейс для використання локальних файлів
- розробити серверну частину та модель бази даних для отримання необхідних даних та медіа файлів в режимі онлайн

- розробити інтерфейс для перемикання між мережевими та локальними ресурсами
- протестувати поступовий веб-додаток у веб переглядаці так і на девайсах в режимі онлайн та офлайн.

1.4 Висновки до розділу 1

В першому розділі розглянуті основні відомі підходи використання електроннийх ресурсів офлайн та онлайн. Розглянуті аналоги вебзастосунку управління стресом для дослідження моделей управління локальними та мережевими ресурсами та використання вебзастосунку в різних умовах доступу до Інтернет.

2 РОЗРОБКА ЗАСТОСУНКУ «АНТІСТРЕС» З СИСТЕМОЮ УПРАВЛІННЯ МЕРЕЖЕВИМИ ТА ЛОКАЛЬНИМИ РЕСУРСАМИ

2.1 Аналіз функцій управління стресом для дослідження можливості розробки прогресивного застосунку з управлінням локальними тамережевими файлами

Важливість застосунку використовувати мережеві та локальні файли для кастомізації полягає у можливості індивідуалізації робочого простору та робочих процесів користувачів. Це дозволяє користувачам створювати персоналізовані налаштування, що відповідають їхнім унікальним потребам і стилю роботи. Такий підхід сприяє збереженню концентрації та ефективності роботи.

Важливість можливості використання при відсутності інтернет-з'єднання полягає в тому, що вона забезпечує доступ до важливих функцій та даних у будь-який час і в будь-яких умовах. Це дозволяє користувачам продовжувати працювати навіть у відсутність Інтернету та забезпечує надійність та стабільність роботи веб-застосунку незалежно від обставин.

Прогресивний веб-застосунок пропонує інноваційний підхід до управління стресом, комбінуючи інструменти для концентрації та зняття стресу. Він дозволяє користувачам ефективно керувати своїм робочим простором та персоналізувати його відповідно до їхніх потреб.

Користувачеві можуть надаватись такі функції як вибір та налаштування фонового відео та фонової музики, які на сайті для релаксації та концентрації можна використовувати для створення сприятливої атмосфери, що сприяє зниженню стресу та підвищенню продуктивності, налаштування таймеру роботи, створення списку завдань та виконання дихальних вправ.

Фонове відео може значно вплинути на емоційний стан та настрій користувачів. Воно може створити візуальну атмосферу, та використовуватись як візуальний стимул, фокусування уваги, створення настрою. Фонова музика також

відіграє важливу роль у створенні сприятливих умов. Наприклад стимуляція мозкової діяльності, зниження рівня стресу, блокування шуму. Такий веб-застосунок для концентрації та боротьби зі стресом може включати різноманітні налаштування фонового відео та музики, дозволяючи користувачам вибирати те, що найкраще відповідає їхнім особистим потребам та вподобанням.

Таймери, для прикладу, дозволяють структурувати робочий день, розбиваючи його на чіткі інтервали роботи та відпочинку. Це допомагає уникнути перевантаження та зберігати високу продуктивність протягом дня. Знання того, що є чіткий час для роботи та перерв, допомагає користувачам зосередитися на завданні. Вони менше відволікаються, оскільки знають, що незабаром буде перерва, під час якої можна відпочити або відволіктися. Зниження втоми та стресу, регулярні перерви допомагають зменшити фізичну та психічну втому. Відпочинок дозволяє відновити енергію та знизити рівень стресу, що позитивно впливає на загальне самопочуття. У такому веб-додатку, таймери можуть бути інтегровані як важливий інструмент для користувачів та поєднувати такі функції як налаштованість, сповіщення та нагадування, мати підказки щодо активностей під час перерв, такі як дихальні вправи, розтяжки або короткі медитації для релаксації.

Список завдань, як інструмент для організації та планування. Маючи чіткий список завдань, користувачі можуть зосередитися на виконанні одного завдання за раз, що сприяє підвищенню продуктивності та зниженню розсіювання уваги. Відстеження прогресу. Список завдань дозволяє відстежувати прогрес у виконанні завдань. Позначаючи виконані завдання, користувачі можуть бачити свої досягнення, що сприяє мотивації та задоволенню від виконаної роботи.

Дихальні вправи, які допомагають в зниження стресу та тривожності, покращення концентрації, підвищення енергії, простота виконання. Дихальні вправи прості у виконанні та не вимагають спеціального обладнання чи підготовки. Їх можна виконувати в будь-якому місці та в будь-який час, що робить їх доступними для всіх користувачів.

За допомогою цього вебзастосунку користувачі можуть створити сприятливі умови для роботи та відпочинку, використовуючи мережеві та локальні ресурси та різноманітні інструменти для підвищення концентрації, відновлення енергії та зниження рівня стресу.

Завдання застосунку можна виконувати в різних режимах – онлайн та офлайн.

2.2 Аналіз даних необхідних для функціонування застосунку

Для максимальної персоналізації застосунку більшість корисних функцій представленні у вигляді інтерфейсів для налаштування власного робочого простору, який найбільше задовільнив би вимого користувача.

Враховуючи вид взаємодії користувача із застосунком, а саме у вигляді віджетів, кожний з яких має власні корисні функції, тому і реалізовані моделі даних для кожного окремого віджета.

Для налаштування зовнішнього вигляду робочого простору, а саме фонового відео або зображення та фонової музики, реалізований віджет як елемент користувацького інтерфейсу, і який має моє можливість отримувати дані із мережі використовуючи HTTP протокол, REST API [3] архітектуру серверної частини для взаємодії із базою даних. Та який має можливість використовувати локальні файли, такі як відео файли у форматі mp4, аудіо файли у форматі mp3, та зображення. Незалежно від джерела отримання даних, мережа чи локальні файли, використовуються наступні моделі даних для використання віде та музики.

Для фонового відео отримуються дані у вигляді масиву обєктів, де кожний об'єкт це група відео об'єднаних за певною категорією. Одна категорія включає в себе назву, фонове зображення, іконку, унікальний ідентифікатор та масив об'єктів для окремого відео. Кожне відео із категорії в свою чергу містить силку на місце зберігання файлу в хмарному сховищі, коротку назву, та унікальний ідентифікатор.

Для фонової музики отримуються дані у вигляді масиву об'єктів, де кожний об'єкт це група аудіо об'єднаних за категорією. Кожна категорія включає в себе назву, іконку, унікальний ідентифікатор та масив об'єктів для кожного аудіо. Кожне аудіо із категорії в свою чергу містить посилання на місце зберігання файлу в хмарному сховищі, коротку назву, та унікальний ідентифікатор. В самому застосунку до фонового аудіо добавляється ще налаштування зручності у вигляді числового значення.

Із інших корисних функцій застосунку це створення списку завдань. Для цього створюється масив в якому кожне завдання це окремий об'єкт, який містить самий текст завдання, булеве значення чи завдання виконане та унікальний ідентифікатор.

Для контролю часу роботи та відпочинку в застосунку використовується таймер із функцією персоналізації. Де відповідно налаштування часу роботи та відпочинку зберігаються у вигляді числових значень. А також для сповіщення про закінчення кожного етапу використовується звуковий сигнал, для якого користувач має можливість налаштувати гучність, яка теж зберігається у вигляді числового значення.

Для зручного налаштування робочого простору для широкоекранних пристрої, а саме контрольованого розміщення віджетів на екрані, реалізована функція Drag&Drop [7]. Відповідно за кожним віджетом створюється та зберігається об'єкт із x та у позицією на екрані у вигляді числових значень.

2.3 Розробка методу та моделі системи налаштування робочого простору

Щоб краще зрозуміти роботу модулю налаштування робочого простору та вибору джерела отримання даних була створена модель системи на прикладі діаграми діяльності UML (рис. 2.1).

Запропонована модель є основою методу налаштування робочого простору управління локальними та мережевими ресурсами, який дозволяє зберігати дані та функціонал для користування застосунком локально та оновлення і використання даних і функціоналу при підключенні до Інтернет.

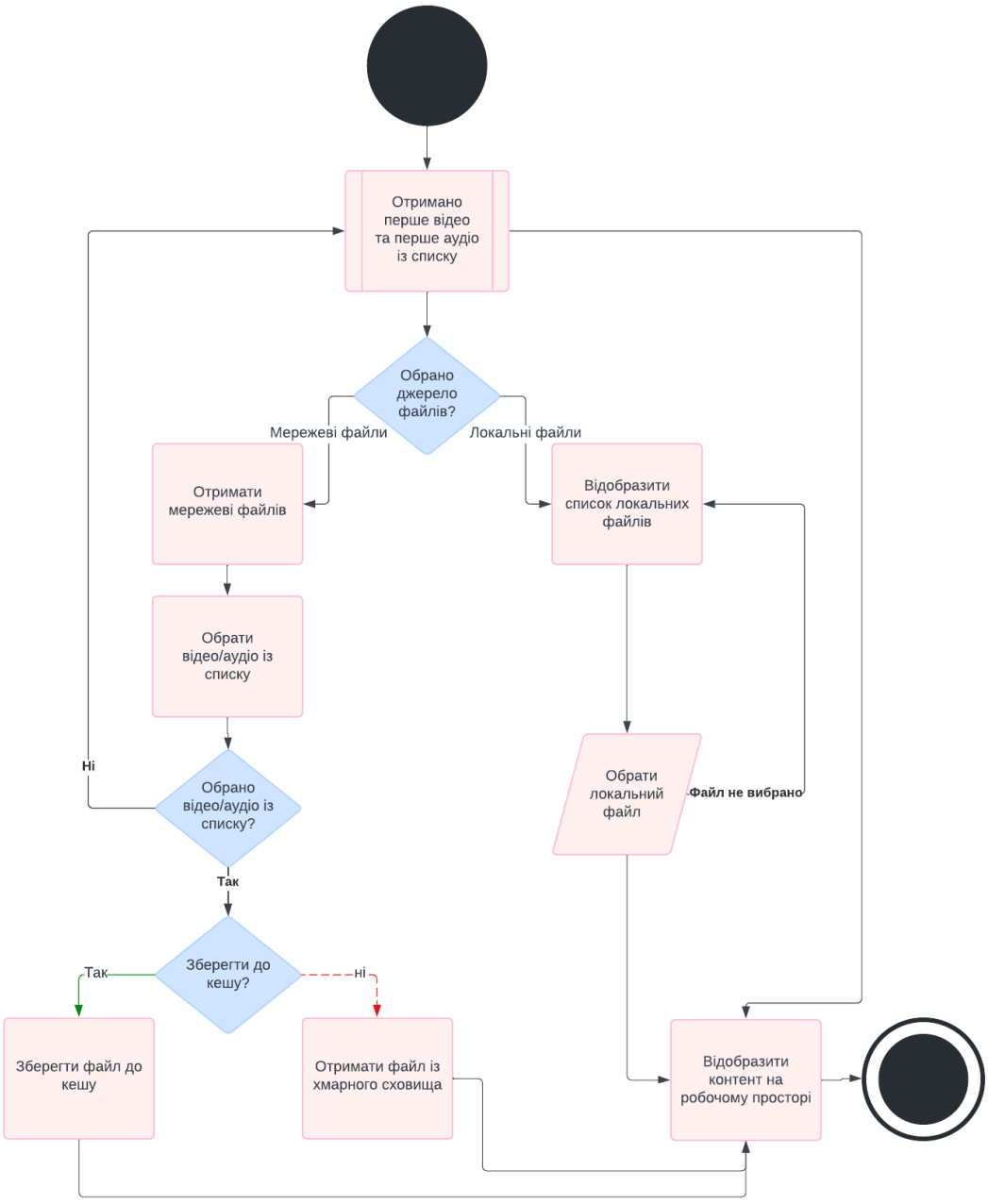


Рисунок 2.1 – Діаграма процесу налаштування робочого простору та вибору джерела отримання даних

Користувач переходить на сторінку із робочим простором та базово відкритим вікном налаштування робочого простору. Спочатку у вікні

налаштувань вибрана робота із мережевими файлами. Застосунок отримує необхідні дані та розпочинає відтворення першого відео файлу та першого аудіо файлу із списку. Користувач має можливість обрати із яким джерелом даних працювати, мережеві файли чи локальні файли. При виборі мережевих даних, користувач може обрати бажане фонове відео та фонову музику із списку отриманих мережевих даних. Для кожного медіа ресурсу, користувач має можливість зберегти до кешу, із якого потім ці дані будуть отримуватись, інакше ці медіа файли будуть запитуватись із мережі. Якщо ж був обраний режим роботи із локальними файлами, та після добавлення файлу до застосунку із пристрою, за допомогою форми, даний медіа файл відтворюється на робочому просторі, зберігається до локального сховища та при перезавантаженні застосунку надходить до застосунку із локального сховища.

2.4 Розробка алгоритмів роботи модулю вибору мережевих чи локальних файлів та кешування

Враховуючи вид взаємодії користувача із застосунком, важливим також є розробка алгоритму роботи застосунку з врахуванням вибору мережевих або локальних ресурсів. Такі ресурси представлені у вигляді віджетів, кожний з яких має власні корисні функції, тому і реалізовані алгоритми обробки даних для кожного окремого віджета.

Застосунок має можливість використовувати локальні файли, такі як відео файли у форматі mp4, аудіо файли у форматі mp3, та зображення.

Для розробки модулю вибору джерела отримання файлів та кешування алгоритм представлено на рисунку (рис. 2.2).

Представлений алгоритм складається з гілок використання мережевих та локальних файлів, перевірки доступу, необхідності оновлення та перезавантаження.

Алгоритм перевіряє доступ, необхідність збереження, кешування. При наявності доступу виконується оновлення файлів та можливість перенастрування параметрів для оновлення та використання ресурсів.

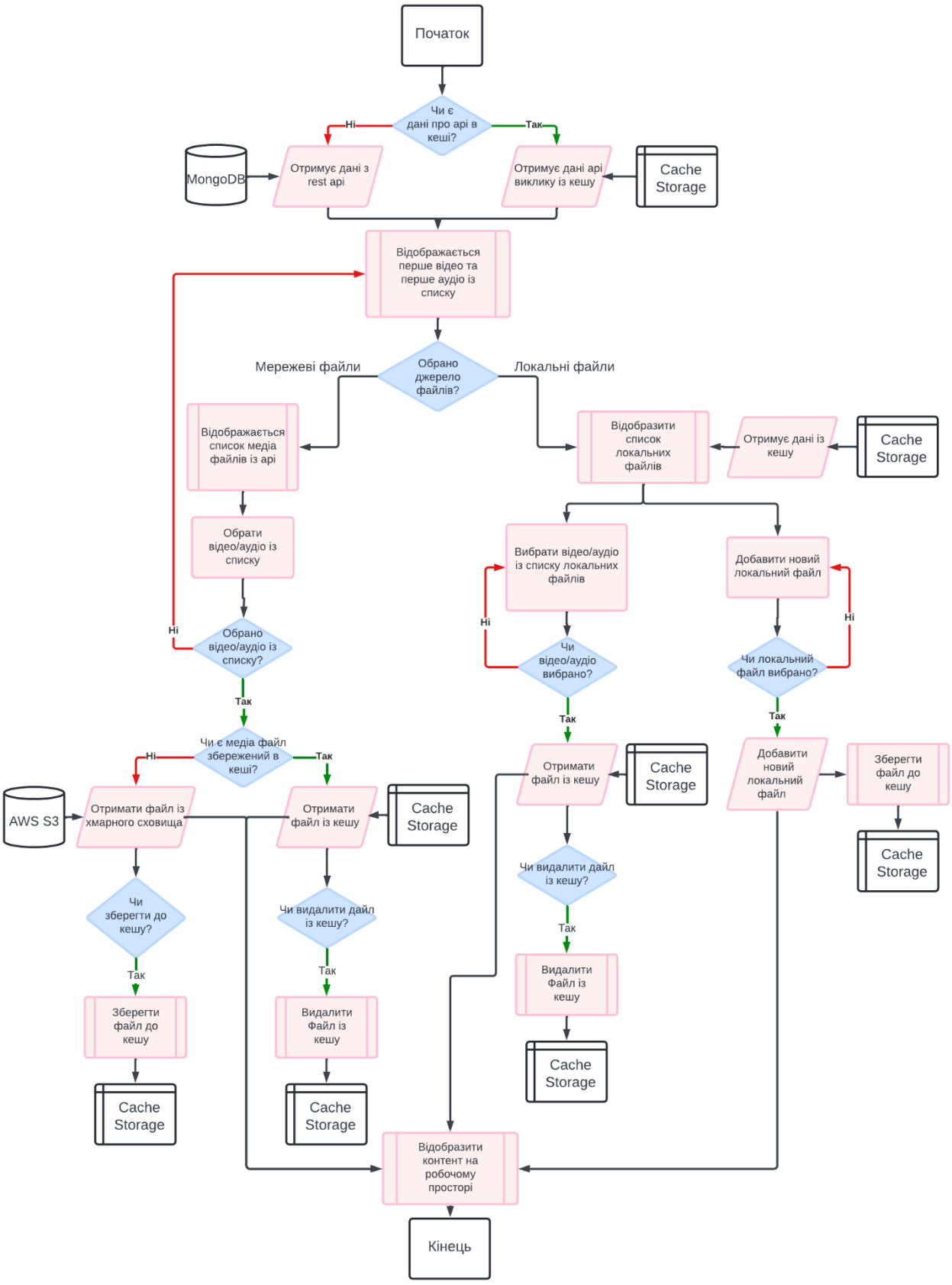


Рисунок 2.10 – Схема роботи модулю налаштування робочого простору, вибору джерела отримання даних та кешування

2.5 Висновки до розділу 2

Було проведено аналіз даних та їх моделей, необхідних для роботи прогресивного веб-застосунку. Розроблені основні елементи користувацького інтерфейсу. Було розроблено та побудовано алгоритм взаємодії користувача із веб-застосунком, алгоритм із забезпеченням застосунку мережевими та локальними файлами, їх кешування, використовуючи UML діаграми.

3 РОЗРОБКА ПРОГРЕСИВНОГО ВЕБЗАСТОСУНКУ З СИСТЕМОЮ УПРАВЛІННЯ ЛОКАЛЬНИМИ ТА МЕРЕЖЕВИМИ РЕСУРСАМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації прогресивного вебзастосунку з можливостями використання локальних та мережевих ресурсів

При розробці прогресивного веб-застосунку для роботи із мережевими та локальними файлами на прикладі застосунку для концентрації та релаксації важливо обрати оптимальні засоби для реалізації інтерфейсу користувача, стилізації, інструменту для збірки, вибору технологій для виконання HTTP запитів, та адаптації застосунку під вимоги прогресивного веб-застосунку.

JavaScript є найпопулярнішою мовою програмування для розробки веб-застосунків завдяки своїй широкій підтримці браузерами та потужній екосистемі.

Переваги:

– Широка підтримка: Всі сучасні браузери підтримують JavaScript, що робить його доступним для розробників.

– Велика екосистема: Безліч бібліотек і фреймворків, таких як React[8], Angular та Vue.js, які спрощують розробку.

– Динамічність: Дозволяє швидко створювати інтерактивні елементи на сторінці.

Недоліки:

– Відсутність статичної типізації: Може призводити до помилок, які важко відстежити.

– Складність у великих проєктах: Без суворої типізації коди можуть стати важкочитаними та складними у підтримці.

TypeScript — це надбудова над JavaScript, яка додає статичну типізацію. Що означає, що все, що доступно в JavaScript, також доступно в TypeScript, і кожна

програма на JavaScript є синтаксично коректною програмою на TypeScript. Однак TypeScript додає перевірку типів під час компіляції, впроваджуючи правила щодо того, як різні типи можуть бути використані та поєднані. Це дозволяє виявляти широкий спектр програмних помилок, які в JavaScript зустрічаються лише під час виконання [9].

Переваги:

– Статична типізація: Допомагає виявляти помилки на етапі компіляції, що зменшує кількість багів.

– Покращена підтримка IDE[9]: Інтеграція з інструментами розробки, що підвищує продуктивність.

– Сумісність з JavaScript: Можна поступово впроваджувати у вже існуючі проєкти на JavaScript.

Недоліки:

– Додаткове налаштування: Потрібна конфігурація компілятора TypeScript.

– Більший вхідний поріг: Для новачків може бути складніше освоїти у порівнянні з JavaScript.

Вибір TypeScript обґрунтований його здатністю знижувати кількість помилок та підвищувати підтримуваність коду, що особливо важливо для великих та складних проєктів,

Вибір фреймворку, бібліотеки для побудови інтерфейсів користувача.

React – це відкрита JavaScript-бібліотека, яка використовується для розробки інтерфейсів користувача. Він був створений компанією Facebook і швидко набув популярності серед розробників з усього світу. React дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю. Одним з ключових концепцій у React JS є компоненти. Вони представляють собою незалежні блоки коду, які відповідають за рендеринг певної частини користувацького інтерфейсу.

Переваги:

– Використання віртуального DOM та ефективного алгоритму оновлення дозволяє робити мінімальні зміни в реальному DOM, що покращує продуктивність застосунків;

– JS React базується на компонентній архітектурі, що дозволяє розбивати інтерфейс на незалежні компоненти. Це спрощує розробку, тестування та підтримку коду, оскільки компоненти можуть бути повторно використані та легко змінюються;

– React пропагує односторонній потік даних, що сприяє простоті та передбачуваності управління станом додатків, полегшує відлагодження та тестування застосунків;

– Спільнота розробників, що використовує React велика та активна. Є безліч ресурсів, бібліотек та інструментів для розробки. Також існує багато сторонніх бібліотек, які підтримують React і допомагають розширити його функціональність.

Недоліки:

– Стрімкі зміни: Швидкий розвиток бібліотеки може призводити до необхідності частих оновлень.

– Відсутність вбудованих рішень: Потреба у використанні додаткових бібліотек для управління станом або маршрутизацією.

Angular — це фреймворк, розроблений Google, який забезпечує повний набір інструментів для створення веб-додатків.

Переваги:

– Повний набір функцій: Включає все необхідне для розробки великих додатків.

– Декларативний підхід: Сприяє зрозумілості та зручності у написанні коду.

Недоліки:

– Складність: Вищий вхідний поріг, особливо для новачків.

– Обмежена гнучкість: Може бути надмірним для невеликих проєктів.

Vue.js — це прогресивний фреймворк для створення користувацьких інтерфейсів, який наголошує на простоті та гнучкості.

Переваги:

– Легкість в освоєнні: Інтуїтивний синтаксис та документація.

– Гнучкість: Легко інтегрується з існуючими проєктами та іншими бібліотеками.

Недоліки:

– Менша спільнота: Порівняно з React та Angular.

– Обмежена масштабованість: Може бути менш ефективним для великих корпоративних додатків.

React обрано за його компонентний підхід, високу продуктивність та велику спільноту, що забезпечує доступ до широкого спектру ресурсів та інструментів. React добре підходить для розробки проектів будь-якого масштабу. Він надає можливості для легкого розширення та перевикористання компонентів, інтеграції з іншими бібліотеками та фреймворками.

Огляд бібліотек для керування станом застосунку.

Zustand [10] — це бібліотека для керування станом в React-застосунках. Вона пропонує простий та зручний інтерфейс для організації стану додатка та його оновлення. Давайте порівняємо Zustand з деякими аналогами, такими як Redux та Context API, щоб побачити його переваги та недоліки.

Redux — це один з найпопулярніших state manager для React. Він базується на принципах однозначності та незмінності даних. Для управління станом в Redux використовується централізований Store, що може бути складним у великих додатках.

Переваги Zustand перед Redux:

– Простота використання: Zustand має простий API та не вимагає великої кількості boilerplate коду, який потрібен у Redux.

– Продуктивність: Бібліотека Zustand має менше накладних витрат, оскільки вона не вимагає копіювання даних, як у Redux.

– Легкість інтеграції: Zustand інтегрується з React з меншими зусиллями, ніж Redux, що полегшує процес розробки.

Недоліки Zustand порівняно з Redux:

– Менша екосистема: Екосистема Redux дуже розгалужена, і вона має багато розширень та плагінів, що дозволяють робити більше речей.

Context API — це частина React, яка дозволяє передавати дані по компонентах безпосередньо, без необхідності використання проміжного компонента.

Переваги Zustand порівняно з Context API:

– Легкість використання: Zustand надає простий API для управління станом, що робить його більш приємним у використанні порівняно з Context API.

– Покращена продуктивність: За допомогою Zustand, ви можете оновлювати стан тільки в компонентах, які підписані на цей стан, що дозволяє уникнути зайвих рендерів.

Zustand — це простий та потужний інструмент для керування станом в React-додатках. Порівняно з Redux, він пропонує простіший підхід до управління станом без необхідності великої кількості шаблонного коду. Порівняно з Context API, Zustand надає кращу продуктивність та легкість використання.

Вибір фреймворку, бібліотеки для стилізації.

Tailwind CSS[11] — це утилітарний CSS-фреймворк, який дозволяє створювати дизайни безпосередньо у HTML Tailwind CSS є одним із найпопулярніших інструментів для швидкої та ефективної розробки стильового дизайну веб-сайтів і додатків. Він надає величезний набір готових класів, що дозволяють розробникам створювати різноманітні компоненти і макети із стилізацією без необхідності написання власного CSS коду з нуля.

Переваги:

– Швидкість розробки. Однією з найбільших переваг Tailwind CSS є його здатність прискорити процес розробки. Замість написання CSS вручну для кожного елемента або компонента, розробники можуть використовувати готові

класи, які вже мають заздалегідь визначені стилі. Це значно скорочує час, необхідний для розробки і дозволяє швидше створювати інтерфейси.

– Простота використання. Tailwind CSS пропонує простий і зрозумілий синтаксис для використання. Він базується на наборі класів, кожен з яких відповідає певному стилевому правилу. Це робить процес розробки більш прозорим і зрозумілим для новачків, а також сприяє швидкому розгортанню проектів.

– Гнучкість і розширюваність. Іншою важливою перевагою є гнучкість Tailwind CSS. Ви можете легко налаштовувати та розширювати бібліотеку за допомогою власних класів або налаштувань. Це дозволяє розробникам створювати унікальні інтерфейси, відповідні конкретним вимогам проекту.

– Адаптивний дизайн. Tailwind CSS також надає потужні засоби для реалізації адаптивного дизайну. Ви можете легко визначати різні класи для різних розмірів екрану, дозволяючи вашому веб-сайту адаптуватися до різних пристроїв і розмірів вікна браузера. Це дозволяє створювати зручний і юзер-центричний дизайн для будь-якої аудиторії.

– Підтримка тем оформлення. У Tailwind CSS є вбудована підтримка тем оформлення, що дозволяє швидко і легко змінювати вигляд вашого додатку або веб-сайту. Ви можете використовувати готові теми або налаштовувати власні, забезпечуючи гнучкість у виборі дизайну.

Недоліки:

– Збільшення розміру HTML: Додавання багатьох класів може збільшити розмір HTML-коду.

Bootstrap — це популярний CSS-фреймворк, який надає готові компоненти та стилі.

Переваги:

– Швидкий старт: Велика кількість готових компонентів та стилів.

– Сумісність: Добре підтримується у всіх браузерах.

Недоліки:

– Обмежена гнучкість: Складно змінювати стиль компонентів.

– Великий розмір: Може призвести до перевантаження стилями.

Material-UI — це набір React-компонентів, що реалізує принципи дизайну Material від Google.

Переваги:

– Сучасний дизайн: Відповідає принципам Material Design.

– Інтеграція з React: Оптимізовано для використання з React.

Недоліки:

– Складність кастомізації: Може бути важко адаптувати під специфічні вимоги.

– Залежність від Material Design: Обмежена гнучкість у створенні унікальних стилів.

Tailwind CSS обрано за його гнучкість, можливість швидкої розробки та налаштування стилів без написання великої кількості власного CSS-коду.

При розробці вебзастосунку важливо обрати оптимальні інструменти для збірки, щоб забезпечити ефективний процес розробки та високу продуктивність кінцевого продукту [12].

Vite[13] — це сучасний інструмент для збірки фронтенд-проєктів, розроблений для забезпечення швидкого старту та високої продуктивності під час розробки.

Переваги:

– Швидкий запуск: Vite використовує нативні ES-модулі для швидкого старту сервера розробки, забезпечуючи миттєве завантаження сторінок.

– Гаряче оновлення модулів (HMR): Миттєве оновлення змін без перезавантаження сторінки, що значно підвищує продуктивність розробки.

– Ефективна збірка: Використовує Rollup для кінцевої збірки, забезпечуючи оптимізований розмір та продуктивність.

– Проста конфігурація: Мінімальні налаштування за замовчуванням, що спрощує початкове налаштування проєкту.

Недоліки:

– Молода екосистема: Менше плагінів та інтеграцій порівняно зі старішими інструментами.

– Не така велика спільнота: Менше ресурсів і підтримки у порівнянні з Webpack.

Webpack – це один з найпопулярніших інструментів для збірки фронтенд-проєктів, який пропонує потужні можливості для налаштування.

Переваги:

– Гнучкість та налаштовуваність: Можливість налаштування майже кожного аспекту збірки.

– Велика екосистема: Широкий вибір плагінів та інтеграцій для будь-яких потреб.

– Велика спільнота: Багато ресурсів, документації та прикладів.

Недоліки:

– Складність конфігурації: Високий вхідний поріг через складність налаштувань.

– Повільний запуск: Довгий час на старт сервера розробки, особливо у великих проєктах.

Parcel — це інструмент для збірки, орієнтований на простоту використання та автоматичну конфігурацію.

Переваги:

– Автоматична конфігурація: Працює з мінімальними налаштуваннями, що знижує час на початкове налаштування.

– Швидкий старт: Швидке завантаження сторінок під час розробки.

– Вбудована підтримка багатьох форматів: Підтримка CSS, HTML, зображень та інших файлів без додаткових плагінів.

Недоліки:

– Менша гнучкість: Менше можливостей для налаштування порівняно з Webpack.

– Молода екосистема: Менше плагінів та ресурсів у порівнянні з Webpack.

Vite обрано за його високу продуктивність під час розробки, завдяки швидкому запуску та гарячому оновленню модулів. Простота конфігурації та ефективна збірка кінцевого продукту також є значними перевагами, особливо для проєктів, що використовують сучасні фреймворки, такі як React [14].

Для виконання HTTP запитів у веб-застосунку є декілька можливих технологій, які необхідно розглянути.

Fetch API [15] — це вбудований інструмент в сучасних браузерах для виконання HTTP запитів, який забезпечує простий та зручний спосіб роботи з ресурсами через мережу.

Переваги:

– Нативна підтримка: Підтримується в більшості сучасних браузерів без необхідності додаткових бібліотек.

– Простота використання: Легкий для розуміння синтаксис, що дозволяє легко виконувати запити.

– Підтримка промісів: Дозволяє обробляти асинхронні операції з використанням промісів, що спрощує код.

– Модульність: Дозволяє використовувати інші бібліотеки та інструменти для обробки HTTP запитів при необхідності.

Недоліки:

– Відсутність деяких можливостей: Не має деяких зручних функцій, таких як автоматичне оброблення JSON. Реалізація функціоналу тайм-аути через додаткові конструкції, такі як AbortController або AbortSignal.

– Менш зручна обробка помилок: Вимагає додаткового коду для повного оброблення помилок, включаючи перевірку статус-кодів.

Axios — це популярна бібліотека для виконання HTTP запитів, яка пропонує багато додаткових функцій і полегшує роботу з асинхронними запитами.

Переваги:

– Автоматична обробка JSON: Автоматично перетворює JSON-дані, що спрощує роботу з ними.

– Проста обробка помилок: Має вбудовану обробку помилок з більш інформативними повідомленнями.

– Підтримка тайм-аутів: Легко налаштовувати тайм-аути для запитів.

– Широка підтримка: Підтримує обробку запитів в браузері та Node.js, що робить його універсальним.

Недоліки:

– Додаткова вага: Потребує встановлення та підключення додаткової бібліотеки, що збільшує розмір проєкту.

– Зовнішня залежність: Залежність від зовнішньої бібліотеки може ускладнити підтримку проєкту в довгостроковій перспективі.

XMLHttpRequest — це старіший інструмент для виконання HTTP запитів, який все ще використовується в багатьох проєктах, але має ряд обмежень у порівнянні з Fetch API та Axios.

Переваги:

– Широка підтримка: Підтримується в усіх браузерах, включаючи старі версії.

– Синхронні запити: Можливість виконання синхронних запитів (хоча це рідко використовується через потенційні проблеми з продуктивністю).

Недоліки:

– Ускладнений синтаксис: Більш складний і менш інтуїтивний синтаксис у порівнянні з Fetch API.

– Відсутність підтримки промісів: Вимагає використання callback-функцій для обробки асинхронних операцій, що може ускладнювати код.

– Меньша гнучкість: Обмежений у порівнянні з Fetch API та Axios щодо налаштування та обробки запитів.

Fetch API обрано за його простоту використання, нативну підтримку у сучасних браузерах та можливість обробки асинхронних операцій за допомогою промісів. Хоча Fetch API має деякі обмеження, його можна легко розширити за допомогою додаткових інструментів або власних функцій для обробки помилок та налаштування запитів. Вибір Fetch API також дозволяє зменшити залежність від зовнішніх бібліотек, що сприяє зменшенню розміру проєкту та підвищенню його продуктивності.

При розробці прогресивного веб-застосунку (PWA[13]) важливо обрати оптимальні засоби для налаштування PWA, щоб забезпечити ефективний процес розробки, високу продуктивність кінцевого продукту та позбавить від необхідності мануального написання та реалізації кожної функціональності Service Worker[14].

Vite-PWA-Plugin[14] — це плагін для Vite, який полегшує інтеграцію функціональності PWA в проєкт.

Переваги:

– Інтеграція з Vite: Прямий інтеграційний шлях з Vite забезпечує зручність налаштування та використання.

– Автоматична генерація: Плагін автоматично генерує необхідні файли (manifest[16], service worker), що спрощує процес налаштування.

– Гнучкість налаштувань: Можливість налаштовувати різні аспекти PWA, такі як кешування, повідомлення та інші.

– Активна підтримка: Часті оновлення та активна спільнота, що забезпечує підтримку нових функцій та виправлення помилок.

Workbox[17] — це набір бібліотек та інструментів від Google для полегшення роботи з service worker та реалізації PWA.

Переваги:

– Потужний функціонал: Надає широкий набір функцій для кешування, управління ресурсами та забезпечення офлайн-роботи.

– Автоматизація: Можливість автоматичного генерації та налаштування service worker.

– Гнучкість: Дозволяє налаштовувати різні стратегії кешування та обробки запитів.

– Підтримка від Google: Підтримується та розробляється Google, що забезпечує високий рівень якості та надійності.

Vite PWA Plugin обрано за його інтеграцію з Vite, що забезпечує зручність налаштування та використання. Цей плагін автоматично генерує необхідні файли та дозволяє налаштовувати різні аспекти PWA, що робить його ідеальним вибором для проєкту, побудованого на Vite.

Workbox обрано за його потужний функціонал та гнучкість у налаштуванні кешування, управління ресурсами та забезпеченні офлайн-роботи. Підтримка від Google та можливість автоматизації процесів роблять Workbox надійним та ефективним інструментом для реалізації PWA [16-18].

3.2 Розробка структури інтерфейсу модулів

При розробці інтерфейсу програмного додатку варто звернути увагу на простоту та адаптивність, так як головною особливістю прогресивного веб-застосунку є крос платформність, застосунок має бути адаптований під різні розширення екранів.

Визначити елементи інтерфейсу які потрібно розробити:

- Панель управління активними віджетами
- Робочий простір із фоновим відео
- Віджет налаштування фонового відео та аудіо, який містить таби для вибору режиму роботи, мережеві чи локальні файли
- Віджет із таймером та його налаштуваннями
- Віджет із списком завдань
- Вікно для виконання дихальних вправ

Панель управління активними віджетами має бути представлена у вигляді бокового меню. Вверху якого знаходиться логотип веб-додатку. Під логотипом розміщуються іконки віджетів, при наведені курсора на які відображається “tooltip”, підказка про те яке саме вікно відкриється чи закриється при натисканні на іконку. Також для інформування які саме віджети активні на робочому просторі, відповідні іконки ідентифікуються рамкою.

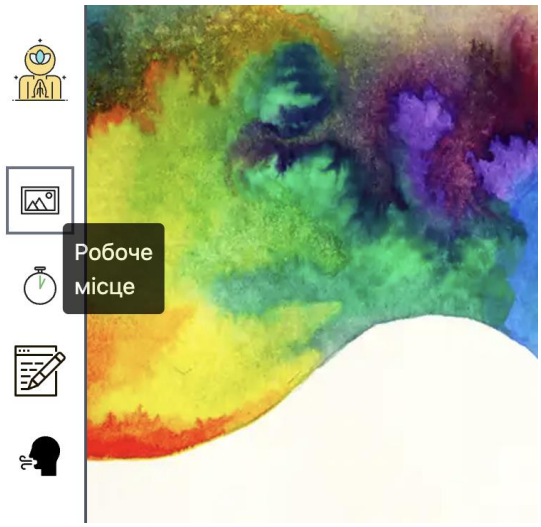


Рисунок 3.1– Панель управління активними вікнами

Робочий простір має представляти собою все інше вільне місце де на фоні відтворюється вибране користувачем з мережевих або локальних файлів відео, у форматі mp4. MP4 з кодеком H.264 є одним з найпоширеніших форматів відео в Інтернеті. Він забезпечує хорошу якість відео при розумних розмірах файлів, що робить його ідеальним для стрімінгового відео на веб-сайтах. H.264 володіє високою ступенем стиснення, що дозволяє ефективно передавати відео навіть при обмеженій ширині смуги. Для зручного та індивідуального розміщення активних вікон на робочому просторі запланована реалізація функціоналу Drag&Drop. Додатково кожне вікно має зберігати дані про своє розміщення, що забезпечує відтворення робочого простору після відновлення сеансу.

Вікно вибору та налаштування фонового відео та аудіо, має містити два режими, вибір із мережевих файлів та з локальних файлів, та інтерфейс для перемикання між ними. В “шапці” вікна знаходиться його назва та іконка для

згортання. У режимі вибору мережевих файлів, в головній частині вікна розмістити елемент “акордеон”, який містить два пункти “фонове відео” та “фонова музика”. В кожному з них відображаються отримані з бази даних списки. Для “фонового відео”, користуючись визначеною моделлю даних, відображається іконка та назва групи і нижче список самих відео. Кожне відео містить його назву та кнопки управління, а саме, відео файл можна зберегти до локального кешу або видалити з кешу. Додатково відео файли що вже збережені до локального сховища візуально мають відображатись фоновим кольором. Вибрана група та саме відео відрізнятимуться рамкою. Той же користувацький інтерфейс реалізований для пункту “фонова музика”. В нижній частині вікна потрібно розмістити елементи управління фоновою музикою. А саме елементи для налаштування гучності звуку, назва відтворюваного треку, та кнопки для переходу на попередні або наступний трек.

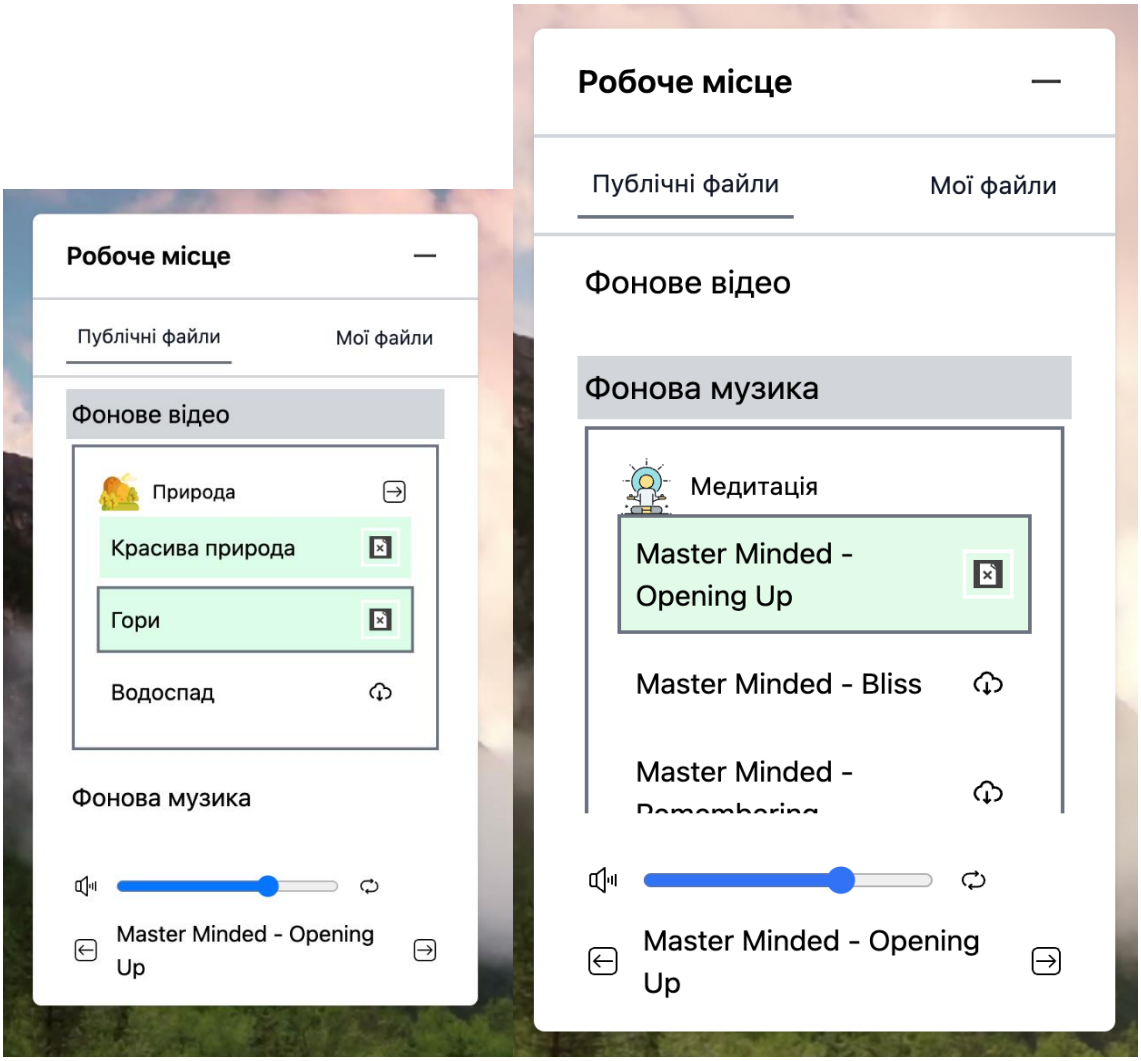


Рисунок 3.2 – Вікно вибору та налаштування фонового відео та аудіо (мережеві файли)

У режимі користування локальними файлами необхідно реалізувати такий же інтерфейс користувача. Із відмінностей це інтерфейс для вибору медіа файлу із ресурсів пристрою, яка і надає можливість користувачеві добавляти локальні файли до застосунку. Дана форма має містити іконки для вибору файлу, текстового поля, яке відповідає за назву ресурсу, та кнопки у вигляді іконки для збереження даних форми.

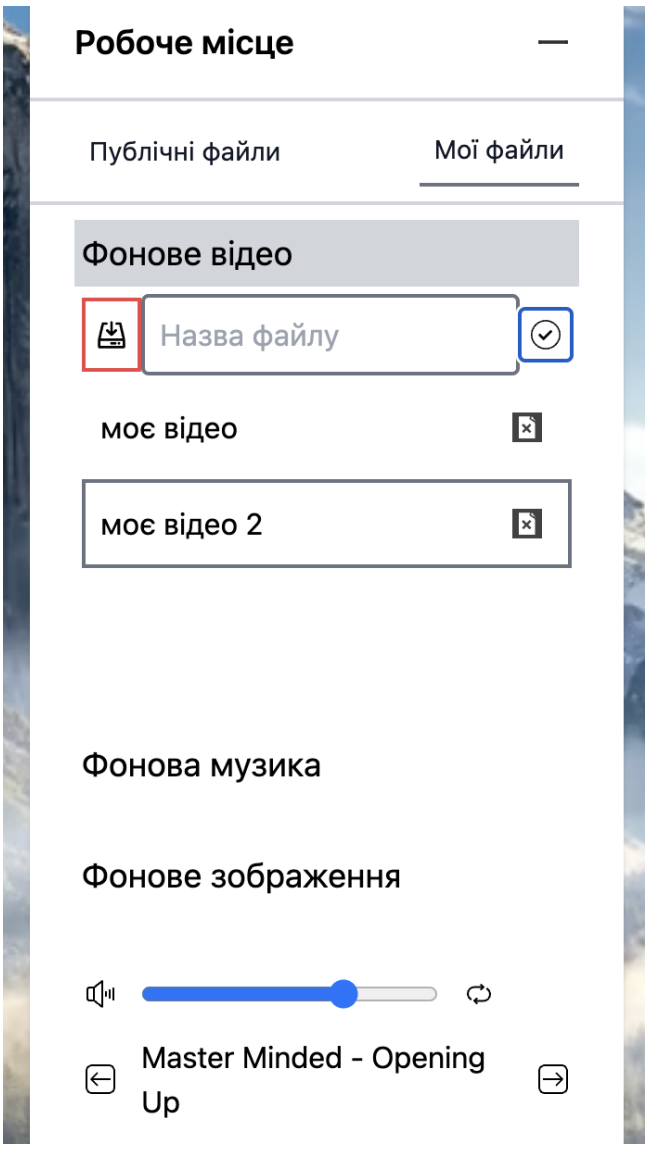


Рисунок 3.3 – Вікно вибору та налаштування фонового відео та аудіо (локальні файли)

Для контролю часу роботи та відпочинку, запланована реалізація віджету таймера. Він має складаєтись із області самого таймера, кнопки “розпочати” або “пауза” та кнопки у вигляді іконки скидання значення таймера до початкового значення. Нижче відобразити таби для перемикання режимів “Робота”, “Коротка перерва” та “Довга перерва”. Також для персоналізації віджета реалізована область налаштувань. Налаштування таймера складаються із форми з трьома текситовими полями, які приймають числові значення та відповідають за відповідний етап роботи чи перерви. Додатково є налаштування гучності

звукового сигналу, який вмикається та сповіщає про закінчення кожного етапу таймера.

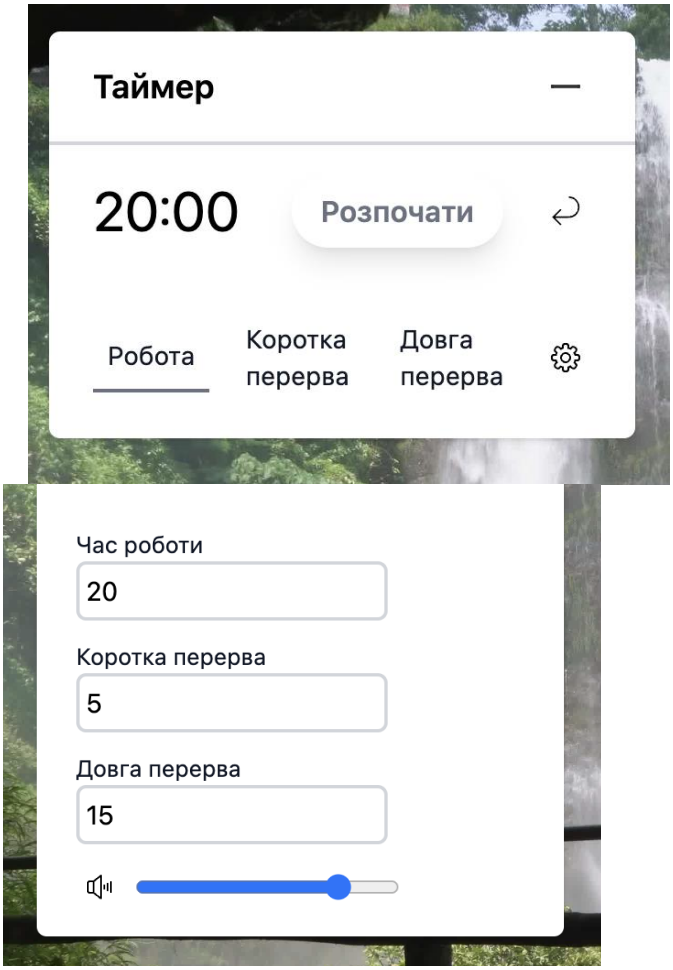


Рисунок 3.4 – Вікно таймера та його налаштування

Для створення списку завдань та контролю їх виконання, необхідно реалізуваит віджет менеджера завдань. Із елементів інтерфейсу користувача в ньому має бути реалізовано самий список завдань, де кожне завдання може бути відмічене як виконане або ж ні та спливаюче меню (“popover”), використовуючи це меню можна дане завдання дублювати або ж видалити. Над списком завдання розмістити кнопку для створення нового завдання та кнопку фільтру. Із можливих варіантів фільтрації списку завдань це - відображати лише не виконані завдання або відображати лише виконані завдання. У нижній частині віджета зображено відношення виконаних завдань до загальної кількості завдань. Також в нижній

частині віджета розмістити іконку для спливаючого меню із функціями управління всім списком, такими як видалити всі завдання або ж видалити всі завершені завдання.

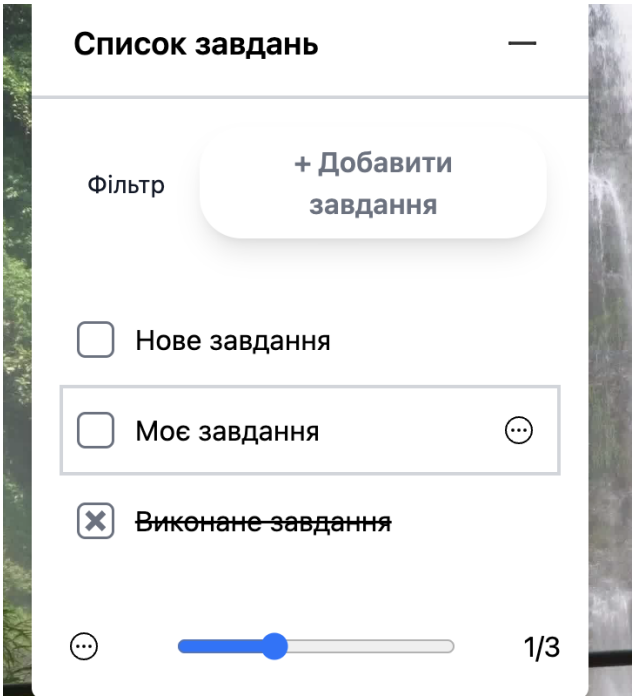


Рисунок 3.5 – Вікно менеджера завдань

Вікно з дихальними вправами. Враховуючи важливість дихальних вправ для процесу концентрації та релаксації, реалізувати інтерфейс що включає в себе анімацію дихання, короткий текст що інформує про фазу вдиху чи видиху та іконки для закриття даного вікна.

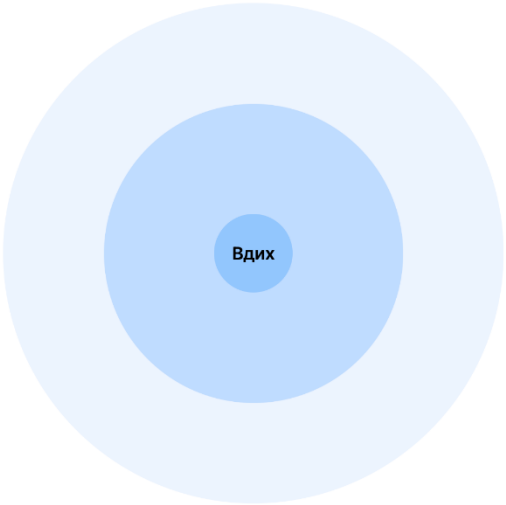


Рисунок 3.6 – Вікно дихальних вправ

При переходить на сторінку із робочим простором, розпочинається робота модулю налаштування зовнішнього вигляду робочого простору та його функціоналу.

Свою роботу модуль розпочинає із запиту до REST API сервер та MongoDB [19] базу даних, при наявності підключення до мережі, після чого отримані дані актуалізують відповідне кеш сховище та передаються далі до компонента. В офлайн режимі цей запит відповідно повертається із помилкою яка інформує про відсутність інтернет з'єднання, тоді модуль звертається за даними до кеш сховища і якщо необхідні дані в локальному сховищі присутні, вони передаються далі до компонента. Дана реалізація надає можливість модулю працювати як в онлайн так і в офлайн режимі.

Отримавши необхідні для виконання компонента налаштування робочого простору, розпочинається відтворення першого відео та першого аудіо із списку. Користувач має можливість обрати із яким джерелом даних працювати, мережеві файли чи локальні файли. При виборі мережевих даних, користувач може обрати бажане фонове відео та фонову музику із списку мережевих ресурсів.

Для кожного медіа файлу, користувач має можливість зберегти до кеш-сховища, із якого надалі ці дані будуть отримуватись, що надає можливість подальшого їх використання в офлайн режимі. Інакше ці медіа файли будуть запитуватись із AWS S3 [20] хмарного сховища. Враховуючи обмеження в доступному об'ємі пам'яті локального кеш-сховища, користувач має можливість видаляти попередньо збережені файли.

Якщо ж був обраний режим роботи із локальними файлами, компонент робить запит до відповідного кеш-сховища, на наявність раніше збережених локальних файлів. Якщо такі є, вони будуть добавлені в список на відображені в компоненті у вигляді списку. І звісно користувач має можливість добавляти нові локальні файли із пристрою, після добавлення файлу до застосунку, даний медіа файл зберігається в кеш-сховищі, відображається у списку файлів в компоненті та відтворюється на робочому просторі. Як і з мережевими файлами, користувач має можливість видаляти дані файли із кешу, та на відміну від мережевих даних, при повторному завантаженню веб-застосунку ці файли не будуть присутні в списку і користувачеві необхідно знову їх завантажувати у застосунок.

Такий підхід надає користувачеві можливість як користуватись підготовленими мережевими даними, так і повністю налаштувати зовнішній вигляд та фонову музику веб-додатку під власні потреби, використовуючи медіа файли із свого пристрою. І поєднуючи це із функціями кешування, користувач отримую можливість використовувати застосунок при відсутності інтернет з'єднання.

3.3 Розробка модуля завантаження та ініціалізації даних

При завантажені додатку для повноцінної роботи застосунку, відбувається надсилання запину на отримання початкових даних. Для підготовки даних та опрацювання запитів був розроблений HTTP сервер із використанням

фреймворків Node.js[21] та Express.js [22], та побудований на архітектурі RESTful API.

Node.js — це середовище виконання JavaScript, побудоване на рушії V8 від Google Chrome. Воно дозволяє запускати JavaScript поза межами веб-браузера, що відкриває нові можливості для розробки серверних додатків і інструментів для командної розробки. Завдяки асинхронній моделі вводу/виводу, ефективному використанню однопоточної моделі з подіями та модульній архітектурі, Node.js забезпечує високу продуктивність та масштабованість додатків.

Express.js — це мінімалістичний і гнучкий веб-фреймворк для Node.js, який надає потужний набір інструментів для створення веб-додатків і API. Express.js є одним з найпопулярніших фреймворків для Node.js завдяки своїй простоті, продуктивності та можливості розширення. Express.js широко застосовується для створення RESTful API, завдяки своїй потужній системі маршрутизації і підтримці middleware[23].

Даний сервер взаємодіє із базою даний реалізованою на MongoDB.

MongoDB — це популярна нереляційна база даних, яка використовує гнучкий підхід до зберігання даних, що відрізняється від традиційних реляційних баз даних. MongoDB є базою даних типу "документ-сховище" і входить до категорії NoSQL баз даних. На відміну від реляційних баз даних, MongoDB не вимагає визначення схеми на момент створення бази даних. Це дозволяє легко змінювати структуру даних у міру розвитку додатка.

Отримані в результаті GET запиту дані надходять до веб-застосунку у вигляді JSON, які потім парсяться до звичайних JavaScript об'єктів. Ці дані являються списками із об'єктами, які містять інформацію про фонові відео та аудіо, та силки на медіа файли, такі як відео файли, аудіо та зображення. Самі ж медіа файли зберігаються та надаються для застосунку із хмарного сховища AWS S3.

Amazon Simple Storage Service (AWS S3) — це веб-сервіс для зберігання та отримання будь-яких об'єктів, які можуть бути досягнуті через Інтернет. AWS S3

є одним з найбільш популярних та розповсюджених сервісів хмарного зберігання даних, який пропонує велику масштабованість, високу доступність та надійність. AWS S3 може зберігати будь-які типи даних, включаючи тексти, зображення, відео, аудіо, архіви тощо.

Таким чином відбувається забезпечення веб-додатку мережевими даними.

3.4 Розробка модуля service worker та кешування даних

Service Worker - це скрипт, який працює в фоновому режимі браузера і має доступ до різних функцій, які не доступні веб-сторінкам, наприклад, можливість керувати кешем, відправляти повідомлення та запускати попередньо завантажені скрипти. Основне призначення Service Worker для прогресивного веб-застосунку включає такі аспекти [17]:

Офлайн режим: Service Worker дозволяє кешувати ресурси, такі як HTML, CSS, JavaScript та зображення, для використання в офлайн-режимі. Коли користувач втрачає з'єднання з Інтернетом, веб-застосунок може використовувати ресурси з кешу, щоб продовжувати функціонування.

Покращення продуктивності: Service Worker може передбачити запити користувача та завантажити відповідні ресурси в кеш, що дозволяє прискорити завантаження сторінок та покращити продуктивність додатка.

Сповіщення та попередження: Service Worker може використовувати Push API для відправлення сповіщень користувачам, навіть коли веб-додаток не відкритий в браузері.

Актуалізація веб-застосунків: Service Worker дозволяє автоматично оновлювати кешовані ресурси, що дозволяє веб-застосунку завжди використовувати останні версії файлів.

Кешування запитів до сервера: Service Worker може кешувати запити до сервера та використовувати їх в разі втрати з'єднання з Інтернетом або для покращення продуктивності.

В застосунку service worker реалізований за допомогою налаштувань vite.config файлу із використанням бібліотеки Vite Plugin PWA та використанням функцій бібліотеки Workbox. А саме використовуються такі функції як skipWaiting, clientsClaim, precacheAndRoute, cleanupOutdatedCaches, registerRoute.

Метод skipWaiting дозволяє пропустити фазу очікування і перейти безпосередньо до фази активації. Коли сервіс-воркер отримує нову версію, він залишається в стані очікування, поки всі старі вкладки не будуть закриті. Це може призвести до затримок у впровадженні оновлення. Виклик skipWaiting дозволяє примусити сервіс-воркер перейти до нової версії, незалежно від наявних вкладок.

Метод clientsClaim дозволяє новому сервіс-воркеру зайняти керування всіма вкладками та сторінками безпосередньо після активації. За замовчуванням, коли новий сервіс-воркер активується, він не керує вкладками, що відкриті до цього моменту, і активний лише для нових вкладок. Виклик clientsClaim дозволяє новому сервіс-воркеру зайняти керування усіма вкладками одразу.

Основне призначення функції precacheAndRoute() полягає у попередньому кешуванні (пре-кешуванні) статичних ресурсів та маршрутизації мережевих запитів. Функція precacheAndRoute дозволяє вказати список статичних файлів (наприклад, HTML, CSS, JavaScript, зображення тощо), які потрібно кешувати в сервіс-воркері під час його установки. Після активації сервіс-воркера, ці статичні ресурси будуть доступні в кеші, що дозволить швидше завантаження сторінок та ресурсів у веб-додатку навіть при відсутності з'єднання з Інтернетом.

Основне призначення функції cleanupOutdatedCaches() полягає в очищенні застарілих кешів, що дозволяє підтримувати чистоту та актуальність кешів у веб-додатку. Функція cleanupOutdatedCaches дозволяє автоматично видаляти застарілі кеші з метою звільнення місця та підтримки актуальності кешованих ресурсів. Застосування цієї функції рекомендується після кешування нових версій статичних ресурсів або при встановленні нового сервіс-воркера, щоб позбутися застарілих кешів, які більше не використовуються.

Основне призначення функції registerRoute полягає в реєстрації кастомних стратегій обробки запитів і встановленні правил маршрутизації для мережевих запитів. Функція registerRoute дозволяє визначити кастомні стратегії обробки запитів, наприклад, кешування запитів, стратегію мережевого звернення, підміну вмісту запиту та інші.

Наприклад для кешування статичних файлів реєструється та використовується стратегія кешування “під час встановлення” застосунку.

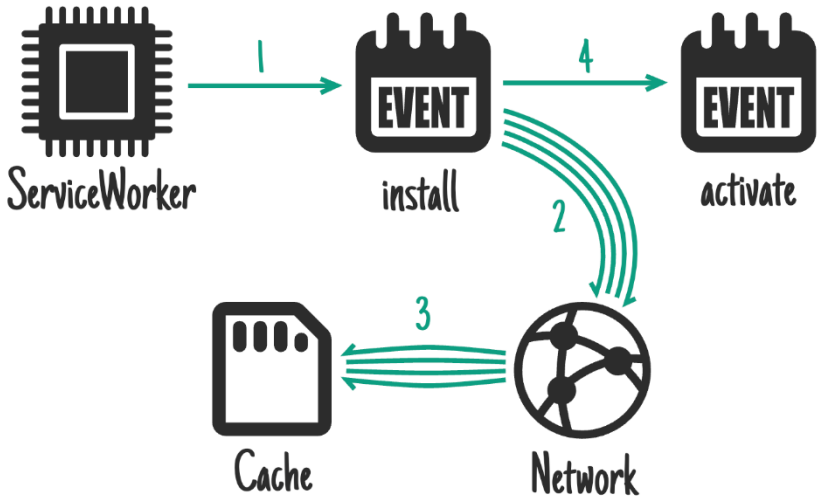


Рисунок 3.7 – Схема стратегії кешування “під час встановлення”

Для мережевих файлів за допомогою функції registerRoute, реєструються стратегі кешування “Спочатку Кеш” (Cache First[21]). Для них же задається додатково максимальна кількість кешованих елементів та час зберігання. Така стратегія забезпечує швидше відображення медіа ресурсу, завдяки тому що дані можуть бути отримані із локального кешу замість мережевого запиту.

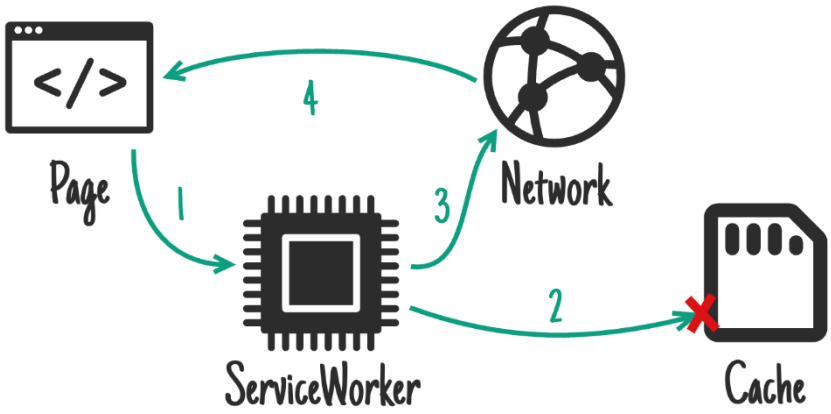


Рисунок 3.8 – Схема стратегії кешування “Спочатку кеш”

Додатково реалізована стратегія кешування “Спочатку мережа” для збереження результату арі виклику. Ця стратегія полягає у виконанні запиту до мережі та збереженні результату до кешу, та використанні збережених даних якщо мережевий виклик не відбувся. Дана функція надає застосунку можливість повноцінно функціонувати за відсутності мережі інтернет.

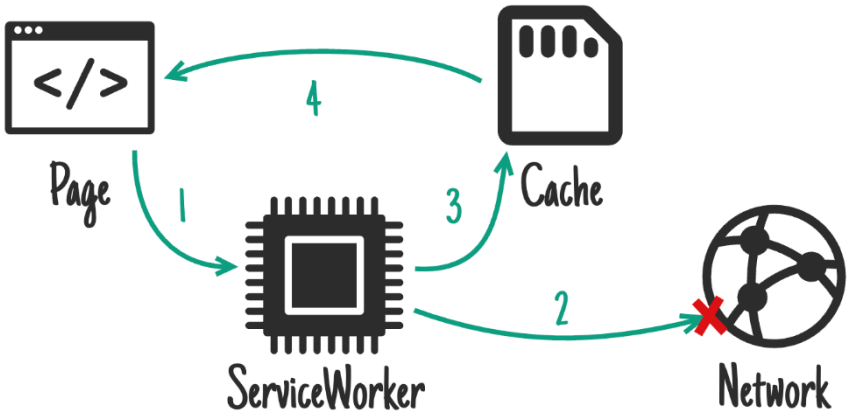


Рисунок 3.9 – Схема стратегії кешування “Спочатку мережа”

Завдяки даним функціям та налаштуванням прогресивний веб-додаток отримує можливість виконуватись не лише у веб-переглядачах, але і на пристроях як нативний застосунок, також в режимі онлайн та офлайн, та ефективно працювати із мережевими та локальними ресурсами.

3.5 Розробка вікна налаштування робочого простору

Вікно налаштування робочого простору це компонент який взаємодіє з API, відображає списки медіа файлів з яких користувач має можливість налаштувати фонове відео та аудіо. Для його функціонування та взаємодії із іншими компонентами, розроблено декілька глобальних сховищ додатку: WorkSpaceStore, WorkSpaceVideoStore, WorkSpaceAudioStore та WorkSpaveLocalStore. Дані сховища розроблені із використанням бібліотеки для контролю стану додатку Zustand.

WorkSpaceStore контролює стан відображення самого вікна та те які дані використовуються, мережеві чи локальні.

WorkSpaceVideoStore містить список відео ресурсів, отриманих з API або кеш сховища та саму функцію для отримання цих даних, і також після надходження даних виконує функції для перевірки чи є відео із списку закешованим, для відображення в інтерфейсі користувача. Унікальний ідентифікатор активної групи відео та активного відео, та методи для їх зміни. Методи для збереження та видалення відео в кешу за ідентифікатором. За схожою моделлю побудоване і сховище WorkSpaceAudioStore.

WorkSpaveLocalStore містить дані про локальні медіа ресурси, добавлені до застосунку. Реалізовує метод для отримання даних із відповідних кеш сховиж, який викликається із компонента при ініціалізації. Також містить функції для добавлення нових медіа файлів. ЇЇ особливість ще в тому що під час однієї сесії силка за файл отримується із виклику функції URL.createObjectUrl і самий файл зберігається в кеш сховищі, та після перезавантаження додатку даний файл вже доступний по посиланню із кешу. Під час завантаження будь якого файлу до веб-додатку, він зберігається в браузерній пам'яті, тому важливо використовувати метод URL.createObjectUrl для отримання посилання на нього для відображення у video, audio та img тегах. Інший метод який реалізований в цьому сховищі даних, це функція для видалення певного файлу за ідентифікатором. Вона виконує

видалення об'єкту даних із самого сховища, очищення даних файлу із браузерної пам'яті за допомогою метода URL.revokeObjectUrl, та видаляє дані про цей файл із кеш сховища.

Ще один тип сховища який використовується в додатку - це localStorage[22]. З додатку взаємоді із ним реалізована з використанням “middleware” функції, із бібліотеки Zustand – persist. Вона першим параметром отримує саме сховище даних застосунку, та другим параметром об'єкт із “ключем” назвою поля в localStorage, та опціонально поля які необхідно зберегти. Даний функціонал дозволяє зберегти налаштування застосунку користувачем, та використовувати їх при повторному відвіданні веб-додатку. Наприклад для вікна налаштування робочого простору це - ідентифікатори активних відео та аудіо файлів.

Додатково для оптимізації деякі операції із кешем, наприклад такі як видалення файлу із кешу, реалізовані в service worker, який виконується в паралельному потоці подій. Взаємодія компонентів із service worker відбувається за допомогою повідомлень. В компоненті викликається метод navigator.serviceWorker.controller.postMessage який генерує повідомлення і як параметри може отримувати будь які інші дані необхідні для виконання у service worker. Він в свою чергу реалізовую метод self.addEventListener із параметром ‘message’, який ловить згенероване повідомлення і виконує необхідну логіку.

Компонент налаштувань робочого простору складається із деількох частин. Це такі компоненти як ControlPanel, DataSourceTabMenu, PlayListController та AudioController. Компонент DataSourceTabMenu реалізовує перемикання із мережевих файлів на локальні. В компоненті PlayListController реалізована логіка по відображенню компонентів для взаємодії із API даними або для локальних файлів.

Для відображення даних з API та надання інтерактивності, реалізовано компоненти VideoPlayListController та AudioPlayListController. В компоненті VideoPlayListController, відбувається відображення груп відео та списку відео для

кожної групи. На етапі ініціалізації компоненту, за допомогою методу бібліотеки React – useEffect із другим параметром пустих залежностей (“[]”), викликається метод getVideoListFromApi, логіка якого, при наявності підключення до мережі інтернет, здійснити запит до REST API, або ж запитати дані з кеш-сховища. Для інтерактивності, реалізовані методи для вибору активної групи відео та самого відео. Додатково, в залежності від того чи є відео файл закешованим, відображається інтерфейс і відповідно методи, для збереження файлу або видалення його з кешу. Схожим чином реалізований і компонент для відображення списку аудіо файлів - AudioPlayListController.

Компоненті AudioController підписаний на сховище даний, використовуючи метод useWorkSpaceAudioStore, і отримує з нього список аудіо файлів, ідентифікатор активної групи та самого аудіо файлу, та реалізовую audio тег, для відтворення самого фонового аудіо, та користувацький інтерфейс для зміни гучності, та перемикання на наступний або попередній трек.

При перемиканні на роботу із локальними файлами відображаються компоненти localVideoListController, LocalAudioListController, LocalImageListController. В цих компнонтах для отримання списку добавлених до застосунку файлів із кешу, викликаються відповідні методи, наприклад getVideoListFromCache, які виконуються при ініціалізації компонента, використовуючи метод useEffect. Окрім функціоналу відображення списку локальних файлі, для добавлення нового файлу до додатку, реалізована форма, із input із типом file, input із типом text, та кнопка для збереження даних. Для реалізації функціоналу форми використовуються методи із бібліотеки React – useRef, для отримання даних полів форми, та useState, для валідації форми.

При ініціалізації застосунку і отримані списків відео файлів, відбувається автоматичний вибір активного відео із першої за списком групи першого за списком відео, якщо таке активне відео не було попередньо вибране. При наявності актичного відео відбувається процес його відтворення в компоненті BG_Video, він містить компонент обгортку, BG_VideoWrapper, функціонал якого

надавати об'єкт активного відео до компоненту реалізації. BG_VideoWrapper компонент підписаний на глобальне сховище, і в залежності від того яке джерело даних використовується, надає компоненту BG_Video необхідний об'єкт даних. Так як компонент BG_VideoWrapper отримує всі списки мережевих відео файлів та локальних відео файлів, але оновлюватись має лише при зміні isLocal та ідентифікатора активного відео, реалізований метод із бібліотеки React – useMemo із масивом відповідних залежностей. Компонент BG_Video віддає video тег, для відображення відео файла, та використовує кастомний “хук” useVideoController. Призначення хука useVideoController, автоматичне відтворення відео файлу при ініціалізації компонента, та при зміні активного відео, завантажувати новий відео файл та автоматично розпочати його відтворення. Дана логіка реалізована за допомогою методів useRef та useEffect.

3.6 Висновки до розділу 3

У третьому розділі було обґрунтовано вибір мов програмування JavaScript та TypeScript: JavaScript є основною мовою програмування для розробки веб-додатків завдяки своїй гнучкості та широкій підтримці браузерами. TypeScript, як надбудова над JavaScript, забезпечує статичну типізацію, що дозволяє зменшити кількість помилок та покращити якість коду.

Бібліотеки для побудови інтерфейсів користувача React: React був обраний завдяки своїй компонентній архітектурі, яка сприяє легкому повторному використанню коду та швидкій розробці. React також забезпечує високу продуктивність та ефективність завдяки віртуальному DOM.

Бібліотеки для стилізації Tailwind CSS: Tailwind CSS було обрано за його утилітарний підхід до стилізації, що дозволяє швидко створювати адаптивні та привабливі інтерфейси без необхідності писати багато кастомного CSS.

Інструменту для збірки Vite: Vite був обраний як інструмент для збірки завдяки своїй високій продуктивності та швидкому часу збірки. Він забезпечує миттєвий перегляд змін під час розробки, що значно прискорює робочий процес.

Менеджеру стану Zustand: Zustand був обраний як простий та легкий у використанні менеджер стану, що дозволяє легко керувати станом додатка без зайвого boilerplate-коду.

Для виконання HTTP-запитів Fetch API: Вибір Fetch API зумовлений його простотою та нативною підтримкою в браузерах. Fetch API забезпечує зручний спосіб роботи з HTTP-запитами та є достатньо гнучким для потреб проекту.

Налаштування прогресивного веб-додатку vite-pwa-plugin та Workbox: Ці інструменти були обрані для налаштування прогресивного веб-додатку завдяки їхній здатності легко інтегруватися з Vite та забезпечувати потужні можливості кешування і офлайн-режиму.

Бази даних MongoDB: MongoDB був обраний завдяки своїй гнучкій документ-орієнтованій моделі даних, що дозволяє легко масштабувати та адаптувати структуру даних до змін у проекті.

Для побудови сервера REST API: Express.js був обраний як веб-фреймворк для серверної частини додатку завдяки своїй простоті, гнучкості та широкій підтримці в екосистемі Node.js.

Розроблено модулі для ініціалізації та запиту даних. Побудований модуль ServiceWorker та функціонал кешування даних. Розроблений модуль для відобращення даних, взаємодії із користувачем та функціональність використання мережевих та локальних даних.

Загалом, вибрані технології забезпечують високу продуктивність, гнучкість та простоту розробки. Вони дозволяють створити прогресивний веб-додаток, який може працювати в офлайн-режимі, пропонує кастомізацію користувацького досвіду та забезпечує зберігання даних.

4 ТЕСТУВАННЯ ПРОГРЕСИВНОГО ВЕБЗАСТОСУНКУ З СИСТЕМОЮ УПРАВЛІННЯ ЛОКАЛЬНИМИ ТА МЕРЕЖЕВИМИ ФАЙЛАМИ

4.1 Особливості тестування

Тестування вебзастосунків - це процес перевірки та оцінки якості, функціональності, безпеки, продуктивності та користувацького досвіду веб-застосунку. Воно є невід'ємною частиною розробки програмного забезпечення, спрямованою на виявлення помилок, недоліків та невідповідностей, щоб забезпечити надійність та задоволення потреб кінцевих користувачів.

Розглянемо основні типи тестування вебзастосунків.

Функціональне тестування перевіряє, чи застосунок виконує свої функції відповідно до вимог. Включає тестування всіх функціональних аспектів, таких як форми, бази даних, внутрішні API, інтерфейси користувача та роботу різних модулів разом.

Тестування продуктивності оцінює швидкість, масштабованість та стабільність веб-застосунків під різними навантаженнями. Включає тестування часу завантаження сторінок, відгуку системи на дії користувача та стійкість до високих навантажень.

Безпекове тестування перевіряє, чи захищений додаток від різних видів атак, таких як SQL-ін'єкції, XSS (міжсайтові скрипти), CSRF (міжсайтові підробки запитів) та інших уразливостей. Забезпечує конфіденційність, цілісність та доступність даних.

Крос-браузерне тестування гарантує, що додаток коректно працює у різних браузерах (Chrome, Firefox, Safari, Edge тощо) та на різних платформах (Windows, macOS, Linux, iOS, Android).

Тестування юзабіліті оцінює зручність використання та користувацький досвід додатка. Включає тестування інтерфейсу, навігації, зручності взаємодії та загального задоволення користувачів.

Регресійне тестування забезпечує, що нові зміни або додавання функціональності не впливають на існуючі функціональні можливості додатка. Це критично важливо при постійному оновленні та вдосконаленні продукту.

Тестування доступності перевіряє, чи додаток доступний для користувачів з обмеженими можливостями. Включає тестування підтримки читачів екранів, навігації клавіатурою та відповідності стандартам доступності (наприклад, WCAG).

Прогресивні веб-застосунки (PWA) поєднують в собі переваги веб-застосунків та нативних мобільних застосунків, що робить їх особливими з точки зору тестування. Тестування PWA включає всі аспекти звичайного веб-тестування, але також вимагає перевірки специфічних функцій та можливостей, які притаманні PWA.

Основні особливості тестування PWA в різних режимах роботи полягають в організації умов тестування офлайн та онлайн режиму.

Тестування офлайн-режиму. PWA повинні працювати навіть при відсутності інтернет-з'єднання. Це вимагає тестування кешування за допомогою Service Worker, перевірки коректної роботи додатку з кешованими даними та поведінки додатку при переході між онлайн та офлайн режимами.

Тестування Service Worker. Service Worker відповідає за кешування, обробку мережевих запитів та управління оновленнями додатку. Важливо тестувати його реєстрацію, активацію та поведінку в різних умовах (наприклад, при оновленні додатку або зміні файлів).

Тестування установки PWA. PWA можна встановлювати як додаток на домашній екран мобільних пристроїв або на робочий стіл комп'ютерів. Це вимагає перевірки коректності встановлення, видалення та оновлення додатку.

4.2 Мануальне тестування

Проведемо мануальне тестування користувацького інтерфейсу. Розпочинаючи із меню бокової панелі, всі вікна мають відкриватись та закриватись натисканням на відповідні іконки, та іконки активних вікон відображаються рамкою.

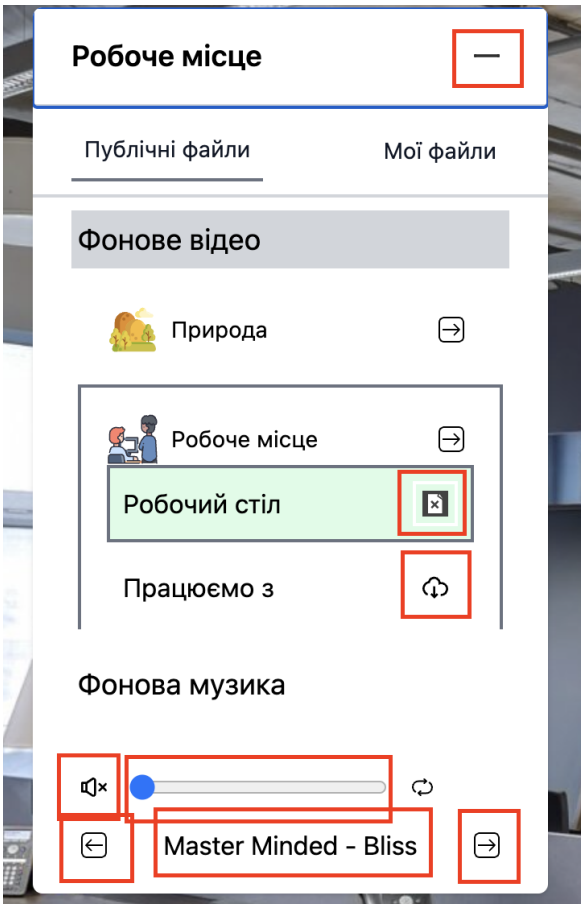


Рисунок 4.1 – Мануальне тестування інтерфейсу користувача “Публічні файли”

У вікні налаштувань робочого місця, відображається коректна інформація та виконується закриття через відповідну іконку. У групах “Фонове відео” та “Фонова музика” відображаються відповідні списки даних, виконується згортання та розгортання при натисканні на певну групу. Натискання на кожний елемент із списку виконує зміну фонового відео та фонового аудіо, вибраний елемент відображається в інтерфейсі рамкою. Для кожного елемента із списку, якщо він не

збережений до кешу, є іконка для збереження, для закешованих, іконка для видалення та самий елемент відрізняється фоновим кольором. В панелі налаштувань фонової музика, іконка із звуком вмикає або вимикає звук, повзунок налаштувань гучності змінює гучність, та іконки перемикання треку змінюють трек на попередній або наступний та змінюється назва треку на коректний.

При перемиканні в таб-меню на “Мої файли”, відображається список збережених локальних файлі, якщо такі є, інакше порожні список, дані локальні медіа-ресурси відтворюються при їх виборі. У формі створення нового елемента, іконка вибору файлу відрізняється коліром якщо файл не добавлений, та форма не зберікається якщо текстове поле не заповнене та файл не добавлений. Після коректного заповнення форми, та її збереження, до списку відповідної групи добавляється новий елемент який при його натисканні відтворюється. Натискання на іконку видалити, видаляє елемент із списку.

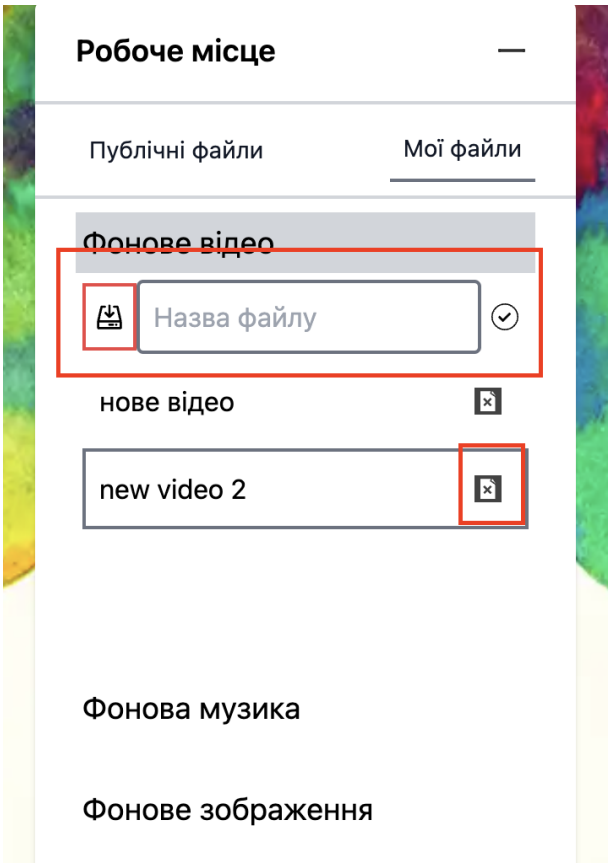


Рисунок 4.2 – Мануальне тестування інтерфейсу користувача “Мої файли”

Проведемо тестування офлайн-режиму. При вимкненні інтернет з'єднання, на екрані з'являється іконка яка про це інформує і застосунок продовжує функціонування. В режимі роботи з мережевими файлами зникають всі не збережені файли і залишаються лише ті що є в кеш-сховищі. Функціонал даних елементів повністю відповідає функціоналу в онлайні. При видаленні збереженого файлу із кешу, відповідний файл зникає із списку так як більше не доступний в офлайн режимі. Список локальних файлів не змінюється та повністю функціонує. При відновленні підключення до мережі інтернет, списки із мережевими файлами автоматично поновлюються, та готові до використання.

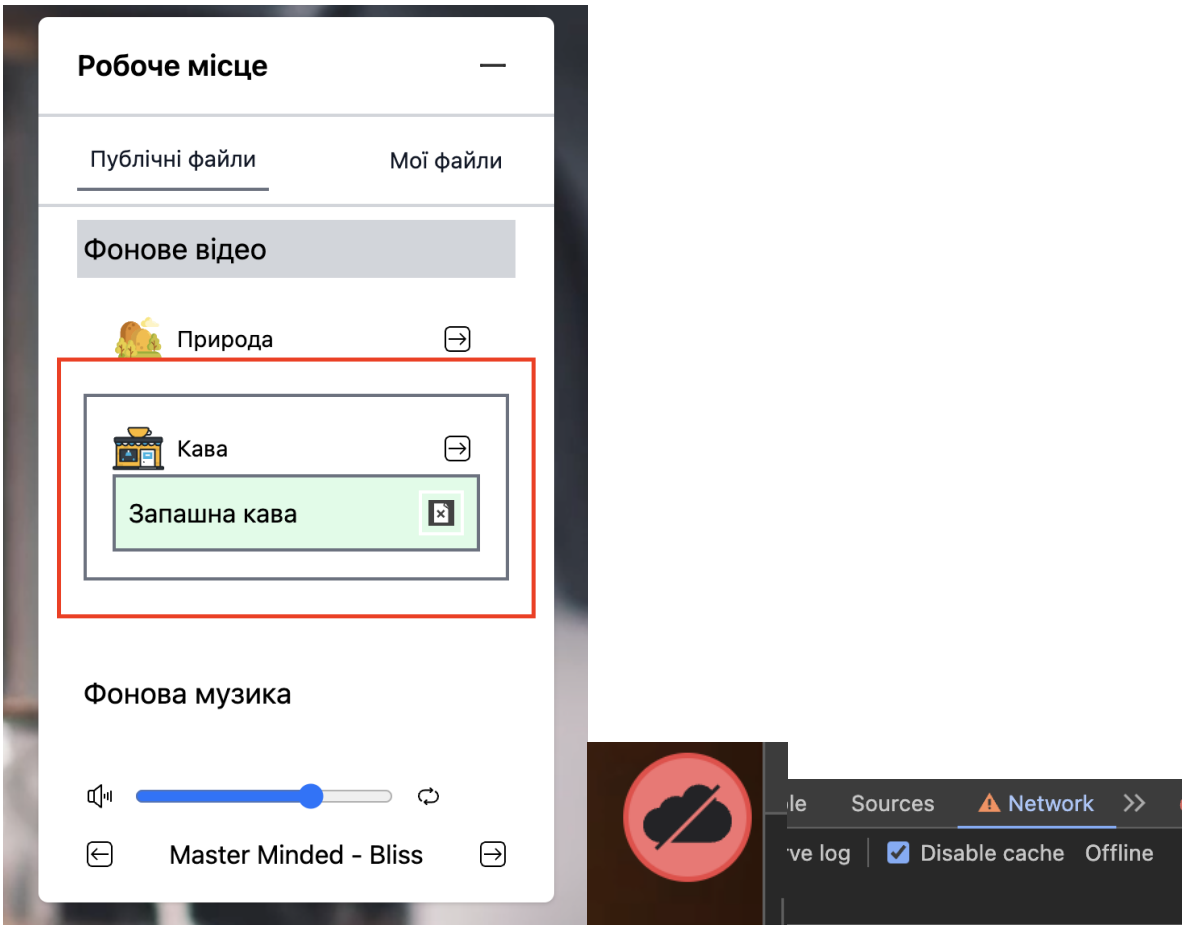


Рисунок 4.3 – Мануальне тестування офлайн режиму

Також так як для роботи офлайн режиму важливе використання локальних сховищ, таких як браузерного кеш сховища та local storage. Тому при очищенні даних, у вкладці “Application”, меді файли мають припинити відтворювення та веб-додаток продовжує функціонування.

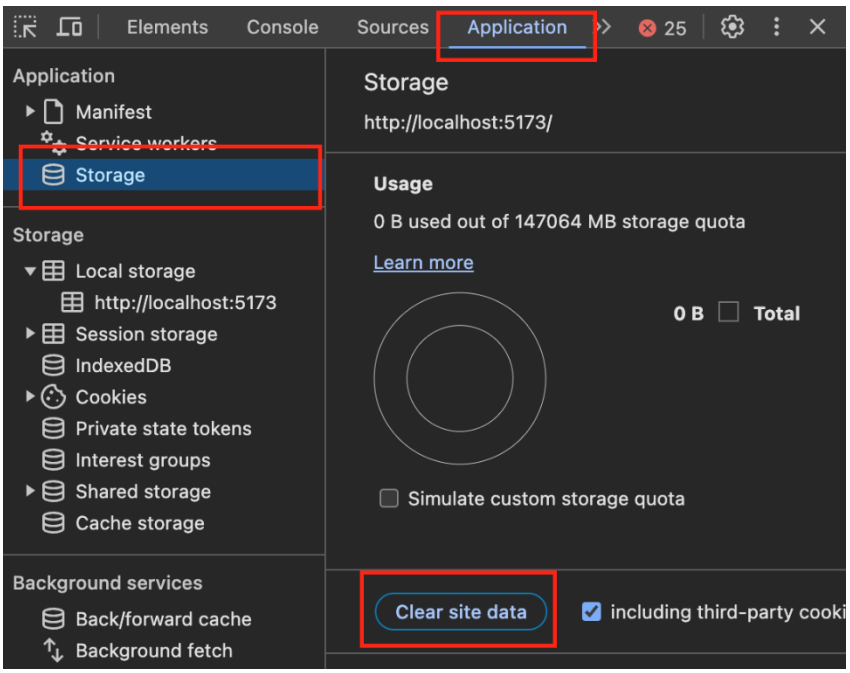


Рисунок 4.4 – Очищення локального сховища даних

Проведемо тестування Service Worker. Разом із завантаженням ресурсів веб-застосунку мають завантажитись і manifest.json та service worker файли. Відкривши в інспекторі вкладку source спостерігаємо їх наявність.

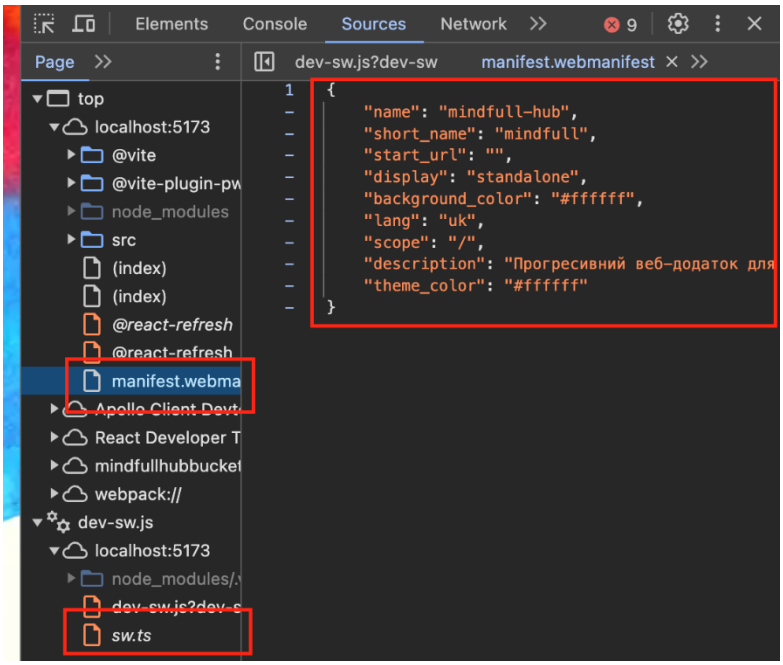


Рисунок 4.5 – Завантаження manifest та service worker файлів

Його виконання спостерігається за прочесом кешування даних, управлінням мережевих запитів, наявністю офлайн режиму та можливістю завантажити додаток.

Для тестування функціоналу PWA, виконуємо завантаження вебзастосунку на десктопний пристрій та на мобільний пристрій, спостерігаємо за виконанням функціоналу додатку, проводимо видалення із пристрою.

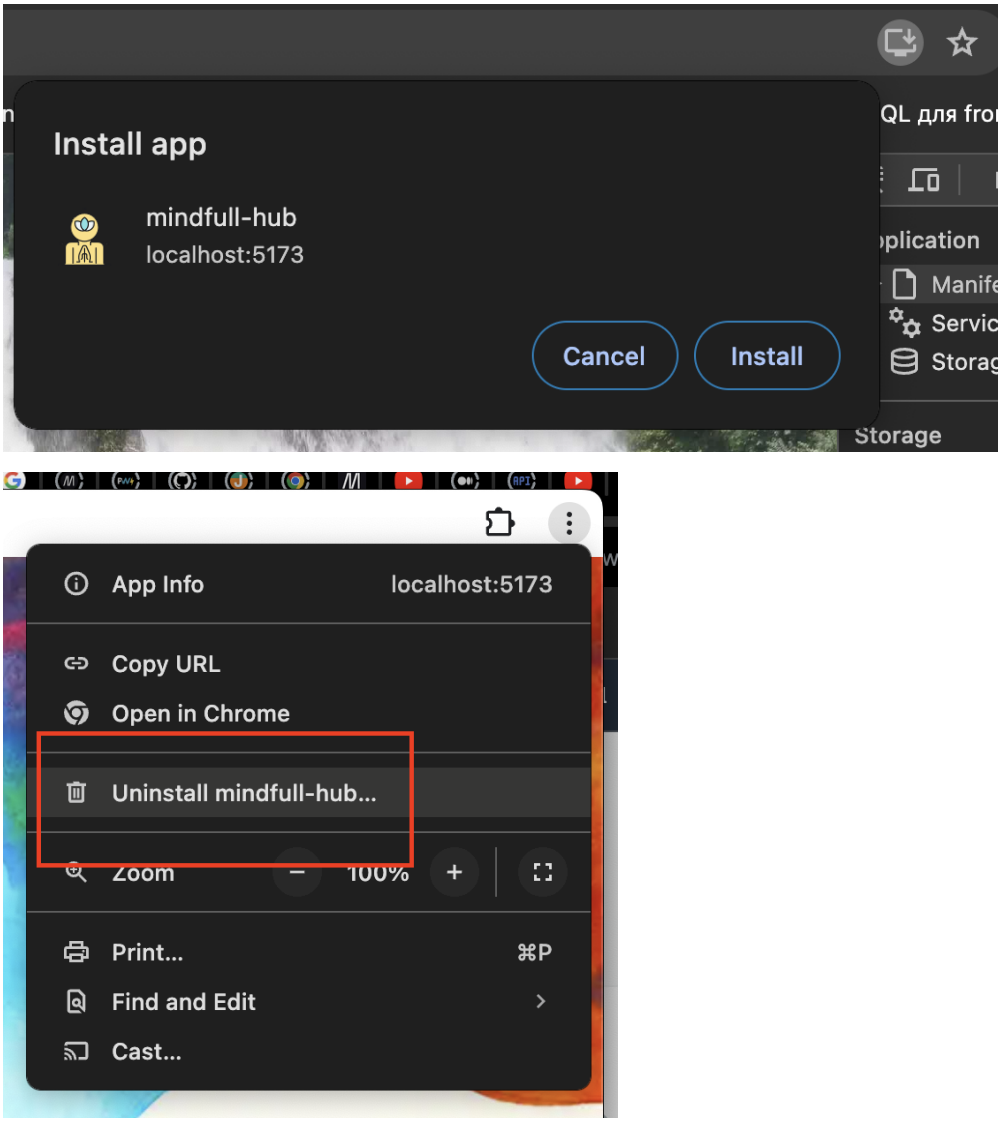


Рисунок 4.6 – Тестування PWA встановлення на пристрій та видалення з пристрою

Тестування прогресивного веб-додатку має свої особливості, зумовлені специфічними можливостями та функціями PWA. Важливо проводити ретельне та

всебічне тестування, яке охоплює офлайн-режим, Service Worker, установку та оновлення додатку, адаптивність, продуктивність, безпеку та юзабіліті. Це дозволяє забезпечити високу якість та надійність PWA, що гарантує позитивний користувацький досвід і задоволення потреб кінцевих користувачів.

Мануальне тестування користувацького інтерфейсу дозволяє перевірити реальний вигляд та функціональність додатку з точки зору кінцевого користувача. Це важливий етап у процесі розробки, який допомагає забезпечити якість та задоволення користувачів. Провівши мануальне тестування всіх функцій та інтерфейсу користувача вебзастосунку, а також основного функціоналу прогресивного вебзастосунку, підтвердив роботу інтерфейсу та коректне відображення інтерфейсу.

4.3 Розгортання та використання вебзастосунку

Розглянемо виконання підготовки клієнтської частини веб-застосунку до збірки.

Завантажити файли проекту із сервісу github, за посиланням https://github.com/SerhiiDeveloper/mindfullhub , використовуючи один із доступних способів: завантажити ZIP файл, або клонувати репозиторій використовуючи HTTPS посилання, SSH посилання або GitHub CLI скрипт. Якщо був вибраний спосіб клонування, після підключення до проекту, на локальному пристрої, необхідно виконати в терміналі команду “git pull origin main”. Так як головна, готова до “production” вітка “main”.

Самий проект складається із клієнтської частини та серверної частини.

Знаходячись в директорії клієнтської частини виконати команду “npm install”, для встановлення необхідних для виконання веб-застосунку бібліотек.

У файлі за адресою “src/const.ts”, вставити нове посилання на API сервер в змінній “BaseURL”.

Для розгортання веб-застосунку використовується сервер, побудований на express, із використанням його методів “express.static”, який надаватиме клієнтські файли користувачеві.

Збірка веб-застосунку та підготовка до розгортання.

В директорії проекту, перейти в папку “server”, та виконати команду “npm install”.

Виконати команду “npm run pre-build", даний виклик виконає збірку клієнтської частини та перемістить зібрані файли застосунку до директорії “public” серверу.

Виконати команду “npm run build”. Яка виконає збірку файлів самого сервера.

Розгортання веб-застосунку на хостингу AWS EC2 [24].

Пройти реєстрацію на AWS EC2, для створення власного віртуального середовища хостингу та ssh ключа.

Завантажити вміст сервера на хостинг, використовуючи команду: "rsync -avz --exclude 'node_modules' --exclude 'build' --exclude '.git' --exclude '.env' \ -e ‘ssh -i path/to/ssh/example-key.pem’ \ . ubuntu@your-host-server-id:~/app".

Використовуючи терміна, підключитись до сервісу за ssh ключем: “ssh -i ‘path/to/ssh/example-key.pem’ ubuntu@your-host-server-id”

Встановити на віртуальну машину актуальну версію node.

Перейти в директорію app, та виконати “npm install”, після чого “npm run build”.

Створити файл “.env” із значенням “MONGODB_URL=path-to-your-mongodb", для підключення сервера до бази даних.

Запустити сервер з використанням systemctl та app.service налаштувань, командою: “sudo systemctl daemon-reload / sudo systemctl enable app.service / sudo systemctl start app.service”.

Налаштувати проксі для приймання http та https запитів з використанням caddy server [25-29].

4.4 Висновки до розділу 4

У четвертому розділі було проведено мануальне тестування прогресивного вебзастосунку, його інтерфейсу користувача, функціональності, виконання кешування даних та основних можливостей управління ресурсами офлайн та онлайн. Під час тестування перевірено правильність відображення інтерфейсу на різних розмірах екранів та у різних браузерах, а також коректність взаємодії з користувачем. Також було підтверджено, що додаток працює стабільно як у режимі онлайн, так і офлайн, завдяки можливості кешування даних. Додаток успішно виконує завдання зі збереженням і відтворенням мережевих та локальних файлів, що забезпечує користувачам зручність та можливість персоналізації робочого середовища. В цілому, результати мануального тестування підтвердили високу якість та надійність прогресивного веб-застосунку, що відповідає його основним цілям та очікуванням користувачів.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмний застосунок, який працює із мережевими та локальними файлами, на прикладі прогресивного веб-застосунку для концентрації та релаксації.

Задачі розробки виконані повністю, з урахуванням особливостей цільової аудиторії та актуальних технологічних тенденцій використання локальних та мережевих ресурсів.

Обґрунтовано вибір теми дослідження, розкрито актуальність проблеми стресу та необхідність наявності засобів для його подолання в сучасному світі. Запропоновано концепцію веб-застосунку як інструменту для концентрації та боротьби зі стресом, враховуючи можливості роботи з файлами онлайн та офлайн, а також персоналізовані налаштування.

У меті та завданнях дослідження було сформульовано основні цілі та завдання проекту, спрямовані на створення ефективного та зручного інструменту для користувачів.

Запропонований метод управління локальними та мережевими файлами та виконано його практичну реалізацію на прикладі вебзастосунку управління стресом.

Практичне значення розробки полягає у можливості використання веб-застосунку для підвищення продуктивності та покращення емоційного стану користувачів.

У процесі розробки було використано сучасні технології та інструменти, такі як JavaScript та TypeScript для програмування, фреймворк React для створення інтерфейсу, а також бібліотеку стилізації Tailwind для оформлення. Zustand як менеджер стану застосункуу. ВикористаноVite як інструмент збірки. Vite-PWA-Plugin та Workbox для розробки service worker та налаштувань PWA. Вибрано MongoDB для бази даних, та NodeJS і ExpressJS для створення REAST API сервера, та також AWS S3 як хмарне сховище для мережевих файлів.

Проведено порівняльний аналіз вибраних технологій з альтернативними варіантами, що дозволило обґрунтувати їх вибір.

Мануальне тестування прогресивного веб-застосунку підтвердило його високу якість та надійність, а також відповідність функціональності та інтерфейсу очікуванням користувачів.

Розроблено інструкцію користувача для клонування проекту, його розгортання, збірки та хостингу.

Отже, в бакалаврській роботі успішно реалізовані поставлені цілі та завдання, розкрито актуальність теми, виконані всі завдання, досліджені режими роботи з локальними та мережевими файлами на прикладі розробки вебзастосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Управління локальними та мережевими ресурсами. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

2. Білоус С. А., Коваленко О. О. Аналіз сучасних програмних застосунків для роботи з файлами онлайн та офлайн на прикладі поступового вебзастосунку для концентрації. Інформаційне суспільство.№ 90.Тернопіль 2024.

3. REST API URL: https://www.ibm.com/topics/rest-apis

4. LifeAt URL: https://lifeat.io/en-us/category/getting-started-1yc2at6/

5. Headspace URL: https://www.headspace.com/about-us?origin=navigation

6. Noisli URL: https://www.noisli.com/how-it-works

7. Drag&Drop URL: https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API

8. React URL: https://uk.react.dev/learn

9. TypeScript URL:https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

10. Zustand URL: https://docs.pmnd.rs/zustand/getting-started/introduction

11. JavaScript URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript

12. Tailwind URL: CSS https://tailwindcss.com/docs/utility-first

13. Vite URL: https://vitejs.dev/guide/

14. Vite-PWA-Plugin URL: https://vite-pwa-org.netlify.app/guide/

15. Fetch API URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

16. PWA URL: https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/

17. Service Worker URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

18. Workbox URL: https://developer.chrome.com/docs/workbox/what-is-workbox

19. MongoDB URL: https://www.mongodb.com/docs/guides/

20. AWS S3 URL: https://aws.amazon.com/s3/faqs/?nc1=h_ls

21. NodeJS URL: https://nodejs.org/en/about

22. ExpressJS URL: https://expressjs.com/en/resources/glossary.html

23. Manifest.json URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json

24. Middlewar URL: https://expressjs.com/en/guide/writing-middleware.html

25. AWS EC2 URL: https://aws.amazon.com/ec2/?nc1=h_ls

26. Systemctl URL: https://hyperhost.ua/info/uk/shho-take-systemctl-upravlinnya-sluzbami-v-linux

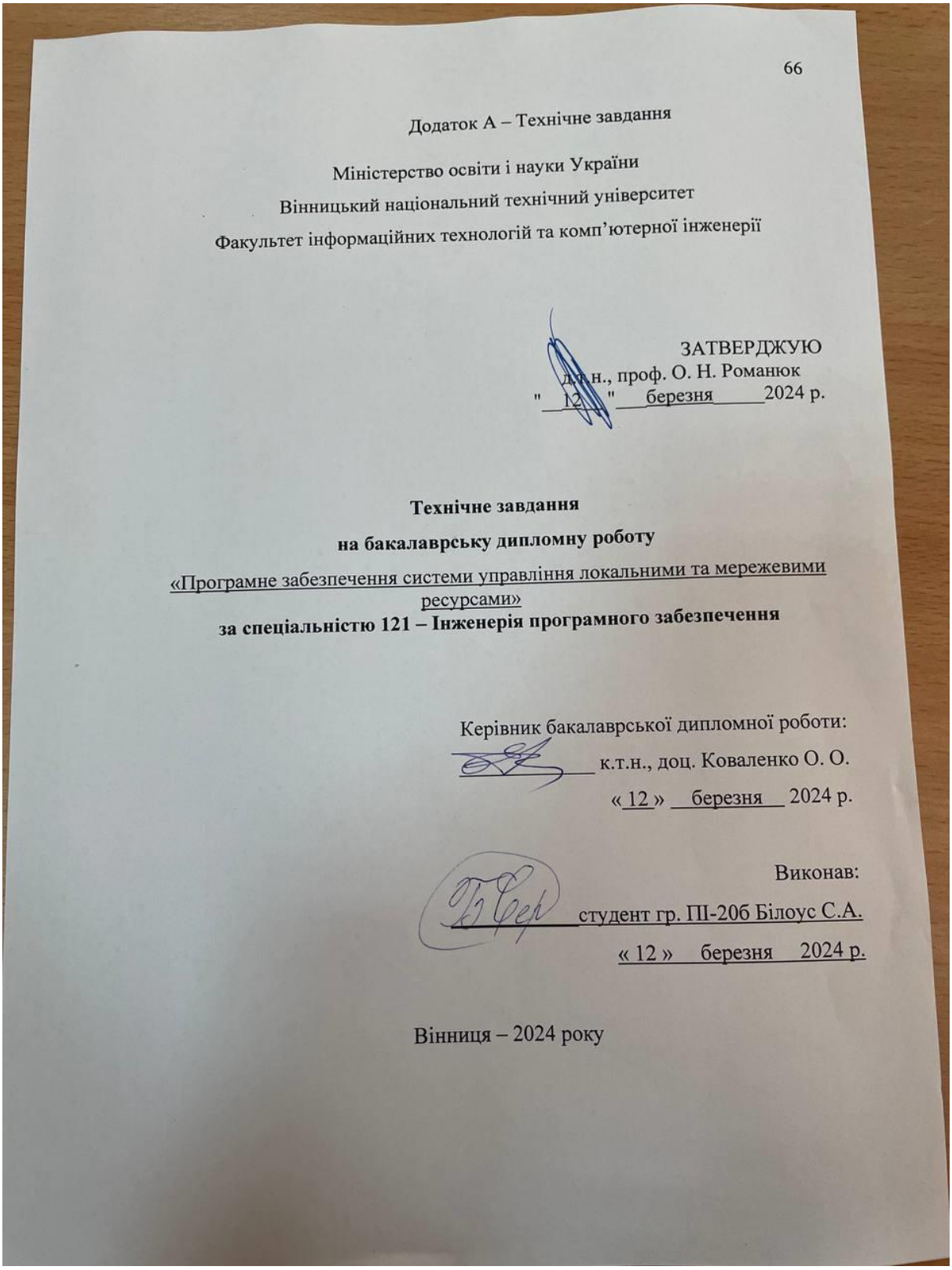
27. Caddy Server URL: https://caddyserver.com/

28. Cache First (The offline cookbook) URL: https://jakearchibald.com/2014/offline-cookbook/#network-falling-back-to-cache

29. Local Storage URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

ДОДАТКИ

Додаток А – Технічне завдання



66

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"12" березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу
«Програмне забезпечення системи управління локальними та мережевими ресурсами»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доц. Коваленко О. О.
« 12 » березня 2024 р.

Виконав:
студент гр. ПІ-20б Білоус С.А.
« 12 » березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Програмне забезпечення системи управління локальними та мережевими ресурсами».

Галузь застосування – вебзастосунки, системи використання локальних та мережевих ресурсів у вебзастосунках

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №17 від 20 березня 2024 р.

3. Мета та призначення розробки.

Метою цього дослідження є розробка програмного забезпечення системи управління локальними та мережевими ресурсами на прикладі прогресивного веб-застосунку, спрямованого на підвищення концентрації та зниження рівня стресу у користувачів за допомогою ефективної роботи з файлами онлайн та офлайн.

Розроблений в рамках дослідження прогресивний вебзастосунок може бути використаний в режимах онлайн та офлайн для зниження рівня стресу.

4. Вихідні дані для проведення НДР

1. Управління локальними та мережевими ресурсами. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
2. Local Storage URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
3. LifeAt URL: https://lifeat.io/en-us/category/getting-started-1yc2at6/
4. Headspace URL: https://www.headspace.com/about-us?origin=navigation
5. Noisli URL: https://www.noisli.com/how-it-works

5. Технічні вимоги

Серверний комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 2 ГБ, розмір жорсткого диску 2 ГБ. Операційна система Windows 7/8/10/11.

З клієнтського боку: ОС Windows та будь-який браузер.

6. Конструктивні вимоги.

Система має відповідати функціональним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до бакалаврської дипломної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку систем для роботи з локальними та мережевими ресурсами	14.03.2024 – 31.03.2024
2	Розробка методу, моделі та алгоритмів роботи вебсистеми	01.04.2024– 15.04.2024
3	Розробка модулів вебсервісу	16.04.2024 – 18.05.2024
4	Тестування програми	19.05.2024 – 24.05.2024
5	Оформлення матеріалів до захисту БДР	25.05.2024 – 30.05.2024

10. Порядок контролю та прийняття

Всі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту. Дозволяється корегування бакалаврської дипломної роботи.

Додаток Б – Протокол перевірки бакалаврської дипломної робобти на наявність текстових запозичень

69

Додаток Б – Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: « Програмне забезпечення системи управління локальними та мережевими ресурсами»

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Коваленко О.О.

Оригінальність	94%
Схожість	2,8%

Аналіз звіту подібності

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

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek

Автор роботи _____ Білоус С.А.

Керівник роботи _____ Коваленко О.О.

Додаток В - Лістинг програми

Sw.ts

```
import { precacheAndRoute, cleanupOutdatedCaches, createHandlerBoundToURL } from 'workbox-precaching';
import { clientsClaim } from 'workbox-core'
import { NavigationRoute, registerRoute } from 'workbox-routing'
import { CacheFirst } from "workbox-strategies"
import { ExpirationPlugin } from "workbox-expiration"
import { CacheableResponsePlugin } from "workbox-cacheable-response"



declare let self: ServiceWorkerGlobalScope
declare global {
interface ServiceWorkerGlobalScope {
location: string
}
}


self.addEventListener('message', (event) => {
console.log(event.data)
if (!caches) return;
switch (event.data.action) {
case "deleteFromCache":
caches.open(event.data.cacheName).then(cache => {
cache.delete(event.data.url)
self.clients.matchAll().then(clients => {
clients.forEach(client => {
client.postMessage({
action: "deletedFromCache",
cacheName: event.data.cacheName,
url: event.data.url
})
})
})
})
break;


default:
break;
}
})



self.skipWaiting();
clientsClaim();


precacheAndRoute([...self.__WB_MANIFEST,], {});


cleanupOutdatedCaches();
registerRoute(new NavigationRoute(createHandlerBoundToURL("index.html"), {
allowlist: [/^\/$/]
}));
registerRoute(/\.(?:png|jpg|jpeg|svg|webp)$/, new CacheFirst({
```

```
"cacheName": "app-image-cache",
plugins: [new ExpirationPlugin({
maxEntries: 50,
maxAgeSeconds: 604800
})]
}), 'GET');
registerRoute(/\.(?:mp3)$/, new CacheFirst({
"cacheName": "app-audio-cache",
plugins: [new ExpirationPlugin({
maxEntries: 50,
maxAgeSeconds: 604800
})]
}), 'GET');
registerRoute(
({ url }) =>
url.origin === 'https://mindfullhubbucket.s3.eu-north-1.amazonaws.com' &&
url.pathname.startsWith('/image/'),
new CacheFirst({
cacheName: 'image-api-cache',
plugins: [
new CacheableResponsePlugin({
statuses: [0, 200, 300],
}),
],
})
);
registerRoute(
({ url }) =>
url.origin === 'https://mindfullhubbucket.s3.eu-north-1.amazonaws.com' &&
url.pathname.startsWith('/icon/'),
new CacheFirst({
cacheName: 'icon-api-cache',
plugins: [
new CacheableResponsePlugin({
statuses: [0, 200, 300],
}),
],
})
);


const AudioCacheableResponse = new CacheableResponsePlugin({
statuses: [0, 200, 300]
})
AudioCacheableResponse.cacheWillUpdate = async ({ request, response }) => {
if (response) {
self.clients.matchAll().then(clients => {
clients.forEach(client => {
client.postMessage({
action: "cacheAPIAudioData",
url: request.url
})
})
})
}
return response
}
registerRoute(
({ url }) =>
url.origin === 'https://mindfullhubbucket.s3.eu-north-1.amazonaws.com' &&
```

```
url.pathname.startsWith('/music/'),
new CacheFirst({
cacheName: 'music-api-cache',
plugins: [
AudioCacheableResponse
],
})
);


const VideoCacheableResponse = new CacheableResponsePlugin({
statuses: [0, 200, 300]
})
VideoCacheableResponse.cacheWillUpdate = async ({ request, response }) => {
if (response) {
self.clients.matchAll().then(clients => {
clients.forEach(client => {
client.postMessage({
action: "cacheAPIVideoData",
url: request.url
})
})
})
}
return response
}
registerRoute(
({ url }) =>
url.origin === 'https://mindfullhubbucket.s3.eu-north-1.amazonaws.com' &&
url.pathname.startsWith('/video/'),
new CacheFirst({
cacheName: 'video-api-cache',
plugins: [
VideoCacheableResponse,
],
})
);
```

WorkSpaceVideoStore.ts

```
import { create } from "zustand";
import { persist } from "zustand/middleware";
import { callFetchAndCache } from "./utils/callFetchAndCache.ts";
import { addIsCachedToVideoList } from "./utils/addIsCachedToVideoList.ts"
import { BaseURL } from "../const.ts";



export type BGVideoType = {
src: string;
_id: string;
isCached?: boolean
```

```
title: string
}
export type BGVideoGroupType = {
video: BGVideoType[],
poster: string
_id: string
icon: string
title: string
}


export type BGVideoDataType = BGVideoType & {
poster?: string
}


export type UpdateCachedType = (src: string) => void


interface IWorkSpaceVideoStore {
getVideoListFromApi: () => void
bgVideoList: BGVideoGroupType[]
activeVideoId: string
activeVideoGroupId: string
setNextActiveVideo: (id?: string) => void
setActiveVideoGroupId: (id: string) => void
updateVideoCached: UpdateCachedType
deleteFromCacheById: (id: string) => void
}


export const useWorkSpaceVideoStore = create<IWorkSpaceVideoStore>()(persist((set, get) => ({
getVideoListFromApi: async () => {
let videoList = await callFetchAndCache<BGVideoGroupType[]>(BaseURL + "video-list")
videoList = await addIsCachedToVideoList('video-api-cache', videoList)


const activeVideoId = get().activeVideoId
const activeVideoGroupId = get().activeVideoGroupId
const isExist = videoList.find(group => group?._id === activeVideoGroupId)
if (activeVideoId && activeVideoGroupId && isExist) {
set(() => ({ bgVideoList: videoList }))
return
}


const newActiveVideoGroupId = videoList[0]?._id || ""
const newActiveVideoId = videoList[0]?.video[0]?._id || ""
set(() => ({ bgVideoList: videoList, activeVideoId: newActiveVideoId, activeVideoGroupId: newActiveVideoGroupId }))
},
bgVideoList: [],
activeVideoId: "",
activeVideoGroupId: "",
setNextActiveVideo: (id) => {
if (id === "") {
set(() => ({ activeVideoId: "" }))
}
else if (!id) {
const activeGroup = get().bgVideoList.find(group => group._id === get().activeVideoGroupId)!
const activeVideoIndex = activeGroup.video.findIndex(video => video._id === get().activeVideoId)
const nextVideoIndex = activeVideoIndex === activeGroup.video.length - 1 ? 0 : activeVideoIndex + 1
set(() => ({ activeVideoId: activeGroup.video[nextVideoIndex]._id }))
```

```
} else {
set(() => ({ activeVideoId: id }))
}
},
setActiveVideoGroupId: (id) => {
if (id === "") {
set(() => ({ activeVideoGroupId: "", activeVideoId: "" }))
}
else {
set(() => ({ activeVideoGroupId: id, activeVideoId: get().bgVideoList.find(group => group._id === id)!.video[0]._id }))
}
},
updateVideoCached: async (src) => {
const bgVideoList = get().bgVideoList.map(group => {
const video = group.video.map(item => item.src === src ? { ...item, isCached: true } : item)
return { ...group, video }
})
set(() => ({ bgVideoList }))
},
deleteFromCacheById: (id) => {
let url;
const bgVideoList = get().bgVideoList.map(group => {
const video = group.video.map(item => {
if (item._id === id) {
url = item.src
return { ...item, isCached: false }
}
return item
})
return { ...group, video }
})
if ("serviceWorker" in navigator && url) navigator.serviceWorker.controller?.postMessage({
action: "deleteFromCache",
url,
cacheName: "video-api-cache"
});


set(() => ({ bgVideoList, activeVideoId: ""}))
}
}), {
name: "workSpaceVideoStore",
partialize: (state) => ({activeVideoId: state.activeVideoId, activeVideoGroupId: state.activeVideoGroupId}),
}));
```

CallFetchAndCache.ts

```
import { CacheableResponse } from 'workbox-cacheable-response';


const cacheable = new CacheableResponse({
statuses: [0, 200, 300],
});


type CallFetchAndCacheType = <T>(url: string) => Promise<T>


export const callFetchAndCache: CallFetchAndCacheType = async (url) => {
const request = new Request(url, {
```

```
method: "GET",
headers: {
'Content-Type': 'application/json'
}
})
let listResponse = await fetch(request)
.then(async (response) => {
const isCacheble = cacheable.isResponseCacheable(response)
if (isCacheble && caches) {
const responseClone = response.clone()
caches.open('api-cache').then(async cache => {
const match = await cache.match(responseClone.url)
if (match) {
await cache.delete(match.url)
}
await cache.put(responseClone.url, responseClone);
})
}
return response
})
.then(response => response.json())
.catch(error => {
console.error("Handle error: ", error)
})
if (!listResponse && caches) {
const cache = await caches.open('api-cache')
const match = await cache.match(request.url)
listResponse = !match ? [] : await match.json()
}
return listResponse
}
```

AddIsCachedToVideoList.ts

```
type AddIsCachedToListType = (cacheName: string, list: BGVideoGroupType[]) => Promise<BGVideoGroupType[]>


export const addIsCachedToVideoList: AddIsCachedToListType = async (cacheName, list) => {
if (!list) return [];
if (!caches) return list;
const cache = await caches.open(cacheName);
const isCachedListPromises = list.map(async (group) => {
const isCachedListedPropertyPromises = group.video.map((item) => {
return cache.match(item.src).then(res => res ? { ...item, isCached: true } : { ...item, isCached: false })
})
return { ...group, video: await Promise.all(isCachedListedPropertyPromises) }
})
const isCachedList = await Promise.all(isCachedListPromises)


return isCachedList || list
}
```

CheckStorageEstimate.ts

```
import { toast } from "react-toastify";


export type StorageEstimateType = {
usage: number
quota: number
```

```
}
async function checkStorageEstimate(): Promise<Error | StorageEstimateType> {
if (!('storage' in navigator && 'estimate' in navigator.storage)) {
console.error('Storage API not supported in this browser');
return new Error("Помилка оцінки об'єму пам'яті браузера");
}
try {
const estimate = await navigator.storage.estimate();
const usedBytes = estimate.usage;
const totalBytes = estimate.quota;
if (usedBytes === undefined || totalBytes === undefined) throw new Error('Storage estimate is not available');
console.log(`Used storage: ${(usedBytes / 1024 / 1024).toFixed(2)} MB`);
console.log(`Total available storage: ${(totalBytes / 1024 / 1024).toFixed(2)} MB`);
console.log(`Available storage: ${((totalBytes - usedBytes) / 1024 / 1024).toFixed(2)} MB`);
toast("Використано пам'яті: " + (usedBytes / 1024 / 1024).toFixed(2) + " MB" + "\n" + "Доступно пам'яті: " + ((totalBytes -
usedBytes) / 1024 / 1024).toFixed(2) + " MB");


return {
usage: usedBytes,
quota: totalBytes
}
} catch (error) {
console.error('Error checking storage estimate:', error);
return new Error("Помилка оцінки об'єму пам'яті браузера");
}
}
```

WorkSpace.tsx

```
import { WidgetTypeEnum } from "../../store/activeWidgetStore";
import { WidgetControlPanel } from "../widgetDnD/widgetControlPanel";
import { WidgetDraggableHOC } from "../widgetDnD/widgetDraggableHOC";
import { AudioController } from "./audioController";
import { PlayListController } from "./playListControllers";
import { DataSourceTabMenu } from "./dataSourceTabMenu";
import { useWorkSpaceStore } from "../../store/workSpaceStore";
import { useEffect } from "react";
import { useWorkSpaceVideoStore } from "../../store/workSpaceVideoStore";
import { useWorkSpaceAudioStore } from "../../store/workSpaceAudioStore";
import checkStorageEstimate from "../../store/utils/checkStorageEstimate";


export const WorkSpace = () => {
const setIsWidgetShow = useWorkSpaceStore((state) => state.setIsWidgetShow);
const isWidgetShow = useWorkSpaceStore((state) => state.isWidgetShow);
const updateVideoCached = useWorkSpaceVideoStore(
(state) => state.updateVideoCached
);
const updateAudioCachedById = useWorkSpaceAudioStore(state => state.updateAudioCachedById)
const cachedListener = (event: MessageEvent) => {
checkStorageEstimate();
switch (event.data.action) {
case "cacheAPIVideoData":
updateVideoCached(event.data.url);
break;
case "cacheAPIAudioData":
updateAudioCachedById(event.data.url);
break;
```

```
default:
break;
}
}
useEffect(() => {
navigator?.serviceWorker?.addEventListener("message", cachedListener);
return () => {
navigator?.serviceWorker?.removeEventListener("message", cachedListener);
}
}, [])


return (
<WidgetDraggableHOC
widgetType={WidgetTypeEnum.WORK_SPACE}
className={!isWidgetShow ? "hidden" : ""}
>
<div className={"w-80 rounded-md bg-white shadow max-h-screen overflow-y-auto"}>
<WidgetControlPanel
header="Робоче місце"
handleClick={setIsWidgetShow}
widgetType={WidgetTypeEnum.WORK_SPACE}
/>
<DataSourceTabMenu />
<PlayListController/>
<AudioController />
</div>
</WidgetDraggableHOC>
);
};
```

VideoPlayListController.tsx

```
export const VideoPlayListController = () => {
const bgVideoListState = useWorkSpaceVideoStore((state) => state.bgVideoList);
const isOffline = useWorkSpaceStore((state) => state.isOffline);
const bgVideoList = useMemo(() => {
if (isOffline) return bgVideoListState.map((group) => {
const video = group.video.filter((video) => video.isCached === true);
if (video.length === 0) return;
return { ...group, video };
});
return bgVideoListState;
}, [isOffline, bgVideoListState]);


const activeVideoGroupId = useWorkSpaceVideoStore(
(state) => state.activeVideoGroupId
);
const activeVideoId = useWorkSpaceVideoStore((state) => state.activeVideoId);
const setActiveVideoGroupId = useWorkSpaceVideoStore(
(state) => state.setActiveVideoGroupId
);
const setNextActiveVideo = useWorkSpaceVideoStore(
(state) => state.setNextActiveVideo
);
const getVideoListFromApi = useWorkSpaceVideoStore(
(state) => state.getVideoListFromApi
);
const deleteFromCacheById = useWorkSpaceVideoStore(
```

```
(state) => state.deleteFromCacheById
);


useEffect(() => {
if (!bgVideoList || bgVideoList.length === 0) {
getVideoListFromApi();
}
}, []);


const handleGroupClick = (id: string) => () => {
if (id === activeVideoGroupId) return setActiveVideoGroupId("");
setActiveVideoGroupId(id);
};


const handleNextVideoClick = (id: string) => () => {
if (id !== activeVideoGroupId) {
setActiveVideoGroupId(id);
} else {
setNextActiveVideo();
}
};


const handleVideoClick = (id: string) => () => {
setNextActiveVideo(activeVideoId === id ? "" : id);
};


const handleVideoCacheClick =
(id: string, cached: boolean | undefined) => (event: MouseEvent) => {
if (cached) {
event.stopPropagation();
return deleteFromCacheById(id);
}
};


return bgVideoList.map((videoGroup) => {
if (!videoGroup) return;
return (
<li
key={videoGroup._id}
className={
"p-4 border-2 " +
(activeVideoGroupId === videoGroup._id
? "border-gray-500"
: "border-white")
}
>
<div
className="flex items-center justify-between cursor-pointer"
onClick={handleGroupClick(videoGroup._id)}
>
<div className="flex items-center">
<img
src={videoGroup.icon}
alt={videoGroup.title}
className="w-8 h-8"
/>
```

```
<p className="text-sm ml-2">{videoGroup.title}</p>
</div>
<SvgController
src={HeadRightSVG}
alt="Наступний відео"
onClick={handleNextVideoClick(videoGroup._id)}
/>
</div>
<ul
className={
"transition-transform duration-500 ease-in-out flex flex-col gap-1 " +
(activeVideoGroupId === videoGroup._id
? "scale-y-100 h-auto"
: "scale-y-0 h-0")
}
>
{videoGroup.video.map((video) => (
<li
key={video._id}
className={
"flex flex-row justify-between items-center p-2 border-2 cursor-pointer hover:bg-gray-100 " +
(activeVideoId === video._id
? "border-gray-500 "
: "border-white ") +
(video.isCached === true ? "bg-green-100 " : "bg-white ")
}
onClick={handleVideoClick(video._id)}
>
<p>{video.title}</p>
<SvgController
src={video.isCached === true ? DeleteSVG : DownloadCloud}
alt={
video.isCached === true
? "Видалити з кешу"
: "Завантажити в кеш"
}
onClick={handleVideoCacheClick(
video._id,
video.isCached
)}
/>
</li>
))}
</ul>
</li>
);
});
};
```

BG_Video.tsx

```
type BG_VideoPropsType = {
videoData: BGVideoDataType
}


export const BG_Video: FC<BG_VideoPropsType> = ({videoData}) => {
const memoVideoData = useMemo(() => videoData, [videoData._id, videoData.poster]);
const { videoRef, handleVideoLoaded } = useVideoController(memoVideoData);
```

```
return (
<div className="flex items-center justify-center h-full w-full z-0 absolute">
<video
ref={videoRef}
onLoadedData={handleVideoLoaded}
loop
muted
id={memoVideoData._id}
className="h-full w-full object-cover"
poster={memoVideoData.poster || DefaultPoster}
>
<source src={memoVideoData.src} type="video/mp4"/>
</video>
</div>
);
};
```

UseVideoController.ts

```
export const useVideoController = (videoData: BGVideoDataType, deleteFromCache: (id: string) => void) => {
const isMounted = useRef(false);
const videoRef = useRef<HTMLVideoElement>(null);
const handleUserInteraction = useRef<() => void>(null!);


useEffect(() => {
handleUserInteraction.current = () => {
if (!videoRef.current) return;
videoRef.current.play().catch((error) => {
console.error(error)
deleteFromCache(videoData._id)
videoRef.current?.pause()
});
document.removeEventListener("click", handleUserInteraction.current);
};


return () => {
document.removeEventListener("click", handleUserInteraction.current);
};
}, []);


useEffect(() => {
if (!videoRef.current) return;
if (!isMounted.current) {
videoRef.current.load();
document.addEventListener("click", handleUserInteraction.current);
isMounted.current = true;
return;
}
videoRef.current.src = videoData.src;
videoRef.current.load();
setTimeout(() => videoRef.current?.play().catch((error) => {
console.error(error)
deleteFromCache(videoData._id)
videoRef.current?.pause()
}), 1500);
}, [videoData])
```

```
const handleVideoLoaded = () => {
console.log("handleVideoLoaded")
if (isMounted.current) return;
document.addEventListener("click", handleUserInteraction.current);
isMounted.current = true;
};


return {
videoRef,
handleVideoLoaded
}
}
```

LocalVideoListController.tsx

```
export const LocalVideoListController = () => {
const videoList = useWorkSpaceLocalStore((state) => state.videoList);
const activeVideoId = useWorkSpaceLocalStore((state) => state.activeVideoId);
const addVideo = useWorkSpaceLocalStore((state) => state.addVideo);
const getVideoListFromCache = useWorkSpaceLocalStore(state => state.getVideoListFromCache)
const setActiveVideoId = useWorkSpaceLocalStore(state => state.setActiveVideoId)
const deleteVideoById = useWorkSpaceLocalStore(state => state.deleteVideoById)


useEffect(() => {
getVideoListFromCache();
}, [])


return (
<ul className="transition-transform duration-500 ease-in-out flex flex-col gap-1 ">
<AddLocalFilesForm saveFileCallback={addVideo} inputAccept="video/mp4"/>
{videoList.map((video) => (
<LocalListItem key={video._id} {...video} activeId={activeVideoId} clickHandler={setActiveVideoId}
deleteHandler={deleteVideoById}/>
))}
</ul>
);
};
```

AddLocalFileForm.tsx

```
type AddLocalFilesFormPropsType = {
saveFileCallback: (file: File, title: string) => void
inputAccept: string
}


export const AddLocalFilesForm: FC<AddLocalFilesFormPropsType> = ({saveFileCallback, inputAccept}) => {
const titleInputRef = useRef<HTMLInputElement>(null);
const fileInputRef = useRef<HTMLInputElement>(null);
const [file, setFile] = useState<File | null>(null);
const handleSubmit = () => (event: FormEvent<HTMLFormElement>) => {
event.preventDefault();
const title = titleInputRef.current?.value;
if (!title || !file) return;
saveFileCallback(file, title);
setFile(null);
if (titleInputRef.current?.value) titleInputRef.current.value = ""
```

```
if (fileInputRef.current) {
titleInputRef.current.files = null
titleInputRef.current.value = ""
}
};
const handleSelectFile = (event: ChangeEvent<HTMLInputElement>) => {
const file = event.target.files?.[0];
if (file) setFile(file);
}


return <form
onSubmit={handleSubmit()}
className="flex flex-row items-center justify-start"
>
<label
htmlFor={"upload_file" + inputAccept}
className={"border-2 py-1 " + (file ? "border-gray-500" : "border-red-500")}
>
<SvgController src={UploadSVG} alt="Добавити файл" />
</label>
<input
accept={inputAccept}
ref={fileInputRef}
type="file"
className="hidden"
id={"upload_file" + inputAccept}
name={"upload_file" + inputAccept}
onChange={handleSelectFile}
/>
<input
ref={titleInputRef}
placeholder="Назва файлу"
required
type="text"
className="w-full bg-white border-2 border-gray-500 rounded p-2"
/>
<button type="submit" className="border-0 bg-white">
<SvgController src={SaveSVG} alt="Зберегти" />
</button>
</form>
}
```

SaveLocalFileToCache.ts

```
export const saveLocalFileToCache = async (file: File, cacheName: string, _id: string, title: string) => {
if (caches) {
const cache = await caches.open(cacheName)
const reader = new FileReader();
reader.onload = (e) => {
const uploadedFile = new Blob([e.target!.result!], { type: file.type });
const headers = new Headers({ "Content-Type": file.type, "_id": _id, "title": btoa(unescape(encodeURIComponent(title)))});
const response = new Response(uploadedFile, { headers });
cache.put(_id, response)
}
reader.readAsArrayBuffer(file);
}
}
```

GetLocalListFromCache.ts

```
export async function getLocalListFromCache(cacheName: string) {
if (!caches) return []
const cache = await caches.open(cacheName)
const responses = await cache.matchAll();
if (responses) {
const list: ILocalList[] = []
const listBlob = await Promise.all(responses.map(response => {
list.push({
src: "",
_id: response.headers.get("_id") || "",
title: decodeURIComponent(escape(atob(response.headers.get("title") || ""))),
})
return response.blob()
}))
listBlob.forEach((item, index) => {
list[index].src = URL.createObjectURL(item)
})


return list
}
return []
}
```

Додаток Г – Ілюстративний матеріал

ІЛЮСТРАТИВНА ЧАСТИНА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ УПРАВЛІННЯ ЛОКАЛЬНИМИ ТА МЕРЕЖЕВИМИ РЕСУРСАМИ.

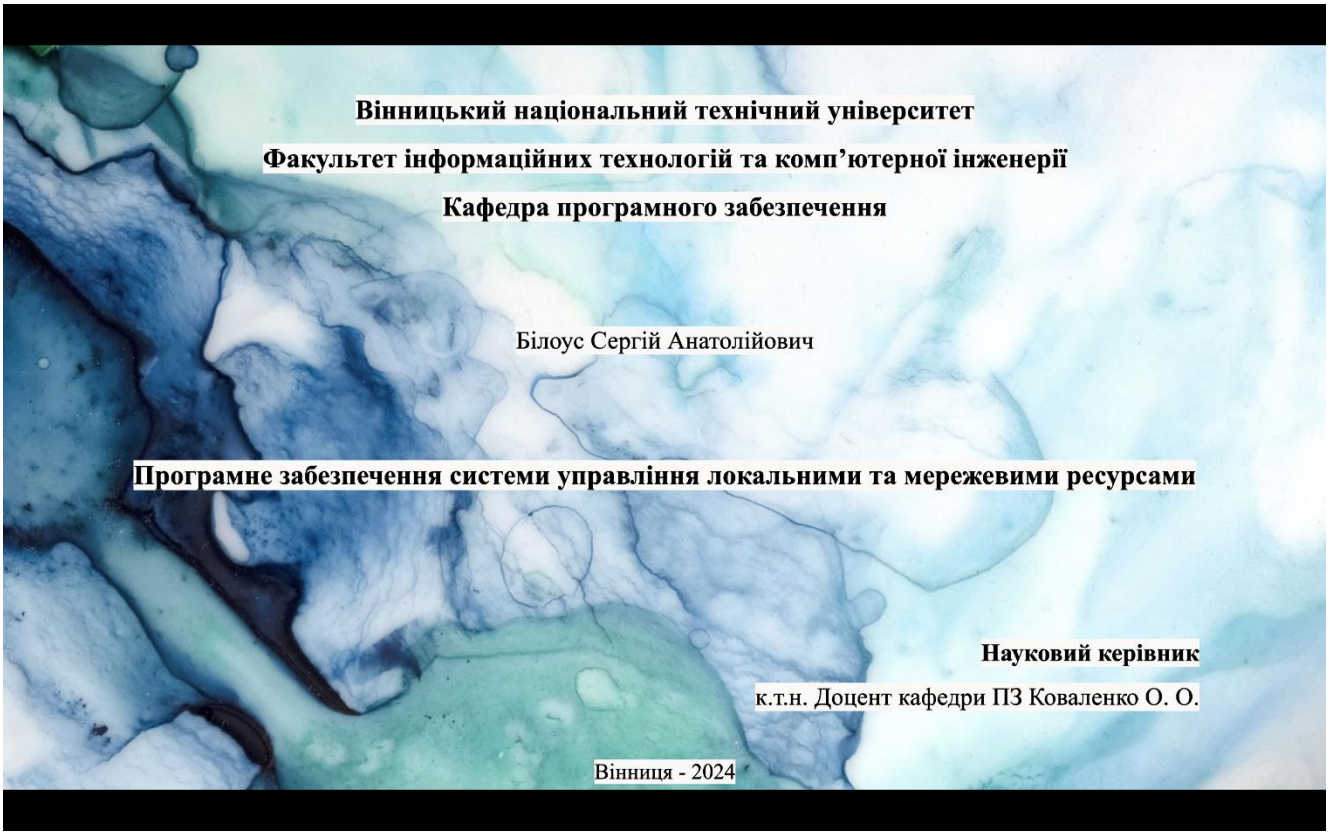


Рисунок Г.1 – Назва роботи

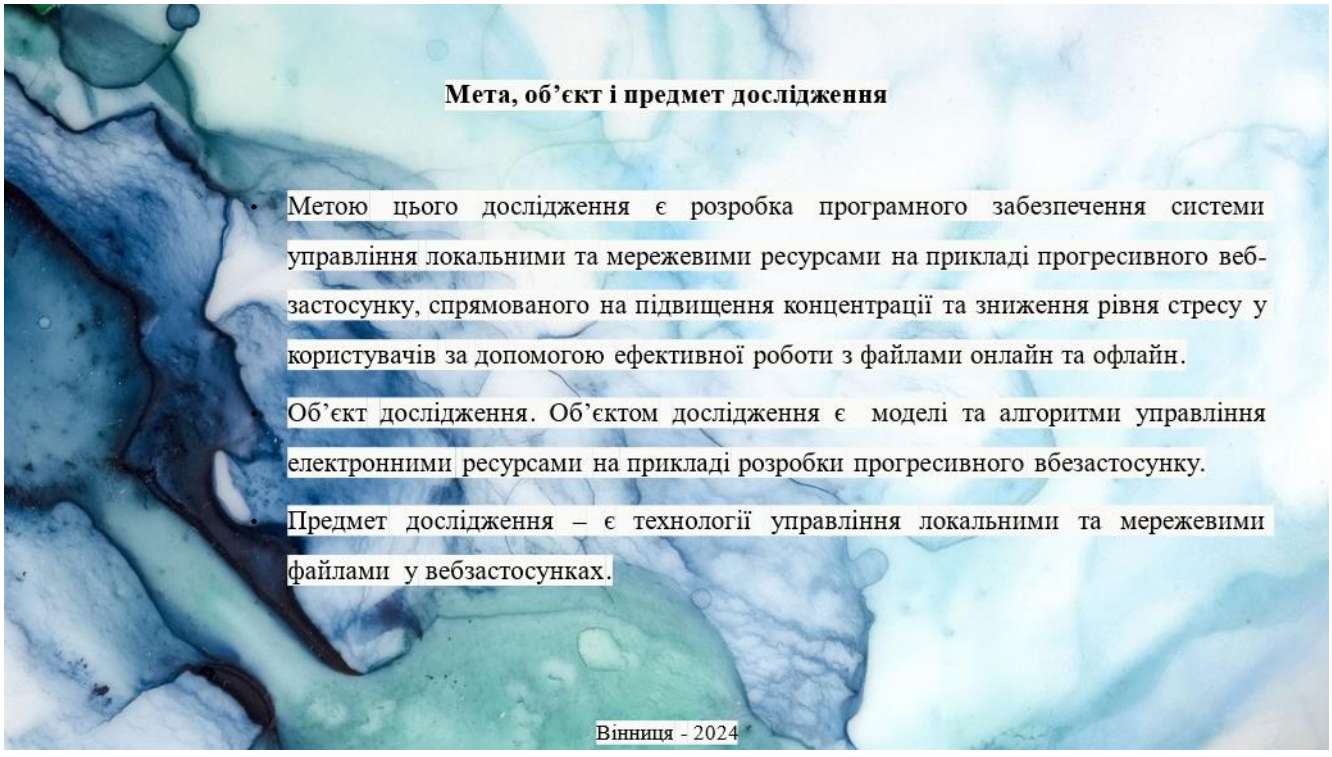


Рисунок Г.2 – Мета, об'єкт та предмет дослідження

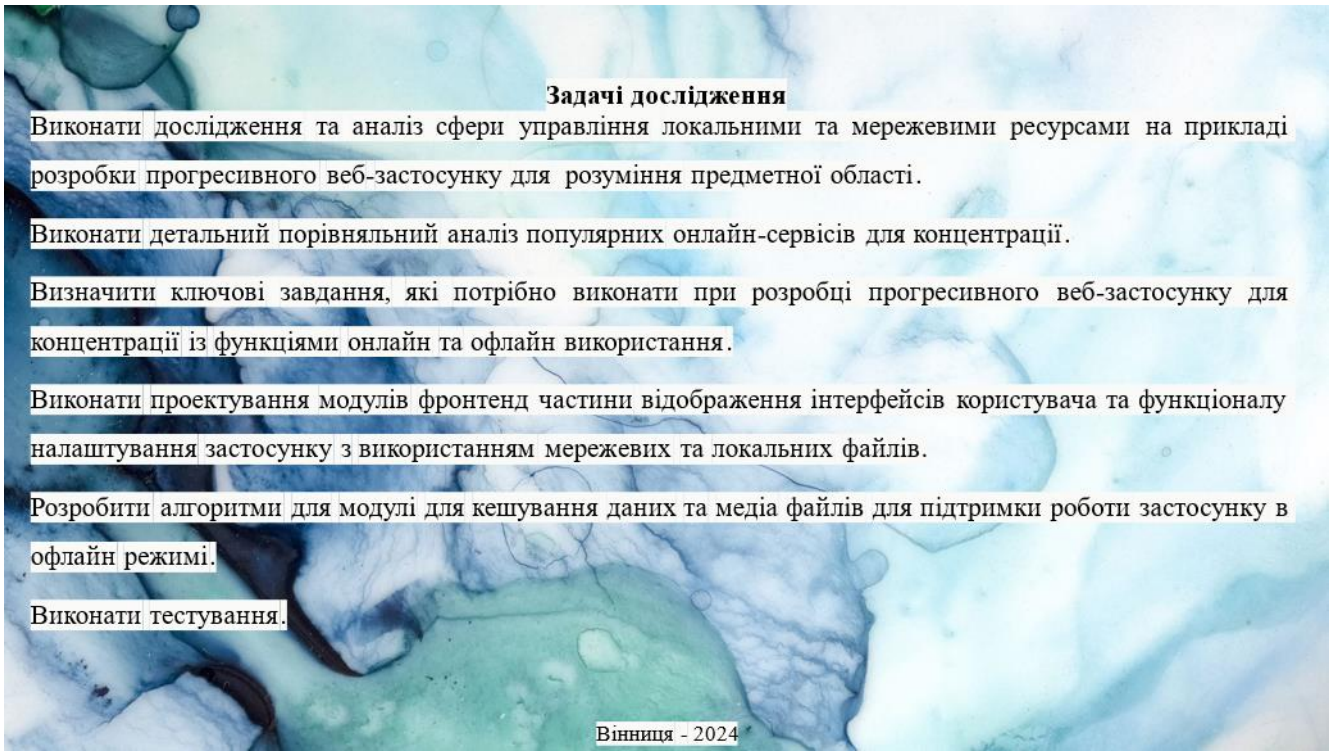


Рисунок Г.3 – Задачі дослідження

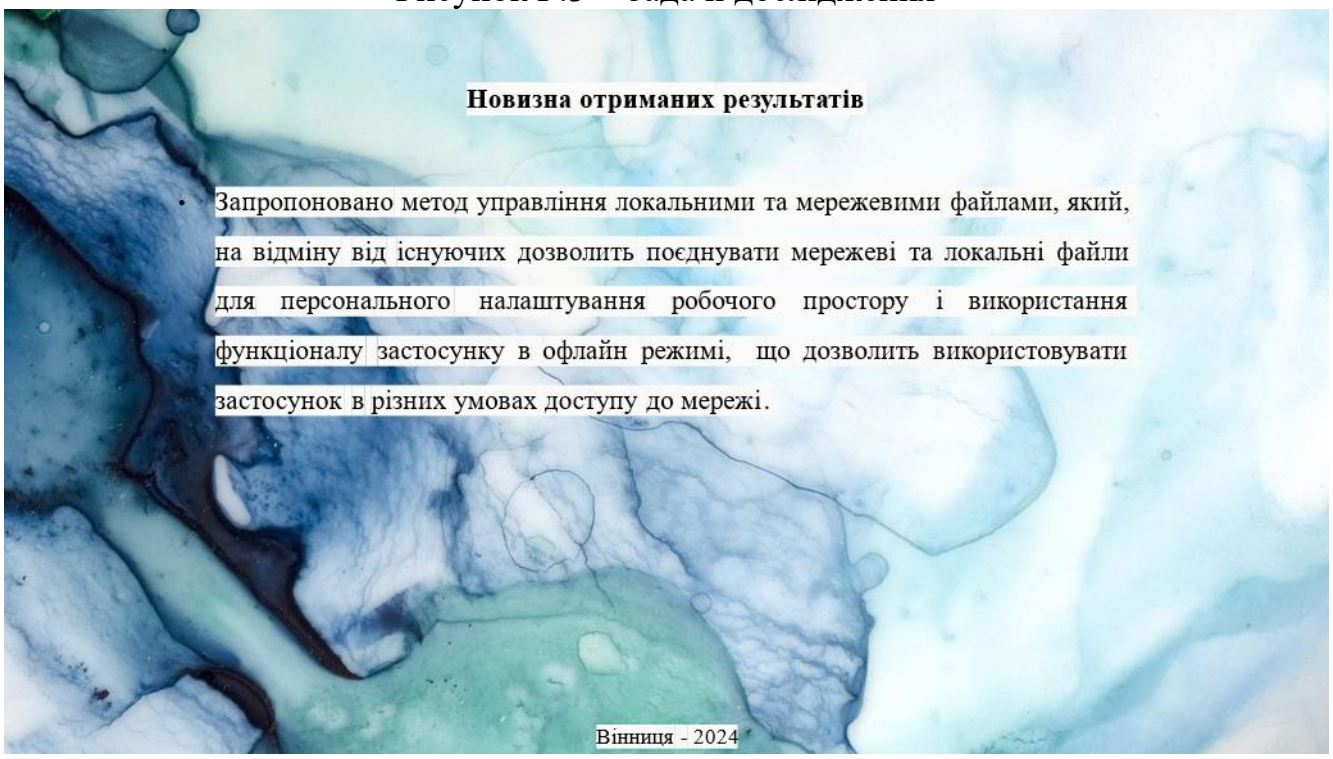


Рисунок Г.4 – Новизна отриманих результатів

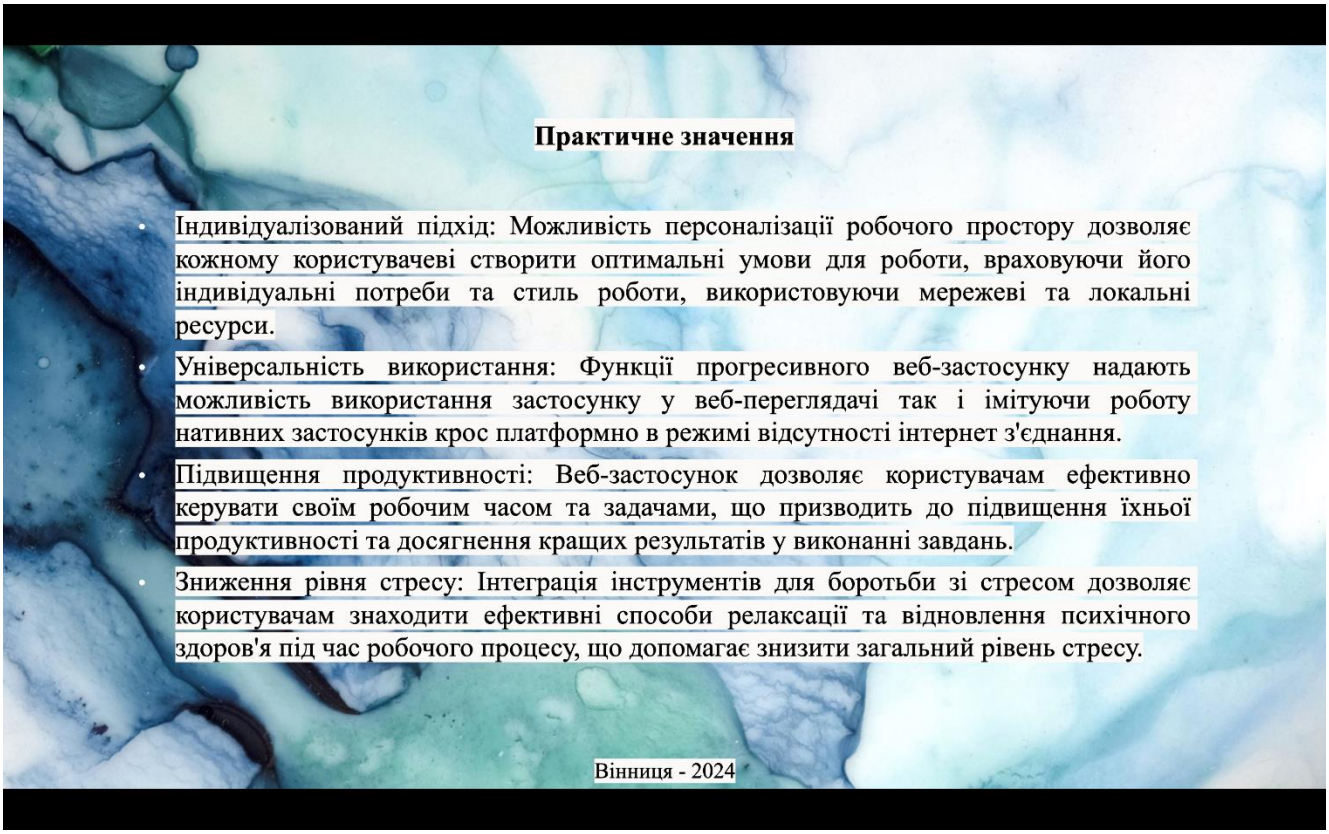


Рисунок Г.5 – Практичне значення

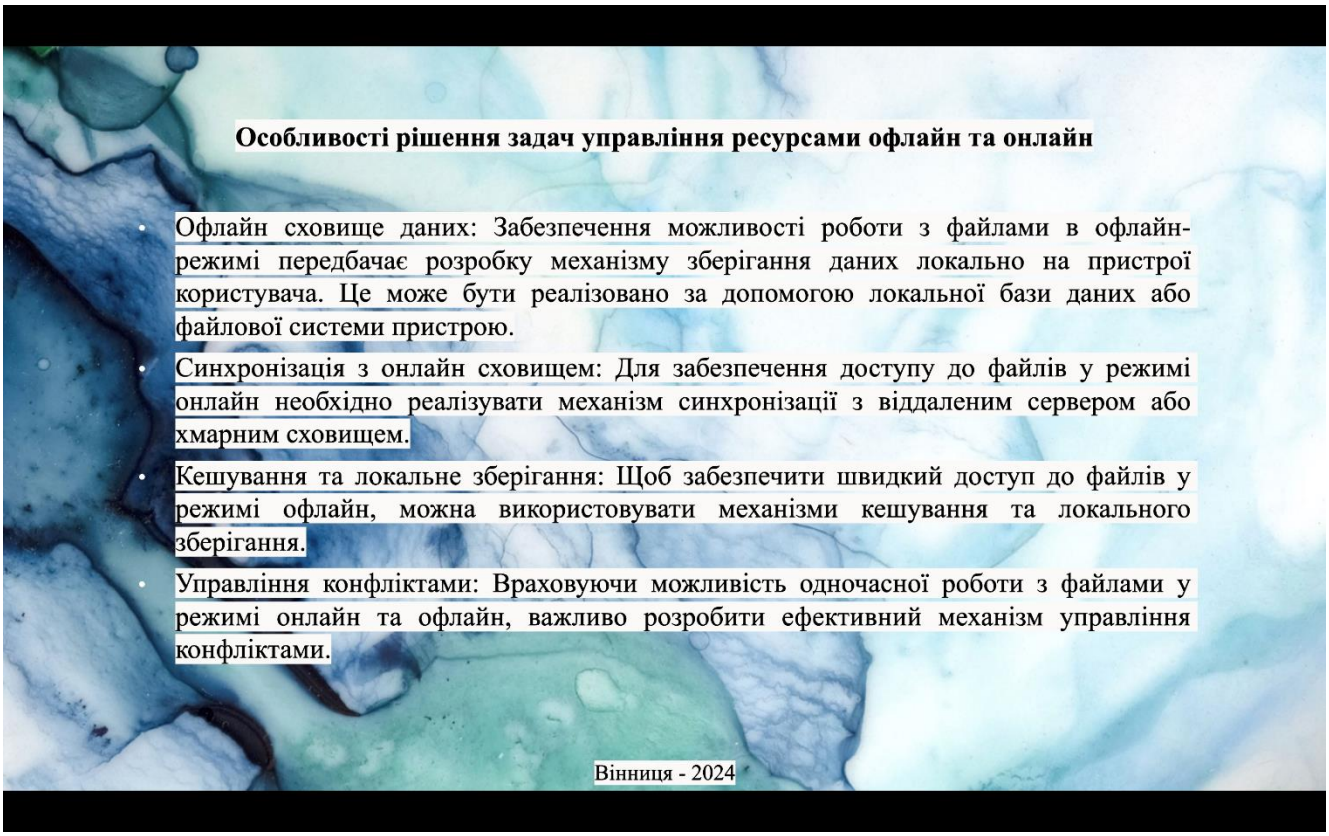


Рисунок Г.6 – Рішення задач управління ресурсами офлайн та онлайн

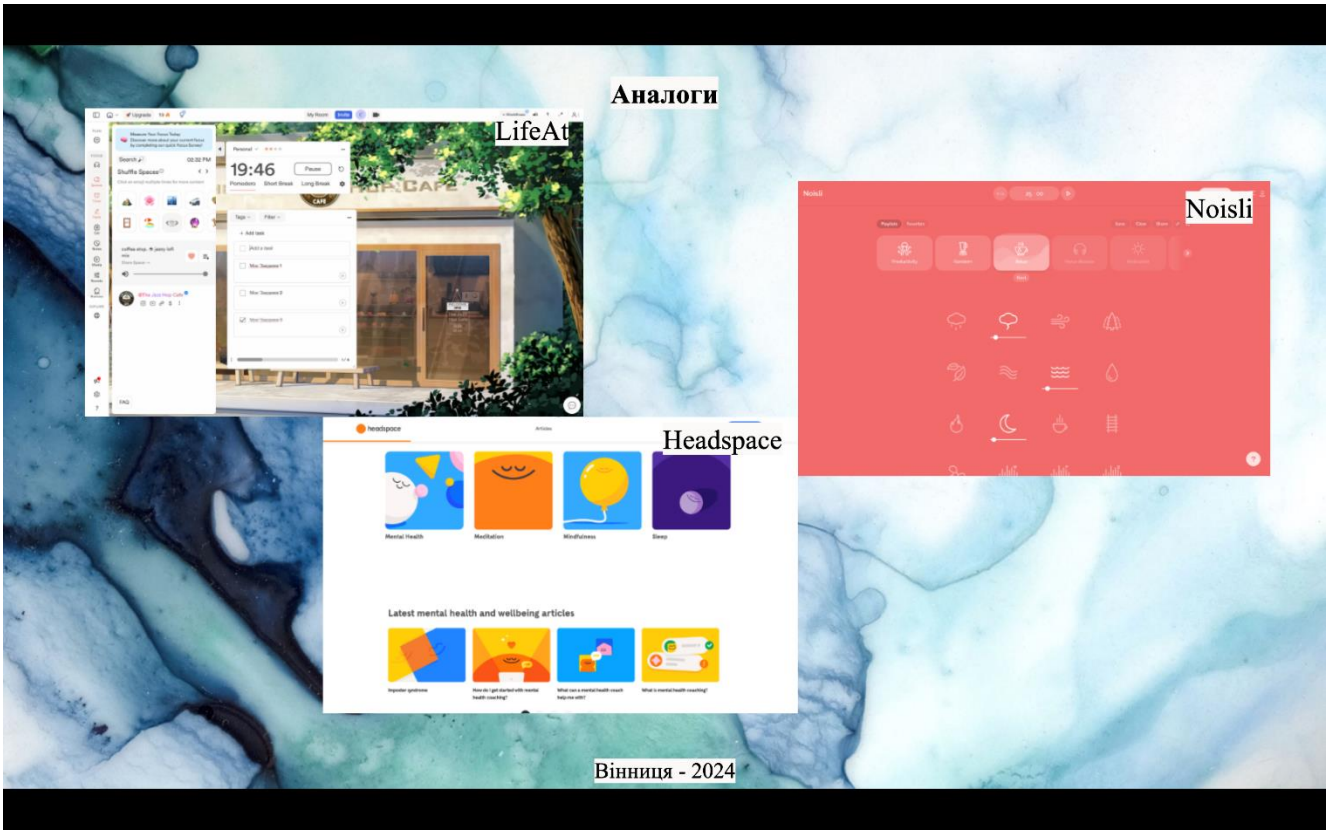


Рисунок Г.7 – Аналоги

Рисунок Г.8 – Діаграма процесу взаємодії

Рисунок Г.9 – Схема роботи модулю та вибору джерела даних

Рисунок Г.10 – Розробка модуля service worker та кешування даних

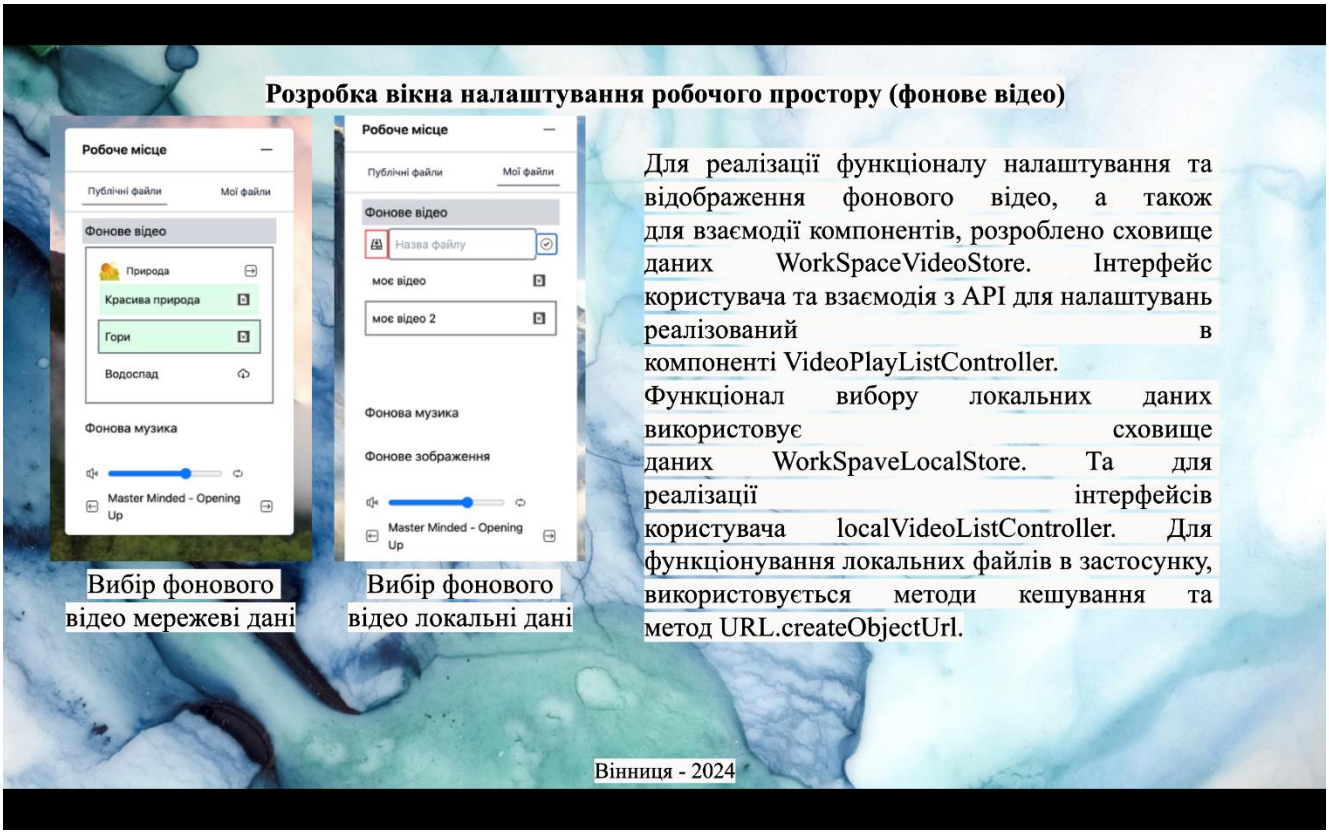


Рисунок Г.11 – Розробка інтерфейсу для фонового відео, мережеві та локальні ресурси

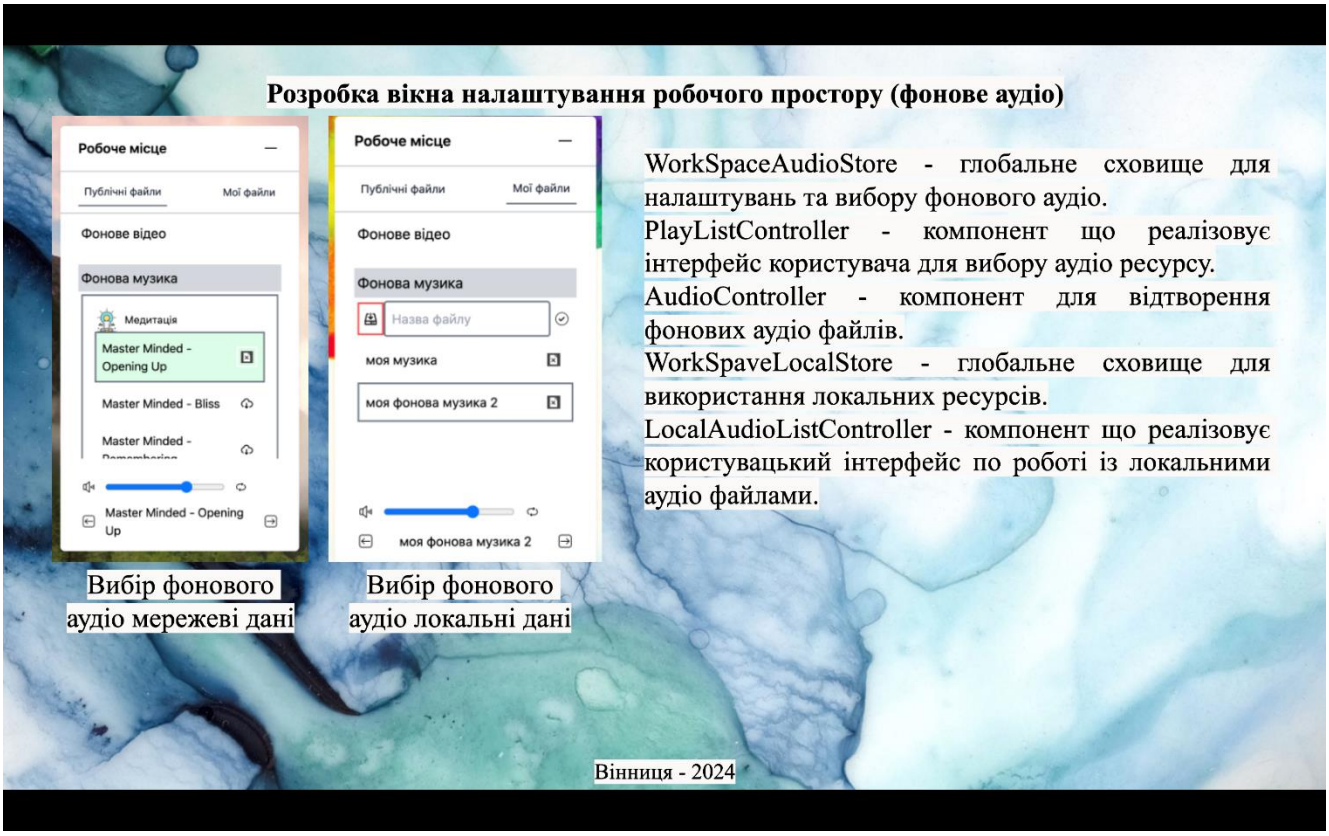


Рисунок Г.12 – Розробка інтерфейсу для фонового аудіо, мережеві та локальні ресурси

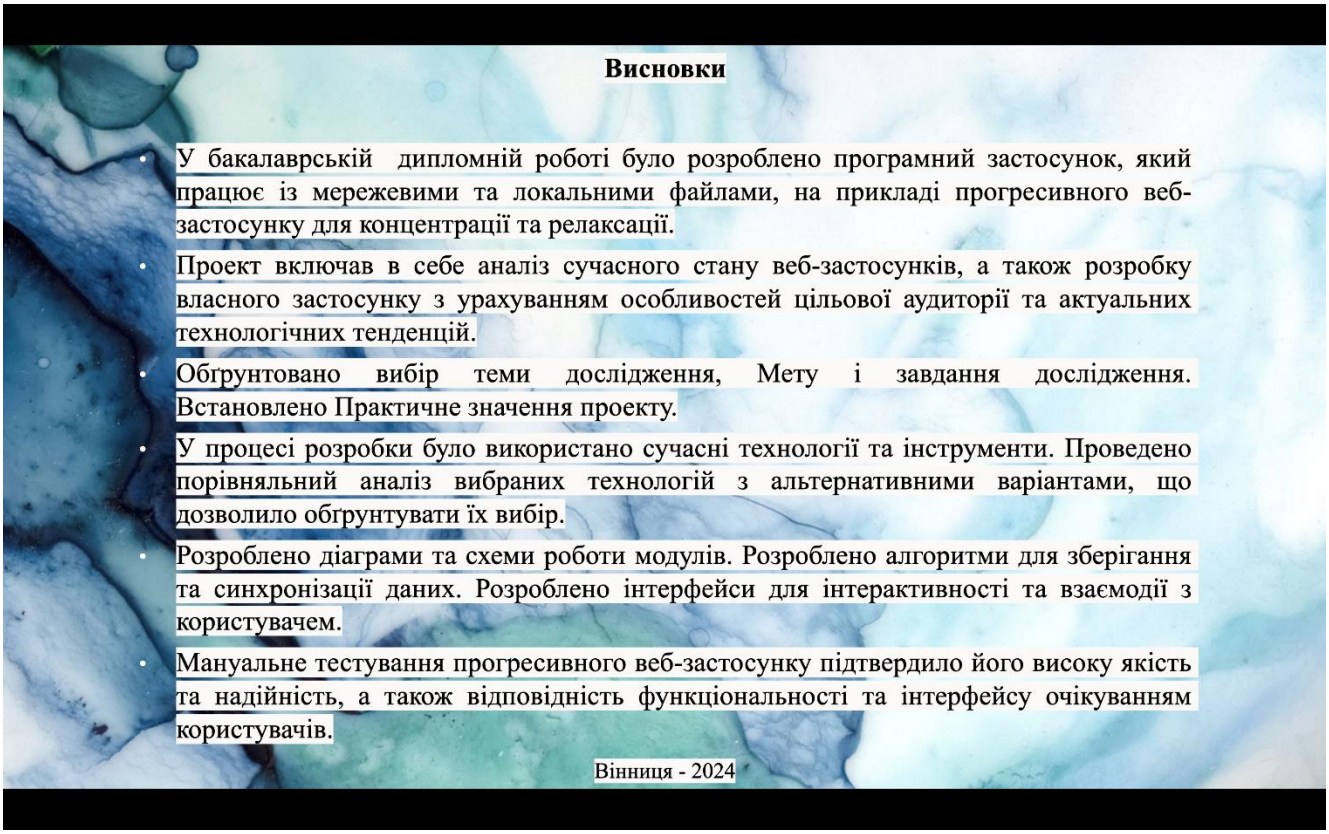


Рисунок Г.13 – Висновки