

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ОНЛАЙН МЕНЕДЖЕРА ЗАДАЧ
SOFTWARE IMPLEMENTATION OF AN ONLINE TASK MANAGER

Виконав: студент 4 курсу, групи КНТ-128сп
Спеціальності 121 Інженерія програмного
забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Желнін М.С.
(прізвище та ініціали)

Керівник Колпакова Т.О.
(прізвище та ініціали)

Рецензент Голуб Т.В.
(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Желніна Михайла Сергійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація онлайн менеджера задач.
Software Implementation of an Online Task Manager

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 17 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми. 5. Технічне завдання. 6. Текст програми. 7. Слайди презентації.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Колпакова Т. О., доцент		
Нормоконтроль	Андрєєв М.О., асистент		

7. Дата видачі завдання “15” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент(ка)

(підпис)

Желнін М.С.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Колпакова Т.О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 113 с., 54 рис., 1 табл., 16 джерел.

ВАЖЛИВІ РЕЧІ, КОМП'ЮТЕРНА ПРОГРАМА, МЕНЕДЖЕР ЗАДАЧ, СПРОЩЕННЯ ЩОДЕННИХ ПЛАНІВ, ВЕБ-ДОДАТОК, ПЛАНУВАЛЬНИК.

Об'єкт дослідження – програмні засоби для планування задач.

Предмет дослідження – браузерні програми для онлайн планування необхідних задач.

Мета роботи – розробка програмного забезпечення для спрощення процесів планування задач.

Матеріали, методи та технічні засоби: функціональне програмування, мови програмування HTML, CSS (SCSS), JS, REACT, REDUX, персональний комп'ютер під управлінням операційної системи Microsoft Windows, Linux, iOS або мобільний пристрій під управлінням Android, iOS та любий веб-браузер.

Результати. Було створено програмний веб-застосунок, який керується браузером комп'ютера, ноутбука, мобільного телефону або планшету, який допомагає визначити основні та додаткові задачі, які можна занести в програму. Вона допомагає впорядкувати задачі та більше ніколи не забувати про їх виконання.

Висновки. Розроблено програмну онлайн систему для користувачів, яка визначає задачі і заносить їх до списку.

Галузь використання – для планування як індивідуальних, так і бізнес задач.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 113 pages, 54 pictures, 1 tabel, 16 sources.

COMPUTER PROGRAM, IMPORTANT THINGS, TASK MANAGER, SIMPLIFICATION OF DAILY PLANS, WEB-APPLICATION, PLANNER.

Object of study is a task scheduling software.

Subject of study are computer programs for online planning of necessary tasks..

The purpose of the work is to develop software to simplify task planning processes.

Materials, methods, and tools: functional programming, programming languages HTML, CSS (SCSS), JS, REACT, REDUX, personal computer running Microsoft Windows, Linux, iOS or mobile device running Android, iOS and any WEB-Browser.

Results. A software web application has been created that is controlled by the browser of a computer, laptop, mobile phone or tablet, which helps to identify the main and additional tasks that can be included in the program. It helps to organize tasks and never forget to perform them again.

Conclusions. A software system for WEB-users has been developed, which defines tasks and lists them.

The field of use - for planning both individual and business tasks.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз предметної області	11
1.1 Об'єкт та предмет дослідження	11
1.2 Теоретичний аналіз предметної області	11
1.2.1 Поняття веб-застосунок	12
1.2.2 Види веб-сайтів	12
1.2.3 Основні етапи проектування веб-застосунків	14
1.2.4 Методи розробки веб-застосунків	17
1.3 Порівняльний аналіз з іншими веб-застосунками	19
1.3.1 Огляд аналогів	20
1.4 Функціональні можливості розроблюваного веб-застосунку	25
1.5 Формальна постановка вирішуваної задачі	25
1.6 Висновки за розділом 1	27
2 Опис програми	28
2.1 Структура веб-застосунку	28
2.1.1 Які бувають веб-застосунки	28
2.2 Обґрунтування вибору типу веб-застосунку	29
2.2.1 Архітектура PWA	30
2.2.2 Які технології використовує PWA	30
2.2.3 Переваги та недоліки PWA	31
2.2.4 Вибір технології PWA	32
2.3 Функціонування застосунку	32
2.4 Середовище розробки застосунку	34
2.5 Висновки за розділом 2	34
3 Розробка програми	35
3.1 Етапи встановлення необхідних модулів та фреймворків	35
3.2 Структура папок та файлів	37

3.3 Реалізація користувацького UI	40
3.3.1 Створення макету	40
3.3.2 Верстка та розробка користувацького UI.....	42
3.4 Реалізація сховища даних Redux	45
3.5 Розміщення застосунку на сервісі Netlify	49
3.6 Підключення PWA	50
3.7 Висновки за розділом 3	52
4 Експлуатація, тестування та експериментальне дослідження програми	53
4.1 Інструкція по роботі з застосунком.....	53
4.2 Методика тестування застосунку	64
4.3 Практичне тестування застосунку.....	65
4.4 Висновки за розділом 4	72
Висновки.....	73
Перелік джерел посилання	74
Додаток А Технічне завдання.....	76
Додаток Б Текст програми.....	79
Додаток В Слайди презентації	105

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE – Integrated Development Environment;

JS – JavaScript;

NPM – Node Package Manager;

PWA – Progressive Web Application;

SPA – Single Page Application;

UI – User Interface;

БЕМ – Блок Елемент Модифікатор;

ПЗ – програмне забезпечення.

ВСТУП

Створення веб-застосунків сьогодні вважається одним з найбільш прибуткових та перспективних напрямків розробки для багатьох компаній або програмістів-одинаків, які працюють в області високотехнологічних і комп'ютерних технологій.

Все більше компаній переходить від розробки звичайного програмного забезпечення та бізнес-інструментів до веб-розробки, тому що це актуально, бо Інтернет розвивається дуже швидко і на сьогоднішній день є дуже багато користувачів Інтернет.

Зараз існують дуже сучасні веб-технології, які дозволяють створювати потужні, масштабовані, кросбраузерні веб-застосунки і сам процес їх створення стає значно простішим і доступнішим практично будь-якому розробнику в області веб-технологій.

Актуальність теми даної роботи обумовлена популярністю та активним розвитком веб-застосунків, використанням і популярністю веб-застосунків, які допомагають користувачам спрощувати ті чи інші повсякденні завдання, вимоги і потреби в повсякденному житті. На сьогоднішній день дуже багато малих та великих компаній, програмістів та звичайних людей, які не пов'язані з розробкою, використовують велику кількість застосунків, які в тій чи іншій мірі полегшують якість розв'язання повсякденних задач, проблем та питань. Тому розробка веб-застосунків, які допомагають вирішувати завдання чи проблеми є дуже актуальною.

Основною метою даної роботи є розробка веб-застосунку, що дозволяє користувачеві швидко і зручно складати список завдань.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

- аналіз функціональних можливостей існуючих застосунків;
- проектування веб-застосунку;
- вибір інструментів розробки;

- розробка і тестування веб-застосунку.

В процесі виконання даної роботи був зроблений аналіз та визначені основні вимоги до застосунку, що допомогло спроектувати зручні та семантичні макети, призначені для користувачів інтерфейсу, а також був описаний процес взаємодії користувача з розроблюваним веб-застосунком.

Після створення макетів був розроблений інтерфейс для крисувачів (зверстані сторінки і написані програмні скрипти). Програми розроблювались за допомогою сучасних веб-інструментів, методів, фреймворків і бібліотек, таких як HTML, CSS (SCSS), JS, React, Redux при розробці користувацького інтерфейсу та сховища даних.

Після розробки веб-застосунку було проведено тестування програми, яке дозволило переконатися у відсутності помилок і зробити висновок, що розроблений застосунок відповідає функціональним вимогам.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В цьому розділі наводиться детальний опис та аналіз предметної області, основних вимог до розроблюваного застосунку, понять, визначень та теоретичний аналіз найбільш популярних на даний момент веб- застосунків, що дозволяють легко та зручно складати список завдань. Це зроблено з метою визначити основні та додаткові функціональні можливості, які можуть покращити користування програмою.

Також буде наведено детальний опис об'єкта та предмета дослідження, формальна постановка вирішуваної проблеми та будуть наведені інші рішення щодо вирішення проблеми.

1.1 Об'єкт та предмет дослідження

Об'єктом дослідження даної дипломної роботи є процес і засоби розробки веб-застосунку «Менеджер задач».

Предметом дослідження дипломної роботи є особливості і методи створення веб-застосунку «Менеджер задач».

1.2 Теоретичний аналіз предметної області

В наш час в епоху дуже розвинутих веб-технологій неможливо уявити людину, яка не користується мережею Інтернет. Кожен день ми заходимо до Інтернету на веб-сторінки у пошуках якої-небудь інформації або просто посидіти в соціальних мережах. Але ми навіть не замислюємося про те, що таке Інтернет, що таке веб-застосунок, його види та інше. У даному підрозділі буде описано, що таке веб-застосунок, які види веб-застосунків існують, в чому їх особливості та інше.

1.2.1 Поняття веб-застосунків

Веб-застосунок – це файли та програмні засоби, документи, які відображаються за допомогою мови програмування так, щоб їх бачили користувачі мережі Інтернет. Клієнтом слугує браузер користувача, а сервером веб-сервер. Дані зберігаються на сервері і включають в себе будь-який текстовий, графічний, аудіо або відеовізуальний контент, зібраний з однієї або декількох сторінок. Унікальність веб-сайту у більшості випадків визначає доменне ім'я. Головною перевагою веб-застосунків є кросплатформенність, тобто клієнт не залежить від операційної системи користувача, будь то комп'ютер, планшет або мобільний пристрій.

1.2.2 Види веб-сайтів

Серед веб-сайтів дуже популярними є такі: Landing Page (односторінкові), сайт-візитка, корпоративні сайти, промо-сайти, інтернет-магазини, персональні сайти, on-line сервіси, портали, веб-застосунки.

Лендінг (від англійського landing page) являє собою спеціально спроектовану сторінку сайту, яка покликана спонукати відвідувачів до вчинення певної дії: придбання товару, оформлення підписки, замовлення послуги тощо.

У наш час наявність власного веб-представництва в Інтернеті є одним з основних способів просування бізнесу. Різноманітність подібних представництв вражає уяву: тут і скромні персональні сторінки, і великі корпоративні ресурси, і інформаційно-розважальні портали. Однак, в тому випадку, якщо компанія тільки починає своє просування в Інтернеті, то найкращим варіантом буде розробка сайту-візитки.

Корпоративний сайт - це повноцінне представництво компанії в Інтернеті. Тут міститься повна інформація про фірму, послуги чи продукцію, які вона пропонує, тощо. Головна відмінність корпоративних сайтів від

сайтів-візиток полягає в розширених функціональних можливостях а також можливої інтеграції з внутрішніми системами компанії.

Починаючи випуск нової продукції або запускаючи новий товарну пропозицію компанії роблять все можливе для отримання лояльності і визнання споживачів, а також для розширення кола постійних покупців або замовників. Зрозуміло, будь-яка новинка, будь то товар або послуга, потребує реклами. Власне, для просування нових пропозицій і створюють промо-сайти.

Від інших сайтів інтернет-магазини відрізняються, в першу чергу, тим, що тут представлено не тільки каталог товарів, а й надається можливість їх покупки безпосередньо на сайті. Популярність цього виду сайтів пояснюється тим, що відкрити свій інтернет-магазин можуть як великі компанії, так і представники малого бізнесу.

На відміну від сайтів-візиток і корпоративних сайтів, персональні сторінки є представництвом не компаній, а приватних осіб. Найчастіше, подібні проекти відрізняються малим об'ємом, містять в собі відомості біографічного або особистого характеру, а також інформацію про рід діяльності власника або про послуги, які він надає.

On-line сервіси надають різні послуги, такі як пошук книги, оплата рахунків та інше.

Портали – це дуже насичені сайти, на яких можна знайти все необхідне: новини, авторські блоги, голосування, пошукові та поштові сервіси і багато іншого.

Веб-застосунками називають різні програмні продукти, доступ до яких здійснюється через браузер. Найчастіше, в розробці веб-застосунків зацікавлені різні комерційні організації. Таким чином, функціональні можливості веб-застосунків, як правило, включають потужні бізнес-орієнтовані інструменти [1].

1.2.3 Основні етапи проектування веб-застосунків

Веб-технології дозволяють розробляти як прості веб- застосунки, так і складні інформаційні веб-системи. Для успішної розробки таких програм потрібно розуміти загальний процес розробки програмного забезпечення (ПЗ): основні дії по розробці, які повинні бути виконані; як ці дії взаємопов'язані між собою; який порядок їх виконання; які фахівці повинні брати участь в цьому процесі.

Процес розробки може бути розділений на набір базових робіт з добре визначеними межами і змістом:

- розуміння проблеми;
- планування рішення;
- виконання плану;
- перевірка точності отриманого результату;
- доопрацювання з метою видалення можливих помилок або неточностей.

У процесі розробки будь-якого ПЗ можна виділити наступні основні види діяльності:

- визначення вимог до ПЗ (інжиніринг вимог): призначений для розуміння розв'язуваної проблеми;
- проектування: призначене для планування та вирішення проблеми;
- реалізація: перетворення плану в працюючий програмний код;
- перевірка і оцінка якості: призначено для виявлення помилок кодування (програмного коду) або невідповідностей між певними вимогами і їх реалізацією;
- розгортання – надання користувачам можливості працювати зі створеним ПЗ;
- підтримка – призначена для відстеження використання діючої системи і збереження її працездатності;

- розвиток – призначений для поліпшення з часом розробленого рішення; надання нових вхідних даних для процесу розробки в формі нових вимог.

Інжиніринг вимог призначений для розуміння необхідних можливостей і характеристик створюваного ПЗ. Даний аналіз спрямований на визначення функціональних вимог (які функції система повинна виконувати) і не функціональних вимог (якість пропонованого рішення). Інжиніринг вимог також передбачає виявлення загальної ідеї, яка стоїть за розроблюваною системою; основних зацікавлених осіб, яким потрібна нова система, і умови, в яких буде використовуватися система. Виявлені вимоги обробляються з метою створення високорівневих моделей даної системи, яка абстрагується від непотрібних подробиць даної проблемної області.

Проектування призначене для опису рішення, яке повинно відповідати функціональним вимогам і вимогам ефективності, а також обмеженням того середовища, в якій вона буде працювати. Раніше зібрані вимоги уточнюються і поліпшуються, щоб задовольняти можливим технологічним обмеженням.

На етапі реалізації проекту розроблені проектні рішення перетворюються у відповідний програмний код (вручну або за допомогою інструментів автоматизації програмування). Можуть знадобитися бібліотеки програм, різні мови програмування, різні комунікаційні протоколи і технічні пристрої.

Перевірка і оцінка якості зазвичай проводиться паралельно з реалізацією, так як правильність і надійність проміжних результатів, а не тільки кінцевого продукту є дуже важливим для гарантування якості всієї програми. Якість в значній мірі пов'язано з наступними критеріями:

- оцінкою функціональності, тобто правильності поведінки застосунку щодо заданих функціональних вимог;
- оцінкою продуктивності, часу очікування відгуку програми в звичайних умовах і при пікових навантаженнях;

- оцінкою зручності використання (usability), тобто легкості використання, комунікаційної ефективності та відповідності стандартам використання.

Розгортання застосунку надає користувачам можливість використання цього застосунку. Залежно від типу застосунку процес розгортання може включати:

- встановлення ПЗ на комп'ютерах клієнтів;
- встановлення центрального застосунку і баз даних на сервері;
- конфігурація проміжного комунікаційного ПЗ;
- інструктування і навчання майбутніх користувачів, особливо, якщо встановлюється зовсім новий застосунок, а не нова версія вже наявної програми.

Підтримка розгорнутого і працюючого застосунку у програмному забезпеченні означає забезпечення його робочого стану, яке полягає в гарантуванні його доступності та зменшенні збоїв. Може включати: періодичну перевірку файлів журналу; звітів про помилки та очищення тимчасових файлів; виправлення помилок; установку виправлень.

Розвиток програми. Застосунок призначений для вирішення реальних завдань користувачів. Однак у зв'язку з розвитком організації, ускладненням розв'язуваних завдань, поліпшенням розуміння користувачем можливостей даного ПЗ неминуче потрібно розвивати створене ПЗ. Нові потреби користувачів з'являються тільки після того, як вони деякий час попрацюють з створеним ПЗ. Після цього вони починають давати свої пропозиції та коментарі. Поява нових вимог може викликати необхідність запустити весь процес розробки заново. Незважаючи на суворе дотримання рекомендацій правильної організації процесу розробки ПЗ, часто тільки після розгортання і накопичення деякого досвіду роботи користувачів з ПЗ стає ясно, що деякі вимоги не були повністю виконані і застосунок повинен бути доопрацьований [2].

1.2.4 Методи розробки веб-застосунків

Сьогодні можна назвати декілька основних методів веб-розробки:

- розробка веб-застосунків використовуючи одну чи більше мов програмування;
- розробка веб-застосунків з використанням інструментів, які автоматизують розробку застосунків.

Коли розробка йде за допомогою першого методу, то застосунки створюються використовуючи одну або більше мов програмування. Також у цьому виді існують інші два види веб-застосунків: статичні та динамічні.

Статичні застосунки розроблюються з використанням таких стандартних мов програмування як HTML та CSS.

HTML – це мова гіпертекстової розмітки, яка використовується аби повідомити браузеру як потрібно розташувати ті чи інші елементи на сторінці. Остання п'ята версія HTML (HTML5) – це оновлена версія гіпертекстової розмітки, яка використовує семантичні теги, такі як <header>, <main>, <aside>, <footer>, <nav>, <section> та інші. Ці теги передають зміст документу для пошукових робіт, які показують застосунок у пошукових запитах та програм, які читають текст з екрану для людей з обмеженими можливостями. Правильне використання цих тегів є дуже важливим для розуміння контенту на сторінці. Також в цю версію були додані різні модулі, які дозволяють використовувати відео, аудіо, векторну графіку SVG, растрову графіку, різні API та локальні бази даних.

CSS – це мова стилізування контенту на сайті. Зараз дуже популярні CSS-препроцесори, такі як SASS (SCSS) та LESS, які значною мірою спрощують процес стилізування веб-застосунків.

Використання тільки HTML та CSS дозволяють створювати лише прості сайти - це так звані лендінги, на яких створюють сайти-візитки або прості ознайомчі сайти, де компанії дають користувачеві якусь корисну їм

інформацію. Якщо вже потрібно додати якусь логіку на сайт, тоді вже не обійтись без мови програмування JavaScript.

JavaScript – це мова програмування, яка робить сторінки на сайті живими, тобто дозволяють якимось взаємодіяти з користувачем. Програми на цій мові називають скриптами, які вбудовуються в HTML і виконуються автоматично при завантаженні веб-сторінки. На теперішній момент JS може виконуватися не тільки в браузері користувача, а також на сервері або будь-якому іншому пристрої, який має спеціальну програму, яка має назву «движок» JavaScript.

Браузер має власний «движок». У різних браузер він називається по-різному. Наприклад, V8 у Google Chrome, SpiderMonkey у Firefox та інші.

Зараз практично всі розробники переходять з написання застосунків на чистому JS до написання програм на фреймворках та JS-бібліотеках. Фреймворк – це програмний продукт з відкритим початковим кодом, який полегшує розробку та об'єднання різних компонентів великих та малих технічно складних застосунків.

Одними з найпопулярніших JS-фреймворків та бібліотек є React, Vue та Angular.

Переваги фреймворків:

- простота написання коду;
- легка реалізація бізнес-логіки та проектів;
- застосунки, які розроблені з використанням фреймворків легко підтримуються;
- застосунки на фреймворках працюють швидше і витримують великі навантаження;
- застосунки на фреймворках більш надійні та безпечні.

Основним недоліком є те, що розробка займає більше часу.

Динамічні застосунки створюються за допомогою серверних скриптів, таких як PHP, Node.js, Python, Ruby, Kotlin, ASP.NET та інші. Серверні скрипти зазвичай взаємодіють із різними базами даних, такими як MySQL,

MongoDB, GraphQL та інші. Такі застосунки дозволяють використовувати форми реєстрації, персональні кабінети, оплату та багато чого іншого.

Статичні та динамічні веб-застосунки вимагають розробки дизайну (графічного оформлення). Вони розробляються у будь-якому графічному редакторі, наприклад, Figma та Photoshop.

Реалізуються такі застосунки здебільшого у будь-якому текстовому редакторі або спеціальній програмі, наприклад, PHP Storm та VS Code.

Розробка цих застосунків більш складна, ніж розробка застосунків другого методу, бо потребують знань програмування, дизайну та більше часу на їх створення. Беззаперечною перевагою такого метода є отримання того, чого ми бажаємо та підтримання коду.

При використанні другого методу створення застосунків використовуються спеціальні конструктори, які направлені на керування контентом (CMS). CMS (Content Managment System) – це спеціальний програмний продукт, який надає змогу створювати і обслуговувати застосунки. Користувачеві надається змога заповнювати застосунок контентом за бажанням чи за шаблоном, що значно спрощує створення програмних продуктів.

Серед переваг цього методу є простота, не обов'язково знати програмування та зменшення витрат часу і сил.

Серед недоліків – зниження гнучкості та обмеження в можливостях при розробці.

Серед популярних CMS є WordPress, Drupal, Joomla та Bitrix.

1.3 Порівняльний аналіз з іншими веб- застосунками

В мережі Інтернеті існує дуже багато популярних веб-застосунків, які допомагають планувати задачі як для бізнес процесів, так і для особистих потреб. Всі ці веб-застосунки відрізняються між собою програмними можливостями і легкістю використання.

Списки завдань допомагають нам організувати особисте життя та запам'ятовувати важливі справи, не забувати важливі речі, розставляти пріоритети та взагалі спостерігати за своєю роботою і життям. Це можуть бути якісь робочі завдання, книжки, які потрібно прочитати, або продукти, які нам потрібно купити в магазині, ми не намагаємося все запам'ятати, а заносимо до застосунку та зручно відслідковуємо, тим самим не забуваючи голову різними думками.

Для того аби організувати наше життя, в епоху Інтернет технологій ми можемо обійтися без ручки, паперу, блокнотів і т.п.. Онлайн менеджери задач популярні як серед десктопних, мобільних застосунків, так і у вигляді веб-застосунків, які у більшості випадків адаптовані під всі пристрої.

В цій роботі проводиться порівняльний аналіз популярних веб-застосунків, їхні можливості, переваги, недоліки і визначаються програмні можливості розроблюваного веб-застосунку.

1.3.1 Огляд аналогів

Для того, щоб порівнювати онлайн менеджери задач, необхідно визначити та описати базові функції розроблюваного веб-застосунку.

Нижче наведені деякі з них:

- додавати групу;
- змінювати назву групи, колір та видаляти її;
- показувати дату створення групи;
- додавати задачу;
- відмічати задачу як зроблену;
- додавати до конкретної задачі фото;
- видаляти задачу;
- змінювати фон всього застосунку на інший;
- при закритті вкладки, браузера не втрачати дані.

Нижче наведені аналоги розроблюваного застосунку.

Trello (рис. 1.1) – один з найпопулярніших застосунків. Це зручні картки зі справами, які розсортовуються за тематичними стовпчиками. Кожному завданню присвоюється назва, прикріплюються документи, файли, додаються учасники і терміни виконання, розписуються подробиці справи і так далі. Найчастіше Trello використовується не тільки в особистих цілях, але і для роботи. Додаток допомагає маленьким або великим компаніям, корпораціям і ЗМІ. Своя робоча дошка в Trello є і у автора цього матеріалу. У Trello організована детальна візуальна настройка: можна встановити задній фон, прикріпити до картки картинку, завантажити відео або пофарбувати в інший колір.

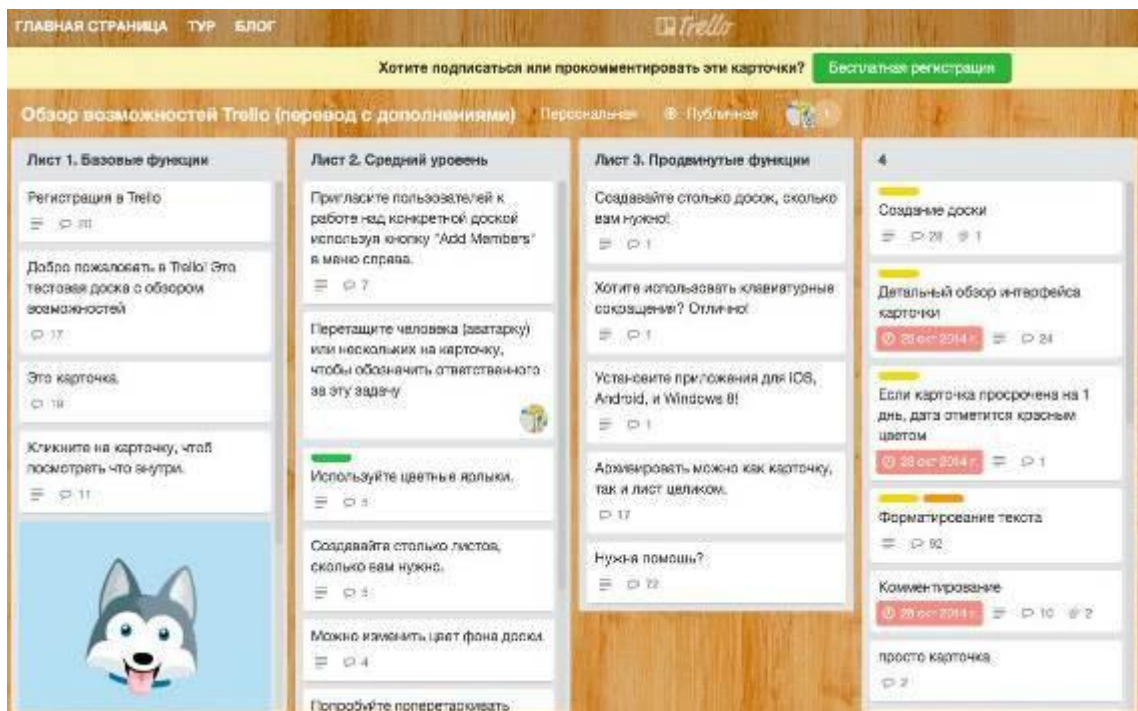


Рисунок 1.1 – Интерфейс Trello

Todoist (рис. 1.2). дозволяє створювати докладні списки справ і нагадування, сортувати їх по категоріях, днях тижня, місяцях і тегами, налаштовувати фільтри, ставити кольору і відправляти завдання іншим користувачам програми. Крім цього Todoist вміє підбадьорювати, показуючи статистику виконаних справ, роздаючи призи та досягнення.

У Todoist є прив'язка до файлів з Google Диска »і Dropbox, спільна робота з низкою інших сервісів, а також підтримка управління голосом.

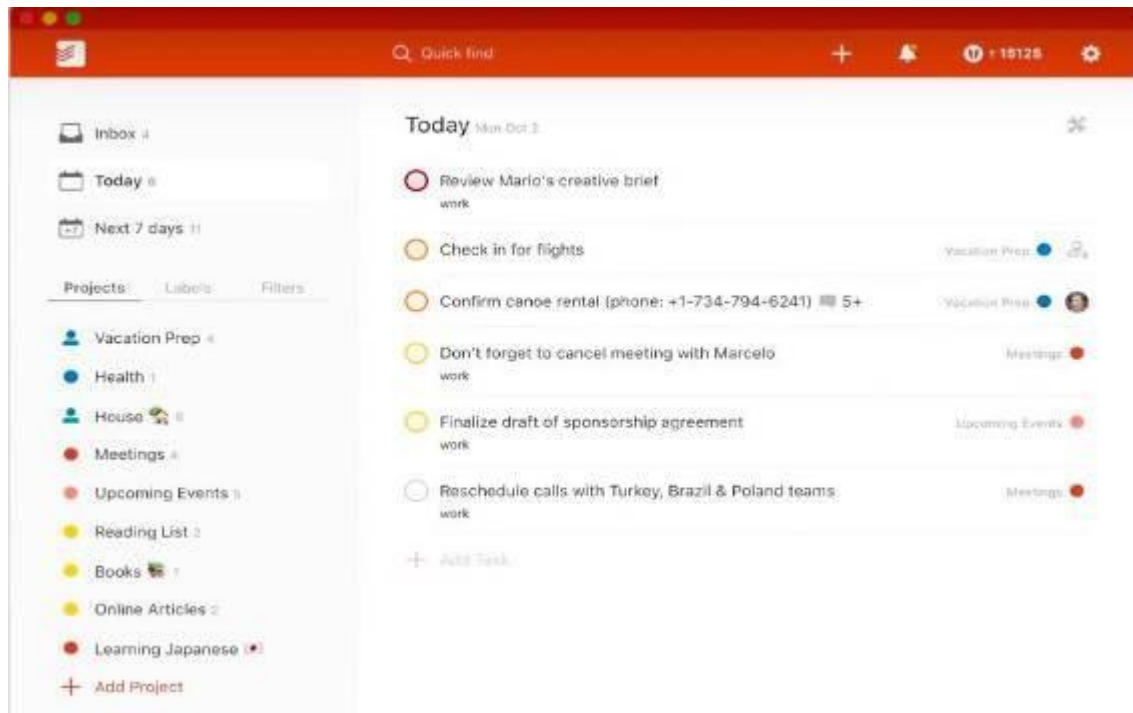


Рисунок 1.2 – Інтерфейс Todoist

Ще один аналог, TickTick, наведено на рис. 1.3. Планувальник TickTick допоможе розібратися зі справами і встановити нагадування, додаючи до завдань підпункти, теги, файли і документи. Справи групуються в тематичні папки і стеки.

У TickTick є підтримка голосу, тому завдання можна ставити не тільки письмово. А для поліпшення концентрації при роботі творці додали класичний і зручний таймер в стилі pomodoro.

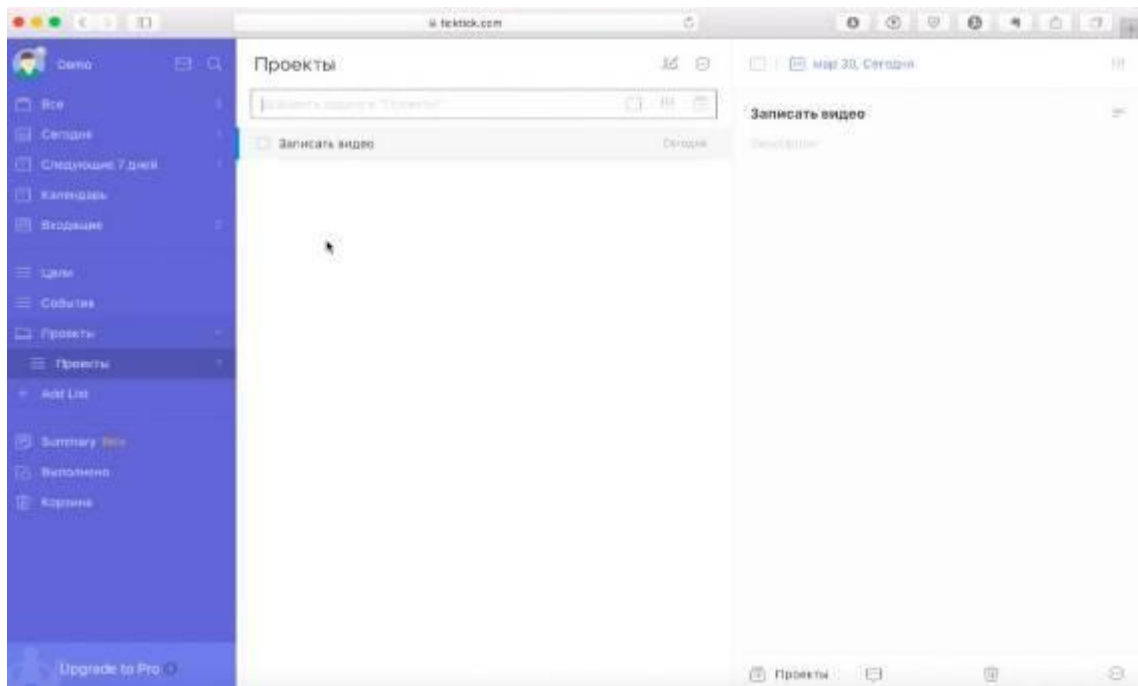


Рисунок 1.3 – Интерфейс TickTick

Google Keep (рис. 1.4). Google Keep часто використовують для нотаток, списків покупок, книг до прочитання і короточасних справ. Завдання тут представлені у вигляді карток, куди кріпляться списки, файли, малюнки, фото, документи та інше. Картками можна змінити колір і прикріпити у верхній частині екрану, якщо цією заміткою (або нотатками) користуються частіше за інших.

У Google Keep є потужна інтеграція з іншими сервісами Google, тому з будь-якого листа або документа в Google Docs або Gmail швидко створюється замітка в Google Keep, а завдання зручно проглядаються через «Google Календар».

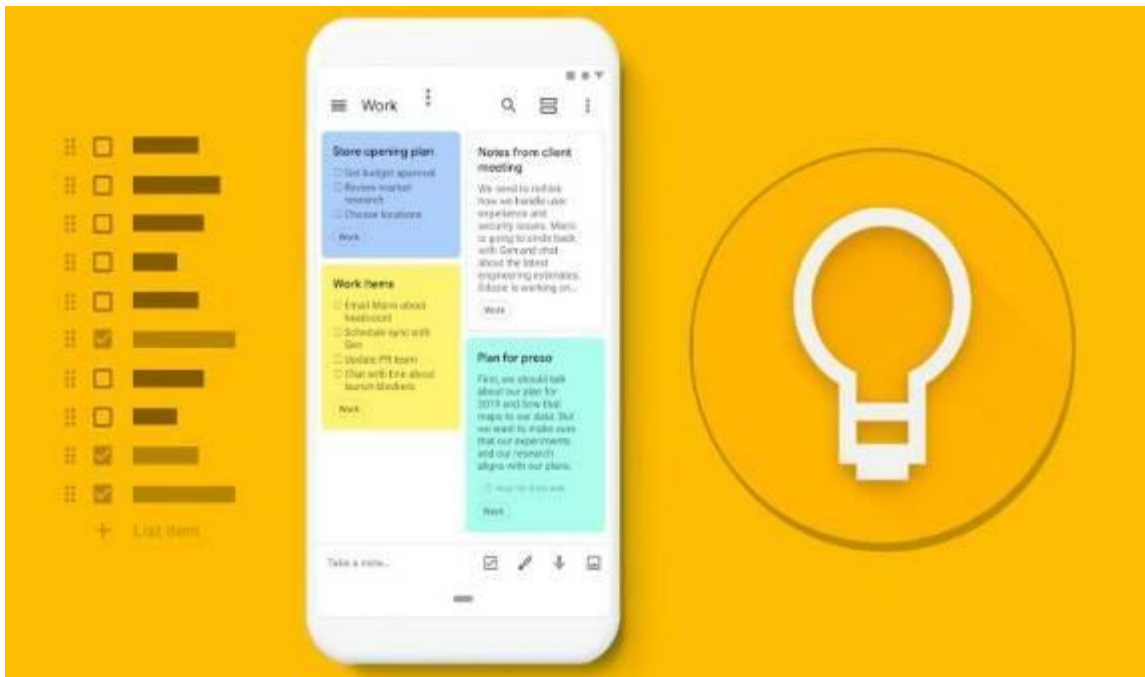


Рисунок 1.4 – Мобільна версія Google Keep

Microsoft To-Do (рис. 1.5) Microsoft To-Do не виділяється великою кількістю можливостей, зате простий у використанні. Є базові списки справ, завдання та нагадування, а також смарт-рекомендації за списками і дат виконання. Все це відправляється за потрібне людям і синхронізується з іншими програмами компанії – Outlook і Office 365 [3].

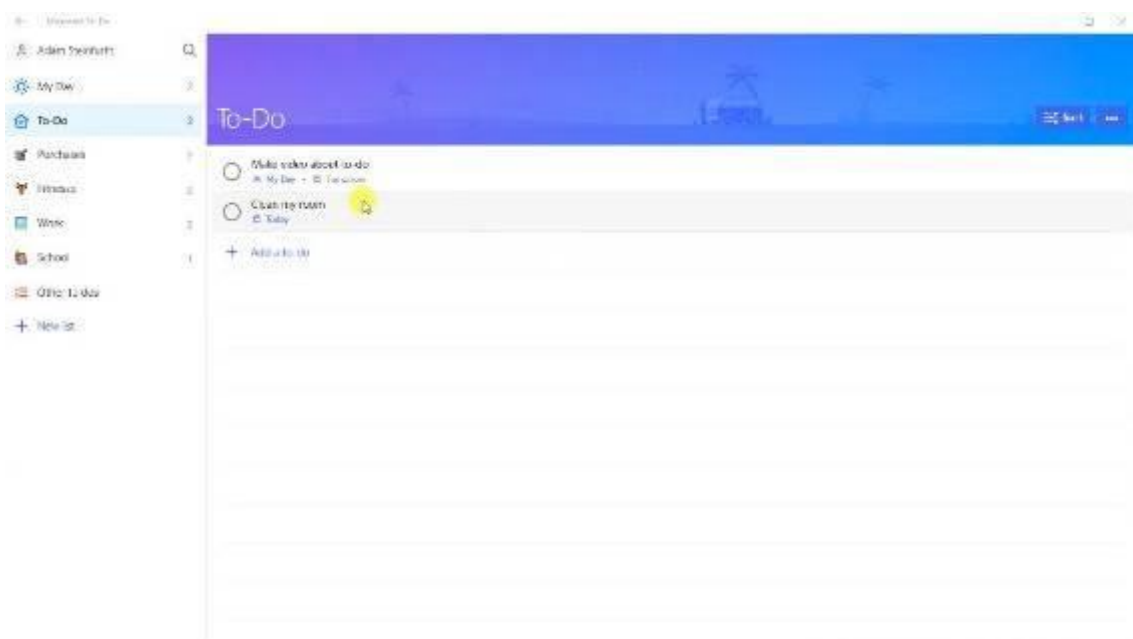


Рисунок 1.5 – Інтерфейс Microsoft To-Do

1.4 Функціональні можливості розроблюваного веб-застосунку

Перед тим як почати розробку веб-застосунку, необхідно визначити якими функціональними та програмними можливостями буде володіти розроблюваний менеджер задач.

У попередньому розділі були розглянуті існуючі аналоги. Отже, тепер можна перейти до формування списку можливостей, які буде мати розроблюваний веб-застосунок.

Основними з них є можливості:

- додавати групу;
- змінювати назву групи та видаляти її;
- змінювати колір фону групи;
- показувати дату створення групи;
- додавати задачу;
- відмічати задачу як зроблену;
- додавати до конкретної задачі фото;
- видаляти задачу;
- змінювати фон всього застосунку на інший;
- бути простим у використанні і інтуїтивно зрозумілим;
- не мати зайвих функцій;
- можливість використовувати застосунок на всіх існуючих пристроях;
- зробити застосунок таким, щоб можливо було його використовувати ще й як десктопний, тобто реалізувати можливість завантаження застосунків з сайту.

1.5 Формальна постановка вирішуваної задачі

В результаті проведеного аналізу та оцінці інших застосунків було виявлено, що менеджерів задач для мобільних та десктопних пристроїв

більше, ніж онлайн (веб) менеджерів. Також їх функціонал і можливості більші.

На підставі цього визначені необхідні функціональні можливості, вимоги до веб-застосунку. Вирішено розробити простий, зрозумілий менеджер задач, який можливо відкрити у браузері будь-якої системи або завантажити його з сайту і використовувати, і тим самим не залежати від місця, часу та використовуваного пристрою.

Зробивши аналіз предметної області та порівняльний аналіз аналогів, були визначені етапи створення веб-застосунку Менеджер задач:

- порівняльний аналіз веб-аналогів;
- розробка макету застосунку;
- створення середовища для зручної розробки програми;
- верстка застосунку;
- розробка користувацького UI та сховища Redux;
- тестування та розгортання проекту до безкоштовного сервісу.

Зробивши огляд першого етапу можна винести важливу та загальну інформацію, яка визначає та виявляє необхідні вимоги до розроблюваного застосунку. Також цей етап допомагає зрозуміти для кого розроблюється застосунок, з якими обмеженнями потрібно його зробити і чого потрібно уникнути.

Другий етап передбачає розробку інтерфейсу застосунку, який буде впливати на зручність використання для користувачів. При розробці інтерфейсу веб-застосунку потрібно враховувати зручність, простоту додатку та пристрої з яким буде взаємодіяти користувач (комп'ютер, планшет або смартфон). Комфорт та зручність повинні бути головним критерієм в побудуванні і створенні дизайну застосунку.

Третій етап передбачає налаштування зручної середи для розробки застосунку. Сюди входять вибір IDE, підключення необхідних модулів, створення папок, файлів та інше.

Четвертий етап включає в себе перетворення другого етапу в електронний вигляд. Тобто ми беремо макет інтерфейсу та за допомогою вибраних інструментів робимо каркас застосунку, з якими надалі буде взаємодіяти користувач.

На п'ятому етапі до готової верстки (готового електронного макету) додається логіка взаємодії з користувачем. Це натиснення якихось кнопок, вибір якої-небудь інформації та інше.

Шостий етап включає тестування застосунку та його розгортання на будь-якому безкоштовному або платному сервісі. Цей процес дозволяє викласти проект у навколишній світ та дати змогу користуватися всім готовим застосунком.

1.6 Висновки за розділом 1

В даному розділі були визначені об'єкт та предмет дослідження дипломної роботи. Був проведений теоретичний аналіз предметної області. Зроблений порівняльний аналіз з аналогами розроблюваного застосунку. Були визначені функціональні можливості розроблюваного веб-застосунку. А також була описана формальна постановка вирішуваної задачі.

2 ОПИС ПРОГРАМИ

В даному розділі буде описана структура програми, вимоги, функціонування програми та середовища її розробки, що дозволяє реалізувати застосунок, описаний у попередньому розділі.

Будова структури застосунку, вибір правильного та зручного середовища розробки є важливою складовою проектування застосунку та подальшого його підтримання.

2.1 Структура веб-застосунку

Веб-застосунок являє собою особливий тип програми, побудований по архітектурі клієнт-сервер, тобто складається з фронтенда та бекенда. Його особливість полягає в тому, що веб-застосунок знаходиться і виконується на сервері, а клієнт при цьому отримує тільки результат роботи. Робота застосунку базується на отриманні запитів від користувача (клієнта). Їх обробці і видачі результату. Передача запитів і результатів їх обробки відбувається через Інтернет.

2.1.1 Які бувають веб-застосунки

Backend (бекенд або серверна частина програми) працює на віддаленому комп'ютері, який може знаходитися де-завгодно. Вона може бути написана на різних мовах програмування: PHP, Python, Ruby, C # та інших.

Frontend (фронтенд або клієнтська частина програми) виконується в браузері користувача. Ця частина застосунку розроблюється на мові програмування Javascript за допомогою так званого чистого JavaScript або різних бібліотек та фреймворків. Якщо застосунку не потрібно зберігати якісь великі об'єми даних на сервері чи якось взаємодіяти з користувачем

більш складними способами, то він може складатися тільки з клієнтської частини. Це можуть бути, наприклад, фоторедактори та інші прості застосунки.

SPA (Single Page Application) – односторінковий інтерактивний застосунок. Він не тільки знаходиться на одній сторінці, а й, подібно повноцінної програмі, є інтерактивним. Так інформаційний вебсайт може складатися з однієї сторінки, але, по суті, не бути SPA. У односторінковому застосунку користувач, перемикаючись між вкладками, залишається на одній сторінці. Причому завантажуються і оновлюються тільки потрібні частини контенту, що грає на користь швидкодії SPA.

MPA (Multi Page Application) - традиційні багатосторінкові веб-застосунки. Коли користувач взаємодіє з веб-сайтом, завантажуються нові HTTP-сторінки. Тому обмін даними відбувається повільніше, ніж в SPA. Особливо, якщо є проблеми з інтернет-з'єднанням або з хостингом сайту.

PWA (Progressive Web Application) – прогресивний веб-застосунок близький за своїми можливостями, функціями і якістю користувацького досвіду до нативних комп'ютерів та мобільних застосунків. Чіткою різниці між HE-PWA і PWA застосунком немає. Але можна виділити ряд характеристик. Зокрема PWA повинно містити проксі-шар (Service Worker) і Web App маніфест. По суті, браузер виступає віртуальною машиною для запуску веб-додатків, подібно до того, як Windows запускає exe-файли, а Android – apk [4].

2.2 Обґрунтування вибору типу веб- застосунку

В цьому підрозділі буде описано що таке PWA, його архітектура, технології, переваги та недоліки та обґрунтування вибору використання цієї технології.

2.2.1 Архітектура PWA

PWA – це веб-застосунок, розроблений за допомогою деяких технологій, які надають змогу впевнено досягти певної мети при розробці.

Основними пунктами цієї мети є:

- надійність – застосунок повинен швидко завантажуватися і відображатися відразу ж, незалежно від якості статусу Інтернет з'єднання;
- швидкість – обмін даними по мережі Інтернет відбувається дуже швидко.

При розробці такого виде веб-застосунку розробник володіє наступними інструментами: Service Worker, Push Notifications та ін.. Також є можливість вказати наступні моменти: веб-застосунок повинен бути швидким в завантаженні, працювати на слабкому з'єднанні чи працювати офлайн при необхідності.

2.2.2 Які технології використовує PWA

Коротко розглянемо основні складові PWA:

- Service Worker – це проксі-шар між серверною і клієнтською частиною. Він знаходиться в браузері, через нього проходять всі запити. Таким чином, є два фронтенд шари – в одному прописується інтерфейс, в іншому – логіка. Це дозволяє робити повноцінні програми для веб;
- HTTPS – у PWA-застосунку вимагається, щоб кожен застосунок і його ресурси передавалися по HTTPS протоколу для більшої безпеки;
- WEB App Manifest – це JSON-файл, в якому прописується назва програми, іконка, тип відображення PWA: standalone, fullscreen або browser. Маніфест дозволяє закріпити PWA на домашньому екрані смартфона разом з нативними додатками;
- Push Notifications – спливаючі підказки та повідомлення [4].

2.2.3 Переваги та недоліки PWA

Переваги PWA:

- вони працюють на будь-якому пристрої. Інтерфейс PWA підлаштовується під ширину екрану комп'ютера або телефону;
- розробнику досить створити одну програму для всіх платформ;
- UX порівнюємо з додатками для ПК і мобільного. Користувач не відчуває різниці між використанням повноцінної, встановленої на пристрій програми і PWA;
- оновлення в режимі реального часу завдяки Сервіс Воркер;
- PWA працює тільки по HTTPS протоколу, що більш безпечно;
- індексованість в пошукових движках;
- інтерактивні функції: push-повідомлення та гео-локація;
- можна закріплювати іконку на домашньому екрані телефону;
- більшість функцій доступні офлайн, якщо виникає необхідність дочекатися відповіді з сервера, PWA включає екран завантаження або окремі елементи інтерфейсу, які сигналізують про те що дані завантажуються. Це направлено на поліпшення користувацького досвіду;
- посиленням на PWA-додаток можна ділитися, як на звичайний вебсайт.

Недоліки PWA:

- не підтримується на старих смартфонах;
- споживання енергії вище, ніж у простого сайту;
- на відміну від мобільних додатків, PWA не представлені в магазинах AppStore і Google Play;
- оскільки PWA не вимагають верифікації в магазинах додатків, гарантії безпеки для користувачів трохи нижче. З іншого боку, для розробника відсутність зайвих погоджень - це плюс;
- у PWA немає доступу до деяких апаратних ресурсів пристроїв. Наприклад, до Bluetooth або NFC [4].

2.2.4 Вибір технології PWA

В застосунку Менеджер задач була використана технологія PWA, тому що, як було вже описано вище, дозволяє швидко та легко використовувати застосунок, запускати його навіть тоді, коли немає доступу в Інтернет. Це також прогресивна та потужна технологія за якою, як багато хто вважає, майбутнє.

Є серед списку тих компаній, які використовують цю технологію, кілька дуже відомих, а саме: Google (Google Maps), Twitter, Pinterest, Starbucks. Тобто актуальність і популярність PWA говорить сама за себе.

2.3 Функціонування застосунку

Для розробки цього Менеджеру задач були використані стандартні мови програмування для створення веб-застосунків. Такі як: HTML, CSS, JavaScript. Крім того було використано деякі інші бібліотеки, інструменти, які перераховані нижче.

Основні бібліотеки, які використовувалися для розробки цього застосунку були React та Redux. Вони значною мірою допомагають швидко та легко створювати як великі програми, так і малі.

React – це JavaScript бібліотека для розробки користувацьких UI. Він дозволяє відображати компоненти в якості видимих елементів на сторінці в браузері. Ціла екосистема навколо React дозволяє створювати односторінкові додатки. В React використовується компонентна структура, яка дозволяє ділити великий застосунок на багато малих компонентів, що дозволяє більш зручно та легко працювати та розробляти веб-застосунки.

Оглянемо деякі з основних концепцій, які входять в React-застосунок:

- компоненти. Це інкапсульовані блоки функціональності, які є основою в React. Вони використовують дані (властивості і стан) для рендеринга призначених для користувача інтерфейсів;

- бібліотеки React. React містить набір основних бібліотек. Основна бібліотека React працює з інтерактивними бібліотеками react-dom і react-native і орієнтована на специфікацію і визначення компонентів. Вона дозволяє створювати дерево компонентів, які можна використовувати для засобів візуалізації (рендеринга) браузера або іншої платформи. React-dom – один з таких засобів, призначений для рендеринга в браузері і на стороні сервера;

- сторонні бібліотеки. React не поставляється з інструментами моделювання даних, HTTP-викликів, бібліотеками стилів або іншими поширеними елементами інтерфейсного застосунку. Тому використовує в своєму застосунку додатковий код, модулі або інші інструменти [5].

Redux – це JavaScript бібліотека, побудована на моделі Flux, розроблена для вирішення проблеми розуміння того, як саме повинні змінюватися дані в веб-застосунку. Ця бібліотека надає стан застосунку за допомогою єдиного незмінного об'єкта, де зберігаються всі дані [6].

Крім цього, при розробці веб-застосунку були використані такі бібліотеки як:

- React-icons – бібліотека для використання у застосунку декоративних іконок;

- LocalStorage – це локальне зберігання даних на стороні користувача, що дозволяє зберігати дані необмежено довго, оскільки дані не будуть видалятися після закриття браузера або перезавантаження комп'ютера;

- SASS – це препроцесор CSS. Це означає, що спочатку пишеться Sass-код, який потім компілюється в CSS-код для браузера. SASS може сприйматися як метамова (розшифровується як Syntactically Awesome Stylesheets - «синтаксично привабливі таблиці стилів») як щось на кшталт скороченого варіанту запису CSS-коду, який дозволяє писати код меншого обсягу [7].

2.4 Середовище розробки застосунку

Інтегроване середовище розробки (Integrated development environment або IDE) – це програма, яка дозволяє швидко та легко писати, розроблювати та тестувати застосунки будь-якої складності. Вона надає спеціальні інструменти, які значною мірою допомагають в написанні коду.

Застосунок розроблювався за допомогою програми Visual Studio Code.

Visual Studio Code (VSCode) – це легке середовище розробки, створене Microsoft, яка працює на Windows, Linux і macOS. Вона включає в себе функції розробки, такі як підсвічування синтаксису, зіставлення дужок, Intellisense, а також підтримує безліч різних мов: TypeScript, JavaScript, JSON, HTML, CSS, C #, C ++, Java і багато інших, що робить її ідеальною для розробки на різних мовах програмування з орієнтацією на веб-сторінки або Node. Основна увага приділяється розробці Node з використанням TypeScript і ASP.NET. Основна розробка з C #. Вона також має сильну підтримку Git з коробки і була створена з використанням TypeScript і Electron [8].

2.5 Висновки за розділом 2

В даному розділі було обґрунтовано вибір типу застосунку. Також було детально проаналізовано архітектуру PWA, які технології використовує, переваги та недоліки. В якості середовища розробки було обрано Visual Studio Code, оскільки він швидкий, популярний, зручний у використанні та підтримує велику кількість плагінів, які спрощують процес написання коду. Були визначені технічні та програмні засоби.

3 РОЗРОБКА ПРОГРАМИ

В цьому розділі будуть описуватися етапи розробки окремих компонентів системи, а також інтерфейсу. Буде деталізовано та обґрунтовано реалізацію кожного елементу архітектури програми. Розробку програми будет реалізовано на локальному сервері, а потім буде розміщено на безкоштовному сервісі.

3.1 Етапи встановлення необхідних модулів та фреймворків

Першим етапом буде встановлення Node.js. Node.js – це серверна технологія, яка основана на JavaScript «рушії» V8. Вона дозволяє створювати серверну частину сайту. Її також можна встановити на комп'ютер, що дозволить системі розуміти мову JavaScript та встановлювати різні залежності, модулі, бібліотеки [9].

Після встановлення Node.js з'являється доступ до двох програм командного рядка: node і системі управління пакетами Node – npm. Npm (node package manager) дає змогу вирішувати багато задач, наприклад, встановити сторонній код, керувати потоком виконання проекту і зовнішніми залежностями, а також надає можливість встановлення вільно розповсюджуваних інструментів розробки [10].

Далі потрібно створити робочий каталог. Після цього запустити консоль (командну строку).

Консоль – це спеціальний програмний продукт, який дозволяє вводити і виводити якісь системні повідомлення, а також вона надає змогу користувачеві працювати напряму з операційною системою. Консоль використовує текстовий інтерфейс, де управління програмою і програмами операційної системи виконується за допомогою текстових команд. Можна використовувати багато різних консолей. Основними є: WindowsPowerShell,

консоль Node.js, GitBush або консоль, яку надає середовище розробки програми (IDE).

Відкривши консоль потрібно вибрати робочий каталог програми. Далі командою `prx create-react-app` встановити необхідні залежності.

Create React App – це інструмент, який дозволяє швидко та зручно налаштувати робоче середовище для роботи з React. Цей інструмент заздалегідь налаштований розробниками React-бібліотеки.

Після всіх цих операцій можна приступати до встановлення необхідних модулів. Першим з них буде `redux`.

Зробити це можна за допомогою команди:

```
npm install redux --save
```

Ключ `--save` говорить npm, у який розділ додати даний модуль, тобто його буде додано в розділ `dependencies` файлу `package.json`.

Таким способом потрібно встановити інші необхідні бібліотеки та модулі, такі як `react-redux`, `node-sass`, `react-icons`, `redux-devtools-extension`. Встановлені бібліотеки та модулі за замовчуванням будуть потрапляти до папки з іменем `node_modules`, звідки Node буде їх витягати і використовувати у проєкті.

React-redux допомагає нам з'єднати дані, які ми будемо зберігати в Redux з бібліотекою React. Це дозволяє об'єднати якісь дані з UI.

Node-sass це пакет NPM, який компілює Sass в CSS (і робить це дуже швидко). Sass – найпопулярніший препроцесор CSS.

React-icons дозволяє використовувати в проєкті необхідні декоративні іконки.

Redux DevTools – це додаткове розширення для браузера, яке дозволяє спостерігати за всіма процесами, які відбуваються з застосунком. Зазвичай його використовують для нагляду за Redux.

3.2 Структура папок та файлів

Правильна структура папок та файлів допомагає швидко та легко орієнтуватися, переміщуватися поміж елементами програми.

На рисунку 3.1 зображено структура папок застосунку Менеджер задач.

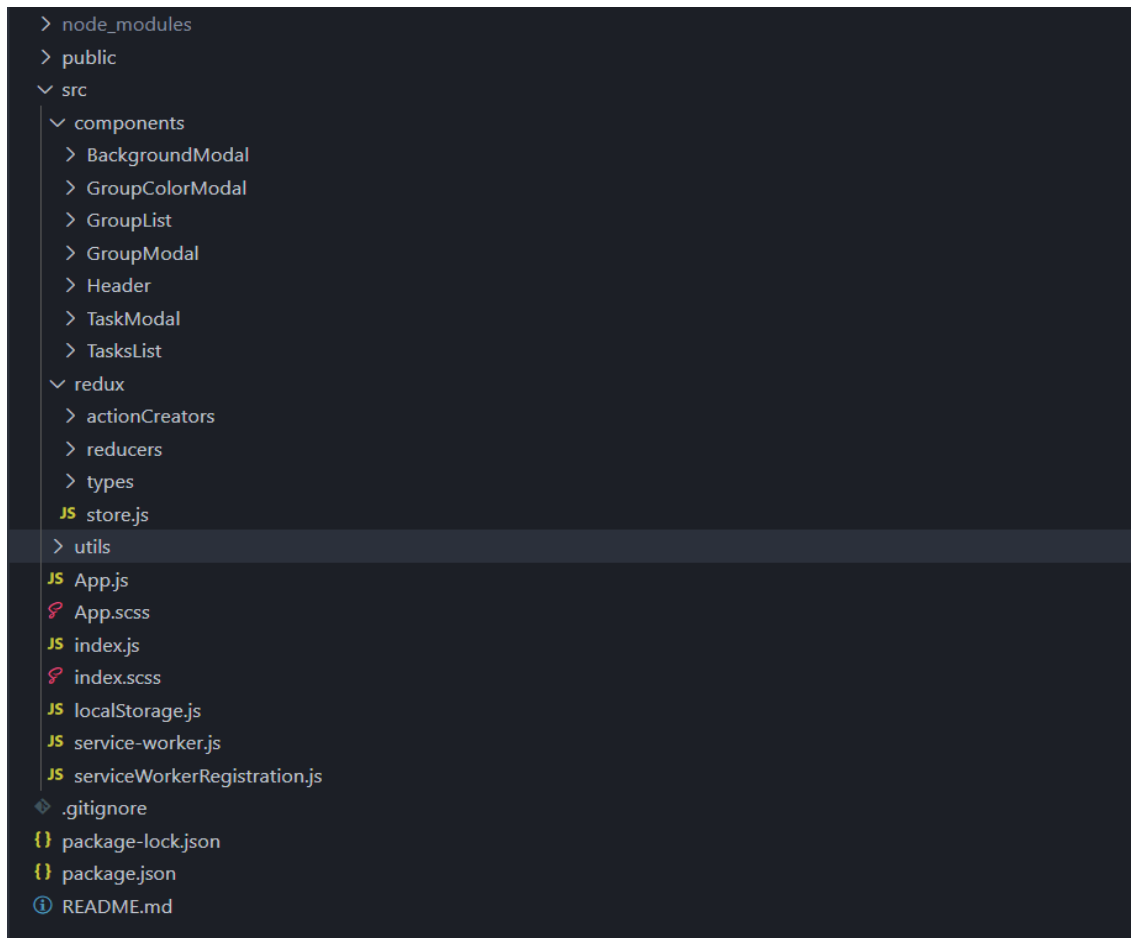


Рисунок 3.1 – Структура папок

В папці node_modules зберігаються модулі, які використовуються в проєкті.

В папці public зберігаються файли, які будуть використовуватися в продакшені. Це файли:

- index.html – основний файл, де підключаються шрифти, css та скрипти;

- ico, png – файли іконок;
- manifest.json – файл, де описуються які розміри, кольори використовуються для різних форматів екранів; описуються назва застосунку, кольори фону та інше.

Вони збираються в кілька єдиних файлів з інших файлів, які розташовуються в папці src за допомогою webpack, який вже входить в збірку create-react-app від розробників бібліотеки React. Можливо самому встановити та налаштувати збірку webpack, але це краще робити для більш великих проектів.

Webpack є інструментом, який являє собою завантажувач і пакувальник модулів, що дозволяє згрупувати файли програми в пакети. Крім цього, він дозволяє оптимізувати їх розміри, щоб в той же самий пакет помістилося більше коду [11].

В папці src зберігаються всі папки та файли, в яких і реалізовується застосунок. В папках components знаходяться основні UI компоненти програми, такі як:

- Header – містить верхню панель з назвою застосунку та перемикачем фону;
- GroupList – це компонент, в якому знаходяться список груп, кнопка додавання групи та компонент GroupItem, де описується логіка однієї групи;
- TasksList – цей компонент відображає задачі;
- GroupModal – це компонент, який відкривається як модальне вікно, коли ми натискаємо на кнопку «Add group» і вводимо назву групи;
- GroupColorModal – компонент, який знаходиться зліва внизу групи та, коли ми на нього натискаємо, з'являється маленьке модальне вікно, де ми можемо вибрати задній фон групи серед представлених кольорів;
- TaskModal – модальне вікно, яке з'являється при натисканні на іконку у вигляді трьох точок та надає можливість відмітити задачу як зроблену, вибрати зображення для задачі та видалити задачу;

- `BackgroundModal` – ще одне модальне вікно, яке з’являється при натисканні у шапці застосунку кнопки `Background` та дозволяє вибрати серед списку зображень і змінити фон всього застосунку.

В папці `redux` зберігаються основні дані програми:

- `actionCreators` – це папка, в якій лежать файли `groupActionCreator.js` та `backgroundActionCreator.js` – це файли які створюють події, пов’язані з групами та задачами і вибором фону застосунку;
- `types` – папка з файлом `types.js` – це файл з набором констант, які описують події;
- `reducer` – це чиста функція яка буде відповідати за оновлення стану; тут реалізовується логіка у відповідність з якою буде відбуватися оновлення полів `store`; тут є файли `rootReducer.js`, який об’єднує два інших ред’юсери `backgroundReducer.js` та `groupReducer.js`;
- файл `store.js` – це файл, в якому створюється глобальне сховище даних.

Також є папка `utils`, в якій є папка `images`, де зберігаються зображення, які використовуються для фону застосунку; файл `background.js` – це файл, в якому зберігається об’єкт з зображеннями, який ми потім відображаємо при виборі кольору групи; файл `groupBackgroundColors.js` – тут зберігається об’єкт з кольорами, які будуть відображені серед списку кольорів для вибору фону групи.

Далі в папці `src` є файл `App.js` – це файл, куди підключаються всі інші файли з папок `components` та `redux`.

Також є файл `index.js`, куди підключається файл `App.js` та деяка інша логіка застосунку. Цей файл ще називають точкою входу всього застосунку. Коли ми будемо збирати наш застосунок у продакшн, то `index.js` буде слугувати таким собі контейнером для всіх інших файлів та за допомогою `webpack` цей файл буде підключатися в файл, який знаходиться в `public/index.html`. Цей файл і буде слугувати основним фалом веб-застосунку.

Файл `localStorage.js` – це файл, де зберігається логіка того, як працює сховище даних `LocalStorage`.

В файлах `service-worker.js` та `serviceWorkerRegistration.js` описується логіка PWA-застосунку.

Файл `.gitignore` слугує файлом, який скриває конфігураційні файли (паролі, тимчасові файли та папки), які не потрібно передавати до віддаленого репозиторію `GitHub`, `Bitbucket` чи `GitLab`.

Віддалений репозиторій (`git`, або система управління версіями) – являє собою версії розроблюваного проекту, який зберігається в Інтернеті або ще десь в мережі. Їх може бути кілька, кожен з яких як правило доступний для тільки для читання, або для читання та запису [12].

Далі розташовується файл `package.json` – це конфігураційний файл та файл `README.md` для опису проекту, який додано до репозиторію.

3.3 Реалізація користувацького UI

У цьому підрозділі буде описано етапи реалізації користувацького інтерфейсу. Які інструменти використовувалися та процес написання програми.

3.3.1 Створення макету

Перш ніж приступити до розробки та верстки застосунку, треба розробити макети, які можна виконати в онлайн-редакторі або завантаживши на комп'ютер редактор.

Для створення макету Менеджера задач було використано онлайн редактор `Figma`.

`Figma` – це онлайн-сервіс для дизайнерів, які проектують інтерфейси і веб-розробників. Розробка інтерфейсів проходить в онлайн-застосунку.

Далі наведені макети Менеджера задач для комп'ютерів (рис.3.2), планшетів (рис. 3.3) та мобільних пристроїв (рис. 3.4).



Рисунок 3.2 – Макет менеджера задач для комп'ютерів

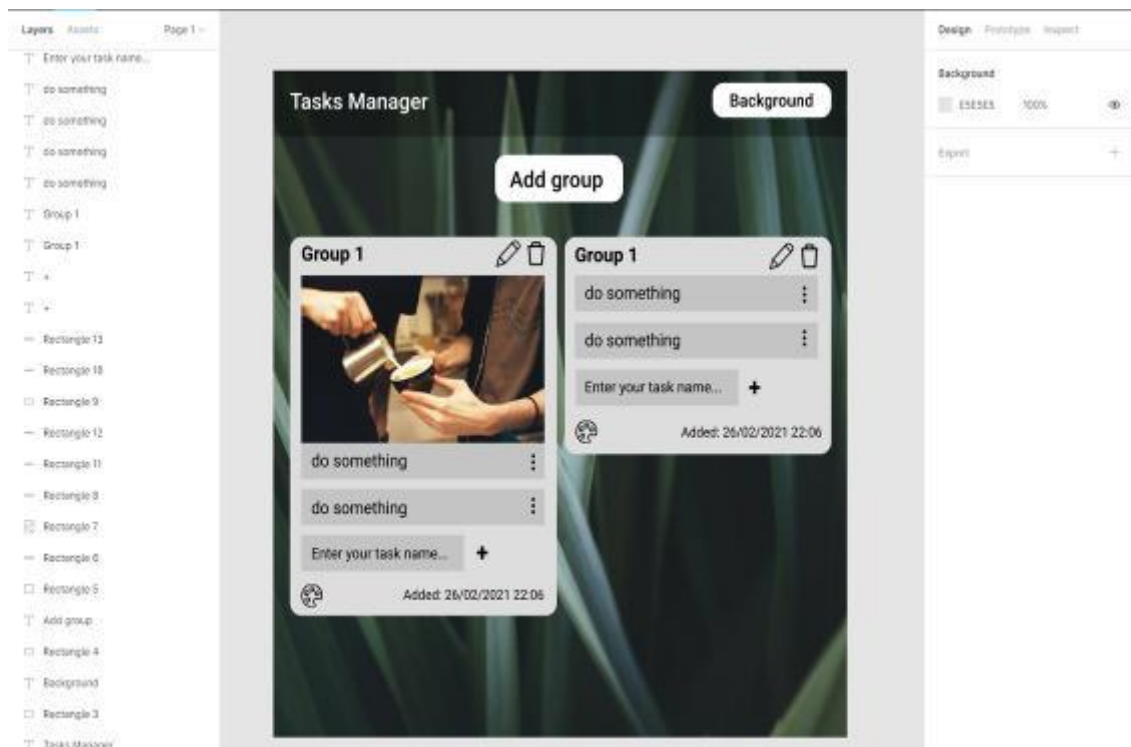


Рисунок 3.3 – Макет для планшетів

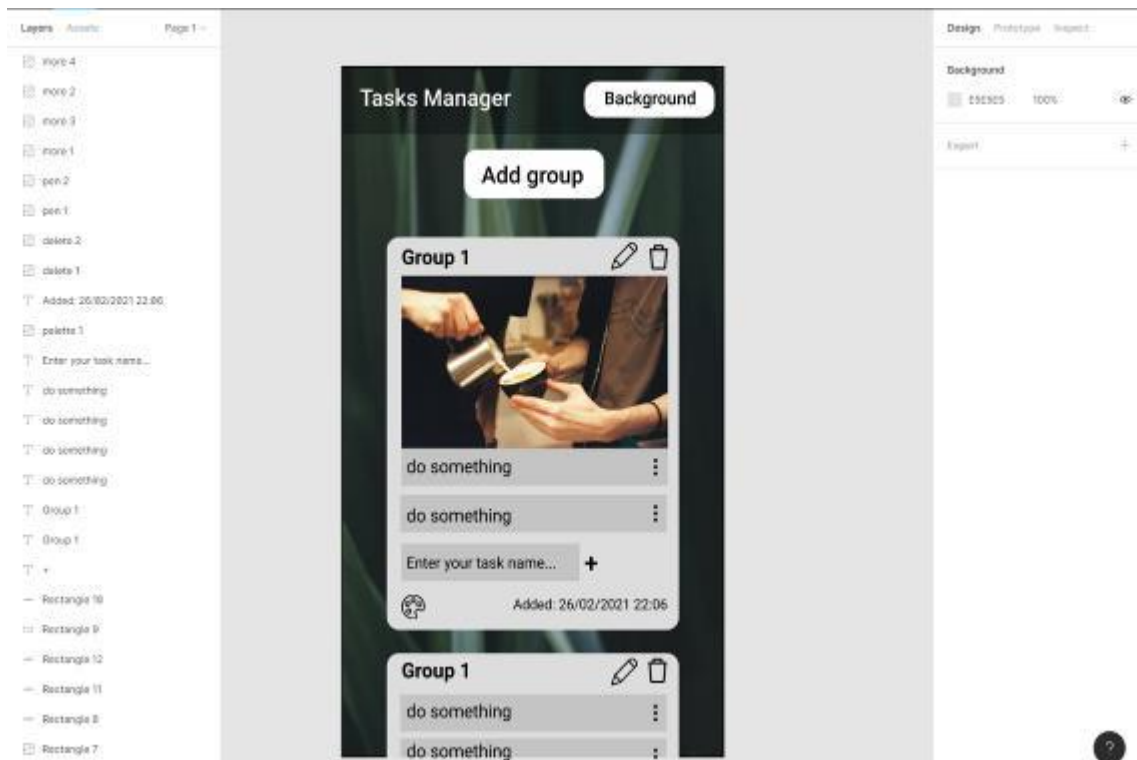


Рисунок 3.4 – Макет менеджера задач для мобільних пристроїв

3.3.2 Верстка та розробка користувацького UI

Після створення макетів можна приступати до верстки застосунку.

Сторінки застосунку Менеджер задач було зверстано за допомогою JSX та SCSS мов програмування.

JSX – це спеціальний розширений синтаксис бібліотеки React, який дозволяє об'єднати та писати HTML та JavaScript в одному файлі. В цьому і є фішка бібліотеки React.

SCSS – це розширений синтаксис мови стилізування сторінок, так званий CSS-препроцесор, який має свій власний синтаксис, який дозволяє писати більш чистий і простий CSS-код. Так як браузер не розуміють такий синтаксис, він компілюється в звичайний CSS, який розуміє браузер. Він є одним з найбільш використовуваних CSS-препроцесорів. Цей препроцесор дозволяє використовувати змінні, міксіни, імпорти і багато чого іншого.

Також було використано кілька іконок для таких кнопок, як змінити тему, відмітити зроблену задачу і видалити задачу. Всі іконки були імпортовані з бібліотеки react-icons та використані в розмітці сторінок.

При ініціалізації проекту за допомогою команди create-react-app була створена стандартна папка React-проекту src. Тут зберігаються всі файли стилів, розмітки, скриптів, зображень та інші. Були створені такі файли, як index.js та index.scss. index.js - це головний файл проекту, де імпортується такий файл як App.js – це другий по важливості файл, куди імпортуються всі інші файли, так звані компоненти, з яких складається застосунок. Index.css – тут зберігаються базові стилі проекту.

Далі була створена папка components. В ній зберігаються компоненти.

Далі будуть описані компоненти по ієрархії, тобто як вони відображаються у застосунку.

Першою підпапкою в папці components є Header – там зберігаються файли, які відносяться до верхньої частини застосунку. Це файли Header.jsx та Header.scss. Тут розташовані назва застосунку та кнопка відкриття модального вікна для вибору фону.

Другою підпапкою є папка GroupList – тут знаходяться два файли GroupList.jsx та GroupList.scss – це файли, де розташовуються кнопка додавання групи, самі групи та їх стилі. Також тут є підпапка GroupItem з файлами GroupItem.js та GroupItem.scss, де описується логіка однієї групи та їх стилі, а саме змінення назви групи, видалення групи, відображення дати створення групи та вибір кольору.

Третьою підпапкою є папка TasksList, де знаходяться папка Task з файлами TasksList.jsx і TasksList.scss. У компоненті TasksList.jsx беруться данні зі сховища redux, а саме задачі та за допомогою функції перебору map відображаються.

Четвертою підпапкою є TaskModal, де розташовані файли TaskModal.jsx і TaskModal.scss. Вони відповідають за логіку та стилі модального вікна, яке відкривається, коли ми натискаємо на кнопку у вигляді

іконки трьох точок, яку ми імпортували з бібліотеки react-icons. У цьому модальному вікні знаходяться також імпортовані іконки у вигляді кнопки, які дозволяють відмічати задачу як зроблену, додавати зображення та видаляти задачу.

П'ятою підпапкою є папка GroupColorModal, де знаходяться файли GroupColorModal.js та GroupColorModal.scss, де реалізується логіка відкривання та вибору кольору.

Шостою підпапкою є GroupModal, де розташовані файли GroupModal.jsx та GroupModal.scss, які відповідають за логіку відкривання модального вікна при натисканні кнопки «Add group» та додавання нової групи до списку.

Останньою підпапкою є BackgroundModal, де знаходяться файли BackgroundModal.jsx та BackgroundModal.scss, які описують логіку відкривання модального вікна при натисканні кнопки «Background» у шапці сайту, що дозволяють змінити фон усього застосунку.

В результаті верстки були отримані такі файли jsx:

- index.js;
- App.js;
- Header.jsx;
- GroupList.jsx;
- GroupItem.jsx
- TasksList.jsx;
- TaskModal.jsx;
- GroupModal.jsx;
- GroupColorModal.jsx;
- BackgroundModal.jsx.

Отримані файли стилів:

- index.scss;
- App.scss;
- Header.scss;

- `GroupList.scss`;
- `GroupItem.scss`;
- `TasksList.scss`;
- `TaskModal.scss`;
- `GroupModal.scss`;
- `GroupColarModal.scss`;
- `BackgroundModal.scss`.

Всі компоненти були зверстані за допомогою методології БЕМ.

БЕМ (Блок Елемент Модифікатор) – це компонентний підхід у веб-розробці. Розшифровується аббревіатура як «Блок Елемент Модифікатор». Це один з найпопулярніших на даний момент підходів, в основі якої лежить домовленість про те, як розбивати інтерфейс на певні незалежні блоки. Він дозволяє легко і швидко розробляти інтерфейси з гнучкою структурою будь-якої складності і повторно використовувати існуючий код, уникаючи «Copy-Paste» [13].

3.4 Реалізація сховища даних **Redux**

React входить до складу State Management архітектури React застосунку. Можна було вибрати state (сховище) класової компоненти чи хуків (hooks), які були додані у React з виходом оновлення 16.8, які дозволяють використовувати стан і інші можливості без написання класів. В менеджері задач використовується `redux`, тому що застосунок великий і є багато даних. В `redux` є багато функцій і можливостей, які дозволяють легко і швидко маніпулювати даними зі сховища, додавати якусь логіку до сховища.

Для початку потрібно створити сховище. Створюємо папку `redux`, де будемо зберігати та описувати всю логіку. Створемо файл `store.js`, де імпортуємо всі необхідні функції для створення сховища. Також тут буде логіка для додавання даних до `localStorage`. За допомогою функції `createStore` створимо сховище та експортуємо його.

Далі підключимо його у файл `index.js` та за допомогою імпортованої функції `Provider` прокинемо наш `store` (сховище) у компонент `App.js`. Тепер сховище доступне для всього застосунку.

Потім потрібно створити папку `types` з файлом `types.js`, де будуть описані константи, які будуть відповідати за певні дії. Ці константи в майбутньому дозволять запобігти багатьом помилкам.

Створимо такі константи для груп:

- `ADD_GROUP` – додавання групи;
- `DELETE_GROUP` – видалення групи;
- `OPEN_GROUP_NAME_EDIT` – відкриття поля для зміни назви групи;
- `CLOSE_GROUP_NAME_EDIT` – закриття поля для зміни назви групи;
- `EDIT_GROUP_NAME` – булевий тип даних, використовуємо для показування поля форми;
- `OPEN_GROUP_COLOR_EDIT` – відкриття модального вікна з вибором кольору фону групи;
- `CLOSE_GROUP_COLOR_EDIT` – закриття модального вікна з вибором кольору фону групи;
- `EDIT_GROUP_COLOR` – зміна кольору фону групи;
- `OPEN_GROUP_MODAL` – відкриття модального вікна для додавання групи до списку;
- `CLOSE_GROUP_MODAL` – закриття модального вікна для додавання групи до списку.

Створимо такі константи для задач:

- `ADD_TASK` – додавання задачі;
- `CHECK_TASK` – відмічання задачі як зроблена;
- `DELETE_TASK` – видалення задачі;
- `OPEN_TASK_MODAL` – відкривання модального вікна з кнопками відмічання, додавання зображення та видалення задачі;

- `CLOSE_TASK_MODAL` – закриття модального вікна з кнопками відмічання, додавання зображення та видалення задачі;

- `ADD_TASK_PHOTO` – додавання зображення до задачі;

- `DELETE_TASK_PHOTO` – видалення зображення у задачі.

Створимо такі константи для фону усього застосунку:

- `SELECT_BACKGROUND` – вибір заднього фону задачі;

- `OPEN_BACKGROUND_MODAL` – відкриття модального вікна для вибору фону застосунку;

- `CLOSE_BACKGROUND_MODAL` – закриття модального вікна для вибору фону застосунку.

Далі створимо папку `actionCreators` з файлами `backgroundActionCreator.js` та `groupActionCreator.js`. Action – це якась дія, за допомогою якої ми говоримо сховищу `state` що потрібно зробити (дія) та які дані потрібно передати до сховища. У кожного `action` обов'язкове поле `type`, де описується дія, яку ми описали у константах та поле `payload` – це дані, які ми передаємо с компоненти.

Створимо такі `action` для `groupActionCreator.js`:

- `addGroupAction` з типом `ADD_GROUP` та дані, які будуть приходити – `id`, `name`, `backgroundColor`, `changedName`, `changedBackgroundColor`, `timeOfCreating` – це поля, які будуть зберігати дані;

- `deleteGroupAction` з типом `DELETE_GROUP` та дані, які будуть приходити – `id` групи, яку потрібно видалити;

- `openGroupNameEdit` – з типом `OPEN_GROUP_NAME_EDIT` та дані, які будуть приходити – `id` групи, в якій потрібно відобразити поле для зміни ім'я;

- `closeGroupNameEdit` з типом `CLOSE_GROUP_NAME_EDIT` та `id` групи, в якій потрібно закрити поле для зміни ім'я;

- `editGroupName` з типом `EDIT_GROUP_NAME` та дані, які будуть приходити – ім'я та `id` групи;

- `openGroupColorEdit` з типом `OPEN_GROUP_COLOR_EDIT` та дані, це `id` групи, у кої потрібно відкрити модальне вікно для вибору фону групи;
- `closeGroupColorEdit` і тип `CLOSE_GROUP_COLOR_EDIT` та дані, це `id` групи, у кої потрібно закрити модальне вікно для вибору фону групи;
- `editGroupColor` з типом `EDIT_GROUP_COLOR` та дані, це `id` групи та колір;
- `openGroupModal` з типом `OPEN_GROUP_MODAL`;
- `closeGroupModal` з типом `CLOSE_GROUP_MODAL`;
- `addTask` з типом `ADD_TASK` та дані, це нова задача;
- `checkTask` з типом `CHECK_TASK` та дані, це `id` задачі;
- `deleteTask` з типом `DELETE_TASK` та дані, це `id` задачі;
- `openTaskModal` з типом `OPEN_TASK_MODAL` та дані, це `id` задачі;
- `closeTaskModal` з типом `CLOSE_TASK_MODAL` та дані, це `id` задачі;
- `addTaskPhoto` з типом `ADD_TASK_PHOTO` та даними, це `id` та вибране фото задачі;
- `deleteTaskPhoto` з типом `DELETE_TASK_PHOTO` та дані, це `id` задачі та пусте поле.

Далі створимо папку `reducers` з файлами:

- `index.js` – тут ми об'єднуємо всі наші ред'юсери;
- `tasks.js` – файл-ред'юсер, де описується логіка, яка пов'язана з задачами;
- `theme.js` – файл-ред'юсер, де описується логіка, яка пов'язана зі зміною теми;
- `filter.js` – файл-ред'юсер, де описується логіка, яка пов'язана з фільтрацією задач.

Створимо такі action для `backgroundActionCreator.js`:

- `selectBackground` з типом `SELECT_BACKGROUND` та дані, це вибране зображення;
- `openBackModal` з типом `OPEN_BACKGROUND_MODAL`;
- `closeBackModal` з типом `CLOSE_BACKGROUND_MODAL`.

Потрібний action ми експортуємо та імпортуємо там, де нам потрібна та чи інша логіка. Відправка action відбувається з компоненти за допомогою імпортованої функції `useDispatch`, яка приймає action та дані, які потрібно передати та обробити.

Reducer – це функція, яка приймає дефолтний state (опис того, як буде виглядати сховище, яке відповідає за кожну логіку) та action, який приходить з компоненти, коли користувач натискає на якусь кнопку. Далі відбувається ререндер (перемальовування) тільки тієї частини коду, де відбулася дія.

Отже, створимо два ред'юсери `backgroundReducer.js` та `groupReducer.js`, де описується логіка зміни та додавання даних. Потім потрібно створити файл `rootReducer.js`, де за допомогою імпортованої функції з бібліотеки `redux` об'єднаємо ці два створені ред'юсери.

3.5 Розміщення застосунку на сервісі Netlify

Netlify – це веб-платформа, яка дозволяє легко розгортати веб-проекти за допомогою простих налаштувань.

Для розгортання застосунку спочатку потрібно авторизуватися на Netlify. Це можна зробити за допомогою GitHub, GitLab, Bitbucket або електронної пошти на сторінці входу [14].

Після реєстрації та заповнення деяких даних потрібно перейти до вкладки Sites, натиснути кнопку New Site From Git. Далі потрібно вибрати до якого з трьох репозиторіїв підключитися і завантажити файли до Netlify. Менеджер задач знаходиться на GitHub, тому вибираємо його. Тепер Netlify підключиться до GitHub. Зі списку проектів (якщо вони є) треба вибрати потрібний проект. Ми перейдемо на іншу сторінку, де потрібно внизу

натиснути на кнопку `Deploy site`. Розпочнеться завантаження та налагоджування застосунку на сайті. Після всіх операцій проект буде опубліковано на Netlify та нам буде видана унікальне посилання, куди можна перейти та побачити, що застосунок працює.

3.6 Підключення PWA

Для створення PWA-застосунку потрібно у папку `src` додати два файли: `service-worker.js` та `serviceWorkerRegistration.js` – в них описується логіка створення такого виду застосунку. Далі у файлі `index.js` потрібно імпортувати файл `serviceWorkerRegistration.js` та внизу файла визвати функцію `serviceWorkerRegistration.register()`, що означає, що застосунок можна буде зареєструвати як Progressive Web Application.

Далі переходимо до браузеру, відкриваємо консоль розробника (F12) та вибираємо зверху серед списку пункт `Lighthouse` (рис. 3.5).

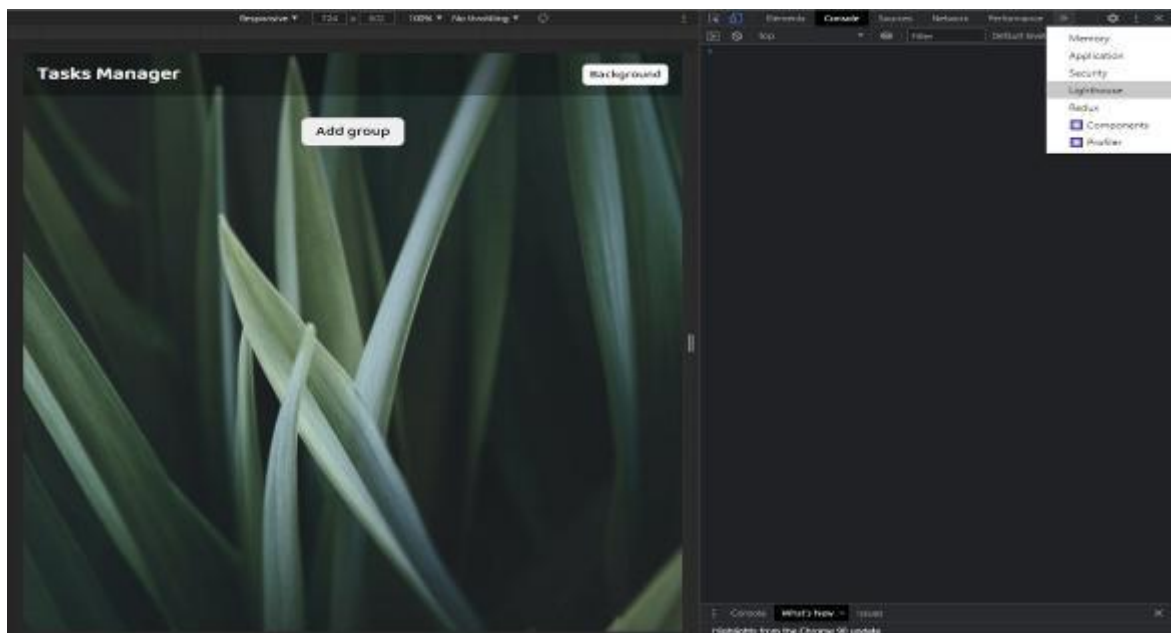


Рисунок 3.5 – Пункт Lighthouse

У вибраному розділі потрібно натиснути на синю кнопку «Generate report» (рис. 3.6). Почнеться формування звіту (рис. 3.7) та буде сформовано

звіт, який означає, що можна використовувати застосунок як PWA (рис. 3.8). Це стандартне налаштування PWA- застосунку, у більш великих проектах все описується вручну.

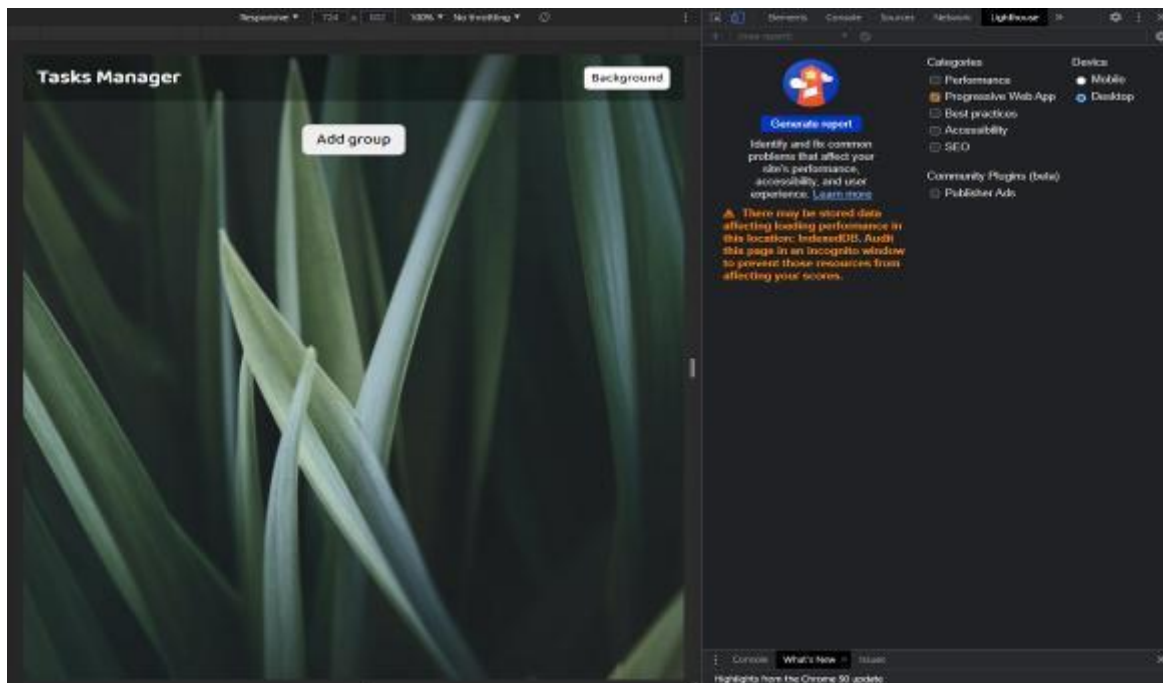


Рисунок 3.6 – Натискання на кнопку «Generate report»

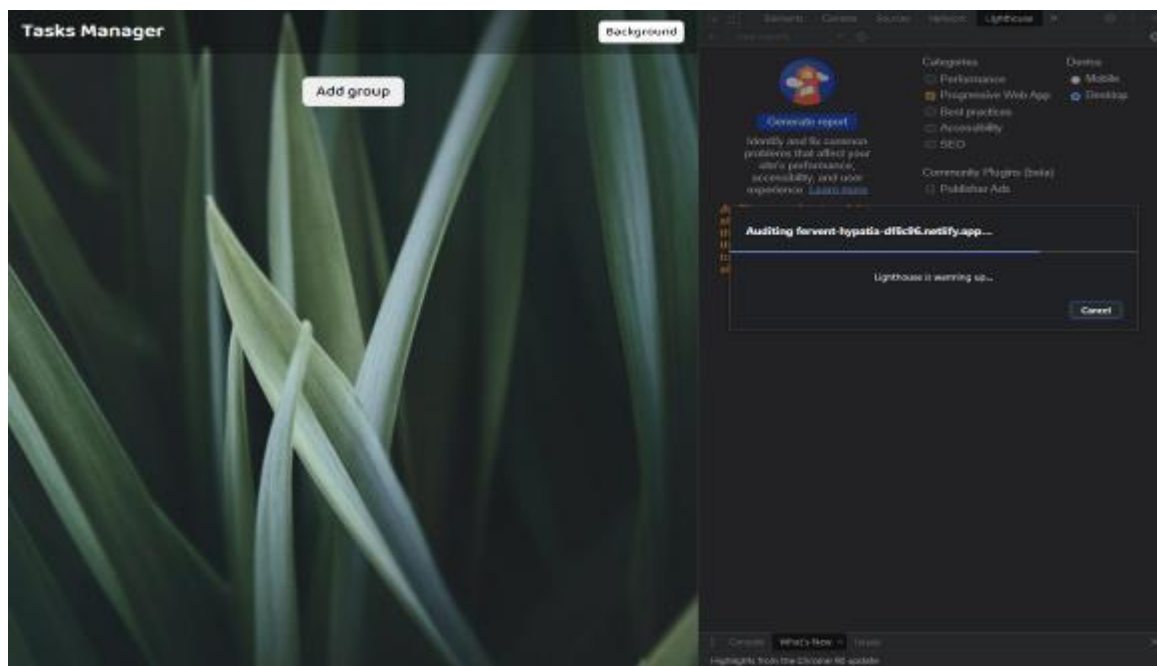


Рисунок 3.7 – Формування звіту

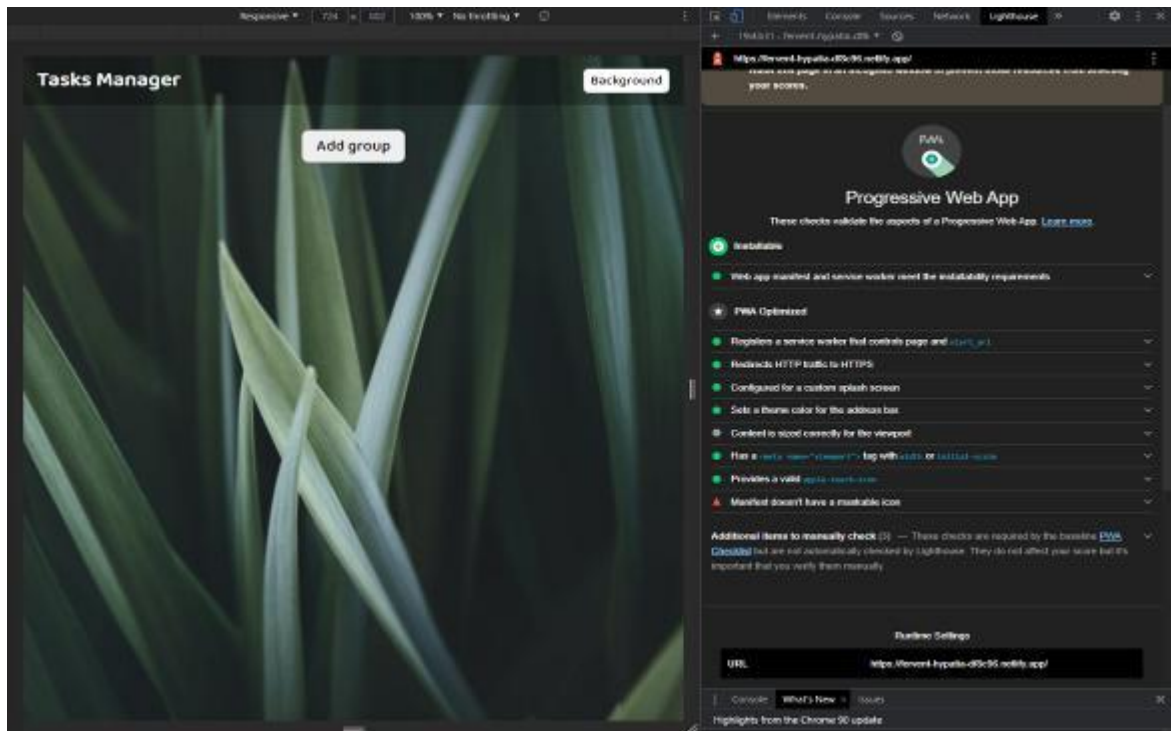


Рисунок 3.8 – Готовий звіт

3.7 Висновки за розділом 3

В даному розділі були описані етапи встановлення необхідних модулів та фреймворків для розробки програми.

Була описана структура файлів та папок, яка дозволяє орієнтуватися в структурі програми.

Був створений макет розроблюваного застосунку, який дає уяву як він буде виглядати у готовому вигляді.

Далі була проведена верстка макету та розробка користувацького UI та сховища даних за допомогою React та Redux.

У завершенні був детально описаний процес розміщення програми на безкоштовному сервісі Netlify.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

В даному розділі буде описана інструкція по використанню застосунку, методам тестування, буде проведено розгорнуте тестування на виявлення помилок при відображенні та роботі коду, валідація HTML та CSS, а також перевірка на кроссбраузерність в найпопулярніших браузерях, таких як Google Chrome, Mozilla Firefox та Microsoft Edge.

4.1 Інструкція по роботі з застосунком

Для запуску готового коду потрібно перейти за тимчасовим посиланням <https://fervent-hypatia-df8c96.netlify.app/> . Відкриється застосунок «Менеджер задач», яким можна буде користуватися як онлайн, так і без доступу до Інтернету. Також є можливість завантажити застосунок на екран робочого столу мобільного пристрою, комп'ютера, ноутбука або планшета. Застосунок адаптується під усі розміри екранів.

Користуватися застосунком можуть користувачі як старшої, так і молодшої аудиторії.

При відкритті буде показана головна сторінка, де і відбуваються всі дії взаємодії з користувачем (рис. 4.1).

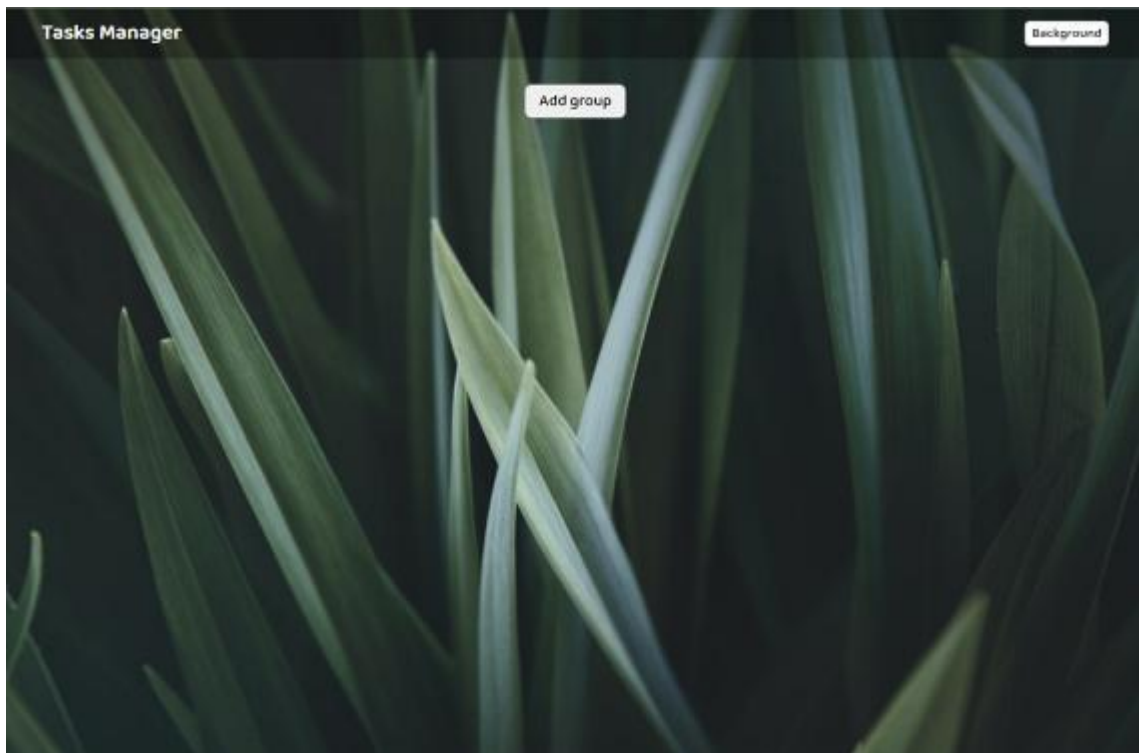


Рисунок 4.1 – Головна сторінка

Якщо натиснути на кнопку «Add group», то відкриється модальне вікно, де можна буде ввести назву групи (рис. 4.2).

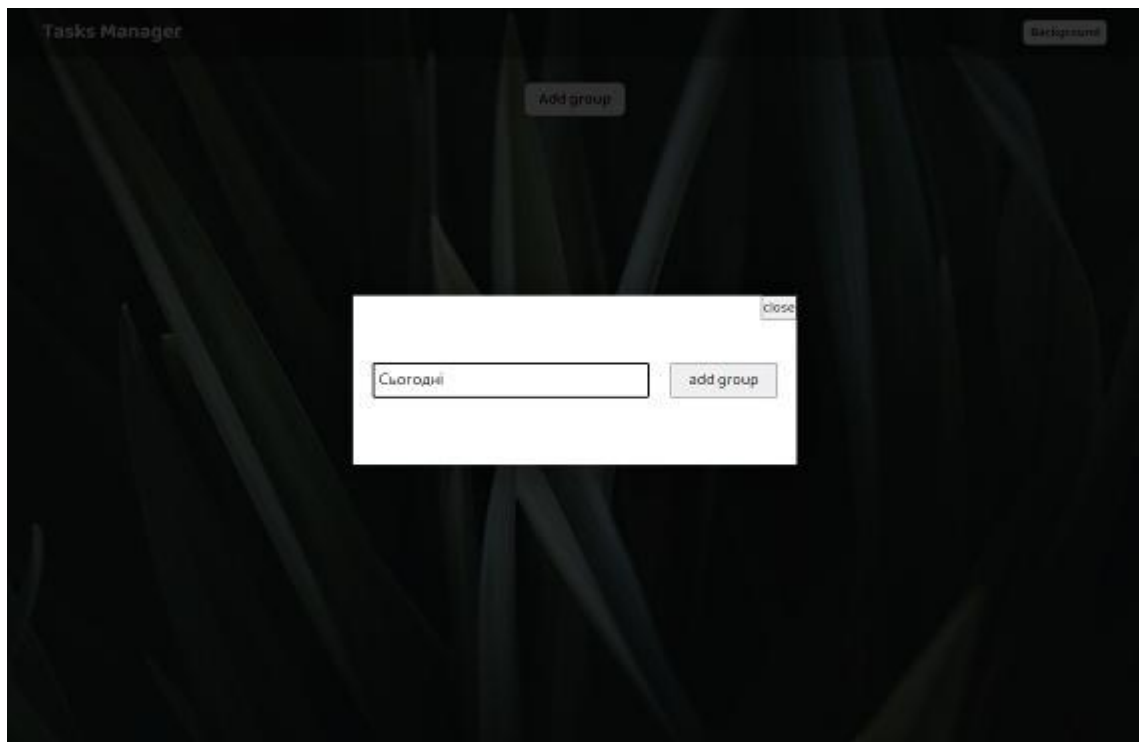


Рисунок 4.2 – Модальне вікно для введення назви групи

Після додавання групи вона відобразиться у списку груп (рис. 4.3).

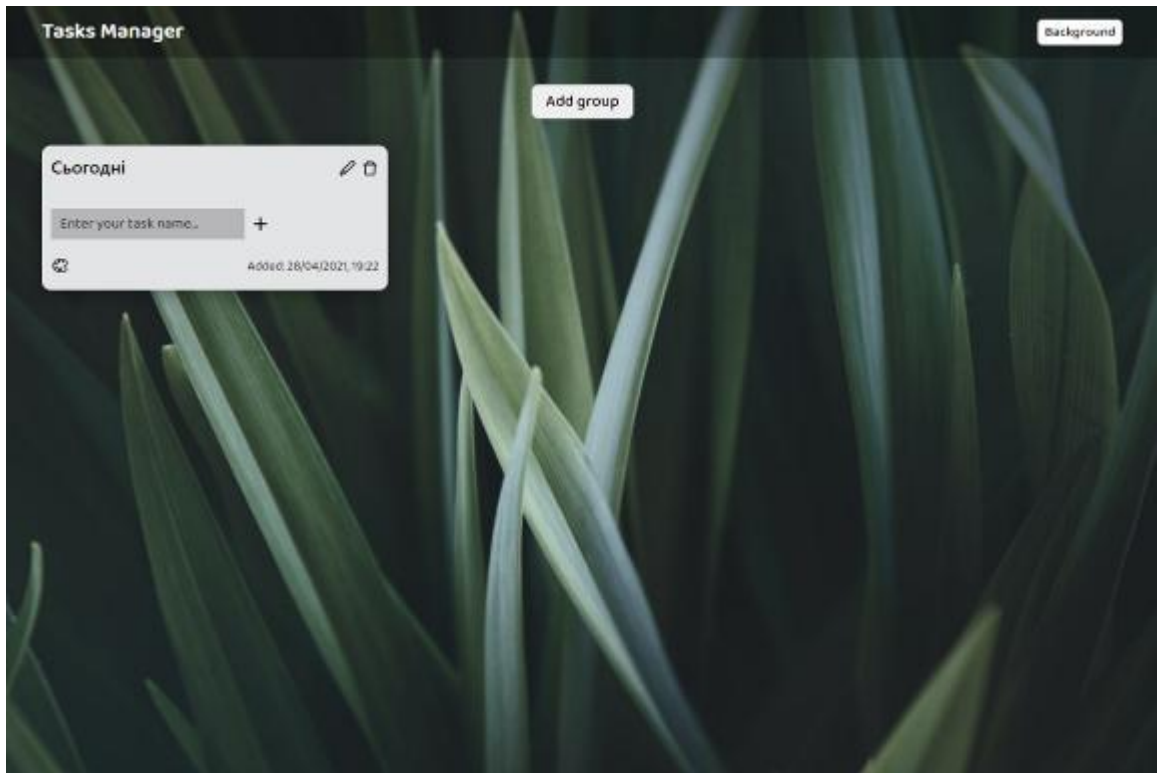


Рисунок 4.3 – Група додана до списку груп

Якщо почати вводити назву задачі в поле з назвою «Enter your task name...» (рис. 4.4) і натиснути кнопку «+», то буде додана задача до групи і буде відображена у списку задач (рис. 4.5).



Рисунок 4.4 – Введення назви задачі

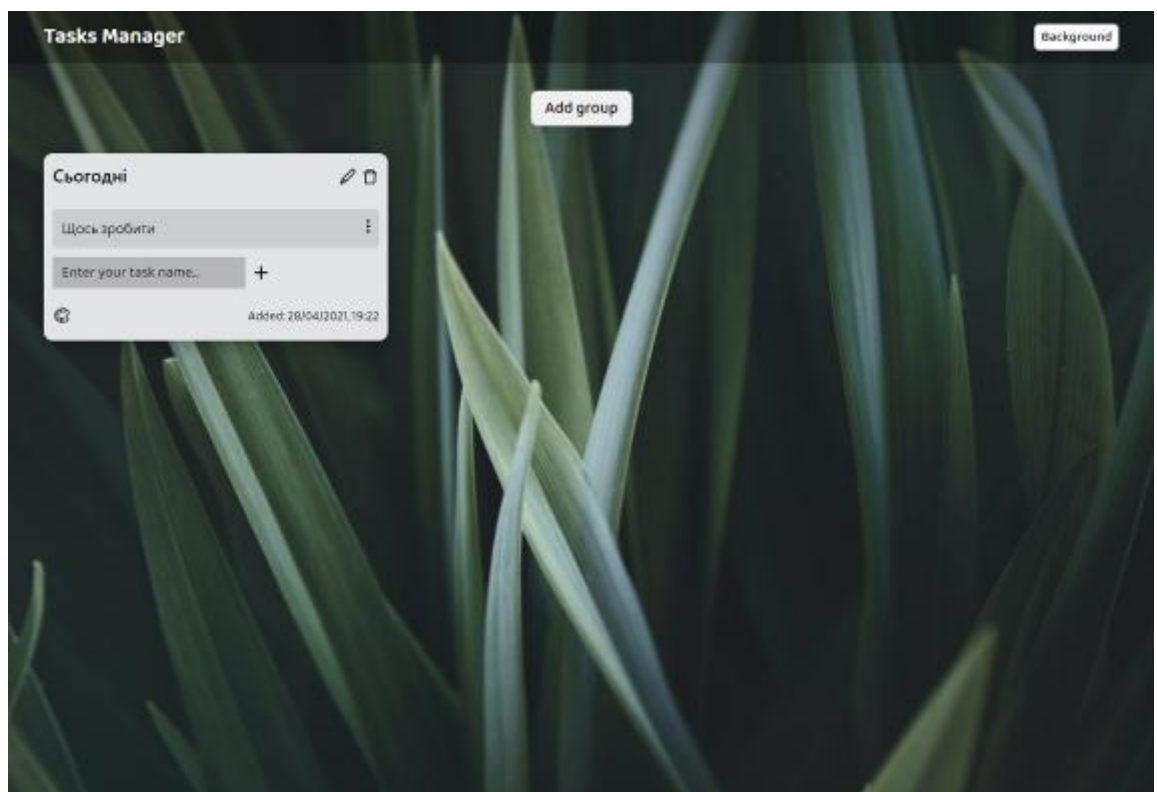


Рисунок 4.5 – Задача додана до списку задач

Якщо натиснути на кнопку у вигляді трьох точок у задачі, то відкриється модальне вікно, де буде список взаємодій з задачею (рис. 4.6).

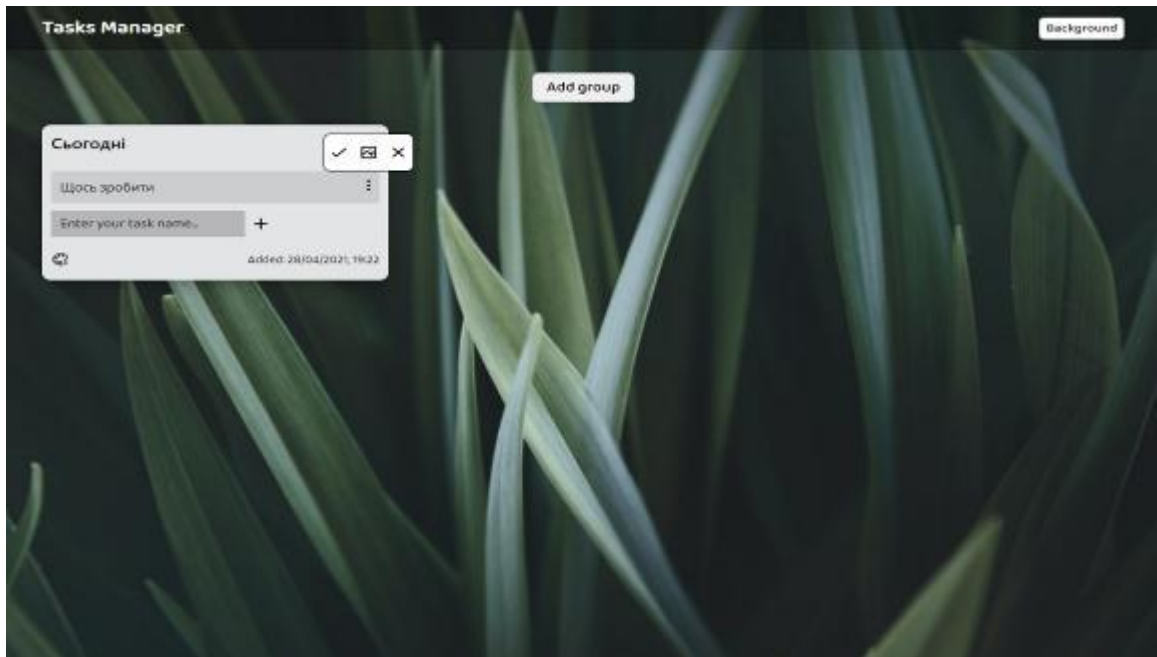


Рисунок 4.6 – Модальне вікно задачі

Якщо натиснути на кнопку у вигляді галки, то задача буді відмічена як зроблена (перекреслена) (рис. 4.7).

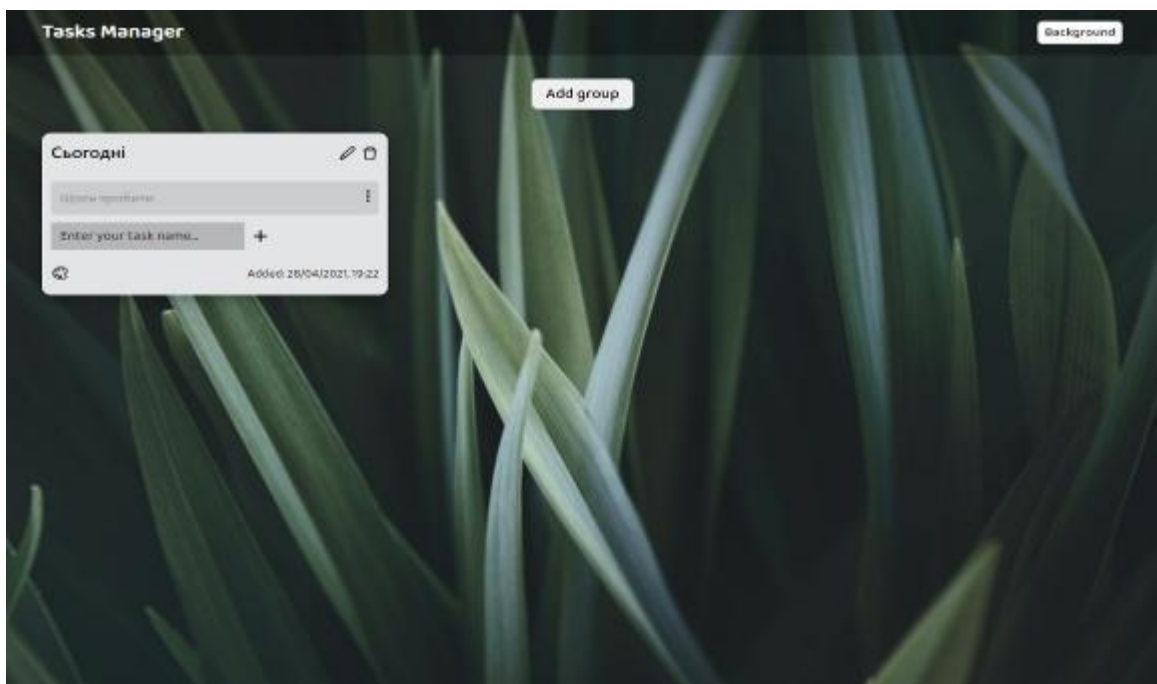


Рисунок 4.7 – Зроблена задача

Якщо натиснути на кнопку у вигляді зображення, то буде відкрито вікно вибору файлу (рис. 4.8) і, коли зображення буде вибрано, то буде додане до задачі (рис. 4.9).

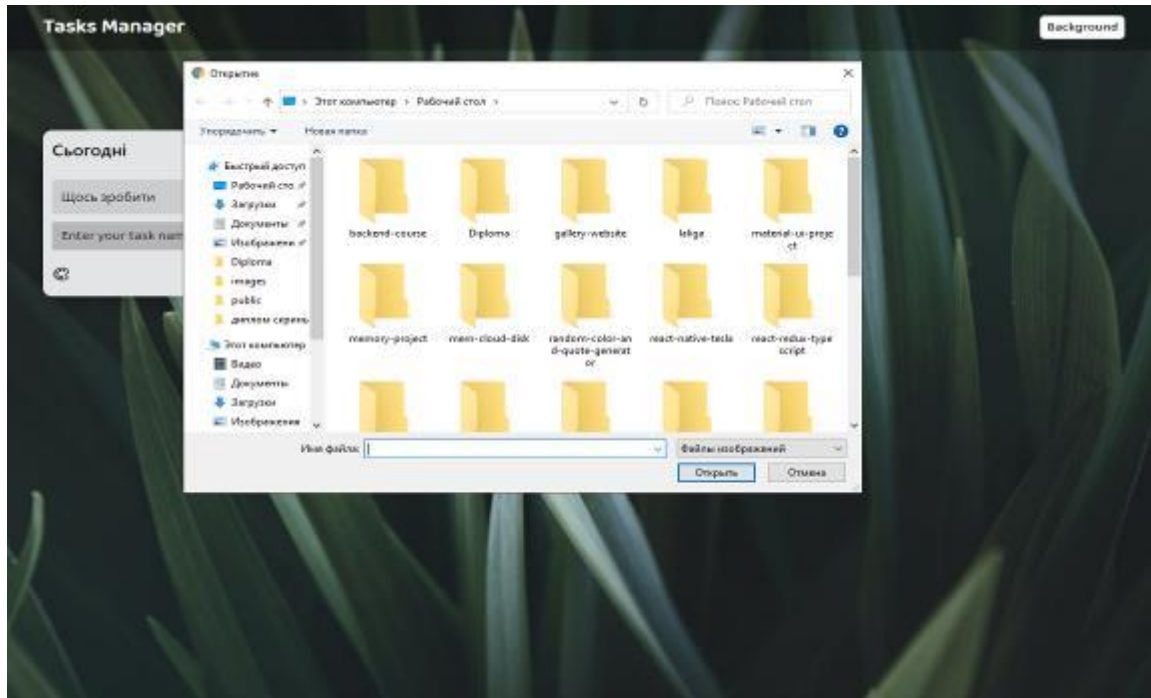


Рисунок 4.8 – Вікно вибору зображення

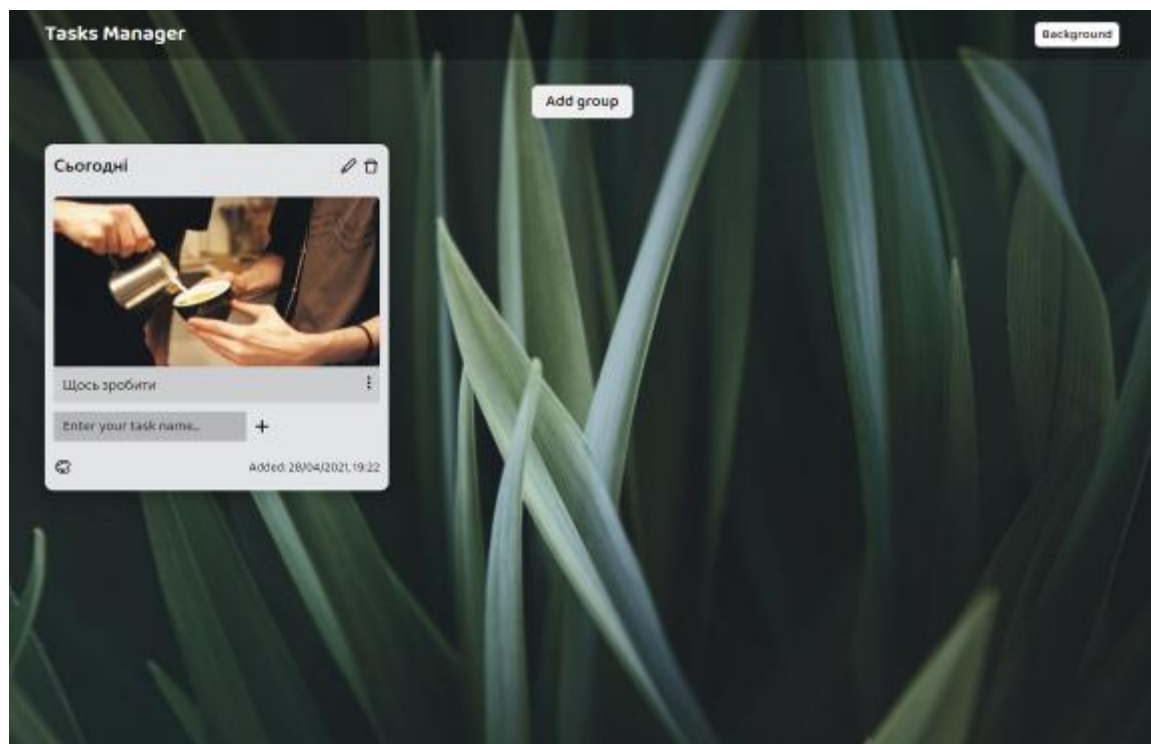


Рисунок 4.9 – Зображення вибране і додане до задачі

Якщо зверху групи натиснути на кнопку у вигляді ручки, то з'явиться поле вводу, де можна змінити назву групи (рис. 4.10). Після цього потрібно натиснути на кнопку у вигляді дискети та назву буде змінено (рис. 4.11).

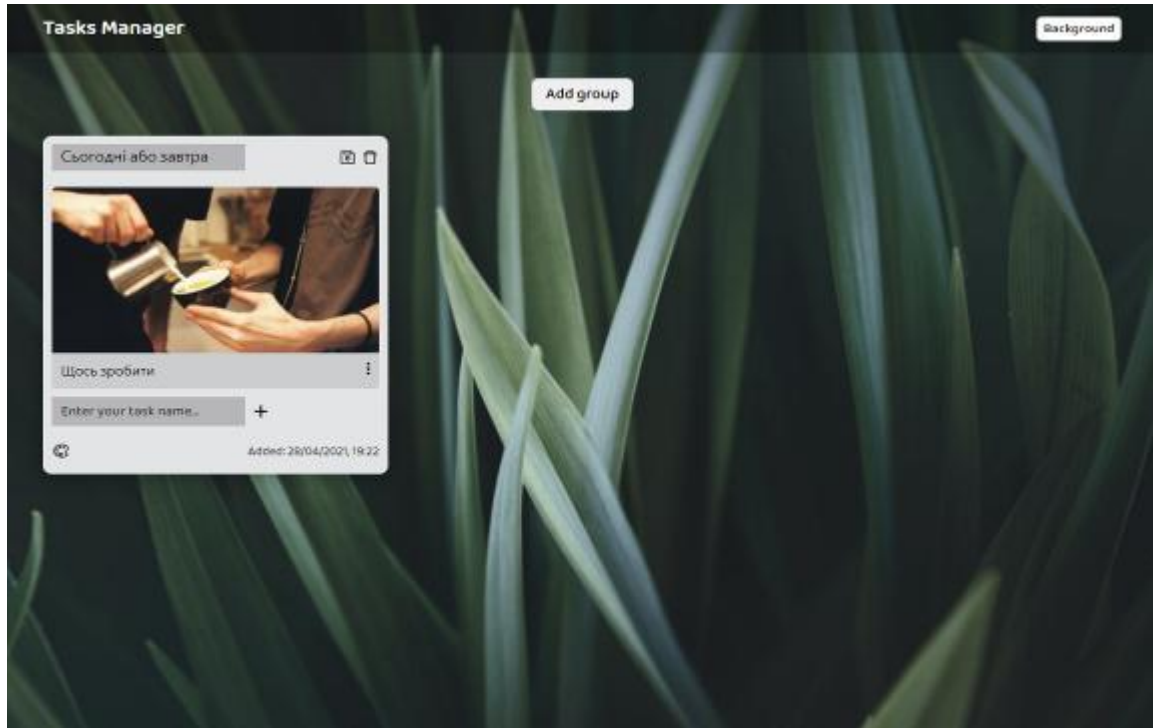


Рисунок 4.10 – Поле зміни назви групи

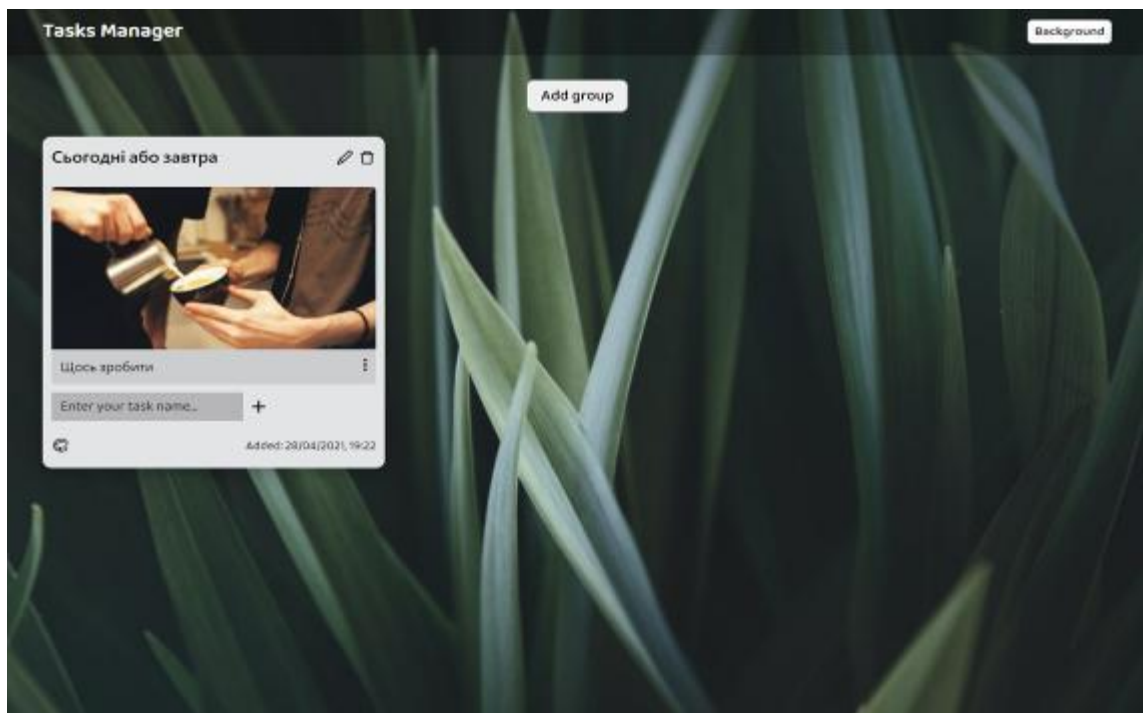


Рисунок 4.11 – Назву групи змінено

Якщо натиснути на кнопку у вигляді палітри зліва внизу групи, то буде відкрито модальне вікно з вибором фону групи (рис. 4.12). Після вибору фон буде змінено (рис. 4.13).

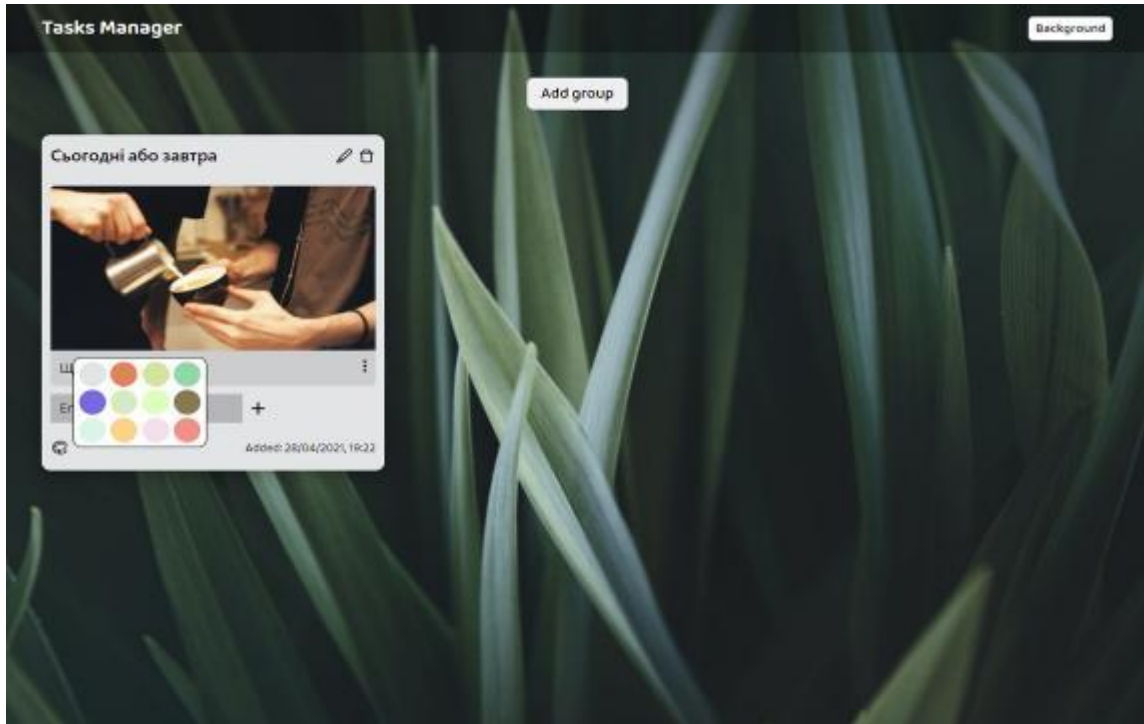


Рисунок 4.12 – Модальне вікно вибору фону групи

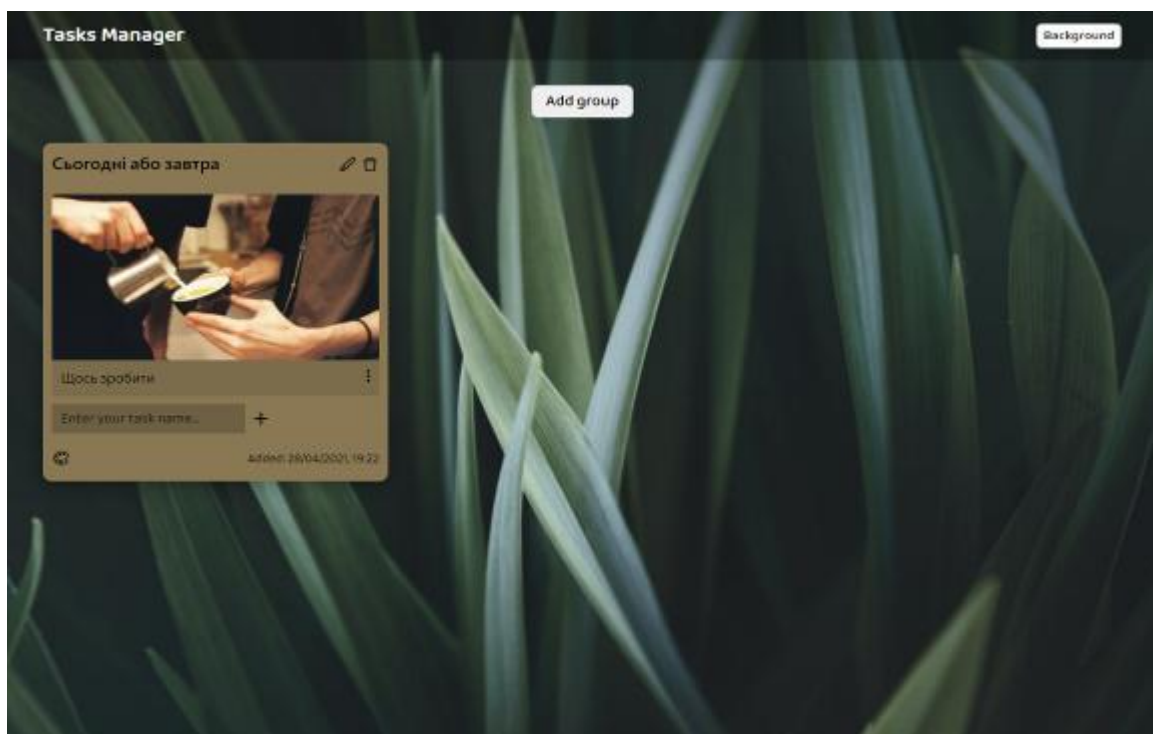


Рисунок 4.13 – Фон групи змінено

Якщо натиснути в шапці сайту на кнопку з назвою «Background», то буде показане модальне вікно зі списком зображень, які можна вибрати як фон всього застосунку (рис. 4.14). Після вибору фон буде змінено (рис. 4.15)

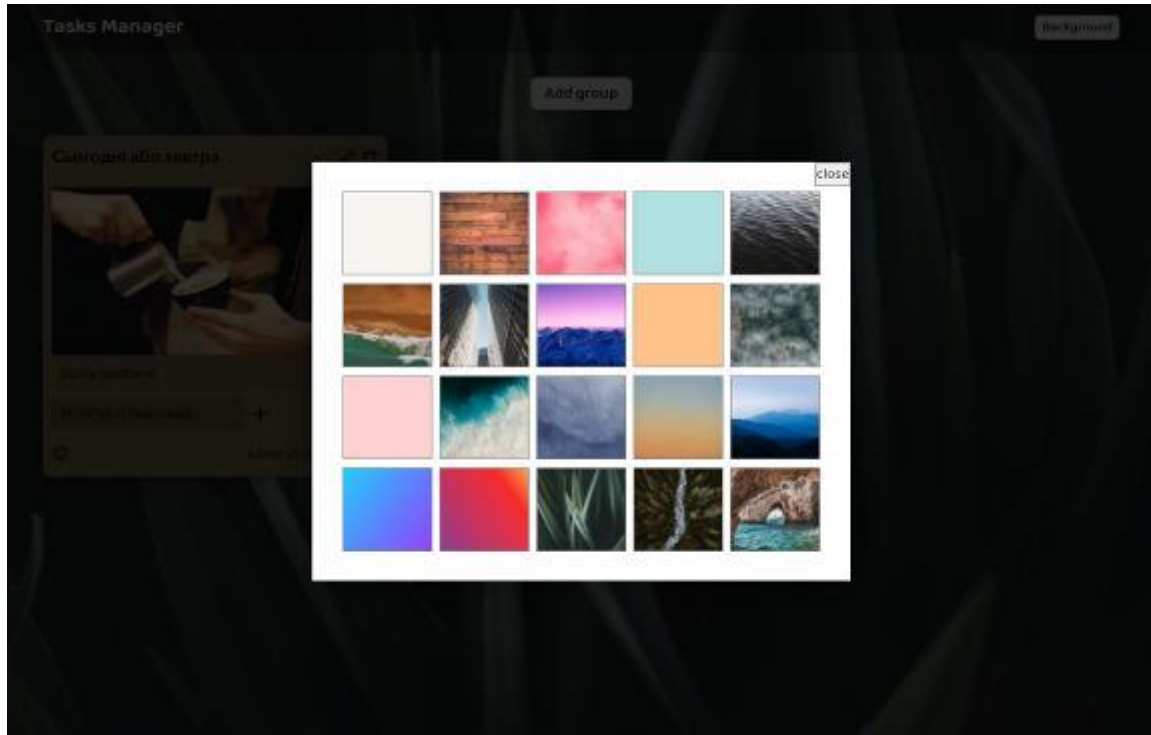


Рисунок 4.14 – Модальне вікно з вибором фону застосунку

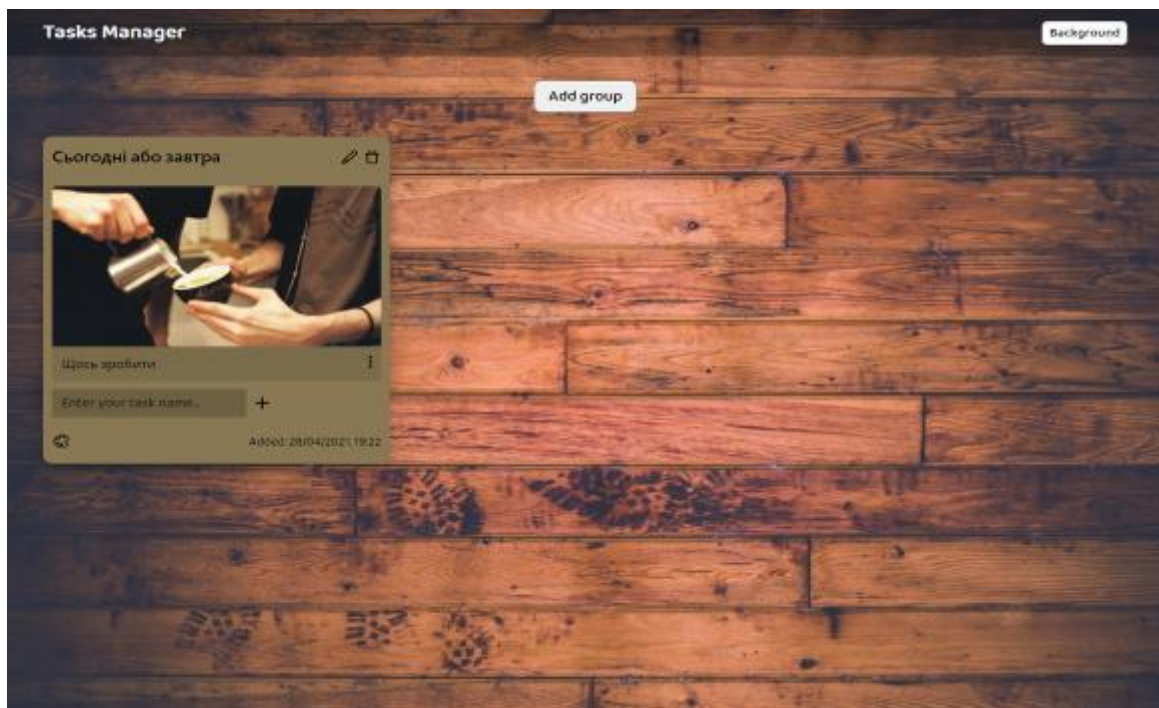


Рисунок 4.15 – Фон застосунку змінено

Якщо натиснути на кнопку у вигляді корзини (показані дві групи для прикладу) (рис. 4.16), то групу буде видалено (рис. 4.17).

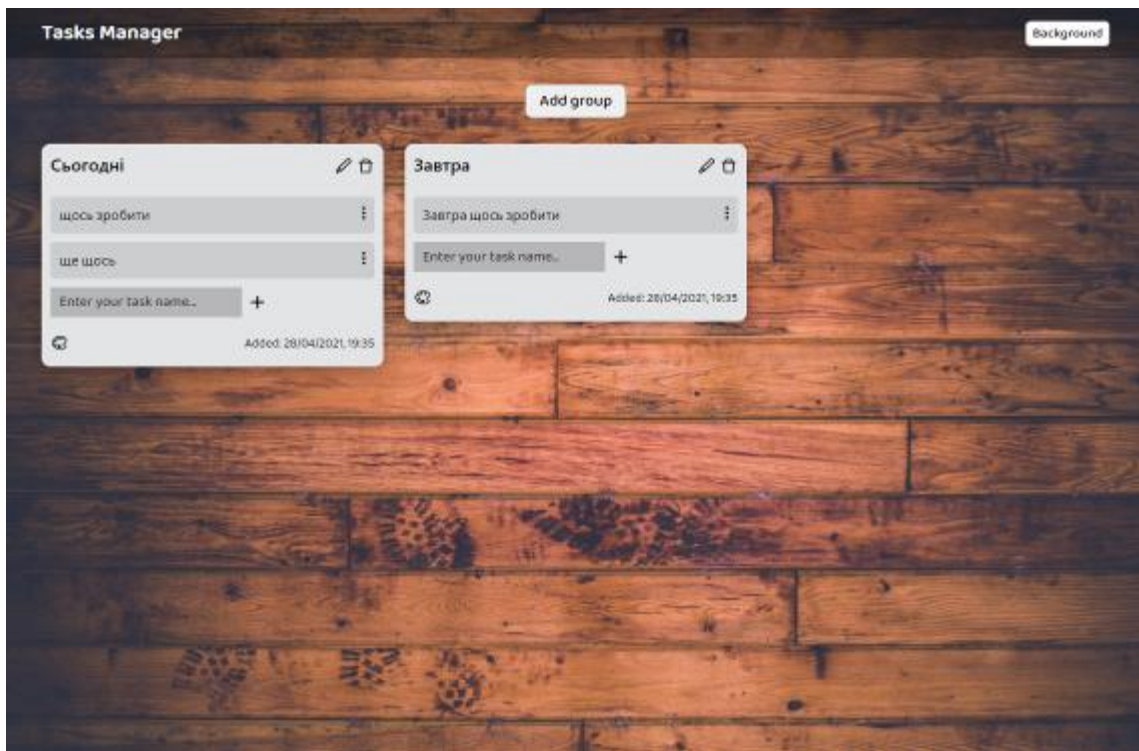


Рисунок 4.16 – Дві групи поруч

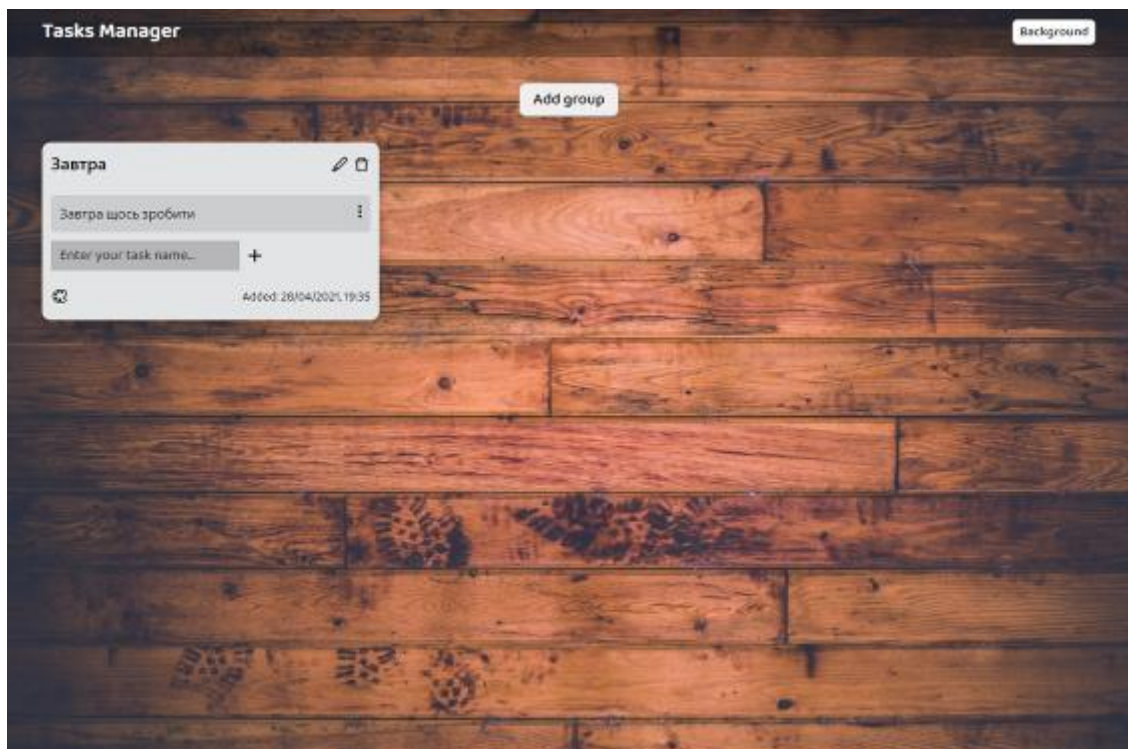


Рисунок 4.17 – Групу видалено

Якщо натиснути на кнопку у вигляді квадрату у полі адреси застосунку, то буде запропоновано відкрити застосунок як PWA- застосунок (рис. 4.18) – він буде доданий на робочий стіл і відкриється як звичайний застосунок (рис. 4.19).

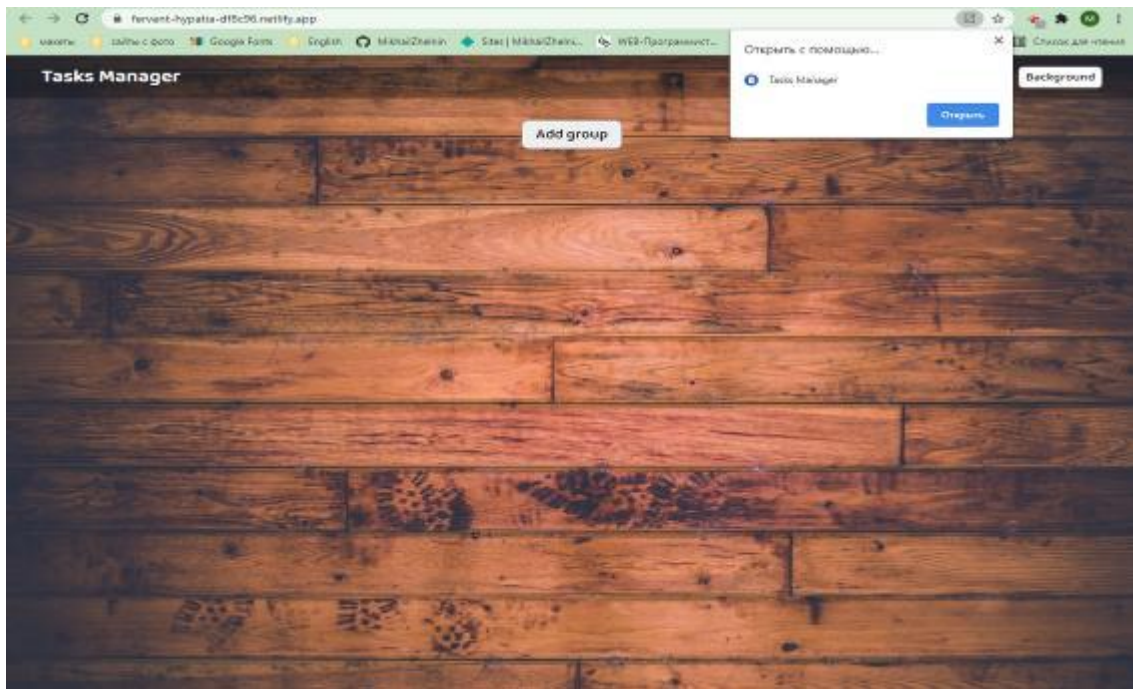


Рисунок 4.18 – Відкривання застосунку як PWA

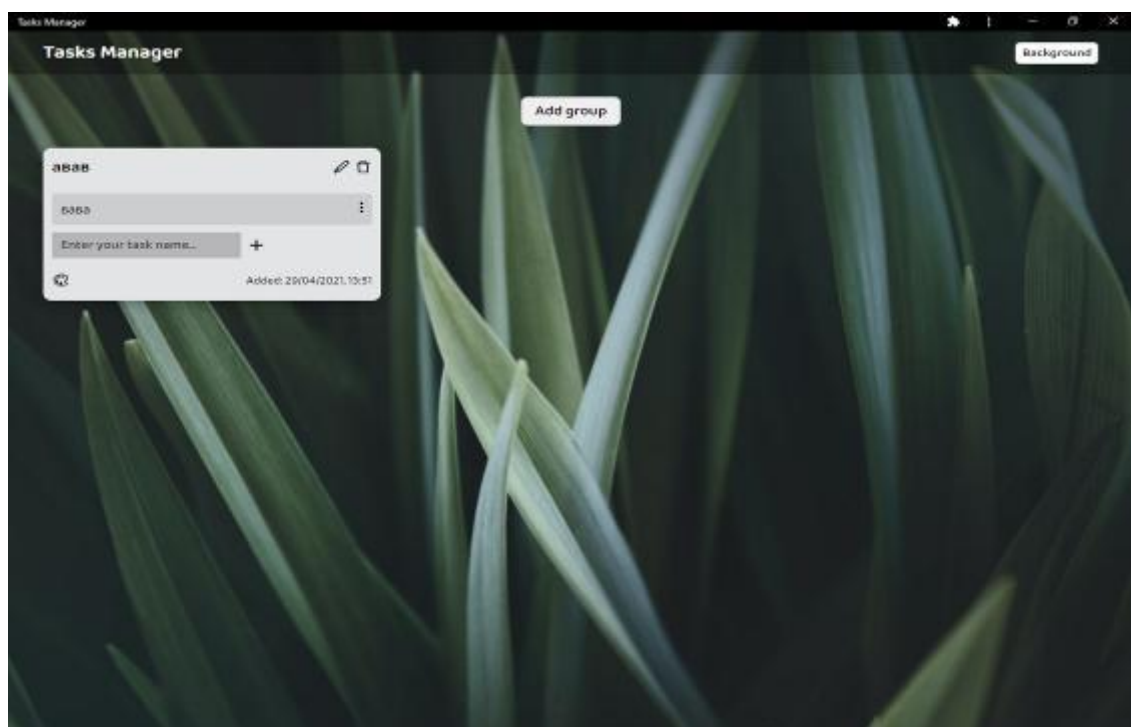


Рисунок 4.19 – Результат роботи PWA- застосунку

4.2 Методика тестування застосунку

Тестування – це найважливіший, складний та обов’язковий процес заключних етапів розробки застосунку, головною метою якого є тестування і виявлення усіх помилок в застосунку при розробці та опис рекомендацій по їх уникненню.

Буває так, що розробники забувають чи просто пропускають цей етап, а потім в продакшені виявляється значна кількість помилок, яка може впливати на продуктивність, швидкість та користування застосунком. Такі помилки значною мірою можуть впливати на великі фінансові втрати, якщо застосунок розроблюється у компанії та просто до незручностей та втраті користувачів, якщо застосунок було розроблено одним розробником. В результаті з’являється необхідність шукати помилки, а це багато часу, грошей та нервів. Тому тестування та просто спостерігання за результатами роботи програми ще на етапі розробки застосунку є великою необхідністю.

Тестувати програму можна ще на етапі розробки застосунку. Зробивши один елемент – можна одразу тестувати. У веб-розробці зазвичай на цьому етапі тестують сумісність з іншими браузерами, а також тестують адаптивність на різних пристроях. Адаптивність – це коректне відображення застосунку на всіх пристроях (мобільних, десктопних, планшетах та інших).

Коли розробка програми закінчена, фахівець з тестування отримує необхідну документацію, матеріали та виконує тестування.

Серед найбільш популярних методів тестування є: блочне тестування, автономне тестування, тестування за допомогою веб-інспекторів, консолі та відладчика.

Блочне тестування відноситься до більш низького рівня, що гарантує очікуване виконання конкретних фрагментів коду в фоновому режимі. Блочне тестування більше схоже на виявлення проблем кроссбраузерності, але воно дозволяє вирішити їх досить швидко, оскільки випробуванню піддаються порівняно невеликі фрагменти коду.

При розробці таких серверних JavaScript-реалізацій, як Node.js і Rhino, з'являється можливість запуску тестів за межами браузера в автономному оточенні з використанням командного рядка. Такі тести вигідно відрізняються швидкістю виконання і простотою налаштування, оскільки тут вже немає ніякого різноманіття браузерів і постійного інтегрованого середовища. Очевидним недоліком є запуск тестів поза реального робочого середовища.

Веб-інспектор доступний при роботі з більшістю браузерів. Веб-інспектор є дуже корисним інструментом, що дозволяє вивчати HTML-елементи, редагувати стилі, займатися налагодженням JavaScript і робити багато чого іншого.

Консоль дозволяє без особливих зусиль виконувати JavaScript і вивчати вміст глобальних змінних, що є на сторінці. Одним з основних переваг консолі полягає в тому, що можна протоколювати свою роботу безпосередньо в самій консолі, використовуючи функцію `console.log ()`. Виклик здійснюється в асинхронному режимі, може приймати кілька аргументів і перевіряти ці аргументи, не конвертуючи їх в рядки. Існують і додаткові види протоколювання. Для підвищення рівня протоколювання, що виявляє перші ознаки неправильної роботи, можна використовувати функції `console.warn ()` і `console.error ()`.

Відладчик JavaScript - це одна з таємниць JavaScript-розробки. Він є повнофункціональним засобом налагодження, що дозволяє розставляти контрольні точки, переглядати вираження, досліджувати значення змінних і в точності з'ясовувати, що відбувається в сценарії [15].

4.3 Практичне тестування застосунку

Тестування застосунку в даній роботі буде проведено методом «чорного ящика».

Тестування методом «чорного ящика», також відоме як тестування, засноване на специфікації або тестування поведінки - техніка тестування, заснована на роботі виключно з зовнішніми інтерфейсами тестованої системи.

Тестування методом «чорного ящика» включає в себе:

- тестування, як функціональне, так і нефункціональне, яке не передбачає знання внутрішнього устрою компонента або системи;
- тест-дизайн, заснований на техніці «чорного ящика» - процедура написання або вибору тест-кейсів на основі аналізу функціональної або нефункціональної специфікації компонента або системи без знання її внутрішнього устрою [16].

Таблиця 1 – Тестування програми

Дія	Очікуваний результат	Результат
1	2	3
Відкриття застосунку	- Застосунок відкрито; - назва застосунку відображається зліва зверху; - кнопка вибору фону відображається з права зверху; - кнопка додавання задачі відображається посередині зверху застосунку.	- Застосунок відкрито; - назва застосунку відображається зліва зверху; - кнопка вибору фону відображається з права зверху; - кнопка додавання задачі відображається посередині зверху застосунку.
Дії з групами	- При додаванні групи вона додається у список груп; - при натискання на кнопку зміни назви відкривається поле введення нової назви, при цьому стара назва	- При додаванні групи вона додається у список груп; - при натискання на кнопку зміни назви відкривається поле введення нової назви, при цьому стара назва

Продовження таблиці 1

1	2	3
Дії з групами	додається у поле; - при натисканні на кнопку закриття зміни назви назва зберігається; - при натисканні на кнопку видаленні групи група видаляється.	додається у поле; - при натисканні на кнопку закриття зміни назви назва зберігається; - при натисканні на кнопку видаленні групи група видаляється.
Дії з групами	- при натисканні кнопку для зміни фону відкривається модальне вікно для зміни кольору фону; - при виборі фону групи він зберігається; - з права внизу групи відображається дата додавання групи.	- при натисканні кнопку для зміни фону відкривається модальне вікно для зміни кольору фону; - при виборі фону групи він зберігається; - з права внизу групи відображається дата додавання групи.
Дії з задачами	- При введенні назви задачі у поле додавання задачі та при натисканні на кнопку «+» задача додається до списку задач; - при натисканні на кнопку у задачі у вигляді трьох точок відкривається модальне вікно для вибору дій з задачами;	- При введенні назви задачі у поле додавання задачі та при натисканні на кнопку «+» задача додається до списку задач; - при натисканні на кнопку у задачі у вигляді трьох точок відкривається модальне вікно для вибору дій з задачами;

Продовження таблиці 1

1	2	3
Дії з задачами	- при натисканні у модальному вікні кнопку у вигляді галки, задача відмічається як зроблена; - при натисканні у модальному вікні кнопку у вигляді зображення відкривається модальне вікно вибору файлу і при виборі зображення відображається коректно; - при натисканні у модальному вікні кнопку у вигляді видалення задача видаляється.	- при натисканні у модальному вікні кнопку у вигляді галки, задача відмічається як зроблена; - при натисканні у модальному вікні кнопку у вигляді зображення відкривається модальне вікно вибору файлу і при виборі зображення відображається коректно; - при натисканні у модальному вікні кнопку у вигляді видалення задача видаляється.
Зміна фону всього застосунку	- При натисканні кнопки «Background» з права зверху у шапці застосунку відкривається модальне вікно; - при виборі фону він змінюється.	- При натисканні кнопки «Background» з права зверху у шапці застосунку відкривається модальне вікно; - при виборі фону він змінюється.
Збереження даних	- При закритті вкладки чи вікна браузера дані зберігаються.	При закритті вкладки чи вікна браузера дані зберігаються.

У таблиці 1 було проведено детальне тестування застосунку методом «чорного ящика».

Далі потрібно перевірити застосунок на валідність HTML та CSS файлів. Зробити це можна на сайті <https://validar.w3org> . На (рис. 4.20) буде наведено результат перевірки на валідність HTML файлів, а на (рис. 4.21) CSS файлів.

Nu Html Checker

This tool is an ongoing experiment in better HTML checking, and its behavior remains subject to change

Showing results for <https://fervent-hypatia-df8c96.netlify.app/>

Checker Input

Show

☐ source
 ☐ outline
 ☐ image report

Options...

Check by

address

<https://fervent-hypatia-df8c96.netlify.app/>

Check

Document checking completed. No errors or warnings to show.

Used the HTML parser. Externally specified character encoding was UTF-8.

Total execution time 58 milliseconds.

[About this checker](#) • [Report an issue](#) • Version: 21.4.26

Рисунок 4.20 – Перевірка на валідність HTML

W3C

сервис проверки CSS от W3C

W3C результаты проверки CSS для <https://fervent-hypatia-df8c96.netlify.app/> (CSS3 + SVG)

Перейти к:

Предупреждения (61)

Проверенный CSS

W3C результаты проверки CSS для <https://fervent-hypatia-df8c96.netlify.app/> (CSS3 + SVG)

Поздравляем! Ошибок не обнаружено.

Этот документ проходит проверку по стандарту [CSS3 + SVG](#) !

Для того, чтобы продемонстрировать посетителям, что вы заботитесь о создании корректных страниц, вы можете разместить этот знак на любой проверенной странице. Вот код XHTML для размещения:

```

<p>
  <a href="https://jigsaw.w3.org/css-validator/check/referer">
    
    </a>
  </p>

```

```

<p>
  <a href="https://jigsaw.w3.org/css-validator/check/referer">
    
    </a>
  </p>

```

Рисунок 4.21 – Перевірка на валідність CSS

Також потрібно виконати перевірку на кроссбраузерність в таких браузерах: Google Chrome (рис. 4.22), Mozilla Firefox (рис. 4.23), Microsoft Edge (рис. 4.24) та мобільна версія (рис. 4.25).

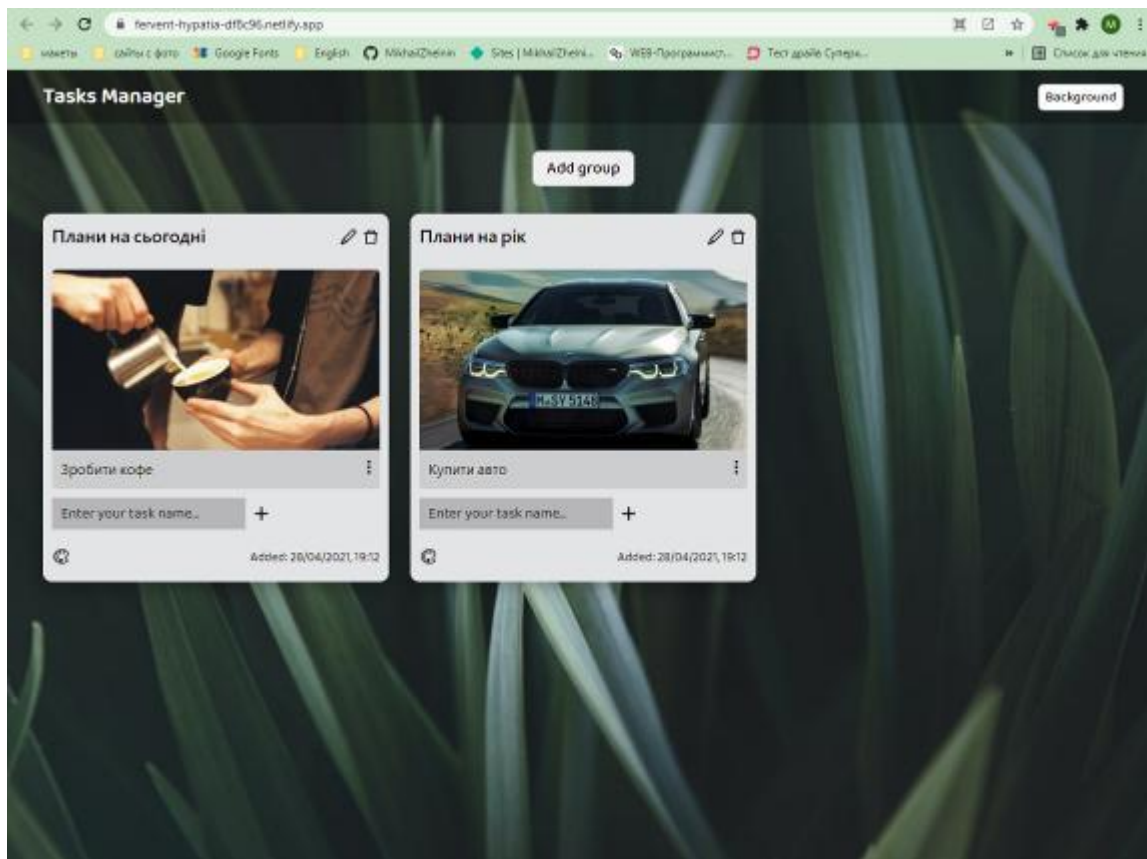


Рисунок 4.22 – Кроссбраузерність Google Chrome

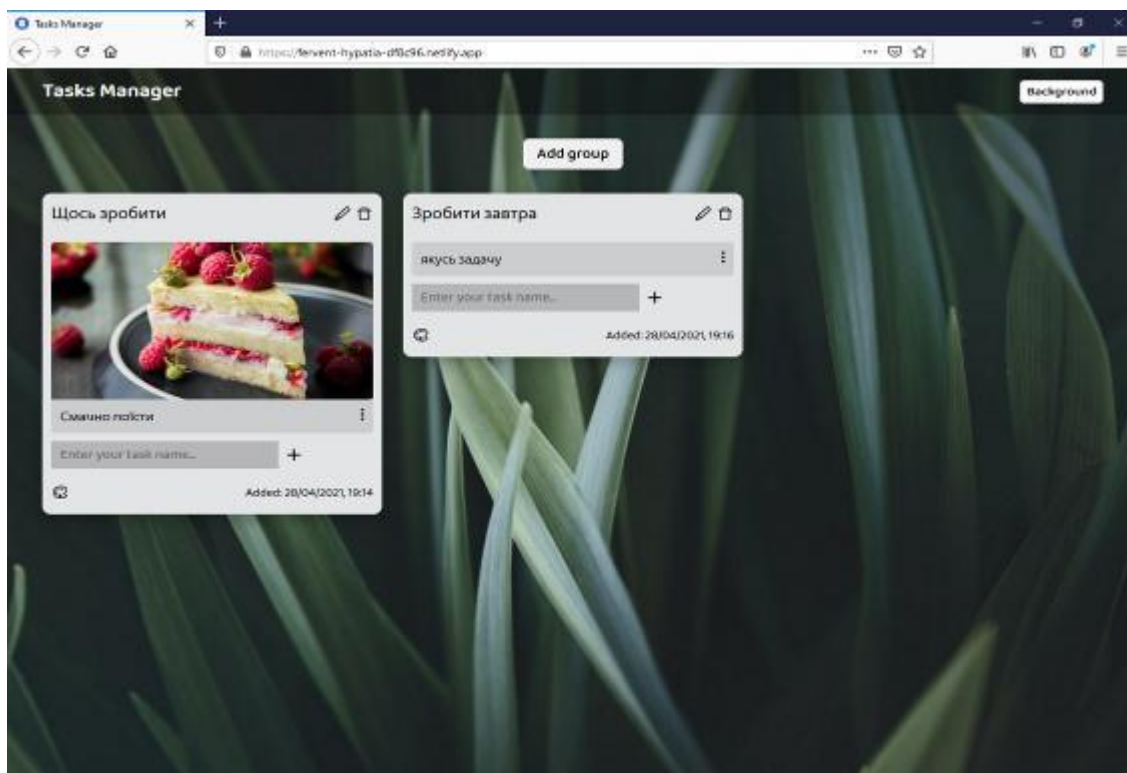


Рисунок 4.23 – Кроссбраузерність Mozilla Firefox

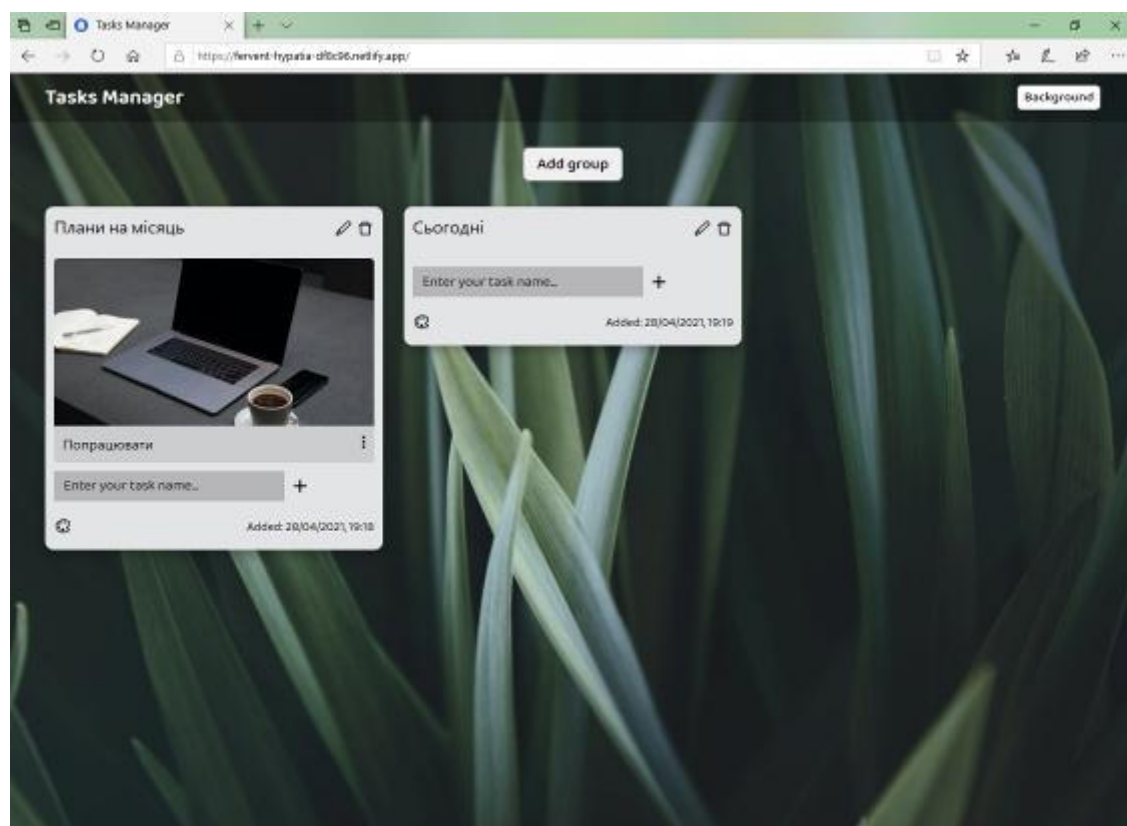


Рисунок 4.24 – Кроссбраузерність Microsoft Edge

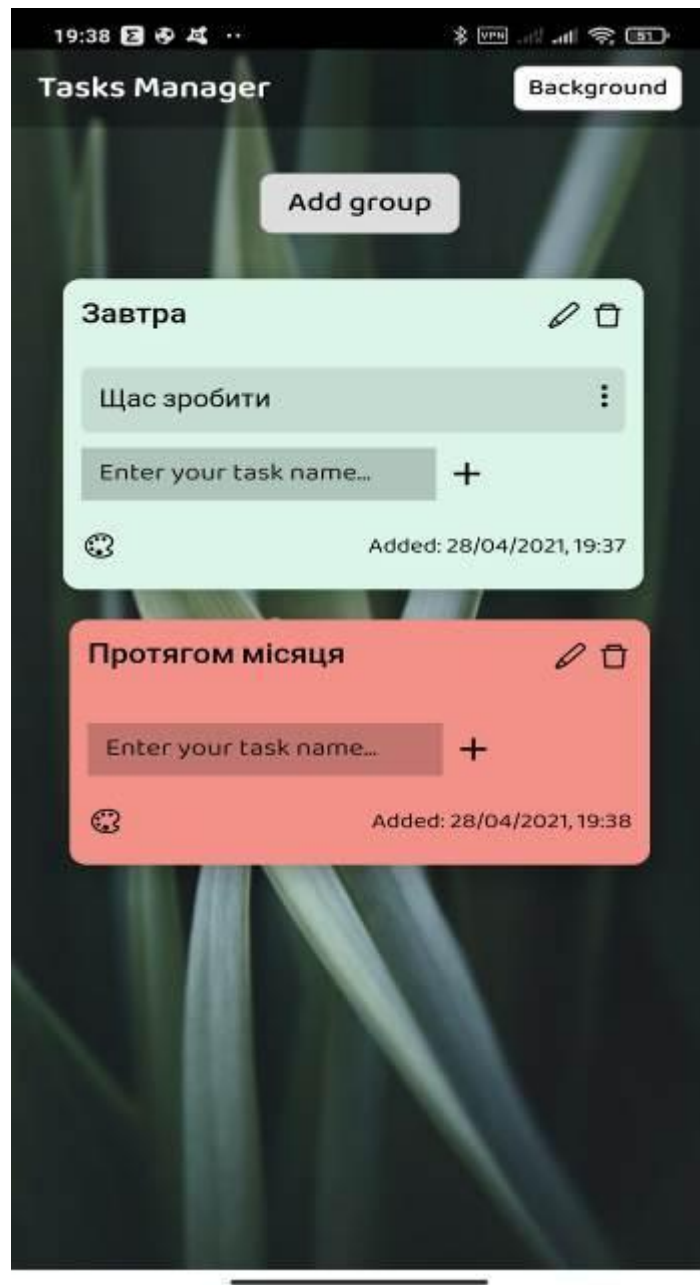


Рисунок 4.25 – Мобільна версія застосунку

4.4 Висновки за розділом 4

В даному розділі була описана детальна інструкція для користувача. Описані види тестування та проведені декілька етапів тестування, а саме тестування функціоналу методом «чорного ящика», тестування на валідність та кроссбраузерність в найпопулярніших браузерях.

При тестуванні не виявлено ніяких помилок, що затверджено рисунками та таблицями.

ВИСНОВКИ

Метою даної дипломної роботи є розробка застосунку, який дозволяє спрощувати процеси планування та виконання повсякденних задач.

Було проведено детальний аналіз предметної області та існуючих застосунків, які допомагають у вирішенні досліджуваної задачі. Були сформовані основні вимоги до застосунку та встановлені основні етапи розробки застосунку.

В роботі було досліджено та описано важливу інформацію щодо розробки веб-застосунків, а саме: їх види, основні етапи проектування та розробки, їх методи розробки.

Після аналізу та визначення основних етапів було вирішено розробити застосунок за допомогою веб-технологій React та Redux, які дозволяють створювати швидкі, приємні та стабільні застосунки, які відображаються та працюють однаково на всіх пристроях.

Середою розробки було обрано Visual Studio Code, тому що він дуже простий, зрозумілий, підтримує багато плагінів і надає значну кількість інструментів, які дозволяють швидко створювати якісні застосунки.

У заключному четвертому розділі було описано важливість тестування та на основі цього було проведено велику кількість тестувань, таких як візуальне та програмне тестування методом «чорного ящика», тестування кроссбраузерності, яке відіграє важливу роль, тому що багато хто використовує різні браузери та тестування на валідність коду.

Таким чином можна сказати, що застосунок «Менеджер задач» працює без помилок та його можна використовувати.

Розробляючи застосунок було закріплені теоретичні знання, були вивчені деякі нові технології та вдосконалені навички розробки застосунків.

Дипломна робота містить 54 рисунки, 1 таблицю, 75 сторінок без додатків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Различные виды и типы сайтов и веб-проектов [Электрон. ресурс]. – Режим доступа : <https://avada-media.ua/tipy-sajtov/>
2. Проектирование web-приложений [Электрон. ресурс]. – Режим доступа : https://studme.org/97678/informatika/proektirovanie_web-prilozheniy
3. 10 современных планировщиков задач [Электрон. ресурс]. – Режим доступа : <https://itc.ua/articles/10-sovremennyh-planirovshhikov-zadach/>
4. Веб приложение. Разница между сайтом, веб-приложением, SPA и PWA [Электрон. ресурс]. – Режим доступа : <https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/>
5. Тиленс, Т. М. React в действии / Т. М. Тиленс. – СПб. : Питер, 2019. – 368 с.
6. Бэнкс, А. React и Redux. Функциональна веб-разработка / А. Бэнкс, Е. Порселло. – СПб. : Питер, 2018. – 336 с.
7. Макфарланд, Д. Новая больша книга CSS / Д. Макфарланд. – СПб. : Питер, 2016. – 720 с.
8. Розенталс, Н. Изучаем TypeScript 3 / Н. Розенталс. – М. : ДМК Пресс, 2019. – 624 с.
9. Пауєрс, Ш. Изучаем Node.js / Ш. Пауєрс. – СПб. : Питер, 2014. – 400с.
10. Аквино, К. Front-end. Клиентска разработка для профессионалов. Node.js, ES6, REST. / К. Аквино, Т. Ганди. – СПб. : Питер, 2017. – 512 с.
11. Файн, Я. Angular и TypeScript. Сайтостроение для профессионалов / Я. Файн, А. Моисеев. – СПб. : Питер, 2018. – 464с.
12. Чакон, С. Git для професійного програміста / С. Чакон, Б. Штрауб. – СПб. : Пітер, 2016. – 496 с.
13. Методологія БЕМ в дії [Електрон. ресурс]. – Режим доступа : <https://avivi.pro/ua/blog/metodologiya-bem-v-deystvii/>

14. Як створити контактну форму за допомогою Netlify Forms та Next.js [Електрон. ресурс]. – Режим доступу : <https://hackit-ukraine.com/1947-how-to-create-a-contact-form-with-netlify-forms-and-next-js>
15. Маккоу, А. Веб-приложение на JavaScript / А. Маккоу. – Спб. : Питер, 2012. – 288 с.
16. White/Black/Grey Box-тестирование [Електрон. ресурс]. – Режим доступу : <https://qalight.ua/ru/baza-znaniy/white-black-grey-box-testirovanie/>

ДОДАТОК А
Технічне завдання

Вступ

Програма призначена для впорядкування та планування повсякденних задач.

A.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація онлайн менеджера задач», затверджене наказом №103 від 30 березня 2021 року по Національному Університету «Запорізька політехніка».

A.2 Призначення розробки

Програма призначена для планування як індивідуальних, так і бізнес задач.

A.3 Вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функції, які мають виконуватись програмним продуктом:

- додавати групу;
- змінювати назву групи та видаляти її;
- змінювати колір фону групи;
- показувати дату створення групи;
- додавати задачу;
- відмічати задачу як зроблену;
- додавати до конкретної задачі фото;
- видаляти задачу;
- змінювати фон всього застосунку на інший.

A.3.2 Вимоги до надійності

Програма повинна коректно працювати на різних пристроях.

A.3.3 Умови експлуатації

Для роботи з програмою користувач має володіти комп'ютером, планшетом, ноутбуком, мобільним пристроєм на достатньому рівні. Також повинен володіти браузером.

A.3.4 Вимоги до складу й параметрів технічних засобів

Для роботи з програмою користувач повинен мати комп'ютер, планшет, ноутбук або мобільний пристрій з встановленим на них будь-яким браузером.

A.4 Вимоги до програмної документації

Програмна документація повинна бути написана українською мовою та включати технічне завдання.

A.5 Порядок контролю й приймання

Дипломна кваліфікаційна робота допускається до захисту керівником, нормоконтролером, затверджується завідувачем кафедри і захищається в присутності членів екзаменаційної комісії.

ДОДАТОК Б
Текст програми

1. Головні файли програми в папці src

src / index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';

import * as serviceWorkerRegistration from './serviceWorkerRegistration';

import App from './App';

import store from './redux/store';

import './index.scss';

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root'),
);

serviceWorkerRegistration.register();
```

src / App.js

```
import React from 'react';
import { useSelector } from 'react-redux';

import Header from './components/Header/Header';
import GroupList from './components/GroupList/GroupList';
import GroupModal from './components/GroupModal/GroupModal';
import BackgroundModal from
'./components/BackgroundModal/BackgroundModal';

import './App.scss';

function App() {
  const groupsModal = useSelector((state) =>
state.groups.groupsModalOpen);
  const backgroundModal = useSelector((state) =>
state.background.backgroundModalOpen);
```

```

    const backgroundImage = useSelector((state) =>
state.background.selectedBackground);

    return (
    <div
    className="App"
    style={{
    backgroundImage: `url(${backgroundImage})`,
    }}>
    <Header />
    <GroupList />
    {groupsModal ? <GroupModal /> : null}
    {backgroundModal ? <BackgroundModal /> : null}
    </div>
    );
  }

export default App;

```

2. Компоненти програми в папці components

src / components / Header / Header.jsx

```

import React from 'react';
import { useDispatch } from 'react-redux';

import { openBackModal } from
'../../redux/actionCreators/backgroundActionCreator';

import './Header.scss';

const Header = () => {
  const dispatch = useDispatch();

  return (
    <div className="header">
      <div className="container">
        <div className="header__wrapper">
          <h2 className="header__title">Tasks Manager</h2>
          <button className="header__btn-background" onClick={() =>
dispatch(openBackModal())}>
            Background
          </button>
        </div>
      </div>
    </div>
  )
}

```

```

    </div>
  );
};

export default Header;

```

src / components / GroupList / GroupList.jsx

```

import React from 'react';
import { useSelector, useDispatch } from 'react-redux';

import { openGroupModal } from
'../../redux/actionCreators/groupActionCreator';

import GroupItem from './GroupItem/GroupItem';

import './GroupList.scss';

const GroupList = () => {
  const dispatch = useDispatch();
  const groups = useSelector((state) => state.groups.groups);

  return (
    <div className="groupList">
      <div className="container">
        <div className="groupList__wrapper">
          <button className="groupList__btn" onClick={() =>
dispatch(openGroupModal())}>
            Add group
          </button>
          <div className="groupList__list">
            {groups.map((group) => (
              <GroupItem key={group.id} {...group} />
            ))}
          </div>
        </div>
      </div>
    </div>
  );
};

export default GroupList;

```

src / components / GroupList / GroupItem / GroupItem.jsx

```
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';

import {
  addTask,
  deleteGroupAction,
  openGroupNameEdit,
  closeGroupNameEdit,
  editGroupName,
  openGroupColorEdit,
} from '../../redux/actionCreators/groupActionCreator';

import { AiOutlineDelete } from 'react-icons/ai';
import { RiAddLine } from 'react-icons/ri';
import { VscEdit } from 'react-icons/vsc';
import { AiOutlineSave } from 'react-icons/ai';
import { IoColorPaletteOutline } from 'react-icons/io5';

import TasksList from '../../TasksList/TasksList';
import GroupColorModal from '../../GroupColorModal/GroupColorModal';

import './GroupItem.scss';

const GroupItem = ({
  id,
  name,
  backgroundColor,
  changedName,
  changedBackgroundColor,
  timeOfCreating,
}) => {
  const [taskName, setTaskName] = useState("");
  const [newName, setNewInputName] = useState(name);
  const dispatch = useDispatch();

  const addNewTask = (id) => {
    if (taskName === "") {
      return false;
    }
    const newTask = {
      groupId: id,
      id: new Date().getTime().toString(),
      name: taskName,
```

```

    photo: "",
    isCompleted: false,
    actionsModalOpen: false,
  };
  dispatch(addTask(newTask));
  setTaskName("");
};

const editNameHandler = (name, id) => {
  dispatch(editGroupName(name, id));
  setNewInputName(name);
  dispatch(closeGroupNameEdit(id));
};

return (
  <div key={id} className="groupList__item" style={{
backgroundColor: backgroundColor }}>
    <header className="groupList__item-header">
      {changedName ? (
        <div className="groupList__item-header__changed-name">
          <input
            type="text"
            value={newName}
            onChange={(e) => setNewInputName(e.target.value)}
            autoFocus
          />
        </div>
      ) : (
        <h3 className="groupList__item-header__name">{name}</h3>
      )}
      {changedName ? (
        <AiOutlineSave
          className="groupList__item-header__btn"
          onClick={() => editNameHandler(newName, id)}
        />
      ) : (
        <VscEdit
          className="groupList__item-header__btn"
          onClick={() => dispatch(openGroupNameEdit(id))}
        />
      )}
      <AiOutlineDelete
        className="groupList__item-header__btn"
        onClick={() => dispatch(deleteGroupAction(id))}
      />
    </header>
  </div>
);

```

```

</header>
<main className="groupList__item-main">
  <TasksList id={id} />
  <div className="group__item-main__actions">
    <input
      type="text"
      className="groupList__item-main__actions-input"
      value={taskName}
      onChange={(e) => setTaskName(e.target.value)}
      placeholder="Enter your task name..."
    />
    <RiAddLine className="groupList__item-main__actions-btn"
      onClick={() => addNewTask(id)} />
  </div>
</main>
<footer className="groupList__item-footer">
  <div className="groupList__item-footer__color">
    <IoColorPaletteOutline
      className="groupList__item-footer__color-icon"
      onClick={() => dispatch(openGroupColorEdit(id))}
    />
    {changedBackgroundColor ? (
      <GroupColorModal id={id} backgroundColor={backgroundColor}
    />
    ) : null}
  </div>
  <span className="groupList__item-footer__time">
    Added:{' '}
    {new Date(timeOfCreating).toLocaleDateString('en-GB', {
      hour: '2-digit',
      minute: '2-digit',
    })}
  </span>
</footer>
</div>
);
};

```

```
export default GroupItem;
```

Src / components / TaskList.jsx

```

import React from 'react';
import { useSelector, useDispatch } from 'react-redux';

import TaskModal from '../TaskModal/TaskModal';

import { BsThreeDotsVertical } from 'react-icons/bs';
import { IoCloseSharp } from 'react-icons/io5';

import { openTaskModal, deleteTaskPhoto } from
'../../redux/actionCreators/groupActionCreator';

import './TaskList.scss';

const TaskList = (props) => {
  const dispatch = useDispatch();
  const tasks = useSelector((state) => state.groups.tasks);

  return (
    <div className="tasksList">
      {tasks.map((item) =>
        item.groupId === props.id ? (
          <div key={item.id} className="tasksList__item">
            {item.photo ? (
              <div className="tasksList__item-header">
                <img src={item.photo} alt="task img" />
                <IoCloseSharp
                  className="tasksList__item-header__btnDelete"
                  onClick={() => dispatch(deleteTaskPhoto(item.id, false))}
                />
              </div>
            </div>
          ) : null}

          <div className="tasksList__item-main">
            <div className="tasksList__item-content">
              <span
                className={
                  item.isCompleted
                    ? 'tasksList__item-name tasksList__item-name__completed'
                    : 'tasksList__item-name '
                }
              >
                {item.name}
              </span>
            </div>
          </div>
        ) : null}
      )}
    </div>
  );
};

```

```

        <div className="tasksList__item-actions">
          <BsThreeDotsVertical
            className="tasksList__item-actions__icon"
            onClick={() => dispatch(openTaskModal(item.id))}
          />
          {item.actionsModalOpen ? <TaskModal taskId={item.id} /> :
null}
        </div>
      </div>
    </div>
  ) : (
    "
  ),
  )}
</div>
);
};

```

```
export default TasksList;
```

src / components / TaskModal / TaskModal.jsx

```

import React from 'react';
import { useDispatch } from 'react-redux';

import {
  checkTask,
  deleteTask,
  closeTaskModal,
  addTaskPhoto,
} from '../redux/actionCreators/groupActionCreator';

import { MdClose, MdCheck } from 'react-icons/md';
import { HiOutlinePhotograph } from 'react-icons/hi';

import './TaskModal.scss';

const TaskModal = ({ taskId }) => {
  const dispatch = useDispatch();

  const checkHandler = (taskId) => {
    dispatch(checkTask(taskId));
    dispatch(closeTaskModal(taskId));
  };

```

```

const deleteHandler = (taskId) => {
  dispatch(deleteTask(taskId));
  dispatch(closeTaskModal(taskId));
};

const addImageHandler = (e) => {
  const reader = new FileReader();
  reader.onloadend = () => {
    if (reader.readyState === 2) {
      dispatch(addTaskPhoto(taskId, reader.result));
      dispatch(closeTaskModal(taskId));
    }
  };
  reader.readAsDataURL(e.target.files[0]);
};

return (
  <div className="taskModal">
    <MdCheck className="taskModal-btn" onClick={() =>
checkHandler(taskId)} />
    <input
      type="file"
      name="taskModal-btn__imgUpload"
      id="input"
      accept="image/*"
      onChange={addImageHandler}
    />
    <label htmlFor="input" className="taskModal-btn__imgUpload-label">
      <HiOutlinePhotograph className="taskModal-btn" />
    </label>
    <MdClose className="taskModal-btn" onClick={() =>
deleteHandler(taskId)} />
  </div>
);
};

export default TaskModal;

```

src / components / GroupModal / GroupModal.jsx

```

import React, { useState } from 'react';
import { useDispatch } from 'react-redux';

import { addGroupAction, closeGroupModal } from
'../../redux/actionCreators/groupActionCreator';

```

```

import { groupBackColors } from '../utils/groupBackgroundColors';

import './GroupModal.scss';

const GroupModal = () => {
  const dispatch = useDispatch();
  const [groupName, setGroupName] = useState("");

  const createGroup = () => {
    if (groupName === "") {
      return false;
    }
    const id = new Date().getTime().toString();
    const name = groupName;
    const backgroundColor = groupBackColors[0];
    const changedName = false;
    const changedBackgroundColor = false;
    const timeOfCreating = Date.now();
    dispatch(
      addGroupAction(
        id,
        name,
        backgroundColor,
        changedName,
        changedBackgroundColor,
        timeOfCreating,
      ),
    );
    setGroupName("");
    dispatch(closeGroupModal());
  };

  const handleClick = (e) => {
    if (e.target.classList.contains('groupModal')) {
      dispatch(closeGroupModal());
    }
  };

  return (
    <div className="groupModal" onClick={handleClick}>
      <div className="groupModal__wrapper">
        <input
          type="text"
          className="groupModal__input"

```

```

      autoFocus
      placeholder="Enter group name..."
      value={groupName}
      onChange={(e) => setGroupName(e.target.value)}
    />
    <button className="groupModal__btn" onClick={createGroup}>
      add group
    </button>
    <button className="groupModal__btn-close" onClick={() =>
dispatch(closeGroupModal())}>
      close
    </button>
  </div>
</div>
);
};

```

```
export default GroupModal;
```

src / components / GroupColorModal / GroupColorModal.jsx

```

import React from 'react';
import { useDispatch } from 'react-redux';

import { editGroupColor, closeGroupColorEdit } from
'../../redux/actionCreators/groupActionCreator';

import { groupBackColors } from '../../utils/groupBackgroundColors';

import './GroupColorModal.scss';

const GroupColorModal = ({ id, backgroundColor }) => {
  const dispatch = useDispatch();

  const selectcolorHandler = (color, id) => {
    dispatch(editGroupColor(color, id));
    dispatch(closeGroupColorEdit(id));
  };

  return (
    <div className="groupColarModal">
      {groupBackColors.map((color, index) => (
        <div
          key={index}
          className="groupColarModal__color"

```

```

onClick={ (e) => selectcolorHandler(e.target.style.backgroundColor, id)}
style={{ background: color }}></div>
    )})
  </div>
);
};

```

```
export default GroupColorModal;
```

src / components / BackgroundModal/ BackgroundModal.jsx

```

import React from 'react';
import { useDispatch } from 'react-redux';

import {
  closeBackModal,
  selectBackground,
} from '../redux/actionCreators/backgroundActionCreator';

import { dataImages } from '../utils/background';

import './BackgroundModal.scss';

const BackgroundModal = () => {
  const dispatch = useDispatch();

  const handleClick = (e) => {
    if (e.target.classList.contains('backgroundModal')) {
      dispatch(closeBackModal());
    }
  };

  const selectImageHandler = (img) => {
    dispatch(selectBackground(img));
    dispatch(closeBackModal());
  };

  return (
    <div className="backgroundModal" onClick={handleClick}>
      <div className="backgroundModal__wrapper">
        {dataImages.map((item) => (
          <div key={item.id} className="backgroundModal__item">
            <img src={item.img} alt="background" onClick={() =>
selectImageHandler(item.img)} />
          </div>

```

```

    )))
    <button className="backgroundModal__btn-close" onClick={() =>
dispatch(closeBackModal())}>
      close
    </button>
  </div>
</div>
);
};

export default BackgroundModal;

```

3. Файли сховища даних Redux

src / redux / store.js

```

import { createStore } from 'redux';
import rootReducer from '../reducers/rootReducer';

import { loadState, saveState } from '../localStorage';

const persistedState = loadState();

const store = createStore(
  rootReducer,
  persistedState,
  window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__(),
);

store.subscribe(() => {
  saveState({
    groups: store.getState().groups,
    background: store.getState().background,
  });
});

export default store;

```

src / redux / types / types.js

```

// Group
export const ADD_GROUP = 'ADD_GROUP';
export const DELETE_GROUP = 'DELETE_GROUP';

```

```

    export const OPEN_GROUP_NAME_EDIT =
'OPEN_GROUP_NAME_EDIT';
    export const CLOSE_GROUP_NAME_EDIT =
'CLOSE_GROUP_NAME_EDIT';
    export const EDIT_GROUP_NAME = 'EDIT_GROUP_NAME';
    export const OPEN_GROUP_COLOR_EDIT =
'OPEN_GROUP_COLOR_EDIT';
    export const CLOSE_GROUP_COLOR_EDIT =
'CLOSE_GROUP_COLOR_EDIT';
    export const EDIT_GROUP_COLOR = 'EDIT_GROUP_COLOR';
    export const OPEN_GROUP_MODAL = 'OPEN_GROUP_MODAL';
    export const CLOSE_GROUP_MODAL = 'CLOSE_GROUP_MODAL';
    // Tasks
    export const ADD_TASK = 'ADD_TASK';
    export const CHECK_TASK = 'CHECK_TASK';
    export const DELETE_TASK = 'DELETE_TASK';
    export const OPEN_TASK_MODAL = 'OPEN_TASK_MODAL';
    export const CLOSE_TASK_MODAL = 'CLOSE_TASK_MODAL';
    export const ADD_TASK_PHOTO = 'ADD_TASK_PHOTO';
    export const DELETE_TASK_PHOTO = 'DELETE_TASK_PHOTO';
    // Background
    export const SELECT_BACKGROUND = 'SELECT_BACKGROUND';
    export const OPEN_BACKGROUND_MODAL =
'OPEN_BACKGROUND_MODAL';
    export const CLOSE_BACKGROUND_MODAL =
'CLOSE_BACKGROUND_MODAL';

```

src / redux / actionCreators / groupActioncreator.js

```

import * as actionTypes from '../types/types';

```

```

export const addGroupAction = (
  id,
  name,
  backgroundColor,
  changedName,
  changedBackgroundColor,
  timeOfCreating,
) => {
  return {
    type: actionTypes.ADD_GROUP,
    payload: {
      id,
      name,
      backgroundColor,

```

```

        changedName,
        changedBackgroundColor,
        timeOfCreating,
    },
};

export const deleteGroupAction = (idGroup) => {
    return {
        type: actionTypes.DELETE_GROUP,
        payload: idGroup,
    };
};

export const openGroupNameEdit = (id) => {
    return {
        type: actionTypes.OPEN_GROUP_NAME_EDIT,
        payload: id,
    };
};

export const closeGroupNameEdit = (id) => {
    return {
        type: actionTypes.CLOSE_GROUP_NAME_EDIT,
        payload: id,
    };
};

export const editGroupName = (name, id) => {
    return {
        type: actionTypes.EDIT_GROUP_NAME,
        payload: {
            name,
            id,
        },
    };
};

export const openGroupColorEdit = (id) => {
    return {
        type: actionTypes.OPEN_GROUP_COLOR_EDIT,
        payload: id,
    };
};

```

```
export const closeGroupColorEdit = (id) => {  
  return {  
    type: actionTypes.CLOSE_GROUP_COLOR_EDIT,  
    payload: id,  
  };  
};
```

```
export const editGroupColor = (color, id) => {  
  return {  
    type: actionTypes.EDIT_GROUP_COLOR,  
    payload: {  
      color,  
      id,  
    },  
  };  
};
```

```
export const openGroupModal = () => {  
  return {  
    type: actionTypes.OPEN_GROUP_MODAL,  
  };  
};
```

```
export const closeGroupModal = () => {  
  return {  
    type: actionTypes.CLOSE_GROUP_MODAL,  
  };  
};
```

```
export const addTask = (newTask) => {  
  return {  
    type: actionTypes.ADD_TASK,  
    payload: newTask,  
  };  
};
```

```
export const checkTask = (idTask) => {  
  return {  
    type: actionTypes.CHECK_TASK,  
    payload: idTask,  
  };  
};
```

```
export const deleteTask = (idTask) => {  
  return {
```

```

    type: actionTypes.DELETE_TASK,
    payload: idTask,
  };
};

export const openTaskModal = (idTask) => {
  return {
    type: actionTypes.OPEN_TASK_MODAL,
    payload: idTask,
  };
};

export const closeTaskModal = (idTask) => {
  return {
    type: actionTypes.CLOSE_TASK_MODAL,
    payload: idTask,
  };
};

export const addTaskPhoto = (idTask, photo) => {
  return {
    type: actionTypes.ADD_TASK_PHOTO,
    payload: {
      idTask,
      photo,
    },
  };
};

export const deleteTaskPhoto = (idTask, empty) => {
  return {
    type: actionTypes.DELETE_TASK_PHOTO,
    payload: {
      idTask,
      empty,
    },
  };
};

```

Src / redux / actioncreators / backgroundActioncreator.js

```

import * as actionTypes from '../types/types';

export const selectBackground = (img) => {
  return {

```

```

    type: actionTypes.SELECT_BACKGROUND,
    payload: img,
  };
};

```

```

export const openBackModal = () => {
  return {
    type: actionTypes.OPEN_BACKGROUND_MODAL,
  };
};

```

```

export const closeBackModal = () => {
  return {
    type: actionTypes.CLOSE_BACKGROUND_MODAL,
  };
};

```

src / redux / reducers / rootReducer.js

```

import { combineReducers } from 'redux';
import groups from './groupReducer';
import background from './backgroundReducer';

const rootReducer = combineReducers({ groups, background });

export default rootReducer;

```

src / redux / reducers / groupReducer.js

```

import * as actionTypes from '../types/types';

const initialState = {
  groups: [],
  tasks: [],
  groupsModalOpen: false,
};

const groups = (state = initialState, action) => {
  switch (action.type) {
    case actionTypes.ADD_GROUP:
      const newGroup = {
        id: action.payload.id,
        name: action.payload.name,
        backgroundColor: action.payload.backgroundColor,
        changedName: action.payload.changedName,

```

```

        changedBackgroundColor:
        action.payload.changedBackgroundColor,
        timeOfCreating: action.payload.timeOfCreating,
    };
    return {
        ...state,
        groups: [...state.groups, newGroup],
    };
case actionTypes.OPEN_GROUP_NAME_EDIT:
    return {
        ...state,
        groups: state.groups.map((group) =>
            group.id === action.payload ? { ...group,
            changedName: true } : group,
        ),
    };
case actionTypes.CLOSE_GROUP_NAME_EDIT:
    return {
        ...state,
        groups: state.groups.map((group) =>
            group.id === action.payload ? { ...group,
            changedName: false } : group,
        ),
    };
case actionTypes.EDIT_GROUP_NAME:
    return {
        ...state,
        groups: state.groups.map((group) =>
            group.id === action.payload.id ? { ...group, name:
            action.payload.name } : group,
        ),
    };
case actionTypes.OPEN_GROUP_COLOR_EDIT:
    return {
        ...state,
        groups: state.groups.map((group) =>
            group.id === action.payload
            ? { ...group, changedBackgroundColor:
            !group.changedBackgroundColor }
            : group,
        ),
    };
case actionTypes.CLOSE_GROUP_COLOR_EDIT:
    return {
        ...state,

```

```

        groups: state.groups.map((group) =>
        group.id === action.payload ? { ...group,
        changedBackgroundColor: false } : group,
        ),
    };
case actionTypes.EDIT_GROUP_COLOR:
    return {
        ...state,
        groups: state.groups.map((group) =>
        group.id === action.payload.id
        ? { ...group, backgroundColor: action.payload.color
        } : group,
        ),
    };
case actionTypes.DELETE_GROUP:
    return {
        ...state,
        groups: state.groups.filter((group) => group.id !==
        action.payload),
        tasks: state.tasks.filter((task) => task.groupId !==
        action.payload),
    };
case actionTypes.OPEN_GROUP_MODAL:
    return {
        ...state,
        groupsModalOpen: true,
    };
case actionTypes.CLOSE_GROUP_MODAL:
    return {
        ...state,
        groupsModalOpen: false,
    };
case actionTypes.ADD_TASK:
    return {
        ...state,
        tasks: [...state.tasks, action.payload],
    };
case actionTypes.CHECK_TASK:
    return {
        ...state,
        tasks: state.tasks.map((task) =>
        task.id === action.payload ? { ...task, isCompleted:
        !task.isCompleted } : task,
        ),
    };

```

```

case actionTypes.DELETE_TASK:
  return {
    ...state,
    tasks: state.tasks.filter((task) => task.id !==
      action.payload),
  };
case actionTypes.OPEN_TASK_MODAL:
  return {
    ...state,
    tasks: state.tasks.map((task) =>
      task.id === action.payload ? { ...task,
        actionsModalOpen: !task.actionsModalOpen } :
      task,
    ),
  };
case actionTypes.CLOSE_TASK_MODAL:
  return {
    ...state,
    tasks: state.tasks.map((task) =>
      task.id === action.payload ? { ...task,
        actionsModalOpen: false } : task,
    ),
  };
case actionTypes.ADD_TASK_PHOTO:
  return {
    ...state,
    tasks: state.tasks.map((task) =>
      task.id === action.payload.idTask ? { ...task,
        photo: action.payload.photo } : task,
    ),
  };
case actionTypes.DELETE_TASK_PHOTO:
  return {
    ...state,
    tasks: state.tasks.map((task) =>
      task.id === action.payload.idTask ? { ...task,
        photo: action.payload.empty } : task,
    ),
  };
default:
  return state;
}
};

```

```
export default groups;
```

src / redux / reducers / backgroundReducer.js

```
import * as actionTypes from '../types/types';

import { dataImages } from '../utils/background';

const initialState = {
  selectedBackground: dataImages[17].img,
  backgroundModalOpen: false,
};

const background = (state = initialState, action) => {
  switch (action.type) {
    case actionTypes.OPEN_BACKGROUND_MODAL:
      return {
        ...state,
        backgroundModalOpen: true,
      };
    case actionTypes.CLOSE_BACKGROUND_MODAL:
      return {
        ...state,
        backgroundModalOpen: false,
      };
    case actionTypes.SELECT_BACKGROUND:
      return {
        ...state,
        selectedBackground: action.payload,
      };
    default:
      return state;
  }
};

export default background;
```

4. Файл localStorage, який відповідає за збереження даних при закритті сторінки

src / localStorage.js

```
export const loadState = () => {
  try {
    const serializedState = localStorage.getItem('state');
```

```

    if (serializedState === null) {
      return undefined;
    }
    return JSON.parse(serializedState);
  } catch (err) {
    return undefined;
  }
};

```

```

export const saveState = (state) => {
  try {
    const serializedState = JSON.stringify(state);
    localStorage.setItem('state', serializedState);
  } catch {}
};

```

5. Головний файл, в якому знаходиться весь застосунок

public / index.html

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1"
  />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <link
      href="https://fonts.googleapis.com/css2?family=Baloo+2:wght@400;500;600&display=swap" rel="stylesheet">
    <title>Tasks Manager</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>

```

6. Файли в папці utils

src / utils / background.js

```
import back1 from '../utils/images/back1.jpg';
import back2 from '../utils/images/back2.jpg';
import back3 from '../utils/images/back3.jpg';
import back4 from '../utils/images/back4.jpg';
import back5 from '../utils/images/back5.jpg';
import back6 from '../utils/images/back6.jpg';
import back7 from '../utils/images/back7.jpg';
import back8 from '../utils/images/back8.jpg';
import back9 from '../utils/images/back9.jpg';
import back10 from '../utils/images/back10.jpg';
import back11 from '../utils/images/back11.jpg';
import back12 from '../utils/images/back12.jpg';
import back13 from '../utils/images/back13.jpg';
import back14 from '../utils/images/back14.jpg';
import back15 from '../utils/images/back15.jpg';
import back16 from '../utils/images/back16.jpg';
import back17 from '../utils/images/back17.jpg';
import back18 from '../utils/images/back18.jpg';
import back19 from '../utils/images/back19.jpg';
import back20 from '../utils/images/back20.jpg';
```

```
export const dataImages = [
  { id: 1, img: back1 },
  { id: 2, img: back2 },
  { id: 3, img: back3 },
  { id: 4, img: back4 },
  { id: 5, img: back5 },
  { id: 6, img: back6 },
  { id: 7, img: back7 },
  { id: 8, img: back8 },
  { id: 9, img: back9 },
  { id: 10, img: back10 },
  { id: 11, img: back11 },
  { id: 12, img: back12 },
  { id: 13, img: back13 },
  { id: 14, img: back14 },
  { id: 15, img: back15 },
  { id: 16, img: back16 },
  { id: 17, img: back17 },
  { id: 18, img: back18 },
  { id: 19, img: back19 },
```

```
    { id: 20, img: back20 },  
  ];
```

src / utils / groupBackgroundColors.js

```
export const groupBackColors = [  
  '#e3e4e6',  
  '#e48257',  
  '#d2e69c',  
  '#8fd9a8',  
  '#7868e6',  
  '#d5ecc2',  
  '#ddffbc',  
  '#897853',  
  '#dbf6e9',  
  '#ffd384',  
  '#f6dfeb',  
  '#f39189',  
];
```

ДОДАТОК В
Слайди презентації

Програмна Реалізація Онлайн Менеджера Задач

1

Виконав

Студент

Групи КНТ-128сп

Желнін Михайло Сергійович

Керівник

к.т.н, доцент

Колпакова Тетяна Олексіївна

Рисунок В.1 – Слайд 1

Мета роботи та задачі дослідження

2

- ▶ **Об'єкт дослідження** – програмні засоби для планування задач.
- ▶ **Предмет дослідження** – браузерні програми для онлайн планування необхідних задач.
- ▶ **Мета роботи** – розробка програмного забезпечення для спрощення процесів планування задач.

Рисунок В.2 – Слайд 2

Таблиця порівняння аналогів

3

Trello	Todoist	TickTick	Google Keep	Microsoft To-Do
- Дуже цікавий інтерфейс; - можливість додавати групу та задачу до неї; - змінювати фон групи та всього застосунку; - гнучке налаштування; - можна додавати теги, терміни виконання.	- Функціональність та мінімалістичний зовнішній вигляд; - підтримка на всіх пристроях; - є можливість додавати справи, розкладати по проектам і спискам; - ставити теги, терміни виконання.	- Зрозумілий інтерфейс; - можна додавати задачі та сортувати; - є можливість додавати коментарі.	- Простий та зрозумілий інтерфейс; - є можливість додавати картки та прикріпити до них списки, файли, рисунок і т.п.; - можна змінювати колір картки; - підтримка різних пристроїв.	- Гнучкість та мінімалістичний інтерфейс; - простий у використанні; - можливість додавати списки справ, задачі і нагадування.

Рисунок В.3 – Слайд 3

Таблиця функціональних можливостей розроблюваного веб-застосунку

4

Основні	Додаткові
- Додавати групу; - змінювати назву групи та видаляти її; - додавати задачу; - відмічати задачу як зроблену; - додавати до задачі фото; - видаляти задачу; - бути простим та зрозумілим у використанні; - використовувати застосунок на різних пристроях; - Можливість відкривати веб-застосунок як звичайну програму.	- Змінювати колір фону групи; - показувати дату створення групи; - змінювати фон всього застосунку.

Рисунок В.4 – Слайд 4

Таблиця видів веб-застосунків та їх порівняння

5

Backend	Frontend	Single Page Application	Multi Page Application	Progressive Web Application
- Працює на віддаленому комп'ютері та написана на мовах програмування PHP, Python, Ruby, C# та інші.	- Виконується в браузері користувача та пишеться на мові JavaScript та бібліотеках і фреймворках на її основі.	- Односторінковий інтерактивний застосунок; - користувач, перемикаючись між вкладками та розділами застосунку залишається на одній сторінці і оновлюються лише потрібні частини контенту.	- Багатосторінковий веб-застосунок; - коли користувач взаємодіє з веб-сайтом, завантажуються нові http-сторінки.	- Схожий за можливостями і функціями до звичайних застосунків; - браузер виступає віртуальною машиною для запуску такого виду застосунку.

Рисунок В.5 – Слайд 5

Таблиця технологій, які використовує PWA

6

Основні складові PWA
- Service Worker - це проксі-шар між серверною і клієнтською частиною; - HTTPS – PWA-застосунок потребує більш надійного протоколу для безпеки; - Web App Manifest – це файл, в якому описується візуальна складова застосунку; - Push Notifications – спливаючі підказки та повідомлення.

Рисунок В.6 – Слайд 6

Таблиця переваг та недоліків PWA

7

Переваги	Недоліки
- Працюють на будь-якому пристрої; - розробнику досить створити одну програму для всіх платформ; - оновлення в режимі реального часу завдяки сервіс Воркер; - працює тільки по HTTPS протоколу; - можна закріпити іконку на домашньому екрані телефону; - більшість функцій доступні офлайн; - посиланням на PWA-застосунок можна ділитися, як на звичайний вебсайт.	- Не підтримується на старих смартфонах; - споживання енергії вище, ніж у простого сайту; - PWA-застосунки не представлені в магазин Google Play та AppStore; - немає доступу до апаратних ресурсів пристроїв (наприклад, блютуз або NFC).

Рисунок В.7 – Слайд 7

Структура папок та файлів

8

- ▶ `public` – файли в цій папці публікуються на хостінгу;
- ▶ `src` – головна папка, де збираються всі файли;
- ▶ `src/components` – компоненти, з яких складається додаток;
- ▶ `src/redux` – файли, які відповідають за збереження даних;
- ▶ `utils` – файли зображень та кольорів.

```

> node_modules
> public
  > src
    > components
      > BackgroundModal
      > GroupColorModal
      > GroupList
      > GroupModal
      > Header
      > TaskModal
      > TaskList
    > redux
      > actionCreators
      > reducers
      > types
    > store.js
    > utils
    > App.js
    > App.scss
    > index.js
    > index.scss
    > localStorage.js
    > service-worker.js
    > serviceWorkerRegistration.js
    > .gitignore
    > package-lock.json
    > package.json
  
```

Рисунок В.8 – Слайд 8

Схема функціонування Redux

9

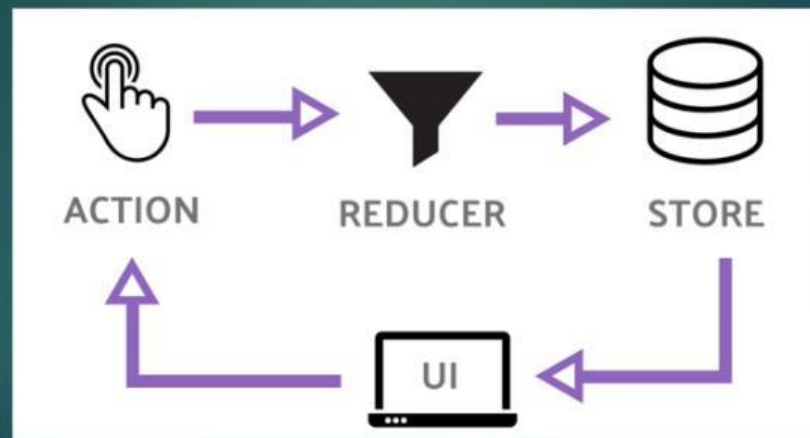


Рисунок В.9 – Слайд 9

Додавання групи

10

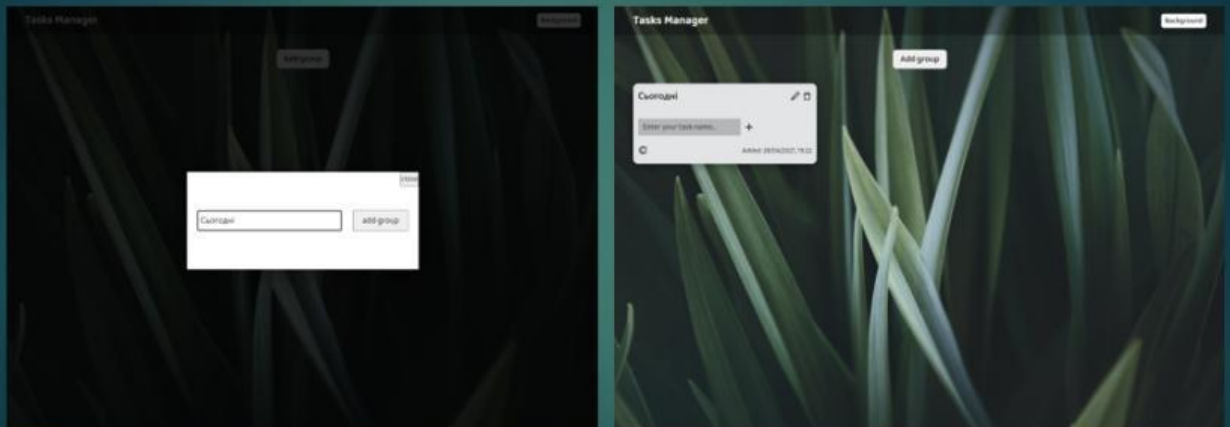


Рисунок В.10 – Слайд 10

Додавання задачі

11

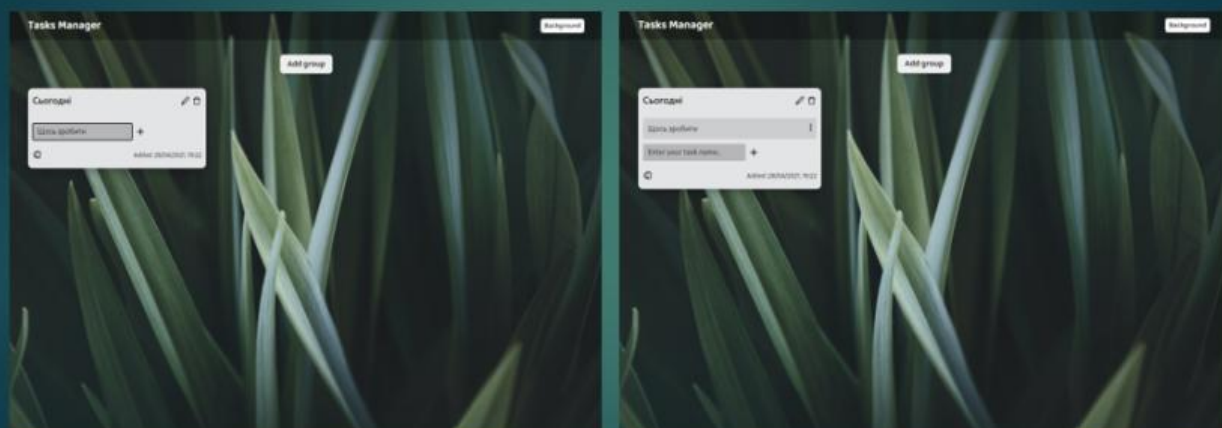


Рисунок В.11 – Слайд 11

Зміна фону застосунку

12

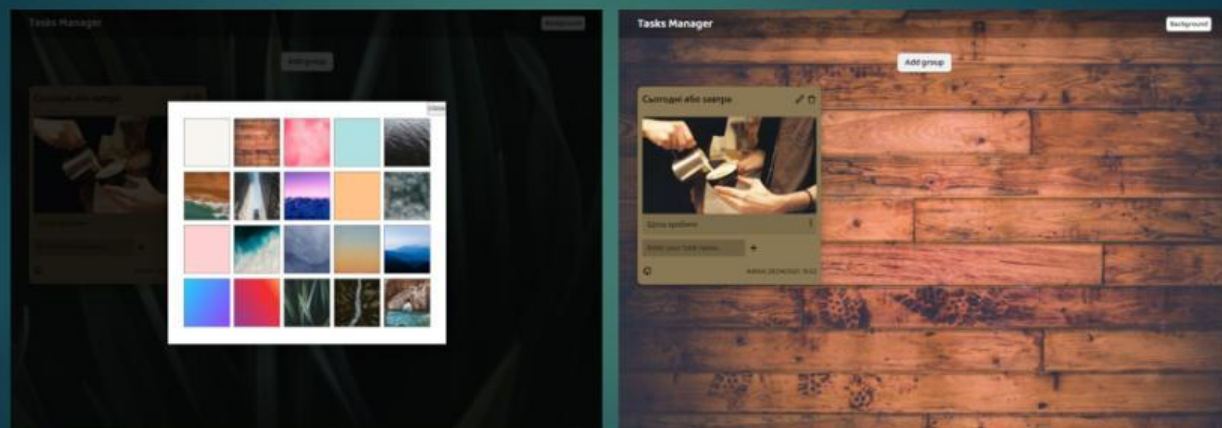


Рисунок В.12 – Слайд 12

Відкривання застосунку як PWA

13

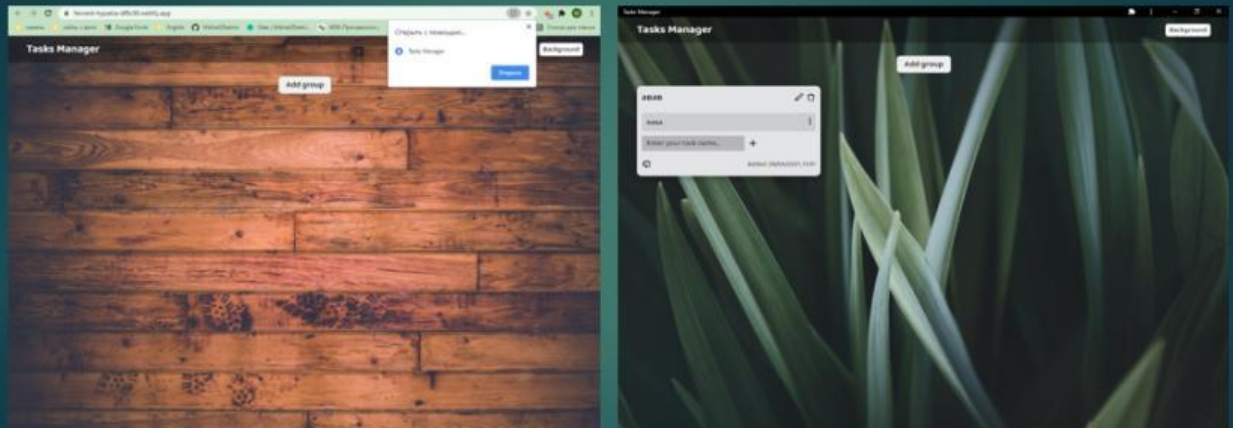


Рисунок В.13 – Слайд 13

Таблиця тестування основних функцій програми

14

Дія	Очікуваний результат	Результат
Дії з групами	- Група додається у список; - назва групи змінюється; - група видаляється; - фон змінюється; - дата створення коректно відображається.	- Група додається у список; - назва групи змінюється; - група видаляється; - фон змінюється; - дата створення коректно відображається.
Дії з задачами	- Задача додається; - Модальне вікно у задачі відкривається; - Задача відмічається як зроблена; - Зображення додається; - Задача видаляється.	- Задача додається; - Модальне вікно у задачі відкривається; - Задача відмічається як зроблена; - Зображення додається; - Задача видаляється.

Рисунок В.14 – Слайд 14

Таблиця тестування додаткових функцій програми

15

Дія	Очікуваний результат	Результат
Відкриття застосунку	- Застосунок відкрито; - назва застосунку, кнопка вибору фону та кнопка додавання задачі коректно відображаються.	- Застосунок відкрито; - назва застосунку, кнопка вибору фону та кнопка додавання задачі коректно відображаються.
Зміна фону застосунку	- При натисканні кнопки "Background" відкривається модальне вікно; - при виборі фону він змінюється.	- При натисканні кнопки "Background" відкривається модальне вікно; - при виборі фону він змінюється.
Збереження даних	- При закритті вкладки чи вікна браузера дані зберігаються.	- При закритті вкладки чи вікна браузера дані зберігаються.

Рисунок В.15 – Слайд 15

Висновки

16

- Розглянуто поняття веб-застосунків та їх видів, основні етапи проектування та методи розробки.
- Зроблений порівняльний аналіз існуючих застосунків.
- На підставі аналізу було сформовано ряд основних вимог до розроблюваного застосунку.
- Розроблені макети та на їх основі корисувачський UI.
- Розроблено сховище даних за допомогою бібліотеки Redux.
- Проаналізовано процес роботи з програмою та її детальне тестування.

Рисунок В.16 – Слайд 16