Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

Самостійна робота

з дисципліни «Технологія створення програмних продуктів» на тему:

«РОЗРОБКА ІНСТРУМЕНТУ УПРАВЛІННЯ ЧАСОМ НА ОСНОВІ МЕТОДУ POMODORO»

Виконав:

ст.групи КНТ-113сп Іван ЩЕДРОВСЬКИЙ

Прийняв:

професор Андрій ОЛІЙНИК

2025

РЕФЕРАТ

Пояснювальна записка до самостійної роботи містить 55 сторінок, 35 рисунків, 0 таблиць, 1 додаток, 9 джерел.

Пояснювальна записка складається з п’яти розділів.

Розділ «Аналіз предметної області» містить в собі аналіз методу Pomodoro, його недоліків та аналіз вимог то застосунку, який буде розроблятись.

Розділ «Проектування програмного забезпечення системи» містить вибір мови програмування, вибір базових технологій, середовища розробки та дизайн інтерфейсу застосунку.

Розділ «Розробка програмного забезпечення системи» містить опис логіки реалізованого програмного забезпечення, включаючи діаграми.

Розділ «Розробка документів на супроводження програмного забезпечення» містить загальний опис, інструкцію для оператора, де є інструкція по запуску програмного забезпечення, а також інструкцію користувача.

Розділ «Тестування програмного забезпечення» містить аналіз застосунку на швидкодія та масштабованість, а також тестування всього програмного забезпечення

УПРАВЛІННЯ ЧАСОМ, POMODORO МЕТОД, ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ, ВЕБСАЙТ, TYPESCRIPT, VUE, GIT

ЗМІСТ

ВСТУП

Управління часом є важливою проблемою в сучасному світі, де зростає обсяг завдань і темп змін. Ефективне використання часу визначає успіх у особистому та професійному житті.

Метод Pomodoro, розроблений Франческо Чірілло в кінці 1980-х років, є одним з найпопулярніших підходів. Він базується на циклах роботи і коротких перерв, з довшою перервою після чотирьох циклів. До переваг цього методу можна віднести простоту, підвищення концентрації та продуктивності для різноманітних завдань [1].

Метод застосовують провідні компанії, як Google, Microsoft, Amazon та Apple. Дослідженням займаються фахівці, зокрема Франческо Чірілло, Томас Манкін та Томас Голд. Популярність методу зростає завдяки його ефективності та доступності.

Актуальність теми цієї роботи полягає в тому, що метод популярний, але має недоліки, які потребують усунення для кращої адаптації.

Мета полягає в створенні інструменту, що поєднає переваги Pomodoro з сучасними технологіями, усуне обмеження та підвищить ефективність.

Інструмент матиме простоту використання, ефективність для різноманітних завдань, підвищення концентрації та продуктивності, адаптацію для довготривалих задач.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Управління часом є дуже важливим аспектом нашого життя в сучасному світі, оскільки час є обмеженим ресурсом та ми хочемо ефективно використовувати його для досягнення поставлених цілей. Саме ефективне управління часом є однією з ключових складових успіху в особистому та професійному житті.

Основними завданнями управління часом є:

– планування, тобто визначення цілей та розробка детального плану їх досягнення;

– організація, розподіл завдань у часі та ресурсах;

– контроль, або ж моніторинг виконання плану та внесення коректівів у разі необхідності.

В світі існує безліч методів для управління часом, кожен з яких має свої переваги та недоліки. Одним з таких методів, а також він один з найбільш популярних, є метод Pomodoro який був розроблений Франческо Чірілло в кінці 1980-х років.

Сама техніка зазвичай описується як цикл роботи-відпочинку, де ми працюємо 25 хвилин, відпочиваємо 5 хвилин і так чотири рази. Після четвертого підходу роботи відпочиваємо не 5 хвилин, а 15 або навіть 30. Цей опис є гарним та показує цикли роботи, які можна просто зрозуміти та почати застосовувати.

Але, на жаль, цей опис є не повним. Наприклад, він не включає в себе стадію аналізу задачі, запису та нотування прогресу по кожній з них, щоденного аналізу, покращення своїх навиків, а також передбачення кількості часу та ресурсів, які займе задача. Повноцінна техніка Pomodoro є набагато більш повнішою, а ніж вона зазвичай описується [1].

Розробити середовище, яке буде повноцінно задовольняти потреби всіх користувачів, які хочуть використовувати цю техніку майже не можливо, оскільки у кожної людини є свої переваги в технологіях, наприклад, одна

людина полюбляє ввести нотатки на папері, коли ж інша веде їх в якісь програмі.

Через це система, яка буде розроблятись, не повинна задовольняти всі аспекти Pomodoro, натомість вона повинна вирішувати лише проблему з часом, коли користувач хоче фокусуватись, а також з введенням статистики. Планування, аналіз, покращення та передбачення часу залишається на користувачі.

Предметною областю цієї самостійної роботи є розробка програмного забезпечення для підвищення особистої продуктивності.

Об'єктом дослідження є процеси управління особистим часом та підвищення продуктивності праці за допомогою спеціалізованих програмних засобів.

1.1 Аналіз вимог

Основною вимогою до застосунку є вирішення проблеми з програмною частиною використання техніки Pomodoro.

Цю техніку можна використовувати навіть зі звичайним таймером на телефоні, або ж фізичним, запускаючи його, а в кінці записуючи результати. Це буде достатньо ефективно, оскільки те, що персона використовує в якості інструменту має меншу вагу, а ніж рефлексія, яка йде після.

На основі цього, застосунок повинен допомагати користувачу простіше аналізувати інформацію через збереженні статистики та, можливо, її візуальне відображення.

Користувачу повинно бути приємно використовувати застосунок, наприклад, через використання приємних кольорів, або ж навіть з можливістю їх зміни.

В сервісі повинен бути таймер, який буде працювати в трьох режимах, а саме в режимі роботи, малої та великої перерв. Користувачу повинно бути

зрозуміло в якому режимі він зараз знаходиться, а також він повинен мати змогу вручну перемикати цей режим.

Застосунок повинен мати багато налаштувань, наприклад, зміну часу роботи, або ж кількість підходів для відпочинку. Якщо користувач хоче працювати не 25 хвилин та 5 відпочивати, а, наприклад, 50 на 10, або ж навіть 1:30 на 15, то застосунок повинен дозволяти це зробити.

Одним з ключових налаштувань є зміна кольору. Кожен режим таймера повинен мати свій унікальний незалежний колір, який буде красити всю сторінку, кнопки тощо в нього. Таким чином користувач зможе візуально зрозуміти чи зараз час працювати, чи зараз час відпочити.

Користувачі можуть працювати над декількома проєктами одночасно, а тому сервіс повинен підтримувати можливість створення категорій. Категорія може бути призначена до таймера та попаде в статистику після його виконання. Це дозволить в майбутньому користувачу аналізувати час, який він витратив на різні справи.

Для забезпечення базової зручності використання вся інформація повинна зберігатись після закриття програми. Тобто, якщо користувач закриває сторінку браузера, або ж весь браузер, то після його відкриття інформація все ще повинна бути доступна.

Для забезпечення максимальної безпеки даних користувача, вся інформація повинна зберігатись тільки в браузері та не відправлятись через мережу. При локальному встановленні застосунку його можна використовувати навіть без підключення до мережі інтернет. Якщо ж користувач очищує кеш браузера, то дані з застосунку повинні бути видалені в тому числі.

Застосунок буде розроблятись для виконання в браузерах, а тому має основну вимогу щодо його роботи в усіх версіях найбільш популярних браузерів, які були випущенні після 2020 року. Такий часовий діапазон охоплює абсолютну більшість користувачів.

Для забезпечення максимальної зручності використання сервісу він повинен бути швидкий, тобто швидко відкриватись, а також швидко виконувати всі операції з ним.

Дизайн повинен бути простим та зрозумілим, щоб у користувачів не виникало запитань як ним користуватись. Також дизайн повинен мати пастельні кольори, щоб створювати гарну атмосферу в випадках, коли користувач захоче відкрити сайт на другому моніторі, або ж на телефоні.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір мови програмування

Основною та єдиною мовою програмування доступною в браузері є JavaScript, тому її використання для розробки сервісу є логічним і необхідним.

JavaScript – легка, JIT-компільована, об'єктно-орієнтована мова з функціями першого класу, а також найвідоміша скриптова мова для веб-сторінок, але також використовується в багатьох не браузерних оточеннях.

JIT-компільована, також це називається Just-in-Time компільована, означає, що код програми спочатку компілюється в проміжковий код, який потім виконується рядок за рядком через JavaScript Virtual Machine. Доказом того, що в JavaScript є компіляціє полягає в обробці помилок, а саме обробці помилок синтаксису. JavaScript рушій не виконає код, якщо в ньому є синтаксична помилка на будь-якому рядку коду, на відміну від, наприклад, Python [2].

JavaScript може виконуватись як за допомогою браузерного оточення або, наприклад, NodeJS. NodeJS та більшість браузерів на даний момент використовують в якості рушія V8 який компілює JavaScript код безпосередньо у власний машинний код, минаючи стадію проміжного байт-коду, а також ефективно керує пам’яттю.

JavaScript є динамічно типізованою мовою програмування, тобто основна частина перевірок типів виконується під час виконання програми, а не під час компіляції. У динамічній типізації значення мають типи, а змінні – ні, тому змінна може містити значення будь-якого типу.

Динамічність дозволяє простіше писати програму, але може заплутати розробника, а в деяких ситуація помилка в коді через типізацію може випадково попасти до клієнта та викликати масу помилок.

Тому над JavaScript існує безліч надбудов, які додають ті, чи інші функції до мови, а потім компілюються в JavaScript який далі виконується. Найбільш популярною надбудовою є TypeScript.

Оскільки JavaScript динамічно-типізована мова програмування її не завжди зручно використовувати без доповнень та надбудов в розробці програмних застосунків. Для більш комфортної, продуктивної та якіснішої розробки застосунку було обрано TypeScript.

TypeScript — це строго типізована мова програмування, яка ґрунтується на JavaScript і надає кращі інструменти будь-якого масштабу.

TypeScript додає до JavaScript додатковий синтаксис для підтримки більш тісної інтеграції з редактором. Дозволяє ловити помилки на ранніх стадіях розробки [3].

В силу повної зворотної сумісності адаптація наявних застосунків з JavaScript на TypeScript може відбуватися поетапно, шляхом поступового визначення типів. Підтримка динамічної типізації зберігається — компілятор TypeScript успішно обробить і не модифікований код на JavaScript.

Основний принцип — будь-який код на JavaScript сумісний з TypeScript, тобто в програмах на TypeScript можна використовувати стандартні JavaScript-бібліотеки і раніше створені напрацювання. Більш того, можна залишити наявні JavaScript-проєкти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли, які не заважатимуть розробці і прямому використанню проєкту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

2.2 Вибір середовища розробки

В якості основного середовища розробки сервісу було обрано Neovim.

NeoVim – цепрозширюваний текстовий редактор, похідний від Vim, який відомий своєю швидкістю, ефективністю та потужними можливостями налаштування.

Хоча NeoVim не є традиційною IDE у повному розумінні (наприклад, як Visual Studio Code або WebStorm), він, завдяки широкому набору плагінів та інтеграцій, дозволяє створити повноцінне робоче середовище для розробки, включаючи автодоповнення, відлагодження, інтеграцію з Git та лінтери. Його легковажність та висока продуктивність є перевагами для оптимізації робочого процесу розробника [4].

В якості систему контролю версіями був обраний Git.

Система контролю версій - це система, що записує зміни у файл або набір файлів протягом деякого часу, так щоб розробник зміг повернутися до певної версії пізніше.

У 2005 році відносини між спільнотою розробників ядра Linux і комерційною компанією, що розробила BitKeeper почали псуватись, і безкоштовне використання продуктом було скасовано. Це підштовхнуло розробників Linux (і зокрема Лінуса Торвальдса, автора Linux) розробити власну систему, ґрунтуючись на деяких з уроків, які вони дізналися під час використання BitKeeper [5].

Як видно з графіку зображеного на рисунку 3.1 популярності різних систем контролю версій 2022 року серед професійних розробників від StackOverFlow на даний момент Git є найбільш популярною системою контролю версій.

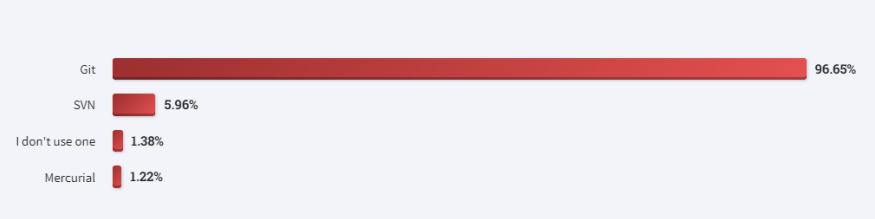

Рисунок 2.1 – Графік популярності різних систем контролю версій 2022 року серед професійних розробників від StackOverflow

2.3 Структурна схема розробки

Для виконання цієї самостійної роботи було використано фреймворк Vue, який має структурну схему View-ViewModel-Model. Більш детально це відображенно на рисунку 2.2.

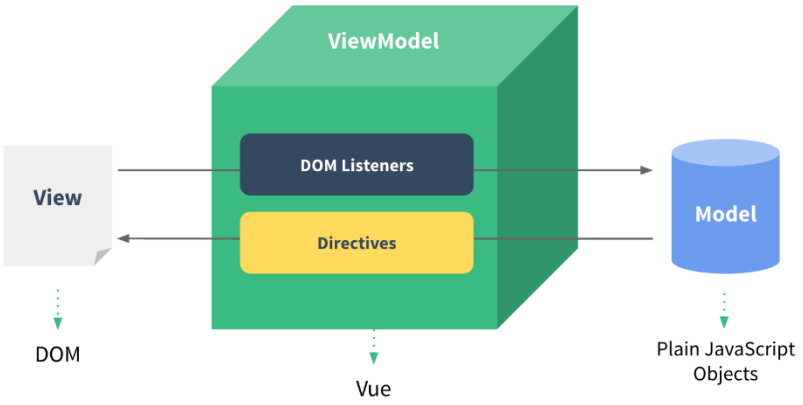

Рисунок 2.2 - View-ViewModel-Model

Архітектура застосунку та його внутрішня структура проєктуватись не будуть, оскільки вони сильно залежать від проблем під час реалізації та всіх можливих нюансів, які не можливо врахувати під час проєктування [6].

2.4 Проєктування інтерфейсу застосунку

Проєктування інтерфейсу застосунку починається зі створення варфремів, або ж замальовок майбутнього інтерфейсу

На рисунках 2.3 – 2.7 зображені розроблені варферйми, а саме два варіанти головного таймера, два варіанти панелі налаштувань та один варіант панелі статистики.

Рисунок 2.3 – Перший варіант головного таймера

Рисунок 2.4 – Другий варіант головного таймера

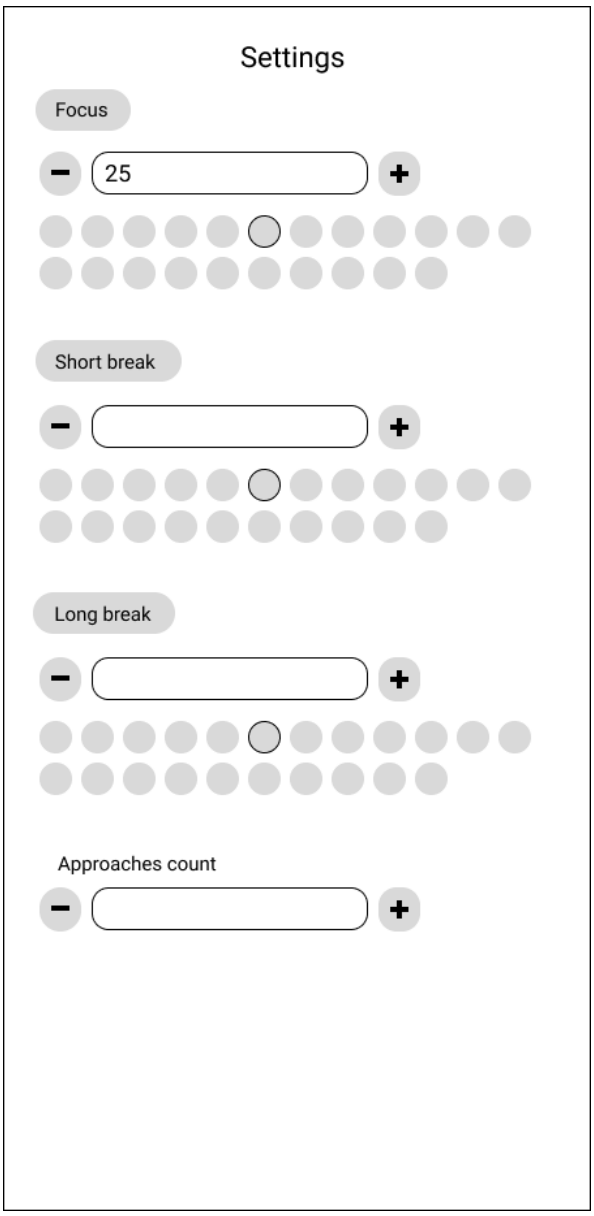

Рисунок 2.5 – Перший варіант налаштувань застосунку

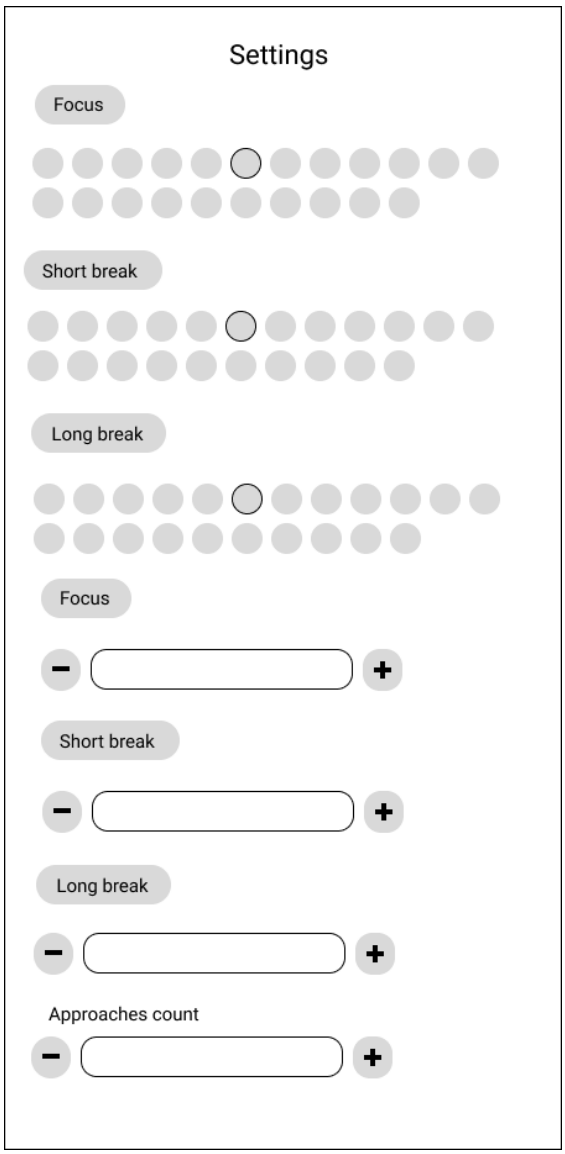

Рисунок 2.6 – Другий варіант налаштувань застосунку

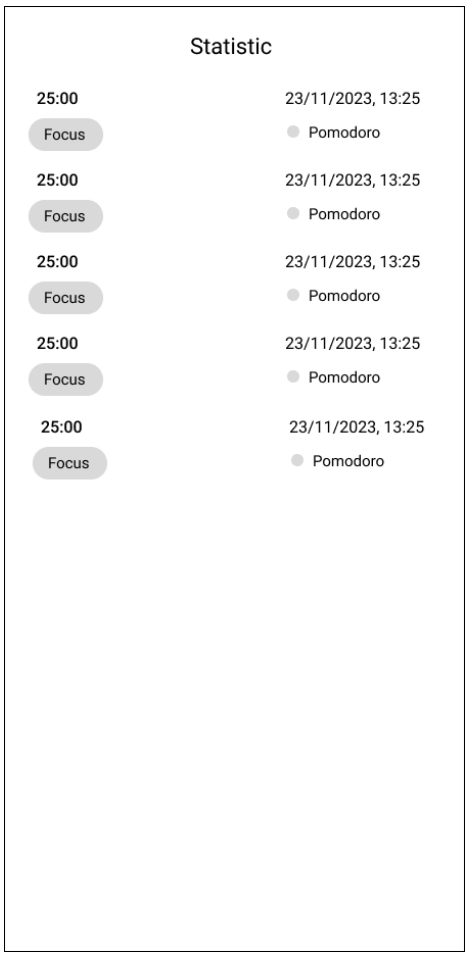

Рисунок 2.7 – Панель статистики застосунку

Було обрано перший варіант головного таймера та перший варіант налаштувань для реалізації на основі аналізу зручності використання.

Далі були створені мокапи – середньо або високо деталізоване статичне уявлення дизайну.

На рисунках 2.8 та 2.9 зображені мокапи головного таймера та панелі налаштувань.

Рисунок 2.8 – Мокап головного таймера

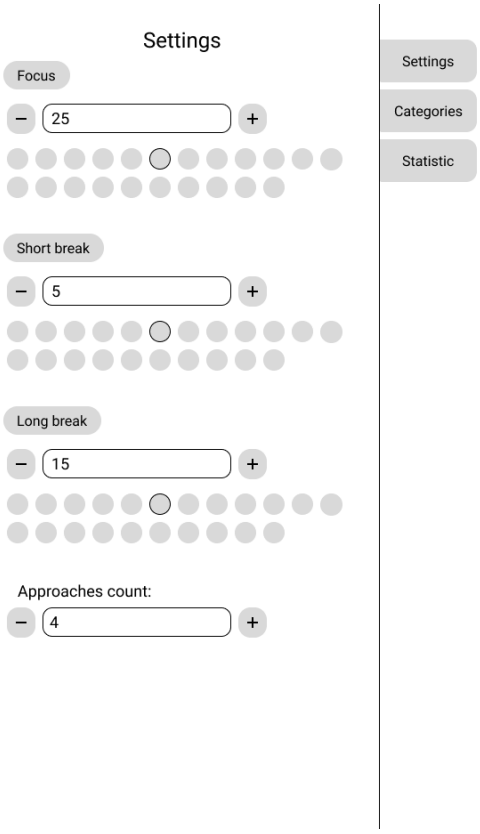

Рисунок 2.9 – Мокап панелі налаштувань

Після створення мокапів було створено прототип –середньо або високо деталізоване уявлення кінцевого продукту, яке імітує взаємодію користувача з інтерфейсом

На рисунках 2.10 та 2.11 зображений створений прототип.

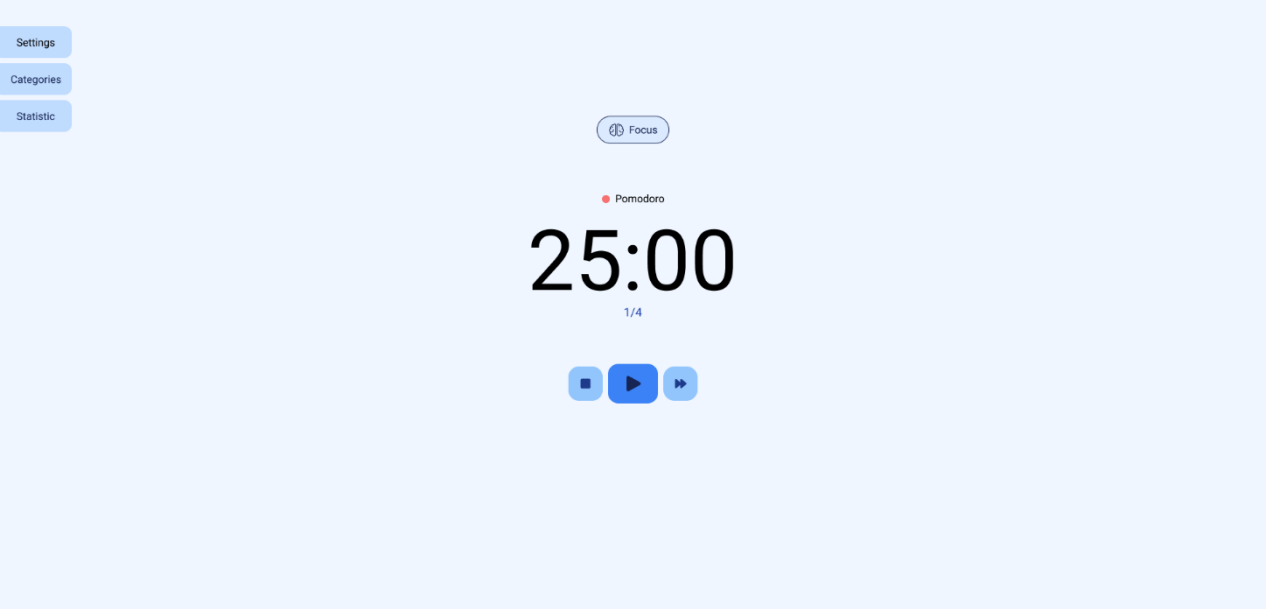

Рисунок 2.10 – Прототип таймера

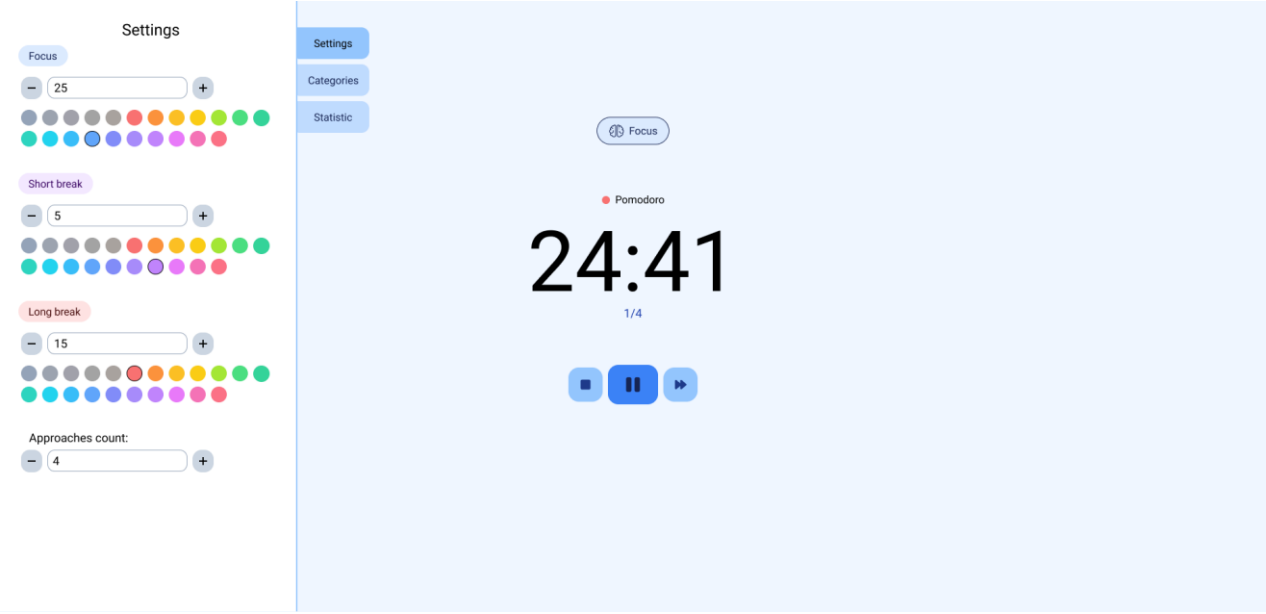

Рисунок 2.11 – Прототип запущеного таймера з відкритою вкладкою налаштувань

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

Як і проєктувалось раніше, в якості основної мови програмування було обрано TypeScript, а як основний фреймворк Vue 3 і систему керування версіями Git.

Для роботи зі стилями було обрано TailwindCSS 3. Це CSS фреймворк який додає велику кількість класів за допомогою яких можна швидко стилізувати елементи на сайті.

Також основну палітру цього фреймворка було використано для механізму перемикання кольорів, про який мова піде трошки згодом.

Як систему керування залежностями проєкту було обрано Performant Node Package Manager, він же pnpm. Було обрано саме його, оскільки він кешує глобально всі встановлені залежності, а тому при встановлені їх локально використовує їх з кешу замість того, щоб кожен раз їх скачувати. Це особливо критично при використанні git worktree, оскільки в кожному worktree потрібно встановлювати залежності незалежно [7].

Для зручної розробки та збірки проєкту було використано Vite. Альтернативою Vite є webpack, але Vite працює в декілька разів швидше, і також його створили розробники, які в тому числі створили Vue, тому при його використанні повинно бути мінімум проблем

Для забезпечення високих стандартів якості коду було встановлено форматувальник коду, який гарантує, що весь код буде мати однакові стилістичні правила. Для prettier також було додатково встановлено плагін для автоматичного сортування імпортів

Також було встановлено eslint, який гарантує що в проєкті використовуються, або ж не використовуються, деякі фукнції/конструкції. Для автоматичного запуску скриптів для перевірки коду при кожному коміті в Git було використано commit pre-hook, а саме husky та lintstaged.

Для роботи з текстами було встановлено vue-i18n, який додає можливість підтримувати багато мов в майбутньому. Наразі цей функціонал не

використовується, але за рахунок його встановлення вже зараз в майбутньому буде набагато простіше додати нові мови до сервісу [8].

Також в проєкті було використано глобальне сховище Pinia. Про його використання буде детально описано згодом

На рисунку 3.1 показано структуру файлової системи в корені застосунку

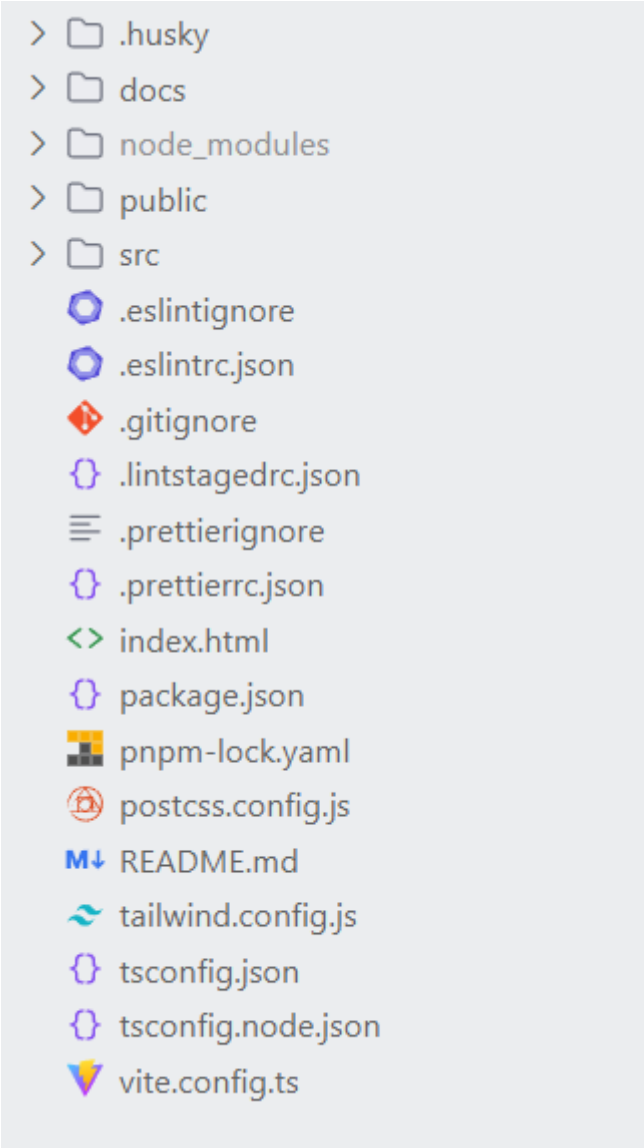

Рисунок 3.1 – Файлова структура кореня програми

На рисунку 3.2 показано структуру файлів вихідного коду

Рисунок 3.2 – Структура файлів вихідного коду

На рисунках 3.3 та 3.4 показано зв’язки між файлами вихідного коду. Діаграма була створена за допомогою використання madge та graphviz [9].

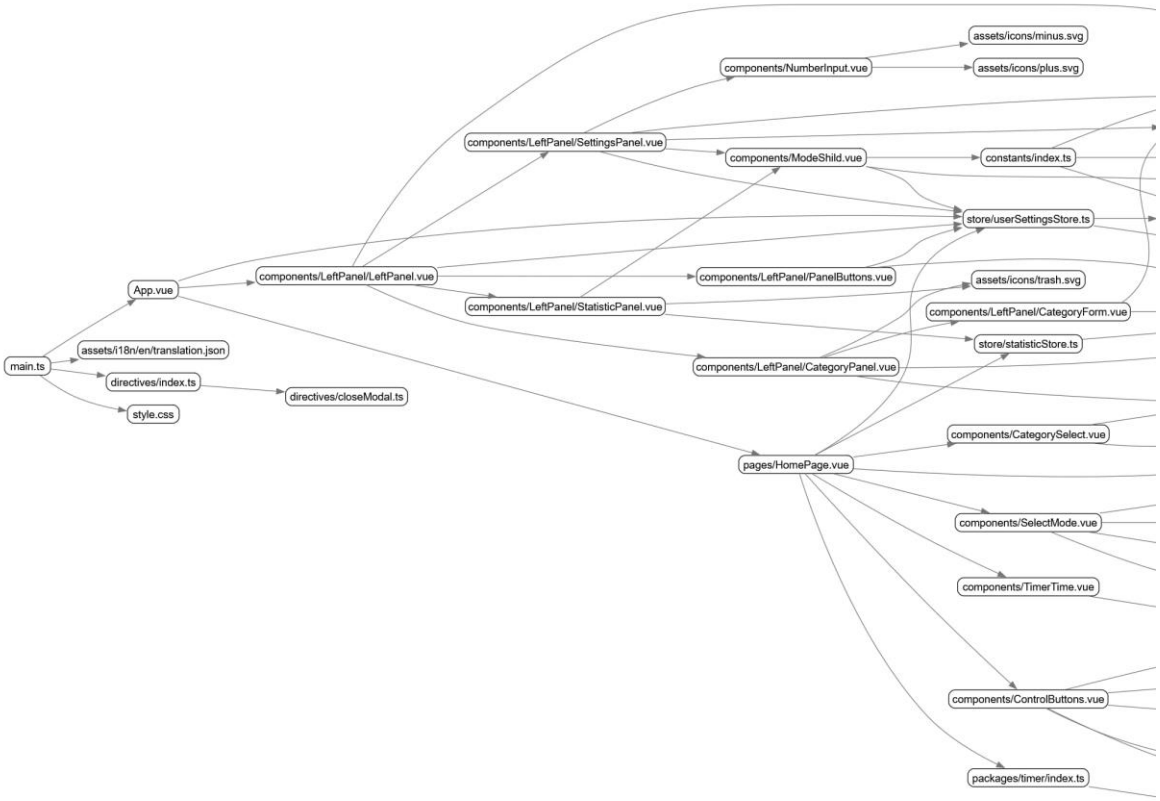

Рисунок 3.3 – Зв’язки між файлами 1

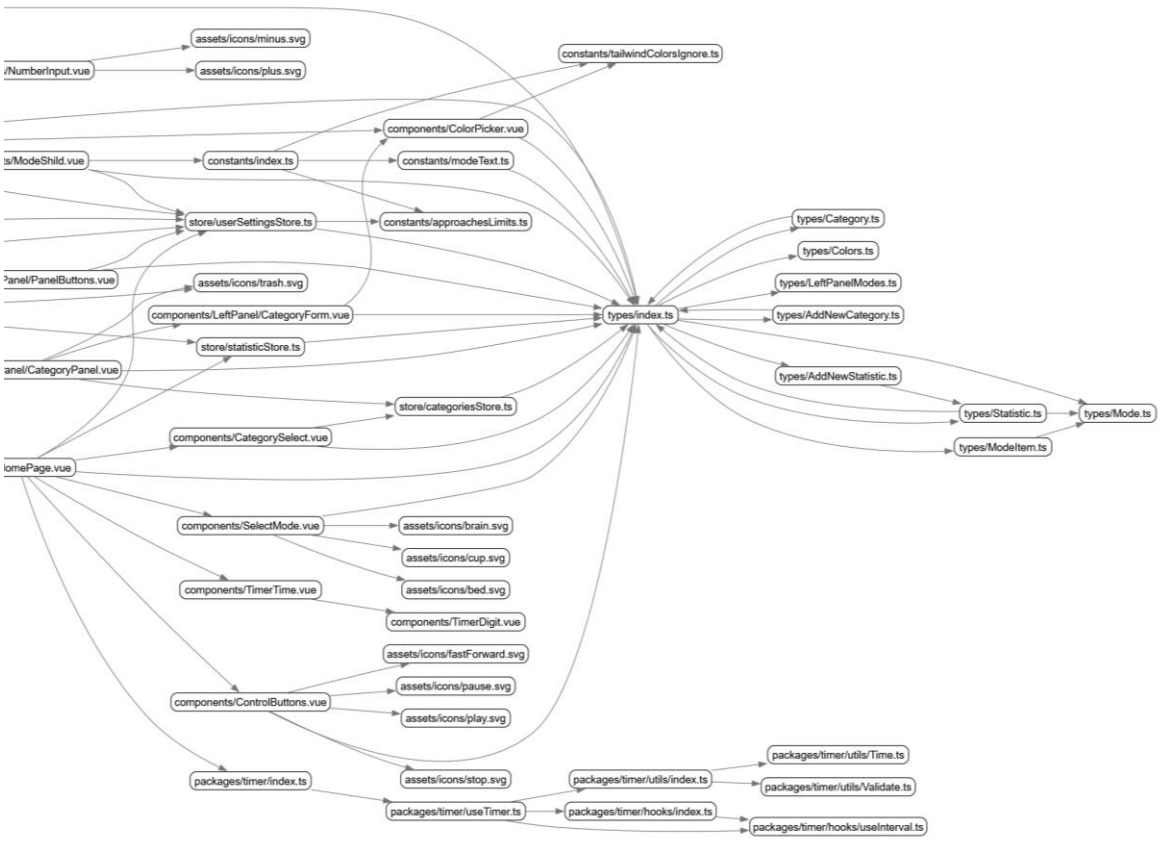

Рисунок 3.4 – Зв’язки між файлами 2

В застосунку використовується три глобальних сховища даних на основі Pinia.

Перше сховище називається «UserSettingsStore» та використовується для зберігання налаштувань користувача та даних користувача

В цьому сховищі для кожного режиму таймера зберігається його унікальний колір, а також час.

Також в цьому сховищі зберігається поточний режим таймера, обрана категорія та кількість підходів сфокусованої роботи для великої перерви

Всі ці значення синхронізовані з екраном таймера та панеллю налаштувань

Додатково в цьому сховищі зберігається поточна кількість виконаних підходів роботи та додатковий внутрішній флаг системи

В другому сховищі, яке називається «StatisticStore», зберігається вся статистика пов’язана з використанням таймеру, а саме всі підходи його використання, в тому числі перерви

В третьому сховищі, яке називається «CategoryStore», зберігаються всі категорії, які створив користувач. Важливо, що при видаленні категорії вона не буде видалена зі статистики

Для реалізації зміни кольорів було використано CSS змінні та об’єкт зі всіма кольорами. При зміні кольору для активного режиму компонент створює свої CSS змінні і використовує їх. Цей підхід має недоліки, але є простішим в реалізації та розумінні системи. Другий варіант це реалізувати глобальні зміні, які б змінювали колір відразу всіх елементів, але тут буде проблема коли потрібно зробити елементи з не поточним кольором

Для правильної розробки системи також було створено sequence діаграму взаємодії користувача з системою. Ця діаграма показує яким чином користувач буде взаємодіяти, що він буде робити, а також яким чином буде на це буде реагувати система. Важливо зауважити, що в системі буде три окремих сховища, коли на діаграмі показано тільки одне, це було зроблено для простоти візуалізації. Також важливо зауважити, що сховище, яке показано на діаграмі є

внутрішнім сховищем сайту та автоматично синхронізується зі сховищем браузера. Діаграма є досить великою, а тому розділена на чотири частини, які показані на рисунках 3.5, 3.6, 3.7 та 3.8.

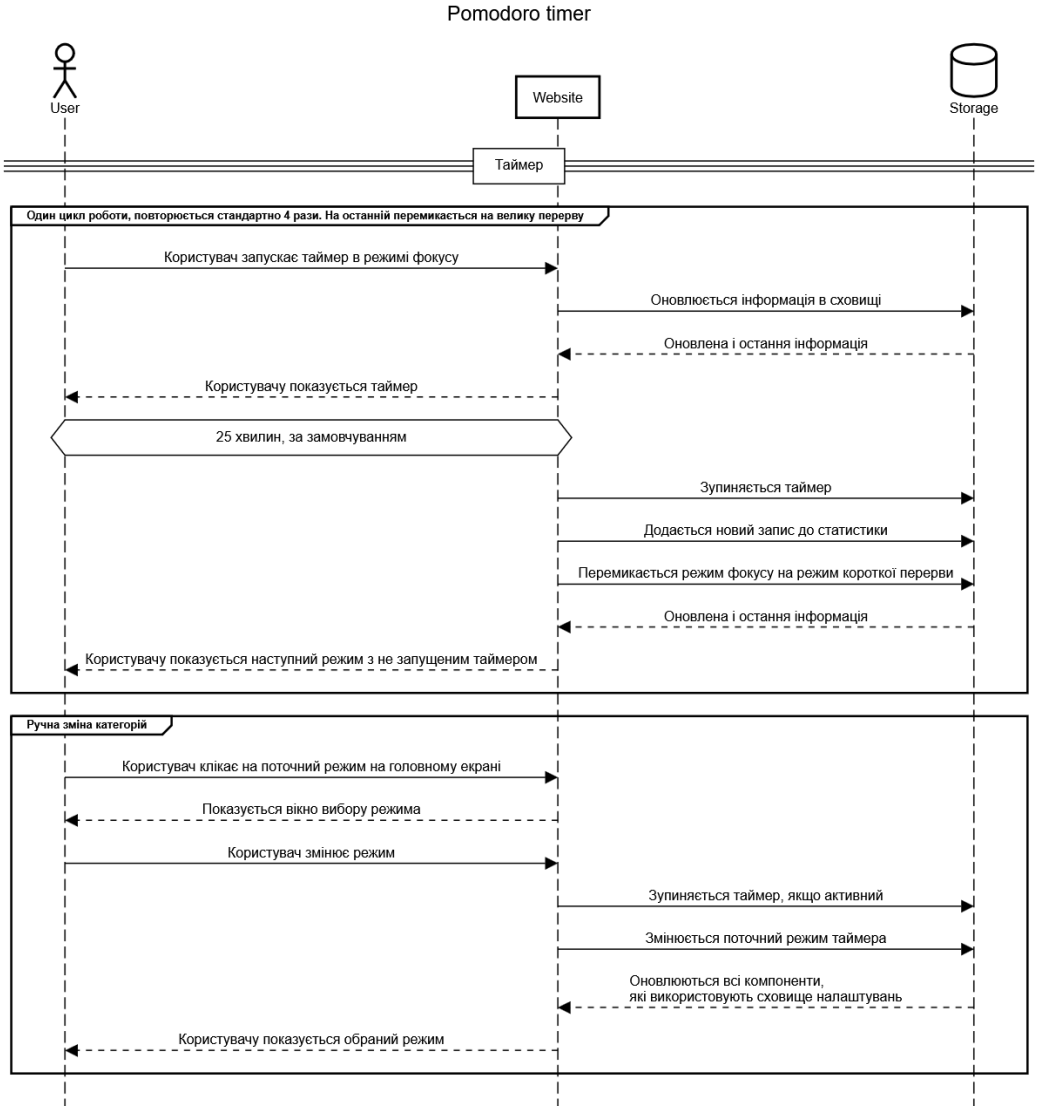

Рисунок 3.5 – Діаграма взаємодії користувача з таймером

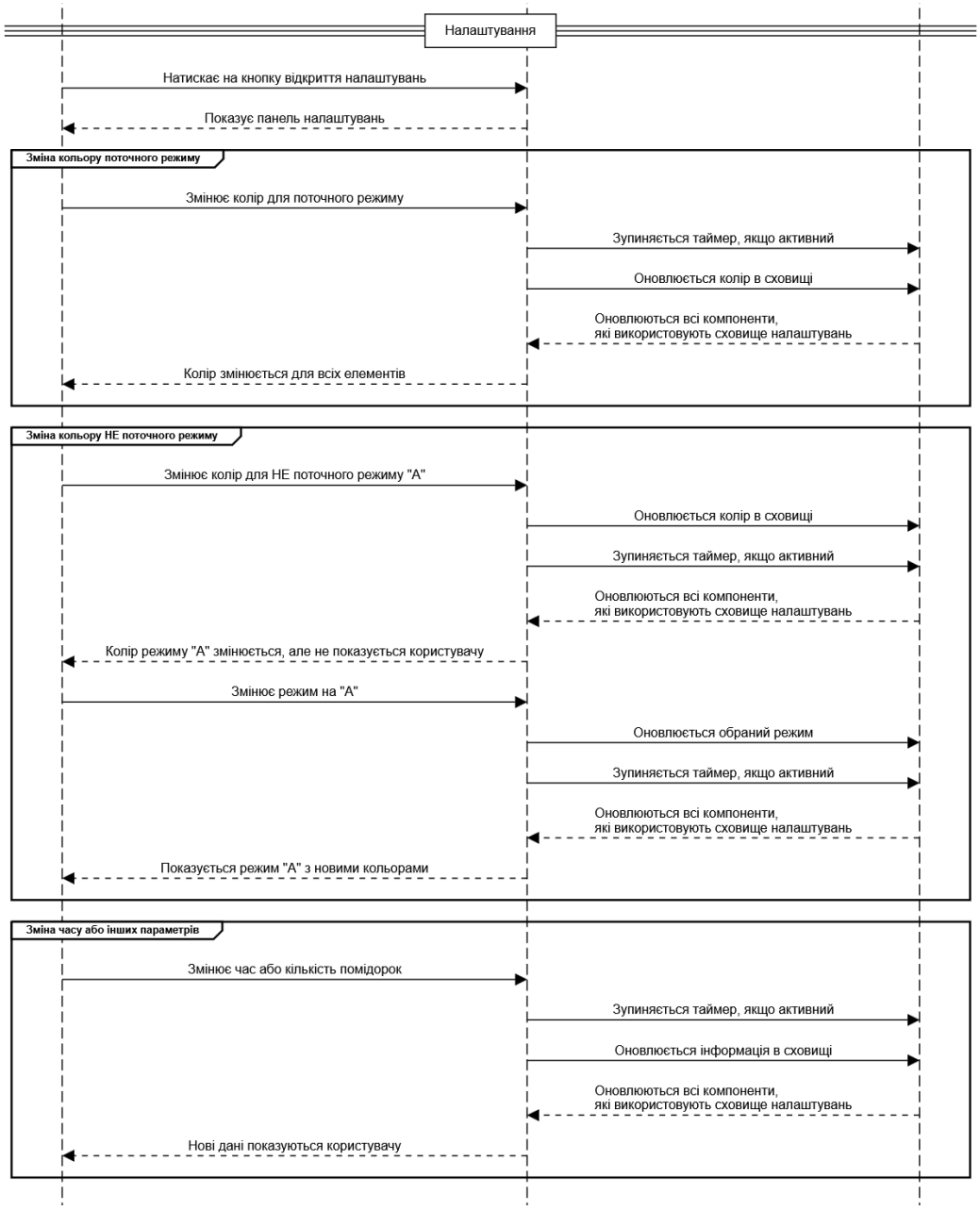

Рисунок 3.6 – Діаграма взаємодії користувача з налаштуваннями

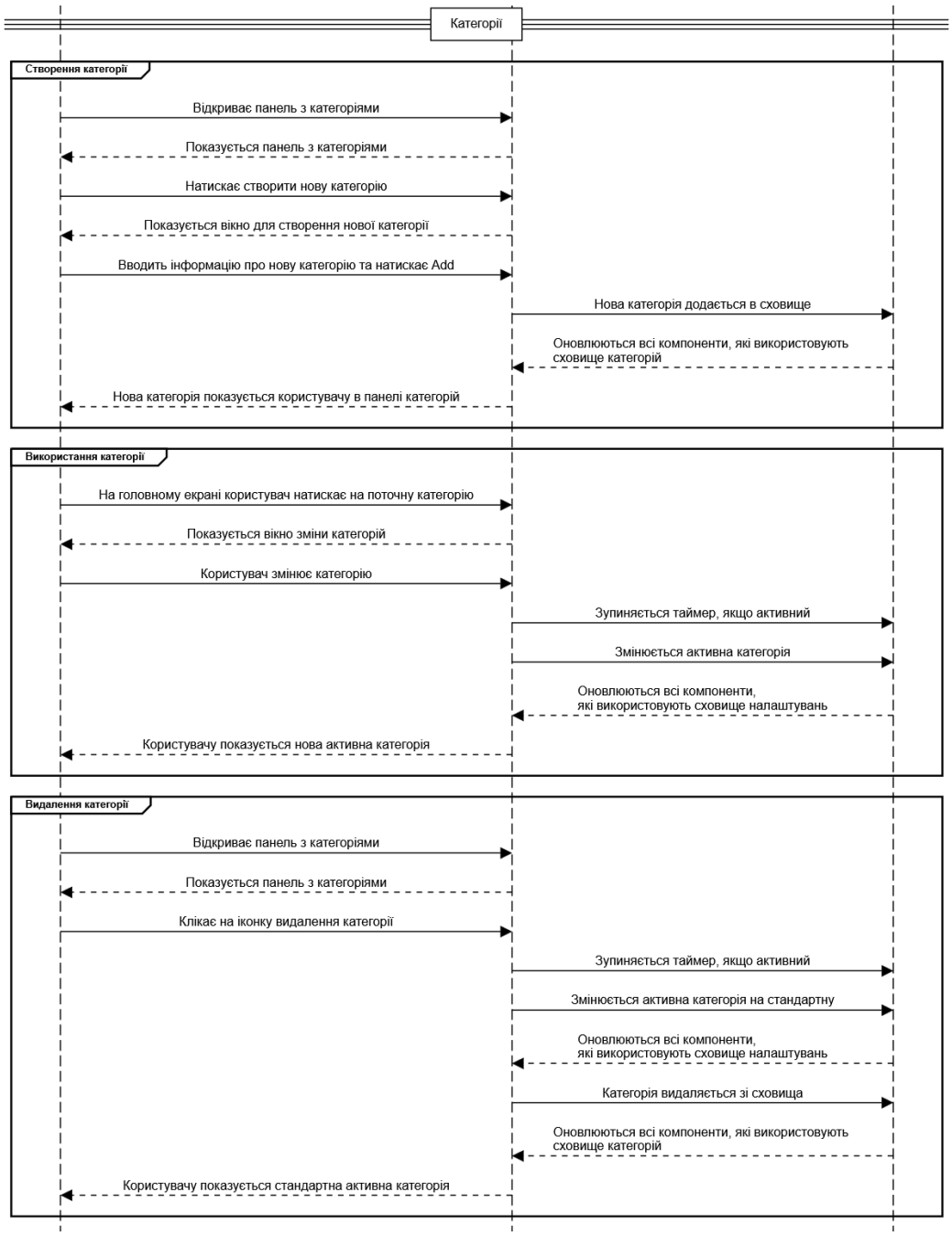

Рисунок 3.7 – Діаграма взаємодії користувача з категоріями

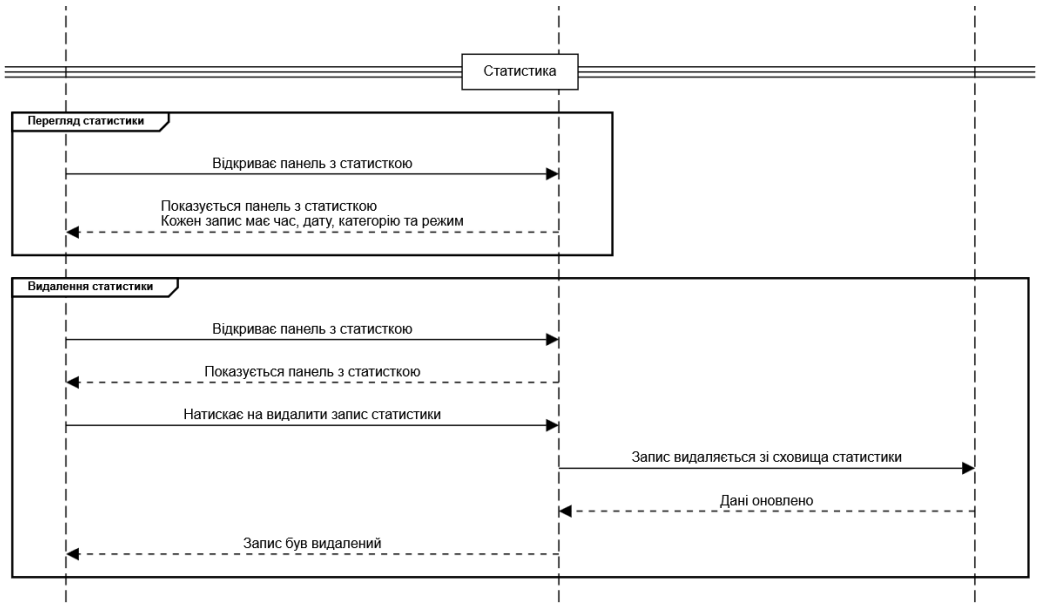

Рисунок 3.8 – Діаграма взаємодії користувача з статисткою

4 РОЗРОБКА ДОКУМЕНТІВ НА СУПРОВОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Призначення й умови застосування програми

Основним призначенням цього застосунку є зручне управління часом та впровадження методу Pomodoro, але не тільки його, для підвищення продуктивності. Цей застосунок повинен допомагати його користувачам досягати поставлених цілей швидше за рахунок кращої концентрації на роботі. Хоча застосунок і не може гарантувати відсутність зовнішніх або внутрішніх відволікань, сам факт встановлення таймера повинен стимулювати людину до роботи

Система має три режими роботи: час для фокусу, мала перерва та велика перерва.

Також система має можливість зміни тривалості часу та кольору для кожного режиму в панелі налаштувань, і підтримує встановлення категорій перед запуском таймера

Основна умова для використання сервісу це наявність інтернету, щоб його відкрити. Наразі в застосунку немає PWA, тому це важливо. Але інтернет не потрібен для роботи застосунку, тому якщо він закешований десь, або запущений локально, або використовуючи функції браузера його встановили як PWA то він буде працювати

Також друга основна умова – застосунок повинен запускатись в браузері або в середовищі, яке підтримує браузерне API. Якщо запустити програму як сервер, або ж в середовищі без, наприклад, можливості зберігати дані в local storage, то застосунок не буде працювати коректно.

4.2 Керівництво оператора

Для розробки застосунку спочатку треба склонувати Git репозиторій та встановити залежності. Склонувати репозиторій можна використовуючи команду git clone

Для керування залежностями в застосунку використовується pnpm, він же Performant Node Package Manager

Далі для розробки застосунку рекомендується використовувати IDE з встановленим плагіном для Vue 3 Volar, не плутати з Vue 2 Vetur

Для локального запуску застосунку використовується скрипт dev. Також в репозиторії встановлений лінтер та форматувальник коду. Лінтер та форматувальник автоматично викликаються при коміті в Git, а тому в віддалений репозиторій ніколи не попаде код, який не проходить наші стандарти, якщо, звичайно, по якісь причині ми не вирішимо вимкнути ці перевірки

4.3 Керівництво користувача

На рисунку 5.1 зображений не запущений таймер

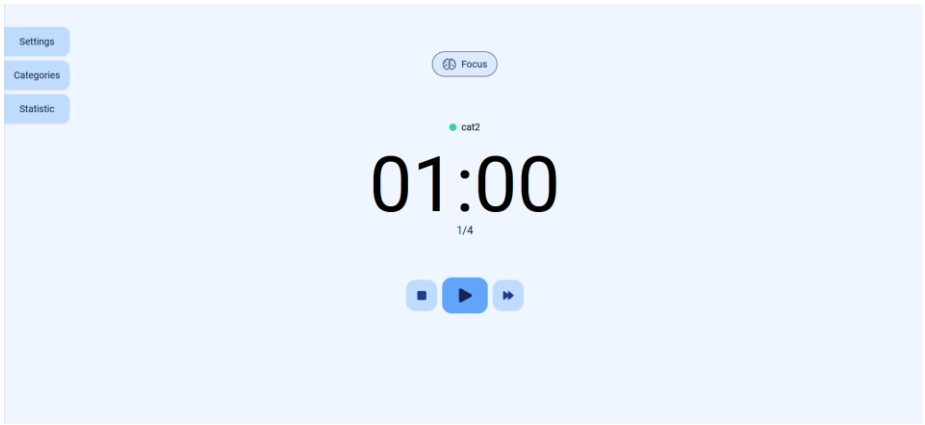

Рисунок 5.1 – Не запущений таймер

На рисунку 5.2 зображений запущений таймер

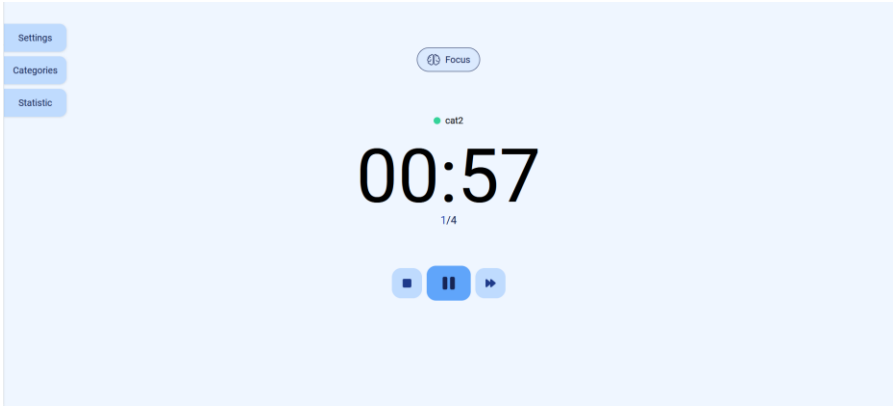

Рисунок 5.2 – Запущений таймер

На рисунку 5.3 зображена панель зміни режиму

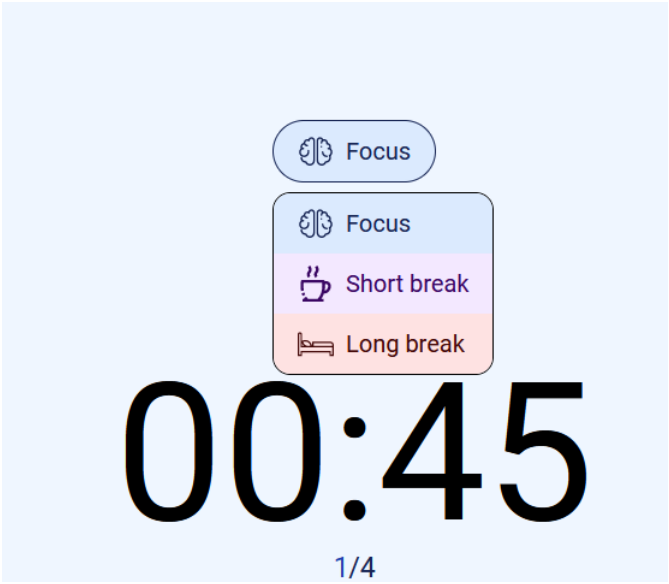

Рисунок 5.3 – Панель зміни режиму таймера

На рисунку 5.4 зображена панель зміни категорії

Рисунок 5.4 – Панель зміни категорій

На рисунку 5.5 зображена панель налаштувань

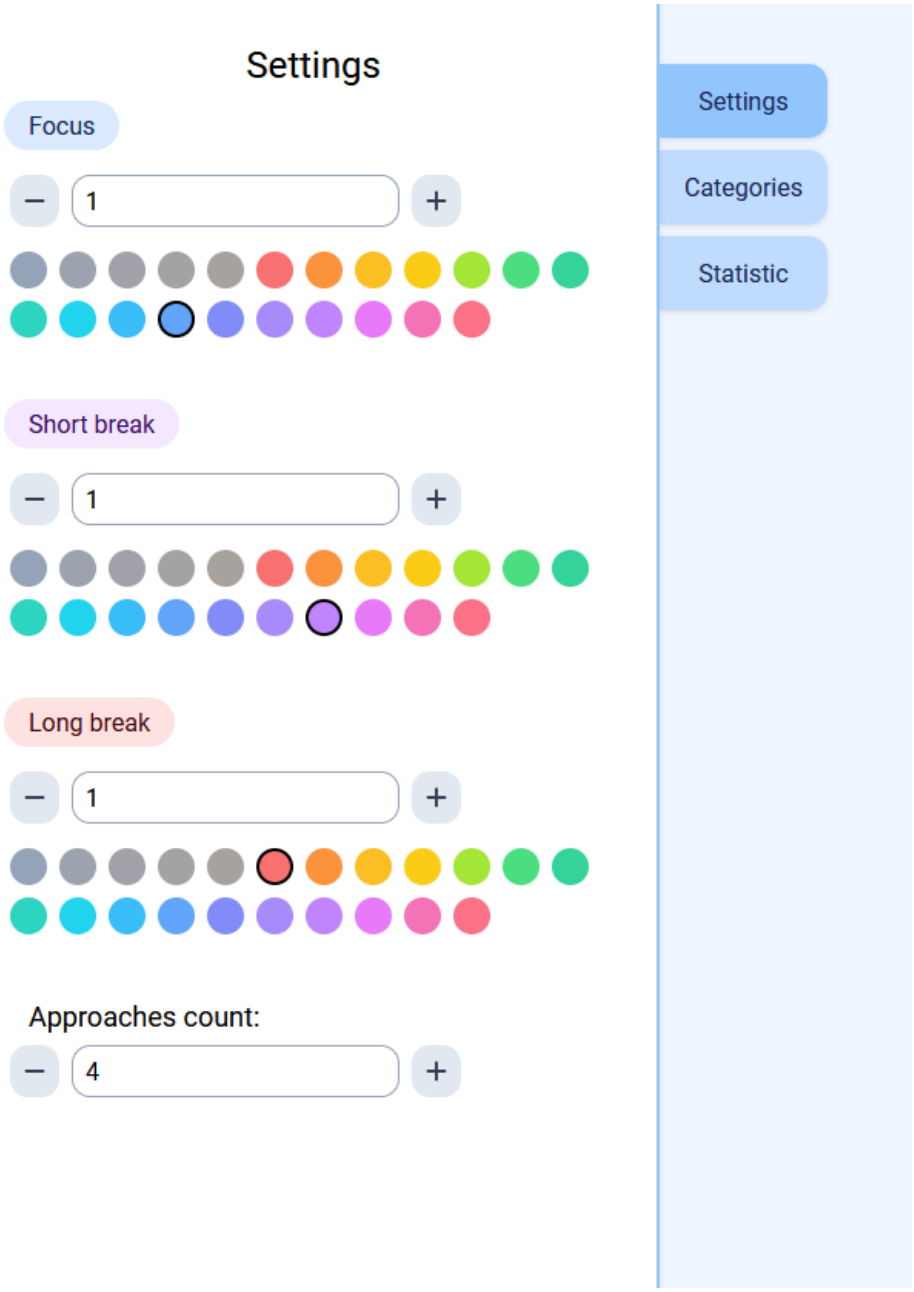

Рисунок 5.5 – Панель налаштувань

На рисунку 5.6 зображена панель категорій

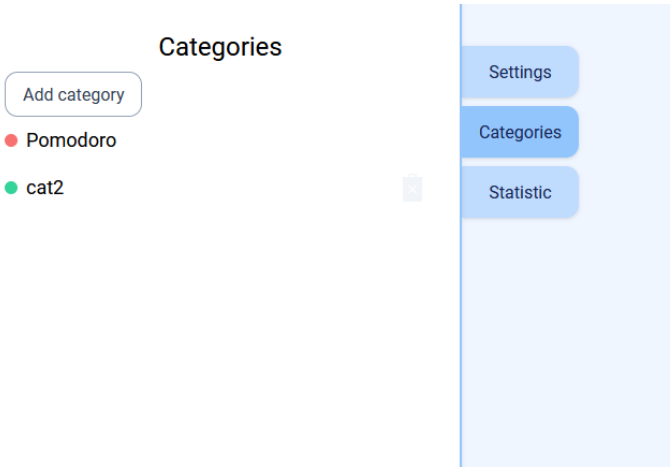

Рисунок 5.6 – Панель категорій

На рисунку 5.7 зображена панель статистики

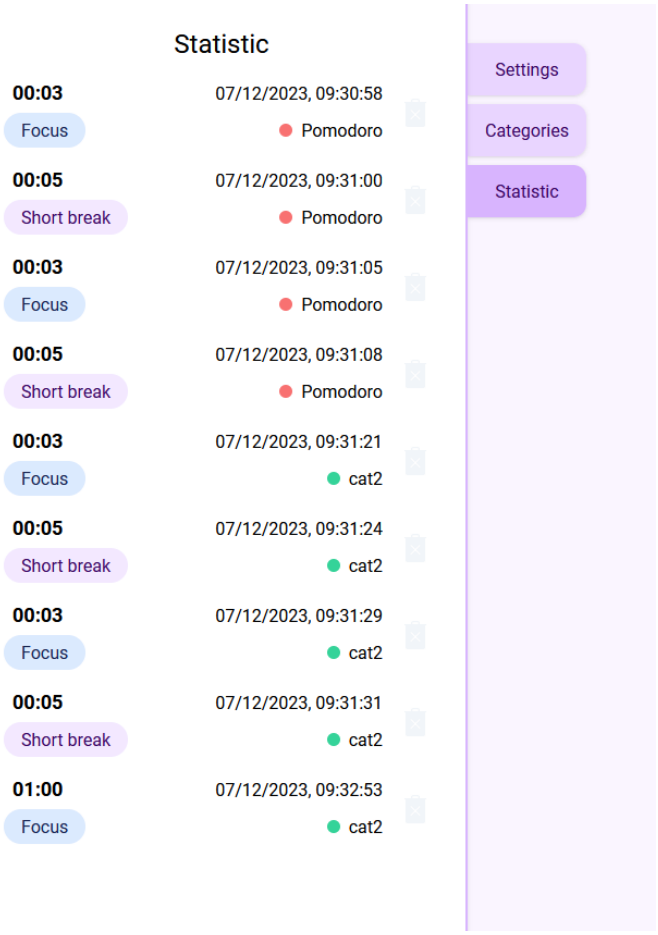

Рисунок 5.7 – Панель статистики

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В результаті тестування застосунку було проведено тестування базового функціоналу, тестування на помилки системи, а також тестування інтерфейсу

Для тестування було використано персональний ноутбук з процесором Intel i5-9300HF, відеокартою NVIDIA GeForce GTX 1650, RAM 16gb. Було використано дві системи, а саме Arch Linux та Windows 11. Також було використано три браузери, а саме Firefox, Google Chrome та GNOME Web для симуляції роботи в Safari.

Також для тестування використовувався телефон Realme 10

В якості першого та базового тесту системи було використано сервіс PageSpeed Insights від Google for Developers. Результати виконання цього тесту показанні на рисунках 5.1 та 5.2.

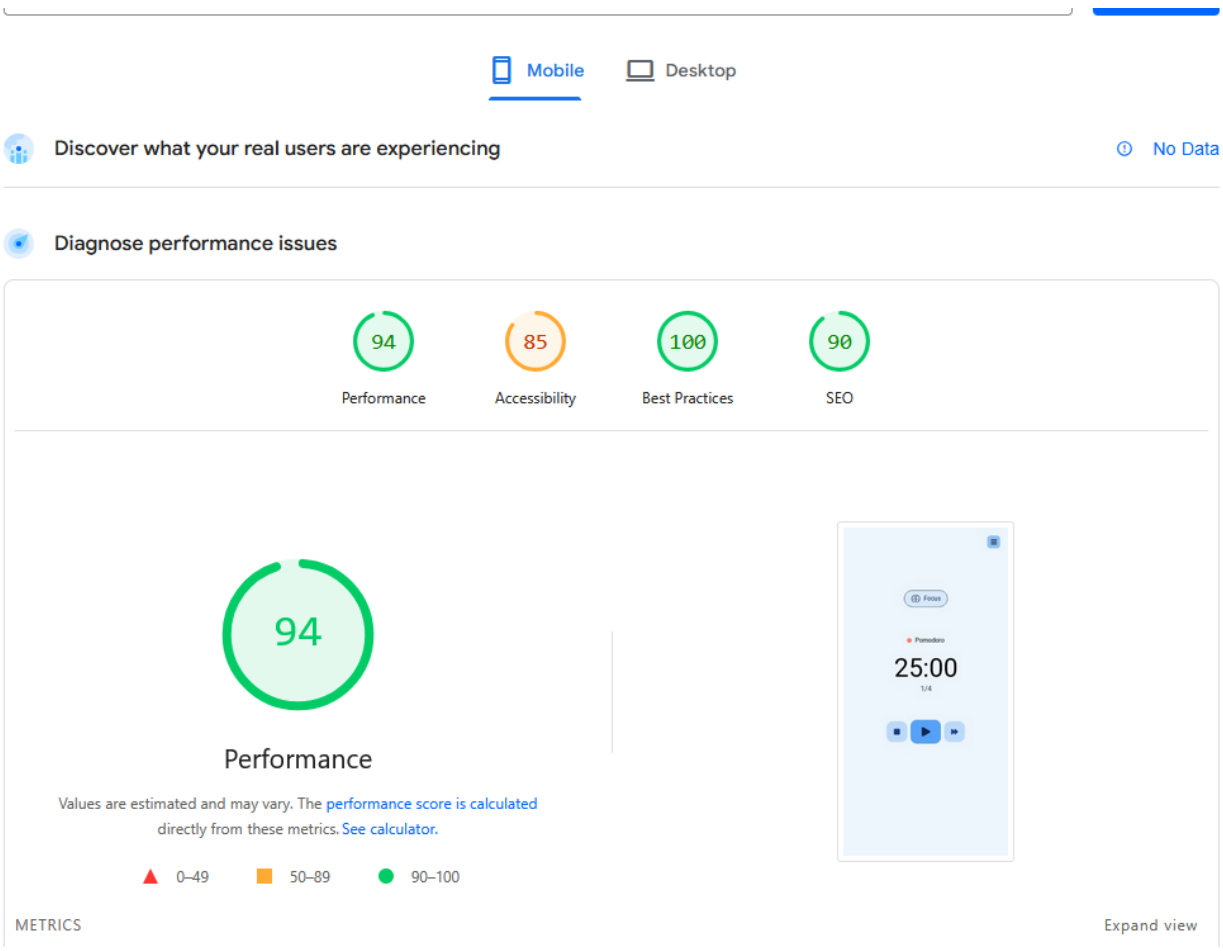

Рисунок 5.1 – Результат тестування вебсайту в PageSpeed Insights для телефонів

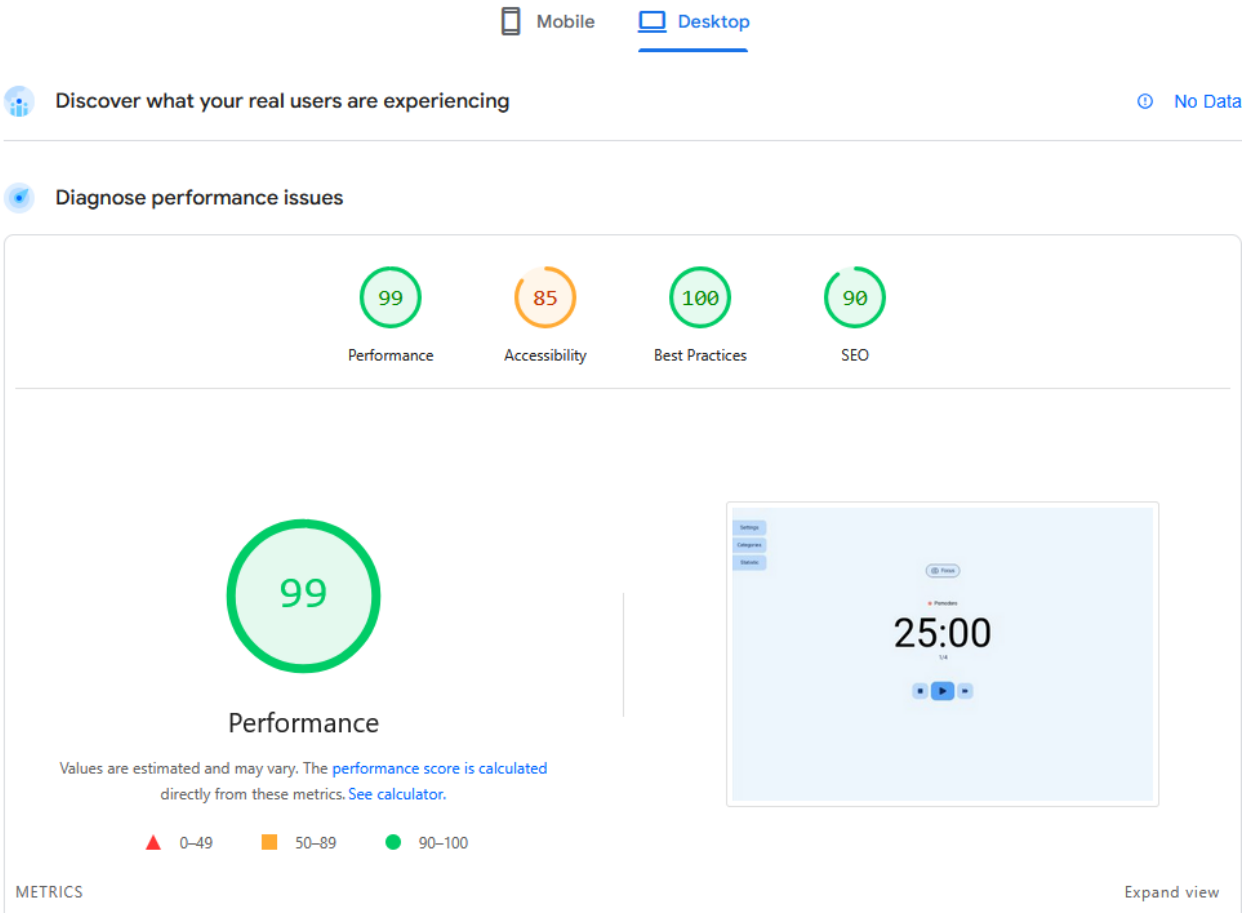

Рисунок 5.2 – Результат тестування вебсайту в PageSpeed Insights для персонального комп'ютеру

Як можна побачити з тестів, сайт завантажується достатньо швидко й використовує всі найкращі практики. Є деякі нюанси з доступністю та SEO оптимізацією, але вони не є критичними та не впливають на роботу

Далі було виконано тестування функціоналу на помилки.

Перевірено роботу таймера, помилок в його роботі при використанні браузеру без додаткових розширень знайдено не було.

Якщо ж в браузері використовується розширення, яке вивантажує сторінки в фоні, то можуть виникати проблеми, оскільки сервіс не зможе записувати інформацію про закінчену сесію, якщо він був вивантажений.

Також таймер не буде працювати, якщо користувач закриє вкладку браузера, або ж сам браузер. Це не є помилкою системи, а скоріше є недоліком її дизайну.

Було перевірено роботу зміни режимів роботи таймера. Проблем не знайдено, переключення відбувається миттєво та скидає значення таймера без запису значення до статистики

Перевірено роботу системи відкриття панелей налаштувань, категорій та статистики. Все працює гарно. На телефонах панель відкривається на повний екран, як це і було задумано

Перевірено роботу налаштувань системи. Знайдено одну не зручність, коли користувач змінює значення таймера, то поточний час скидається, але такий механізм був задуманий з самого початку

Перевірено роботу запису та перегляду статистики. Все працює правильно, запис до статистики створюється лише тоді, коли закінчується таймер, а при скидані таймера запис не створюється

Перевірено роботу категорій, їх створення та використання. Проблем не знайдено

Далі було виконано тестування інтерфейсу на його зручність, на сприйняття, на достатній контраст кольорів, логіку розташування елементів тощо. Згідно тестування було знайдено декілька тривіальних проблем, але нічого критичного

Було перевірено безпеку застосунку. Оскільки вся інформація зберігається в браузері у користувача, так само як і все виконання, існує декілька загроз для програми.

По-перше, виконання в браузері означає, що будь хто може переглянути як програма працює всередині, а також змінити деякі параметри на window об’єкті.

По-друге, дані зберігаються також в браузері і їх можна абсолютно легко переглянути. Навіть якщо імплементувати шифрування інформації, оскільки весь код є майже відкритим, то це не буде мати сенсу

Основна причина чому цей застосунок є безпечним полягає в тому, що інформація, яку він зберігає не коштує абсолютно нічого при спробі її продати. Це застосунок потрібен, щоб користувачі могли працювати більш сфокусовано

по схемі pomodoro. Вся інформація, яку зберігає цей застосунок корисна тільки її власнику. Застосунок не зберігає більше ніяких ключів доступу та ключів для авторизації, оскільки він працює тільки в браузері.

Всі тести виконувались на двох пристроях, в трьох браузерах на двох, якщо не рахувати телефон, системах. У всіх варіантах все працює однаково

У результаті тестування система пройшла всі тести та відповідає вимогам, які були визначені в розділі "Постановка завдання". Система є функціональною, продуктивною, безпечною та зручною у використанні.

В якості покращення роботи можна покращити доступність вебсайту для людей з вадами зору, покращити роботу вебсайту тільки з клавіатури для людей, які не використовують мишку, а також покращити SEO

Другим покращенням можна зробити сервіс абсолютно асинхронним. Тобто, щоб користувач міг його запустити, закрити браузер, а коли відкриє все працювало правильно

5.1 Тестування за сценарієм

Також було виконано тестування за сценарієм

Спочатку користувач запускає систему. На рисунку 5.3 зображений не запущений таймер, який з’являється коли користувач заходить в систему.

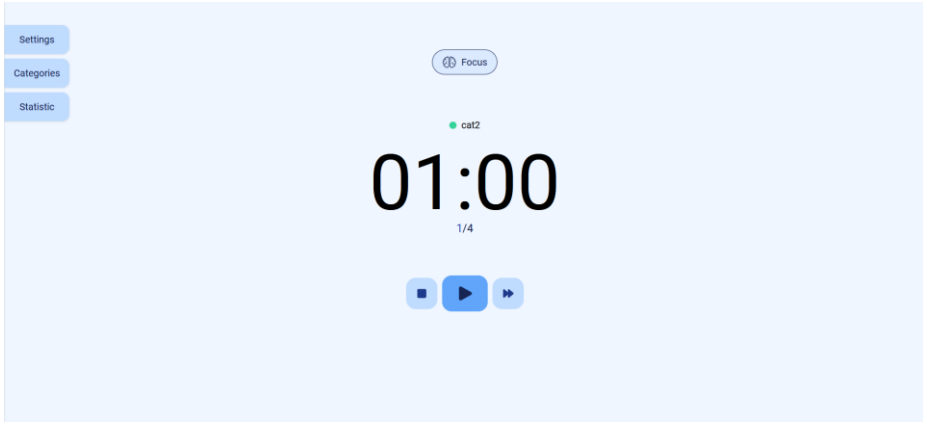

Рисунок 5.3 – Не запущений таймер

Далі користувач встановлює тривалість часу для кожного режиму. На рисунку 5.4 показана панель налаштувань, на якій користувач може встановити значення часу для кожного режиму.

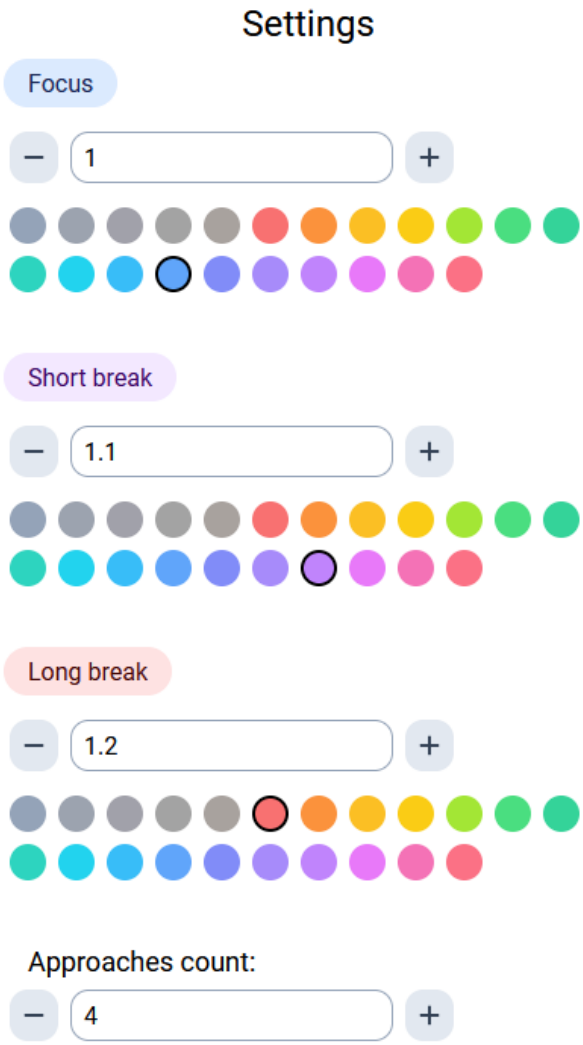

Рисунок 5.4 – Панель налаштувань

Користувач запускає таймер у режимі "сфокусований час". На рисунку 5.5 зображений запущений таймер в режиму "сфокусований час".

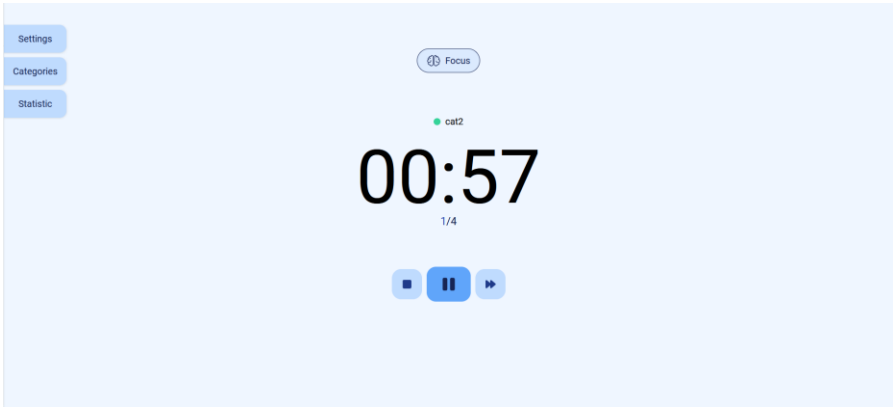

Рисунок 5.5 – Запущений таймер в режимі "сфокусований час"

Користувач працює в режимі "сфокусований час" протягом встановленого часу, зупиняє таймер і встановлює тривалість часу для режиму "мала перерва". На рисунку 5.6 зображено встановлення часу для режиму "мала перерва"

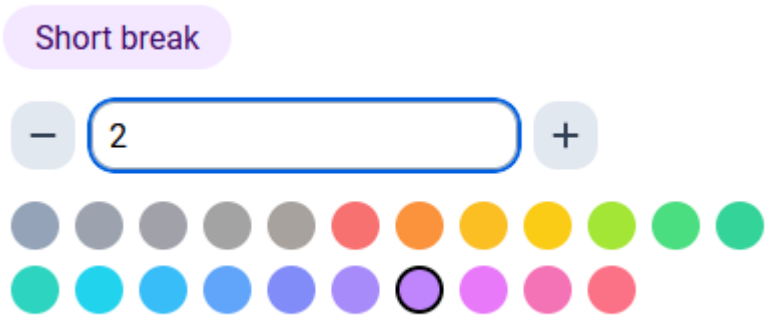

Рисунок 5.6 – Встановлення значень для режиму "мала перерва"

Далі запускає таймер у режимі "мала перерва". На рисунку 5.7 зображений запущений таймер в режимі "мала перерва"

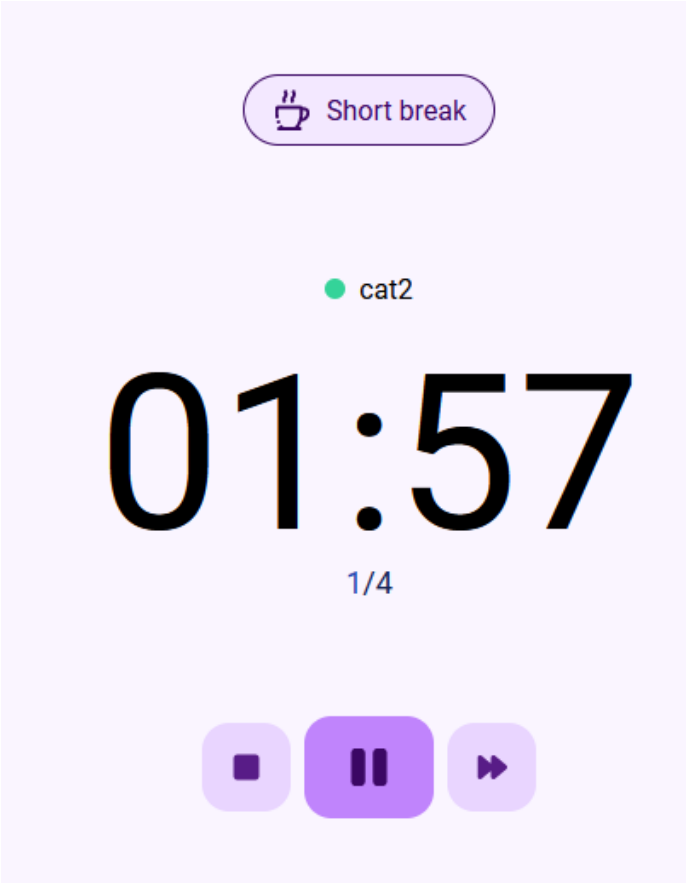

Рисунок 5.7 – Запущений таймер в режимі "мала перерва"

Після закінчення малої перерви користувач встановлює тривалість часу для режиму "велика перерва". На рисунку 5.8 зображено встановлення часу для режиму "велика перерва"

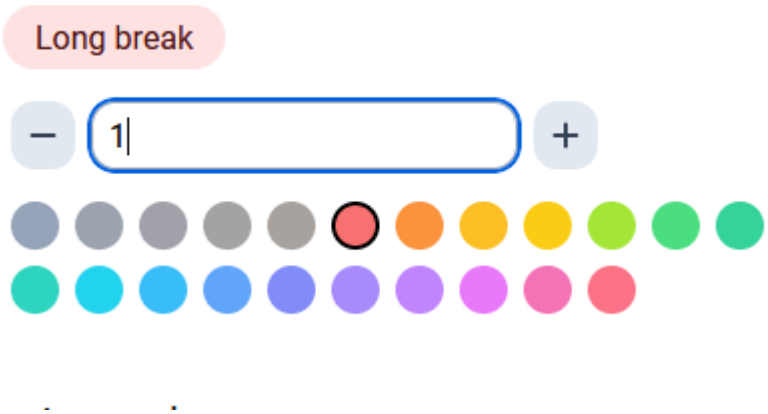

Рисунок 5.8 – Встановлення значень для режиму "велика перерва"

Користувач запускає таймер у режимі "велика перерва". На рисунку 5.9 зображений запущений таймер в режимі "велика перерва".

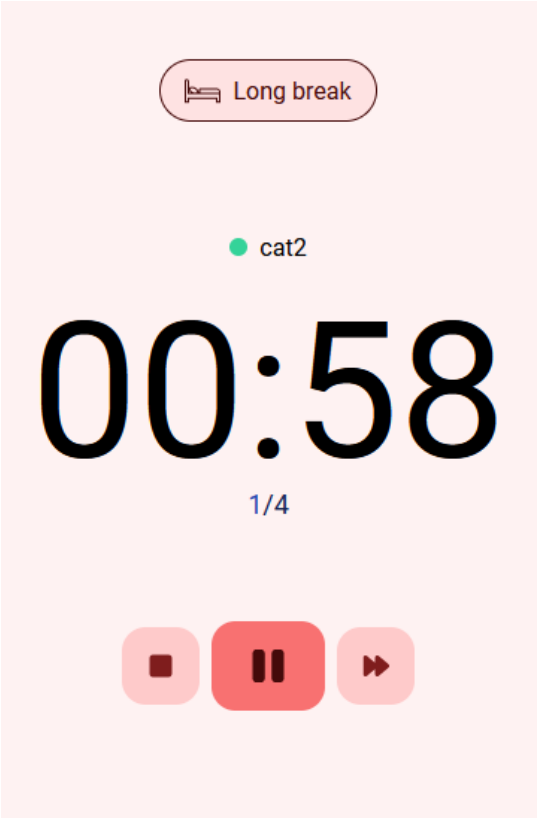

Рисунок 5.9 – Запущений таймер в режимі "мала перерва"

У результаті тестування за сценарієм система таймера пройшла всі тести та відповідає вимогам, які були визначені в розділі "Постановка завдання".

Система таймера є функціональною та продуктивною. Вона може бути використана для підвищення продуктивності користувачів.

ВИСНОВКИ

У ході виконання самостійної роботи було виконано аналіз предметної області та вимог, спроєктовано систему та її дизайн, розроблено систему, створено супровідну документацію та виконано тестування.

Було проаналізовано предмету область та встановлено технічні, функціональні та поведінкові вимоги, а також вимогу дизайну та вимоги продуктивності до системи, що розробляється.

У ході проєктування системи було обрано мову програмування, середовище розробки, архітектурних підхід, а також спроєктовано інтерфейс застосунку з використанням варфреймів, мокапів та прототипів.

Під час розробки програмного забезпечення було уточнено список технологій та мотивацію їх використання, розроблено діаграми зв’язків модулів системи та діаграму взаємодії користувача, системи та сховища інформації. Також на цьому етапі було створено повністю функціональний застосунок.

Далі було створено документи для супроводу застосунку, які включають короткий опис призначення й умов застосування, керівництво оператора з детальною інструкцією по подальшій розробці та керівництво користувача з детальним описом та картинками використання застосунку

Під кінець було виконано тестування, яке показало, що застосунок відповідає всім вимогам, не має критичних багів та працює достатньо швидко. Також тестування показало, що в застосунку є речі, які можна покращити в майбутньому для ще більшої зручності його використання.

Розроблена система є функціональною та продуктивною. Вона може бути використана для підвищення продуктивності користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cirillo F. The Pomodoro Technique [Електронний ресурс]. 2006–2007. 45 с. URL: https://cirillocompany.de/pages/pomodoro-technique

2. JavaScript [Електронний ресурс] // MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript

3. TypeScript Handbook [Електронний ресурс]. URL: https://www.typescriptlang.org/docs/handbook/intro.html

4. Neovim Documentation [Електронний ресурс]. URL: https://neovim.io/doc

5. Git Documentation [Електронний ресурс]. URL: https://git-scm.com/docs

6. Vue.js Documentation [Електронний ресурс]. URL: https://vuejs.org/guide/introduction.html

7. pnpm Documentation [Електронний ресурс]. URL: https://pnpm.io/

8. Vue I18n Documentation [Електронний ресурс]. URL: https://vue-i18n.intlify.dev/

9. Madge [Електронний ресурс] // npm. URL: https://www.npmjs.com/package/madge

ДОДАТОК А - КОД ПРОГРАМИ

src\main.ts
```
import { createApp } from 'vue'

import { createPinia } from 'pinia'
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'

import { vCloseModal } from '@directives'

import App from './App.vue'
import './style.css'

const app = createApp(App)
const pinia = createPinia()

pinia.use(piniaPluginPersistedstate)

app.use(pinia)

app.directive('close-modal', vCloseModal)

app.mount('#app')
```

src\vite-env.d.ts
```
/// <reference types="vite/client" />
```

src\constants\approachesLimits.ts
```
export const approachesLimits = {
	max: 100,
	min: 0
}
```

src\constants\index.ts
```
export { modeText } from './modeText'
export { tailwindColorsIgnore } from './tailwindColorsIgnore'
export { approachesLimits } from './approachesLimits'
```

src\constants\modeText.ts
```
import { Mode } from '@types'

export const modeText: Record<Mode, string> = {
	[Mode.pomodoro]: 'Focus',
	[Mode.short]: 'Short break',
	[Mode.long]: 'Long break'
}
```

src\constants\tailwindColorsIgnore.ts
```
export const tailwindColorsIgnore = [
	'inherit',
	'current',
	'transparent',
	'black',
	'white',
	'lightBlue',
	'warmGray',
	'trueGray',
	'coolGray',
	'blueGray'
```

```
]


src\directives\closeModal.ts
import { ObjectDirective } from 'vue'

// eslint-disable-next-line
export const vCloseModal: ObjectDirective<any> = {
	mounted(el, binding) {
		el.__CloseModalClick__ = (e: MouseEvent) => {
			if (!(el === e.target || el.contains(e.target))) {
				binding.value()
			}
		}

		el.__CloseModalKeydown__ = (e: KeyboardEvent) => {
			if (e.code === 'Escape') {
				binding.value()
			}
		}

		document.addEventListener('click', el.__CloseModalClick__)
		document.addEventListener('keydown', el.__CloseModalKeydown__)
	},
	unmounted(el) {
		document.removeEventListener('click', el.__CloseModalClick__)
		document.removeEventListener('keydown', el.__CloseModalKeydown__)
	}
}


src\directives\index.ts
export { vCloseModal } from './closeModal'


src\packages\timer\index.ts
export * from './useTimer'
export { useTimer } from './useTimer'


src\packages\timer\useTimer.ts
import { Ref, reactive, toRef } from 'vue'

import { useInterval } from './hooks'
import { Interval } from './hooks/useInterval'
import { Time, Validate } from './utils'

const DEFAULT_DELAY = 1000

function getDelayFromExpiryTimestamp(expiryTimestamp: number) {
	if (!Validate.expiryTimestamp(expiryTimestamp)) {
		return null
	}

	const seconds = Time.getSecondsFromExpiry(expiryTimestamp)
	const extraMilliSeconds = Math.floor((seconds - Math.floor(seconds)) * 1000)
	return extraMilliSeconds > 0 ? extraMilliSeconds : DEFAULT_DELAY
}

export interface UseTimer {
	seconds: Ref<number>
	minutes: Ref<number>
```

```
	hours: Ref<number>
	start(): void
	pause(): void
	resume(): void
	restart(newExpiryTimestamp?: number, newAutoStart?: boolean): void
	isRunning: Ref<boolean>
	isExpired: Ref<boolean>
}

export const useTimer = (expiry = 60, autoStart = true): UseTimer => {
	let interval: Interval

	const state = reactive({
		expiryTimestamp: expiry,
		seconds: Time.getSecondsFromExpiry(expiry),
		isRunning: autoStart,
		isExpired: false,
		didStart: autoStart,
		delay: getDelayFromExpiryTimestamp(expiry)
	})

	function _handleExpire() {
		state.isExpired = true
		state.isRunning = false
		state.delay = null
		if (interval) interval.remove()
	}

	function pause() {
		state.isRunning = false
		if (interval) interval.remove()
	}

	function restart(newExpiryTimestamp: number = expiry, newAutoStart = true) {
		pause()
		state.delay = getDelayFromExpiryTimestamp(newExpiryTimestamp)
		state.didStart = newAutoStart
		state.isExpired = false
		state.expiryTimestamp = newExpiryTimestamp
		state.seconds = Time.getSecondsFromExpiry(newExpiryTimestamp)
		if (state.didStart) start()
	}

	function resume() {
		const time = new Date()
		const newExpiryTimestamp = time.setMilliseconds(
			time.getMilliseconds() + state.seconds * 1000
		)
		restart(newExpiryTimestamp)
	}

	function start() {
		if (state.didStart) {
			state.seconds = Time.getSecondsFromExpiry(state.expiryTimestamp)
			state.isRunning = true
			interval = useInterval(
				() => {
					if (state.delay !== DEFAULT_DELAY) {
						state.delay = DEFAULT_DELAY
					}
					const secondsValue =
Time.getSecondsFromExpiry(state.expiryTimestamp)
```

```
                                state.seconds = secondsValue
                                if (secondsValue <= 0) {
                                        _handleExpire()
                                }
                        },
                        state.isRunning ? state.delay : null
                )
        } else {
                resume()
        }
}

restart(expiry, autoStart)

return {
        ...Time.getTimeFromSeconds(toRef(state, 'seconds')),
        start,
        pause,
        resume,
        restart,
        isRunning: toRef(state, 'isRunning'),
        isExpired: toRef(state, 'isExpired')
}
}


src\packages\timer\hooks\index.ts
export { useInterval } from './useInterval'


src\packages\timer\hooks\useInterval.ts
const isNumber = (val: unknown): val is number => typeof val === 'number'

export interface Interval {
        remove: () => void
        start: (_ms?: number | undefined) => NodeJS.Timeout | undefined
}

export function useInterval(
        callback: () => void,
        ms?: number | boolean | null
): Interval {
        let intervalId: NodeJS.Timeout | undefined = undefined

        const remove = () => {
                if (!intervalId) return
                clearInterval(intervalId)
                intervalId = undefined
        }

        const start = (_ms?: number) => {
                remove()
                if (!_ms && !ms) {
                        return
                }
                const m = (_ms || ms) as number
                return (intervalId = setInterval(callback, m))
        }

        if (isNumber(ms)) {
                start()
        }
```

```
	return { remove, start }
}


src\packages\timer\utils\index.ts
import Time from './Time'
import Validate from './Validate'

export { Time, Validate }


src\packages\timer\utils\Time.ts
import { Ref, computed } from 'vue'

interface TimeAmpm {
	seconds: Ref<number>
	minutes: Ref<number>
	hours: Ref<number>
	ampm: Ref<string>
}

interface TimeNum {
	seconds: Ref<number>
	minutes: Ref<number>
	hours: Ref<number>
}

export default class Time {
	static getTimeFromSeconds(secs: Ref<number>): TimeNum {
		const totalSeconds = computed(() => Math.ceil(secs.value))
		const hours = computed(() =>
			Math.floor((totalSeconds.value % (60 * 60 * 24)) / (60 * 60))
		)
		const minutes = computed(() =>
			Math.floor((totalSeconds.value % (60 * 60)) / 60)
		)
		const seconds = computed(() => Math.floor(totalSeconds.value % 60))

		return {
			seconds,
			minutes,
			hours
		}
	}

	static getSecondsFromExpiry(expiry: number, shouldRound?: boolean): number {
		const now = new Date().getTime()
		const milliSecondsDistance = expiry - now
		if (milliSecondsDistance > 0) {
			const val = milliSecondsDistance / 1000
			return shouldRound ? Math.round(val) : val
		}
		return 0
	}

	static getSecondsFromPrevTime(
		prevTime: number,
		shouldRound: boolean
	): number {
		const now = new Date().getTime()
		const milliSecondsDistance = now - prevTime
```

```
            if (milliSecondsDistance > 0) {
                  const val = milliSecondsDistance / 1000
                  return shouldRound ? Math.round(val) : val
            }
            return 0
      }

      static getSecondsFromTimeNow(): number {
            const now = new Date()
            const currentTimestamp = now.getTime()
            const offset = now.getTimezoneOffset() * 60
            return currentTimestamp / 1000 - offset
      }

      static getFormattedTimeFromSeconds(
            totalSeconds: Ref<number>,
            format: '12-hour' | '24-hour'
      ): TimeAmpm {
            const {
                  seconds: secondsValue,
                  minutes,
                  hours
            } = Time.getTimeFromSeconds(totalSeconds)
            const ampm = computed(() =>
                  format === '12-hour' ? (hours.value >= 12 ? 'pm' : 'am') : ''
            )
            const hoursValue = computed(() =>
                  format === '12-hour' ? hours.value % 12 : hours.value
            )

            return {
                  seconds: secondsValue,
                  minutes,
                  hours: hoursValue,
                  ampm
            }
      }
}


src\packages\timer\utils\Validate.ts
export default class Validate {
      static expiryTimestamp(expiryTimestamp: number): boolean {
            const isValid = new Date(expiryTimestamp).getTime() > 0
            if (!isValid) {
                  console.warn(
                        '{ useTimer } Invalid expiryTimestamp settings',
                        expiryTimestamp
                  )
            }
            return isValid
      }
}


src\store\categoriesStore.ts
import { defineStore } from 'pinia'

import { AddNewCategory, Category } from '@types'

interface CategoryStore {
      categories: Category[]
```

```
}

export const useCategoryStore = defineStore('category', {
	state: (): CategoryStore => ({
		categories: [
			{
				_id: '0',
				color: 'red',
				mode: 'time',
				name: 'Pomodoro'
			}
		]
	}),

	actions: {
		add(payload: AddNewCategory) {
			this.categories.push({
				_id: String(new Date(Date.now()).getTime()),
				mode: 'time',
				...payload
			})
		},

		delete(categoriesItem: Category) {
			this.categories = this.categories.filter(
				item => item._id !== categoriesItem._id
			)
		}
	},

	persist: true
})


src\store\statisticStore.ts
import { defineStore } from 'pinia'

import { AddNewStatistic, Statistic } from '@types'

interface StatisticStore {
	statistic: Statistic[]
}

export const useStatistic = defineStore('statistic', {
	state: (): StatisticStore => ({
		statistic: []
	}),

	actions: {
		add(payload: AddNewStatistic) {
			this.statistic.push({
				_id: String(new Date(Date.now())),
				date: String(new Date(Date.now() - payload.count)),
				count: payload.count,
				mode: payload.mode,
				category: payload.category
			})
		},
		delete(statisticItem: Statistic) {
			this.statistic = this.statistic.filter(
				item => item._id !== statisticItem._id
			)
```

```
		}
	},

	persist: true
})


src\store\userSettingsStore.ts
import { defineStore } from 'pinia'

import { approachesLimits } from '@constants/approachesLimits'
import { Category, Colors, Mode } from '@types'

interface UserSettingsStore {
	settings: {
		colors: Record<Mode, Colors>
		times: Record<Mode, number>
		selectedMode: Mode
		selectedCategory: Category
		approachesCount: number
	}
	currentApproach: number
	activeCompletedPomodoro: boolean
}

export const useUserSettingsStore = defineStore('settings', {
	state(): UserSettingsStore {
		return {
			settings: {
				colors: {
					[Mode.pomodoro]: 'blue',
					[Mode.short]: 'purple',
					[Mode.long]: 'red'
				},
				times: {
					[Mode.pomodoro]: 3 / 60,
					[Mode.short]: 4 / 60,
					[Mode.long]: 5 / 60
				},
				selectedMode: Mode.pomodoro,
				selectedCategory: {
					_id: '0',
					color: 'red',
					mode: 'time',
					name: 'Pomodoro'
				},
				approachesCount: 4
			},
			currentApproach: 1,
			activeCompletedPomodoro: false
		}
	},
	getters: {
		colors: state => state.settings.colors,
		times: state => state.settings.times,
		selectedMode: state => state.settings.selectedMode
	},
	actions: {
		setColor(mode: Mode, color: Colors) {
			this.settings.colors[mode] = color
		},
		setSelectedMode(mode: Mode) {
```

```
                this.settings.selectedMode = mode
            },
            setApproachesCount(value: number) {
                if (value > approachesLimits.max || value < approachesLimits.min)
return

                this.settings.approachesCount = value
            }
        },
        persist: true
})


src\types\AddNewCategory.ts
import { Category } from '.'

export type AddNewCategory = Omit<Category, '_id' | 'mode'>


src\types\AddNewStatistic.ts
import { Statistic } from './Statistic'

export type AddNewStatistic = Omit<Statistic, '_id' | 'date'>


src\types\Category.ts
import { Colors } from '.'

export interface Category {
        _id: string
        name: string
        mode: 'number' | 'time' // Тільки time, number для помідорок не потрібен
        // comment: string
        color: Colors
        // order: number // Подивимось) Не буде використанно в цій версії
        // dimension?: string // Не буде використанно в цій версії
        // group: string[] // Не буде використанно в цій версії
        // archived: boolean // Подивимось) Не буде використанно в цій версії
        // trash: boolean // Подивимось) Не буде використанно в цій версії
        // trash_expires: number // Подивимось) Не буде використанно в цій версії
}


src\types\Colors.ts
import type { DefaultColors } from 'tailwindcss/types/generated/colors'

export type Colors = keyof Omit<
        DefaultColors,
        'inherit' | 'current' | 'transparent' | 'black' | 'white'
>


src\types\index.ts
export * from './ModeItem'
export * from './Category'
export * from './Colors'
export * from './Mode'
export * from './LeftPanelModes'
export * from './Statistic'
export * from './AddNewCategory'
export * from './AddNewStatistic'
```

src\types\LeftPanelModes.ts
```
export type LeftPanelModes = 'settings' | 'statistic' | 'category'
```

src\types\Mode.ts
```
export enum Mode {
	'pomodoro' = 'POMODORO',
	'short' = 'SHORT',
	'long' = 'LONG'
}
```

src\types\ModeItem.ts
```
import { Mode } from './Mode'

export interface ModeItem {
	id: Mode
	title: string
	icon: string
}
```

src\types\Statistic.ts
```
import { Category } from '.'
import { Mode } from './Mode'

export interface Statistic {
	_id: string
	date: string // Date as ISO string
	mode: Mode // Lapricot diff
	count: number
	// comment: string
	category: Category
}
```