ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка розумного анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Едуард МАРТИНЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Едуард МАРТИНЕНКО

Керівник: Ігор ГАМАНЮК
ст.викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мартиненку Едуарду Вікторовичу

1. Тема кваліфікаційної роботи: «Розробка розумного анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями»
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ Ігор ГАМАНЮК, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Теоретичні відомості про прокрастинацію та її наслідки, Огляд принципів гейміфікації в навчальних і робочих процесах, Опис техніки Pomodoro та динамічних нагадувань, Технічна документація: вибір стеку технологій (C#, ASP.NET Core, Blazor, EF Core), опис бібліотек для реалізації таймера та сповіщень

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз проблеми прокрастинації та існуючих підходів до її подолання
2. Огляд наявних аналогів (Forest, RescueTime, Focus To-Do тощо) і їхні обмеження.
3. Вимоги до системи (функціональні та нефункціональні).

4. Проектування архітектури.
5. Опис реалізації основних модулів.
6. План і результати тестування.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма прецедентів.
5. Діаграма предметної галузі.
6. Діаграма діяльності.
7. Діаграма станів.
8. Діаграма класів.
9. Екранні форми.
10. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз застосунку для боротьби з прокрастинацією	14.03-20.03.2025	
4	Огляд та аналіз засобів реалізації застосунку для боротьби з прокрастинацією	21.03-28.03.2025	
5	Програмна реалізація та тестування застосунку для боротьби з прокрастинацією	29.03-18.04.2025	
6	Програмна реалізація та тестування застосунку для боротьби з прокрастинацію	19.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Едуард МАРТИНЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 1 табл., 30 рис., 20 джерел.

Мета роботи – покращення боротьби з прокрастинацією шляхом впровадження розумного веб-застосунку анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями.

Об'єкт дослідження – процес прокрастинації та методи її подолання.

Предмет дослідження – веб-застосунок для боротьби з прокрастинацією.

Короткий зміст роботи: Проаналізовано сучасні підходи до подолання прокрастинації та існуючі рішення (Forest, RescueTime, Pomodoro To-Do, Focus Keeper). Спроєктовано й реалізовано веб-застосунок із основними функціями управління завданнями й підзавданнями, Pomodoro-таймером із кнопками «Зупинити»/«Пропустити перерву», системою бонусів і штрафів, сервісом динамічних нагадувань та модулем візуалізації статистики (гістограми й графіки Pomodoro-сесій). Проведено функціональне й модульне тестування. Для серверної логіки використано C# і .NET 8 з ASP.NET Core для REST API, клієнтська частина створена на Blazor WebAssembly, дані зберігаються через Entity Framework Core із SQLite. Сфера застосування — особистий тайм-менеджмент і мотивація користувачів, а також підтримка організації роботи й навчання в корпоративних і освітніх середовищах.

КЛЮЧОВІ СЛОВА: АНТИ-ПРОКРАСТИНАТОР, ГЕЙМІФІКАЦІЯ, POMODORO, ДИНАМІЧНІ НАГАДУВАННЯ, ASP.NET CORE, BLAZOR WEBASSEMBLY, ENTITY FRAMEWORK CORE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

C# – мова програмування C Sharp

.NET – платформа .NET (версія 8)

ASP.NET Core – фреймворк для створення веб-API та серверної логіки

Blazor WASM – Blazor WebAssembly (SPA на WebAssembly)

EF Core – Entity Framework Core (ORM-бібліотека)

SQLite – вбудована реляційна СУБД SQLite

API – Application Programming Interface

REST – Representational State Transfer

HTTP – HyperText Transfer Protocol

UI – User Interface

UX – User Experience

CRUD – Create, Read, Update, Delete

DI – Dependency Injection

WASM – WebAssembly

CSS – Cascading Style Sheets

ВСТУП

Актуальність: У сучасному світі прокрастинація — поширене явище, що негативно впливає на продуктивність, мотивацію та психологічний стан людини. Незважаючи на широке розмаїття мобільних додатків і веб-сервісів для планування та тайм-менеджменту, багато з них не враховують індивідуальні поведінкові патерни користувачів і не інтегрують мотиваційні механізми у вигляді гейміфікації. Ураховуючи зростаючу роль віддаленої та самостійної роботи й навчання, потреба в інструменті, який би адаптував інтервали нагадувань на основі історії виконання завдань та використовував механіку Pomodoro у поєднанні з бонусами й штрафами, є надзвичайно високою.

На цій основі розроблено веб-застосунок FocusUp, який поєднує Pomodoro-таймер, систему гейміфікації та динамічні нагадування для підвищення особистої ефективності.

Об'єктом дослідження є процес прокрастинації та методи її подолання.

Предметом дослідження є веб-застосунок для боротьби з прокрастинацією.

Метою роботи є покращення боротьби з прокрастинацією шляхом впровадження розумного веб-застосунку анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями.

Методи дослідження:

– Аналіз існуючих аналогів (Forest, RescueTime, Pomodoro To-Do) для формулювання функціональних і нефункціональних вимог;

– Проектування архітектури клієнт-серверної моделі з використанням UML-діаграм;

– Програмна реалізація на C#, .NET 8, ASP.NET Core, Blazor WebAssembly, EF Core (SQLite);

Наукова новизна роботи визначається такими функціональними складовими:

– адаптивний сервіс динамічних нагадувань на основі історії виконання;

– система бонусів і штрафів;

– комбінування Pomodoro-таймера з гейміфікацією в єдиному інтерфейсі.

Практична значущість результатів полягає в можливості використання застосунку для особистого тайм-менеджменту, підвищення мотивації та ефективності роботи чи навчання, а також інтеграції в корпоративні та освітні системи.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТА БОРОТЬБИ З ПРОКРАСТИНАЦІЄЮ

1.1 Веб-застосунок «Розумний анти-прокрастинатор з гейміфікацією та динамічними нагадуваннями»

Веб-застосунок «Розумний анти-прокрастинатор з гейміфікацією та динамічними нагадуваннями» — це спеціалізований онлайн-інструмент, що об’єднує методи Pomodoro та адаптивні сповіщення для планування й виконання завдань. Завдяки зручному SPA-інтерфейсу користувачі можуть створювати й організовувати завдання, запускати робочі сесії та відстежувати прогрес в режимі реального часу.

Основні компоненти веб-застосунку включають:

– Pomodoro-таймер із налаштовуваними інтервалами робочих сесій і перерв, кнопками «Зупинити» й «Пропустити перерву»;

– система нарахування балів за успішне завершення Pomodoro-сесій;

– модуль динамічних нагадувань, що автоматично коригує інтервали сповіщень на основі історії виконання завдань;

– панель управління завданнями й підзавданнями з функцією drag-and-drop, сортування та групування;

– блок графічної статистики з гістограмами й лінійними графіками Pomodoro-сесій за днями та годинами.

Основні цілі веб-застосунку для боротьби з прокрастинацією:

– зниження рівня відкладання важливих справ;

– підвищення продуктивності та концентрації уваги;

– формування корисних робочих звичок;

– оптимізація управління часом відповідно до індивідуальних потреб;

– наочна візуалізація власної активності;

– підтримка мотивації через накопичення балів.

Оцінимо переваги та недоліки веб-застосунку як засобу для боротьби з прокрастинацією.

Переваги:

– висока доступність у будь-якому сучасному браузері без встановлення додаткових плагінів;

– адаптивність сповіщень відповідно до поведінкових патернів користувача;

– гнучке налаштування Pomodoro-таймера та нагадувань;

– наочна візуалізація прогресу сприяє кращому розумінню власної продуктивності;

– мінімальні технічні вимоги до розгортання серверної частини.

Недоліки:

– потреба у стабільному інтернет-з’єднанні для синхронізації даних;

– залежність від самодисципліни користувача щодо виконання нагадувань;

– початковий час на освоєння інтерфейсу і налаштувань;

– можливе відволікання при надмірній частоті сповіщень;

– обмежена офлайн-функціональність без активного з’єднання з сервером.

1.2 Специфіка інтеграції Pomodoro-таймера та гейміфікаційних механік

Специфіка інтеграції Pomodoro-таймера та гейміфікаційних механік полягає у створенні середовища, яке поєднує дисципліну тайм-менеджменту з мотиваційними елементами, що стимулюють користувача до регулярної роботи. Розглянемо ключові особливості цього поєднання.

Pomodoro-таймер:

– автоматичний запуск перерв або наступної сесії без додаткових дій

– звукові та візуальні сигнали про початок і завершення сесій

Гейміфікація:

– система нарахування балів за кожне успішно завершене Pomodoro-вікно

– мотиваційні повідомлення між сесіями для підтримки інтересу

Динамічні нагадування:

– автоматичне коригування частоти сповіщень на основі історії виконання сесій

– контекстні сигнали при простої або низькій активності користувача

– врахування часу доби та персональних налаштувань для гнучкої адаптації.

Архітектурні особливості SPA:

– односторінкова архітектура забезпечує мінімальні затримки та плавну взаємодію

– виконання клієнтської логіки в браузері за допомогою Blazor WebAssembly

– серверна частина на ASP.NET Core відповідає за збереження даних та обробку запитів CRUD

Інтерфейс користувача:

– адаптивний дизайн для підтримки різних розмірів екранів

– реактивні компоненти для списку завдань, таймера й аналітики

– інтуїтивно зрозумілі елементи керування: кнопки «Старт/Пауза», перемикачі інтервалів.

Система аналітики:

– збір даних про тривалість сесій і перерв, пропущені Pomodoro-вікна

– візуалізація у вигляді гістограм та лінійних графіків активності за днями та годинами.

1.3 Цільова аудиторія

Застосунок для боротьби з прокрастинацією та підвищення продуктивності орієнтований на широке коло користувачів, які прагнуть оптимізувати власний час і сформувати корисні робочі звички. До основних категорій користувачів належать:

– студенти та аспіранти, які виконують навчальні завдання, пишуть курсові або дипломні роботи та прагнуть краще організувати свій робочий

графік

– фрилансери та самозайняті фахівці, які працюють за власним розкладом і потребують інструменту для контролю часу без нагляду з боку роботодавця

– офісні працівники з рутинними завданнями, яким необхідно розбити день на структуровані інтервали і уникнути вигорання

– працівники з віддаленим або гнучким графіком, які відповідають самі за планування й виконання проєктів у різні години доби

– творчі спеціалісти (дизайнери, програмісти, письменники), чия діяльність вимагає тривалої концентрації і регулярних перерв для відновлення креативності.

1.4 Дослідження існуючих аналогів

На сьогоднішній день одними з найпопулярніших веб- та мобільних застосунків для боротьби з прокрастинацією та підвищення продуктивності є Forest, RescueTime, Pomodoro To-Do та Focus Keeper. Розглянемо їх детальніше.

1.4.1 Forest

Forest — це крос-платформений застосунок із гейміфікаційною механікою «вирощування дерев» під час концентрації на завданнях.

Основні функції Forest включають:

– посадка віртуального насіння перед початком Pomodoro-сесії, яке виростає в дерево за відведений час;

– створення «лісу» зі всіх завершених сесій, що візуально відображає прогрес;

– можливість додати список «білого» та «чорного» сайтів (через розширення браузера), щоб блокувати відволікаючі ресурси;

– соціальні елементи — спільні «ліси» з друзями для взаємної мотивації.

Переваги Forest:

– висока візуальна мотивація через зростаючі дерева;

– простий інтерфейс і гнучкі налаштування тривалості сесій;

– підтримка мультиплатформеності (iOS, Android, веб-розширення).

Недоліки Forest:

– обмежена аналітика по сесіях (немає детальних графіків);

– відсутність адаптивних нагадувань — фіксовані інтервали;

– частина функцій доступна лише в платній версії.

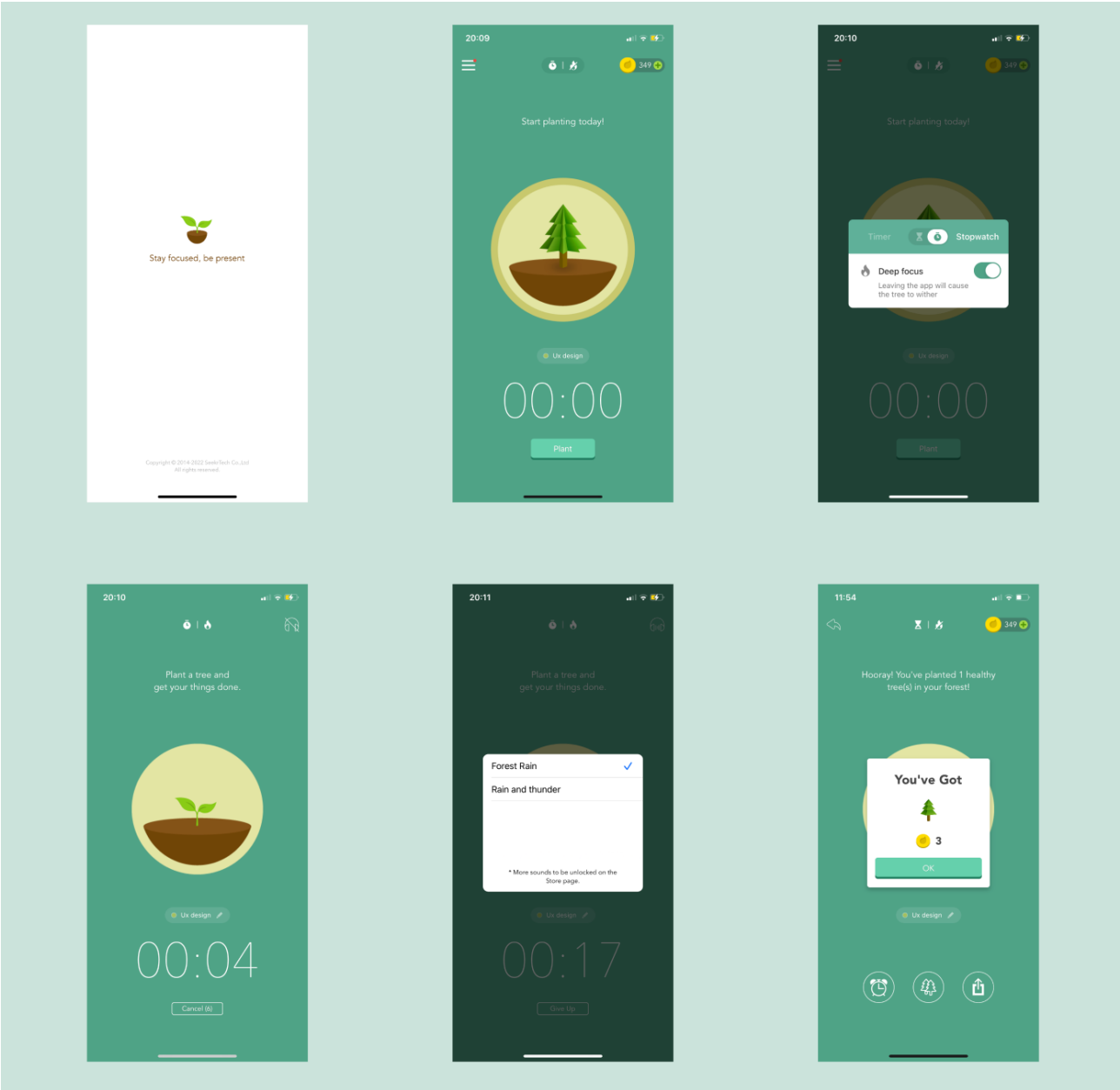


Рис.1.1 Приклад інтерфейсу Forest

1.4.2 RescueTime

RescueTime — це застосунок для автоматичного відстеження часу роботи на комп’ютері та мобільних пристроях із деталізованими звітами.

Основні функції RescueTime включають:

– фоновий моніторинг використання програм і сайтів;

– категоризація діяльності за рівнем продуктивності;

– налаштування цілей (наприклад, не перевищувати 2 години в соціальних мережах на день);

– функція FocusTime для блокування відволікаючих сайтів.

Переваги RescueTime:

– точний облік часу без ручного запуску таймера;

– докладні звіти та аналітика за категоріями діяльності;

– можливість автоматизації блокувань через FocusTime.

Недоліки RescueTime:

– складні налаштування для непідготовленого користувача;

– частина звітів доступна лише у платних планах;

– відсутність гейміфікаційних елементів.

Рис.1.2 Приклад інтерфейсу RescueTime

1.4.3 Pomodoro To-Do

Pomodoro To-Do — це комбінація списку завдань і Pomodoro-таймера з базовою статистикою.

Основні функції Pomodoro To-Do включають:

– створення та організація списку справ із можливістю пріоритизації;

– вбудований Pomodoro-таймер з налаштовуваними інтервалами;

– перегляд історії сесій у вигляді таблиць та простих графіків;

– нагадування про початок і кінець сесії через сповіщення ОС.

Переваги Pomodoro To-Do:

– зручна інтеграція тасків і таймера в одному застосунку;

– легкий інтерфейс і мінімальні налаштування;

– підтримка крос-платформеності (мобільні ОС, веб).

Недоліки Pomodoro To-Do:

– обмежена аналітика (відсутні глибші інсайти щодо продуктивності);

– немає адаптивності в нагадуваннях — фіксовані інтервали;

– мінімальна гейміфікація (лише базові бали без рівнів чи бейджів).

Рис.1.3 Приклад інтерфейсу Pomodoro To-Do

1.4.4 Focus Keeper

Focus Keeper — це простий Pomodoro-таймер із базовими статистичними інструментами.

Основні функції Focus Keeper включають:

– вибір між декількома готовими режимами Pomodoro (25/5, 50/10 тощо);

– підрахунок завершених сесій за день;

– звукові та візуальні сигнали про початок і кінець інтервалів;

– історія сесій у вигляді лістингу за датою.

Переваги Focus Keeper:

– дуже простий і зрозумілий інтерфейс;

– мінімальні відволікання під час роботи;

– стабільна робота без необхідності складних налаштувань.

Недоліки Focus Keeper:

– відсутність інтеграції з списком завдань;

– немає адаптивних чи динамічних нагадувань;

– обмежена статистика без графічної візуалізації.

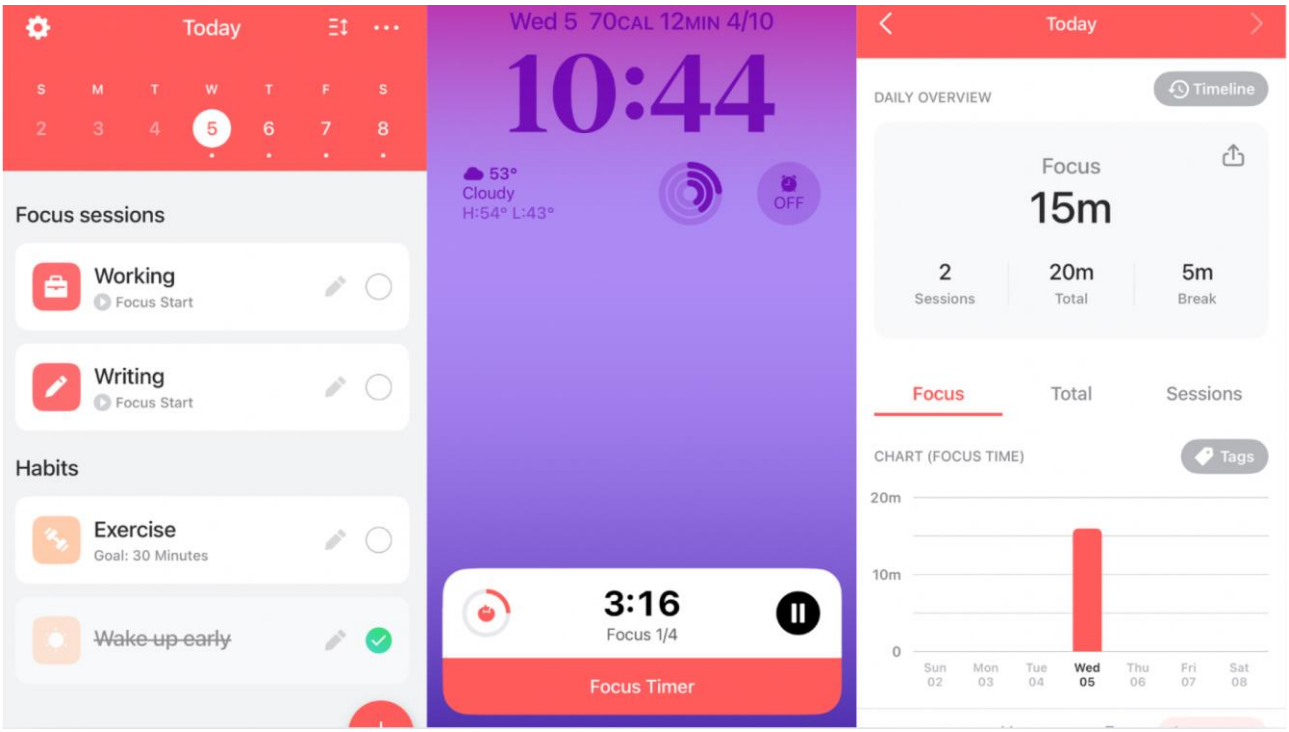


Рис.1.4 Приклад інтерфейсу Focus Keeper

В таблиці 1.1 зведено результати аналізу існуючих застосунків для боротьби з прокрастинацією та їх порівняння.

Таблиця 1.1

Зведена таблиця аналізу існуючих застосунків для боротьби з прокрастинацією та їх порівняння

Показник	**Forest**	**RescueTime**	**Pomodoro To-Do**	**Focus Keeper**
Платформи	iOS, Android, розширення Chrome/Firefox	Windows, macOS, Linux, Android, iOS, Web	iOS, Android, Web	iOS, Android
Pomodoro-таймер	25/5 хв за замовчуванням	FocusTime, але не класичний Pomodoro	Із довільними інтервалами	Із кількома готовими режимами
Налаштування інтервалів	Обмежене (25/5, 50/10)	Відсутнє	Будь-які робочі/перервні інтервали	Обмежене (три фіксовані режими)
Гейміфікація	Вирощування віртуальних дерев	Відсутня	Базова: бали за завершені сесії	Відсутня
Динамічні нагадування	Відсутні	Відсутні	Відсутні	Відсутні

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих застосунків для боротьби з прокрастинацією та їх порівняння

Управління завданнями	Відсутнє	Відсутнє	Вбудований список справ	Відсутнє
Аналітика	Візуалізація «лісу» завершених сесій	Детальні звіти: категорії діяльності, час у програмах/на сайтах	Базова історія сесій у табличному вигляді	Історія сесій у вигляді списку
Мотивація	Візуальний прогрес (ліс)	Цілі та попередження про перевищення ліміту часу	Бали та лічильник сесій	Звукові та візуальні сигнали без мотиваційних елементів
Ціна	Freemium (платні фічі)	Freemium/Premium	Freemium/Преміум-функції	Безкоштовно з in-app покупками

1.5 Визначення вимог до застосунку

Проаналізувавши сутність застосунку, його специфіку, цільову аудиторію та існуючі аналоги, було визначено наступні вимоги:

– Управління завданнями: створення, редагування, видалення й групування завдань та підзавдань із можливістю призначення пріоритетів;

– Pomodoro-таймер: налаштування тривалості робочих сесій і перерв, ручний запуск, пауза та зупинка сеансів;

– Система нарахування балів: підрахунок балів за успішно завершені Pomodoro-сесії та відображення поточного рейтингу користувача;

– Динамічні нагадування: адаптивне коригування інтервалів сповіщень на основі історії виконання сесій;

– Пошук і фільтрація завдань: можливість знаходити завдання за ключовими словами, пріоритетом, датою створення та статусом виконання;

– Аналітика та звітність: візуалізація продуктивності у вигляді гістограм і лінійних графіків Pomodoro-сесій за обраний період;

– Інтерфейс користувача: односторінковий додаток (SPA) з адаптивним дизайном для десктопів і мобільних пристроїв, що завантажує сторінки менше ніж за 2 секунди;

– Технологічний стек: реалізація на C#, .NET 8, ASP.NET Core, Blazor WebAssembly та Entity Framework Core з SQLite для збереження даних;

– Масштабованість і підтримка: модульна архітектура, яка дозволяє легко додавати нові функції та підключати зовнішні сервіси в майбутньому.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) — це комплекс програмних інструментів, який об’єднує редактор коду, можливості автодоповнення, відлагодження, збірки та візуалізації структури проєкту. Використання потужного IDE дозволяє суттєво підвищити ефективність створення, тестування та підтримки коду веб-застосунку.

2.1.1 Visual Studio 2022

Visual Studio 2022 Community — основне середовище для роботи із серверною частиною на .NET 8 і клієнтським кодом Blazor WebAssembly. Містить потужний редактор з автодоповненням IntelliSense, вбудований відлагоджувач із підтримкою “гарячого” перезавантаження (Hot Reload), інструменти для роботи з Entity Framework Core (міграції, генерація моделей) та глибоку інтеграцію з Git. Дозволяє одним кліком запускати локальний веб-сервер, переглядати деталі викликів HTTP API та налагоджувати Razor-компоненти без додаткових налаштувань.

2.1.2 Visual Studio Code

Visual Studio Code — легкий редактор від Microsoft, який розширюється плагінами C# і Blazor Debugger та використовується для швидкої правки Razor-шаблонів, JavaScript-сценаріїв та CSS-стилів. Його гнучкість і швидкість запуску роблять VS Code зручним інструментом для роботи з фронтендом і конфігураційними файлами проєкту.

2.2 Мови програмування

Мова програмування — формалізована система інструкцій для створення

алгоритмів та програм. Правильний вибір мови впливає на продуктивність розробки, оптимізацію продуктивності та підтримку екосистеми.

2.2.1 C#

C# — основна мова розробки бізнес-логіки та клієнтського інтерфейсу. Забезпечує строгі типи, патерни ООП, підтримку async/await для побудови асинхронних потоків виконання та можливість ре-юзабельності коду між сервером і фронтендом завдяки .NET Standard. Використання LINQ значно спрощує роботу з колекціями й запитами до бази даних через EF Core.

2.3 Веб-фреймворки

Веб-фреймворки надають набір готових компонентів та шаблонів для швидкої побудови клієнт-серверних додатків, вирішуючи типові завдання аутентифікації, маршрутизації та доступу до даних**.**

2.3.1 ASP.NET Core

ASP.NET Core — кросплатформний фреймворк для створення високонавантажених RESTful API та MVC-застосунків. Надає вбудовані middleware для обробки запитів, налаштування CORS, JWT-аутентифікації і авторизації, можливість створення гранульованих політик безпеки та підтримку OpenAPI/Swagger для документації.

2.3.2 Blazor WebAssembly

Blazor WebAssembly- фреймворк, який дозволяє виконувати C#-код у браузері через WebAssembly. Дає змогу створювати реактивний SPA-інтерфейс з компонентною моделлю, повторно використовувати сервіси та моделі даних із серверної частини, а також інтегрувати JavaScript-бібліотеки через JS-interop.

2.4 Система управління базою даних

SQLite у поєднанні з Entity Framework Core (Code-First) — легка вбудована реляційна СУБД, що не потребує окремого сервера. EF Core генерує схему бази даних за моделями C# і керує міграціями, забезпечує параметризовані запити та кешування, а також дозволяє легко замінити SQL-Lite на іншу СУБД (PostgreSQL, SQL Server) з мінімальними змінами в коді.

2.5 Система контролю версій

2.5.1 Git

Git — розподілена система контролю версій, яка дозволяє відстежувати кожен коміт, працювати з гілками (branches) для нових фіч і підтримувати історію змін.

2.5.2 GitHub

GitHub — хостинг-сервіс для віддаленого зберігання репозиторію, code review через pull requests, інтеграції CI/CD та автоматичного деплою. Використовується для організації командної роботи та резервного копіювання коду.

2.6 Інструменти проєктування інтерфейсу користувача

Figma — застосунок для створення макетів UI/UX і прототипів. Дає змогу в команді оперативно обговорювати дизайн, експортувати іконки та стилі для безпосередньої інтеграції в Razor-компоненти, налаштовувати гайдлайни по кольорах і типографіці, а також підтримує плагіни для генерації CSS-перемінних та SVG-спрайтів.

3. ПРОЄКТУВАННЯ

3.1 Проєктування архітектури застосунку

Проєктування архітектури веб-застосунку — це фундаментальний етап, який визначає успішність усіх подальших стадій розробки: від написання коду й тестування до розгортання та супроводу. Правильно спроєктована архітектура забезпечує ясність відповідальностей, полегшує масштабування системи та підвищує гнучкість у впровадженні нових функцій.

У цьому проєкті обрано багаторівневу (N-tier) модель із трьома чітко відокремленими шарами:

– Presentation Layer — відповідає за взаємодію з користувачем;

– Business Logic Layer — містить сервіси й алгоритми, що реалізують бізнес-правила;

– Data Access Layer — забезпечує збереження та отримання даних із бази.

Такий підхід базується на найкращих практиках інженерії ПЗ та дає:

– Масштабованість. Кожний шар можна незалежно масштабувати або замінити новими модулями без втручання в інші шари.

– Підтримуваність. Зміни в реалізації доступу до даних не «протікають» у Presentation Layer, а еволюція UI-компонент не зачіпає бізнес-логіку.

– Тестованість. Чіткі контракти (інтерфейси API, сервіси, репозиторії) дозволяють юніт- та інтеграційне тестування окремих шарів без їхнього переплетення.

– Безпеку. Центральна обробка валідації даних, аутентифікації та авторизації у Business Layer мінімізує вектори атак (SQL-ін’єкції, CSRF, XSS).

– Повторне використання. Presentation- та Business-компоненти можна легко застосувати в інших клієнтських додатках (мобільних або десктопних), зберігаючи загальні правила та API.

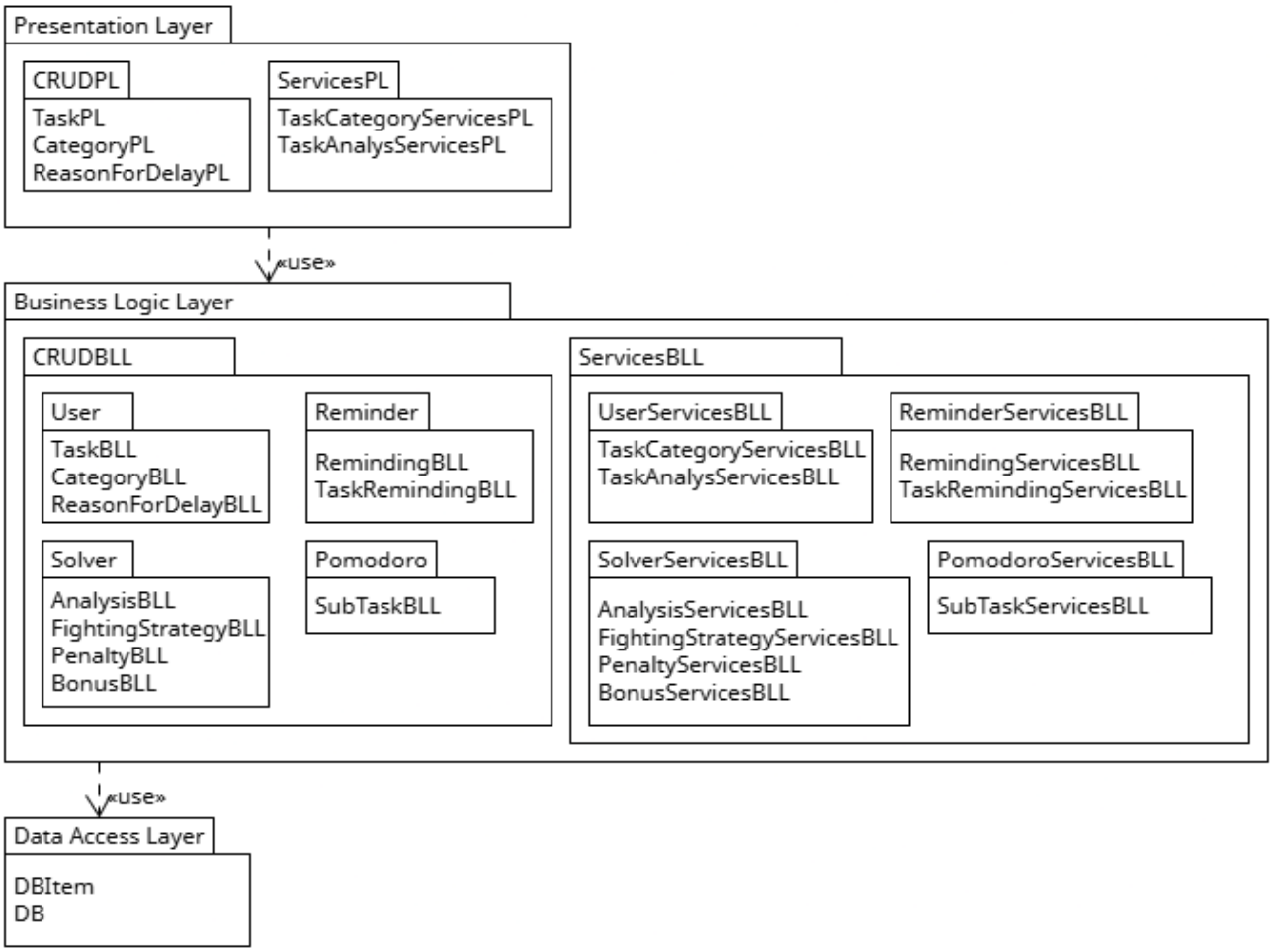


Рис. 3.1 Діаграма рівневої архітектури застосунку

Presentation Layer реалізовано на Blazor WebAssembly. Цей шар відповідає за:

– побудову інтерактивного інтерфейсу за допомогою Razor-компонентів (об’єднання HTML-розмітки й C#-логіки);

– маршрутизацію SPA, що зберігає стан програми між переходами без перезавантаження;

– обробку подій користувача та формування HTTP-запитів до Web API;

– відображення повідомлень про помилки, стан Pomodoro-сесій і оновлення статусів завдань.

Завдяки такому рішенню:

– знижується поріг входження для розробників (одна технологія — C# для фронтенду й бекенду);

– забезпечується висока продуктивність завдяки WebAssembly;

– гарантується безпечне виконання коду в пісочниці браузера.

Business Logic Layer складається з окремих сервісів, кожен із власним інтерфейсом (див. рисунок 3.3.):

– TaskService — керування завданнями й підзадачами (фільтрація, сортування, оновлення статусів);

– PomodoroService — запуск/зупинка таймера, збереження історії сесій, розрахунок оптимальних інтервалів;

– ReminderService — адаптивні сповіщення, що підлаштовуються під продуктивність користувача;

– SolverService — алгоритми нарахування бонусів і штрафів за дотримання дедлайнів;

– загальні сервіси валідації, логування подій та обробки виключень.

Кожен сервіс можна протестувати ізольовано, що значно скорочує час розробки та підвищує стабільність.

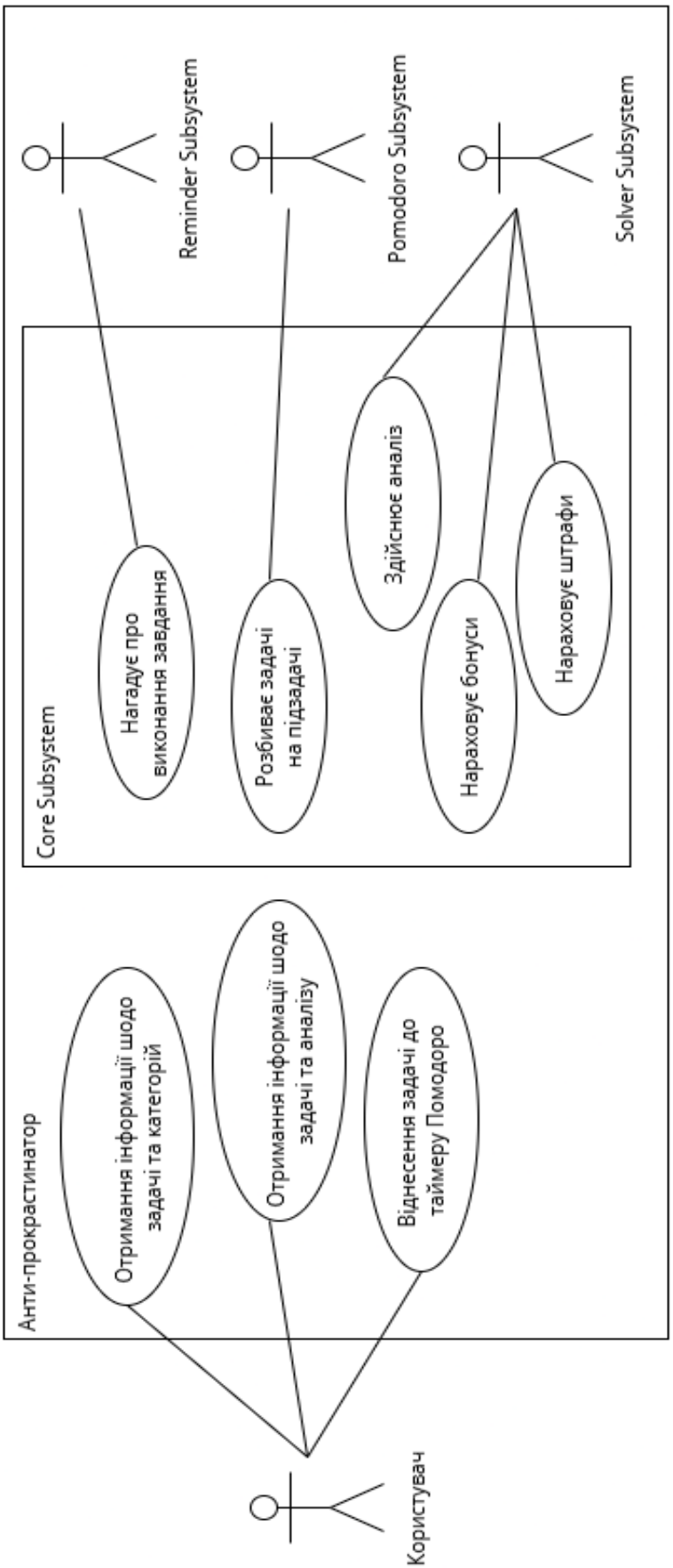


Рис. 3.2 Діаграма прецедентів «Основні функції»

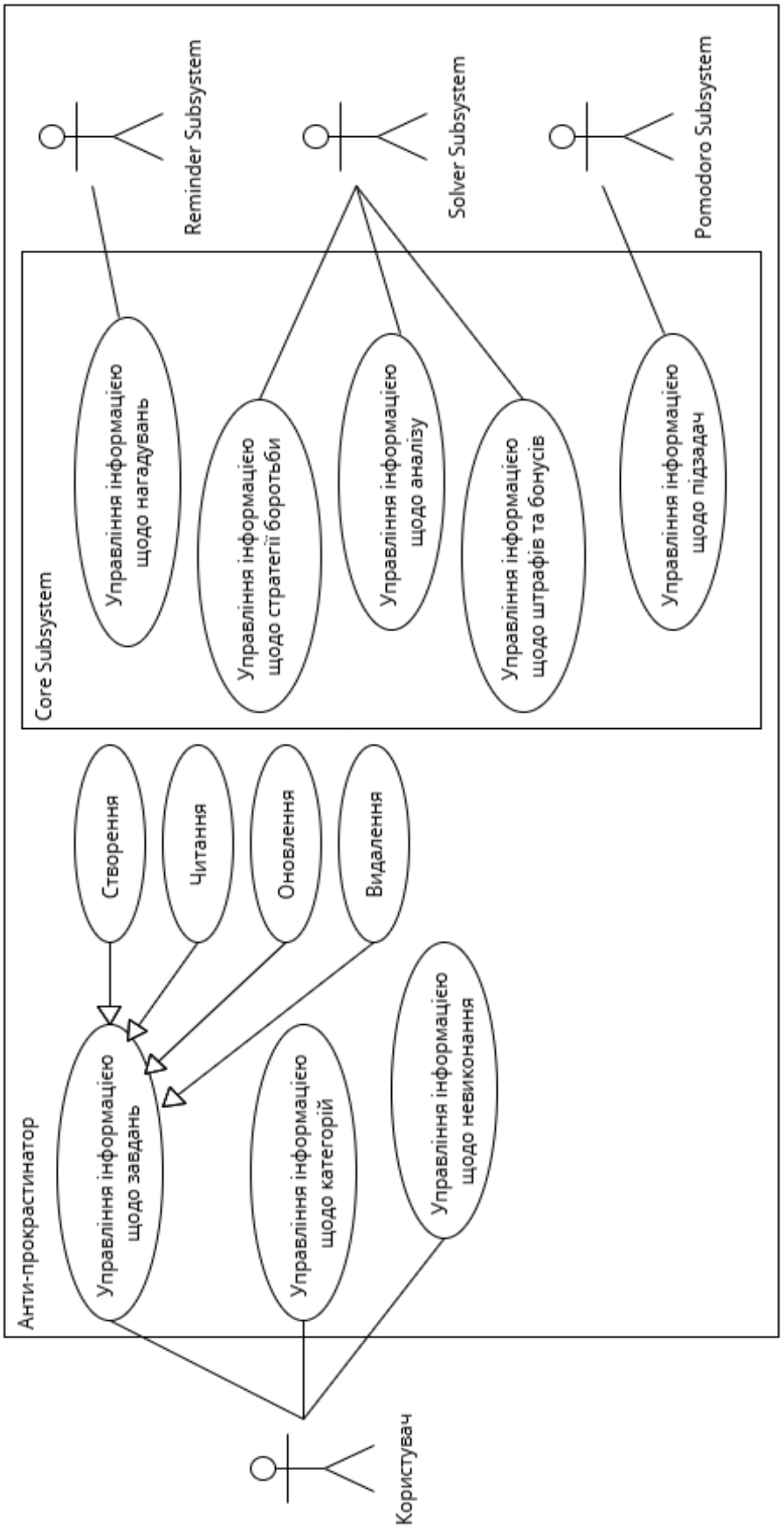


Рис. 3.3 Діаграма прецедентів CRUD

Data Access Layer побудовано на Entity Framework Core із SQLite за підходом Code-First:

– сутності (User, TaskItem, SubTask, PomodoroLog, Reminder, Point) описано в коді — міграції автоматично створюють і оновлюють схему БД;

– патерн Repository інкапсулює CRUD-операції, Unit of Work гарантує атомарність транзакцій;

– Fluent API налаштовує зв’язки “один-до-багатьох” і “багато-до-багатьох” та індекси для прискорення запитів.

Рис. 3.4 Діаграма діяльності «Життєвий цикл завдання»

Усі шари взаємодіють через чітко визначені DTO та інтерфейси, а залежності керуються через вбудований у ASP.NET Core контейнер Dependency Injection, що спрощує підміну й тестування реалізацій сервісів.

3.2 Архітектура клієнтської частини

Клієнтська частина застосунку побудована на Blazor WebAssembly, що дає змогу писати інтерфейсну логіку на C# і запускати її в браузері без проміжних перетворень у JavaScript. Така архітектура забезпечує єдиний стек технологій, підвищену швидкість завантаження та безпеку виконання коду в «пісочниці» браузера.

У межах цього шару можна виділити чотири основні підсистеми:

– Razor-компоненти

– кожен екран і елемент UI реалізовано як окремий .razor-файл, у якому поєднано HTML-розмітку та C#-логіку;

– компоненти інкапсулюють візуальні блоки (список завдань, кнопки керування Pomodoro, діаграми статистики тощо) та можуть повторно використовуватися на різних сторінках.

– Сторінки (Pages)

– маршрутизація задається атрибутом @page (наприклад, @page "/dashboard");

– кожна сторінка агрегує набір компонентів і відповідає за початкове підвантаження даних і обробку глобальних подій (наприклад, оновлення списку завдань при переході між розділами).

– Клієнтські сервіси (Services)

– інкапсулюють HTTP-виклики до REST API та бізнес-логіку презентаційного рівня;

– зареєстровані в DI-контейнері ASP.NET Core як singleton або scoped;

– приклад: TaskService із методами GetAllAsync(), CreateAsync(TaskItem),

CompleteAsync(id).

– Сховище стану (State Management)

– реалізовано за патерном Flux (бібліотека Fluxor): через дії (actions), редюсери (reducers) та ефекти (effects) централізується зберігання стану (користувач, завдання, статистика);

– це спрощує трасування змін, юніт-тестування та відлагодження.

– Навігація та маршрутизація

– компонент Router відповідає за динамічну підстановку компонентів згідно з URL без перезавантаження сторінки;

– забезпечує плавний UX, близький до настільних застосунків.

Загалом, такий підхід дає змогу легко розширювати інтерфейс: додавання нового екрану або механіки гейміфікації потребує мінімальних змін у вже існуючих компонентах. Сумісність з .NET 8 і Blazor WebAssembly гарантує можливість подальшої еволюції клієнтської частини.

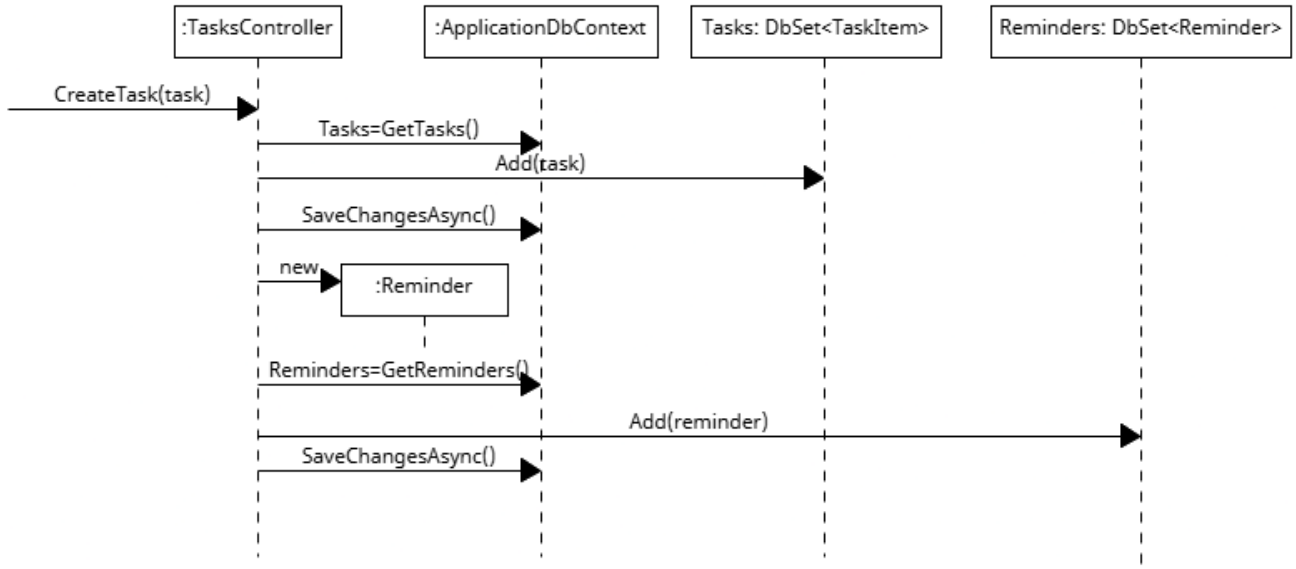


Рис. 3.5 Діаграма послідовностей клієнтської частини

Щоб показати, як змінюються стани інтерфейсу під час роботи з головним екраном, наведено діаграму станів.

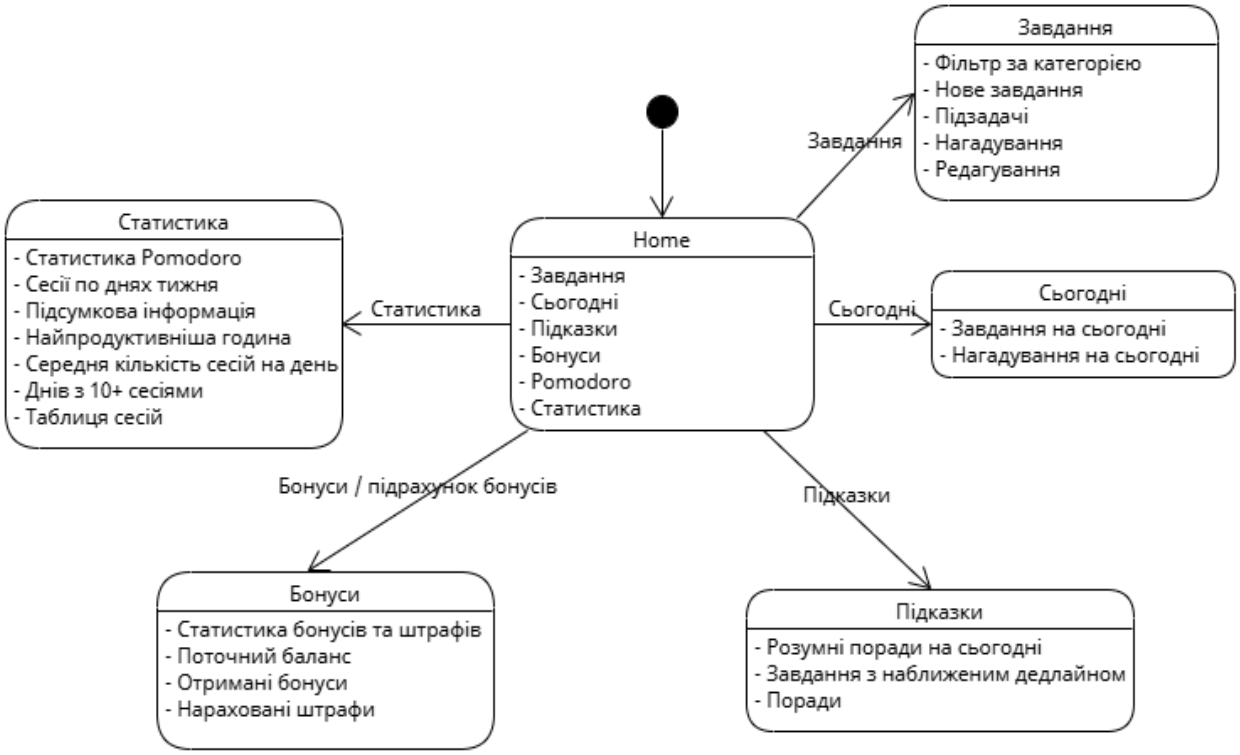


Рис. 3.6 Діаграма станів інтерфейсу

3.3 Архітектура серверної частини

Архітектуру серверної частини побудовано за багаторівневою N-tier моделлю на базі ASP.NET Core. Кожен рівень відповідає за свою зону відповідальності й взаємодіє з іншими через чітко визначені інтерфейси та DTO, що забезпечує модульність, тестованість і безпеку.

- Presentation API Layer містить контролери, які приймають HTTP-запити, виконують валідацію та аутентифікацію, формують і повертають JSON-відповіді; серед них TasksController для CRUD-операцій із завданнями, PomodoroController для керування Pomodoro-сесіями, RemindersController для адаптивних нагадувань, BonusesController і PenaltiesController для нарахування балів і штрафів.

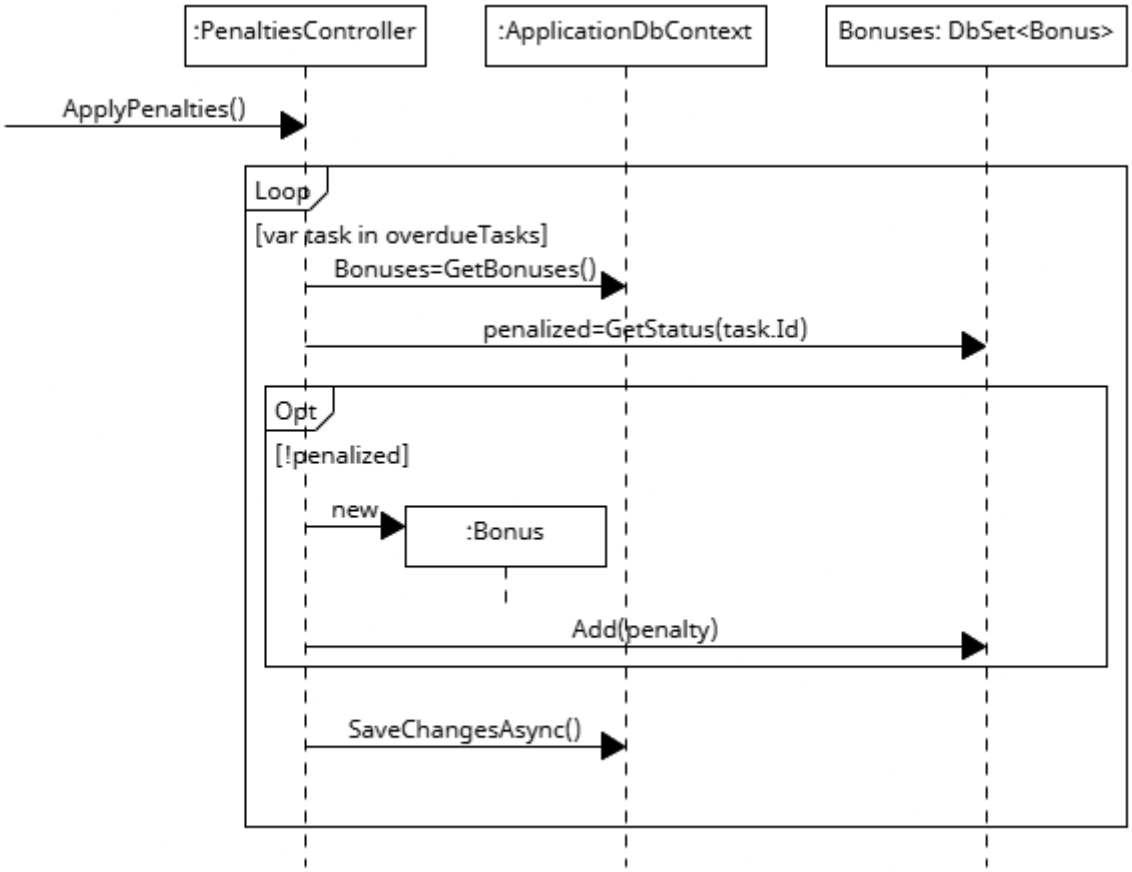


Рис. 3.7 Діаграма послідовностей застосування штрафів

- Business Logic Layer укладається з окремих сервісів з чіткими контрактами: TaskService реалізує фільтрацію, сортування й оновлення TaskItem та SubTask; PomodoroService — запуск, паузу, збереження PomodoroLog і розрахунок відпочинкових інтервалів; ReminderService — адаптивні правила сповіщень; SolverService — нарахування бонусів і штрафів; окремі ValidationService, LoggingService та ExceptionHandlingService опрацьовують валідацію, логування і помилки.
- Data Access Layer побудовано на Entity Framework Core із Code-First міграціями та SQLite: ApplicationDbContext визначає DbSet для сутностей TaskItem, SubTask, PomodoroLog, Reminder, Bonus, FightingStrategy, ReasonForDelay, Category тощо; Fluent API налаштовує зовнішні ключі, каскадні правила й індекси; патерни Repository і Unit of Work інкапсулюють CRUD-операції й гарантують їхню атомарність.

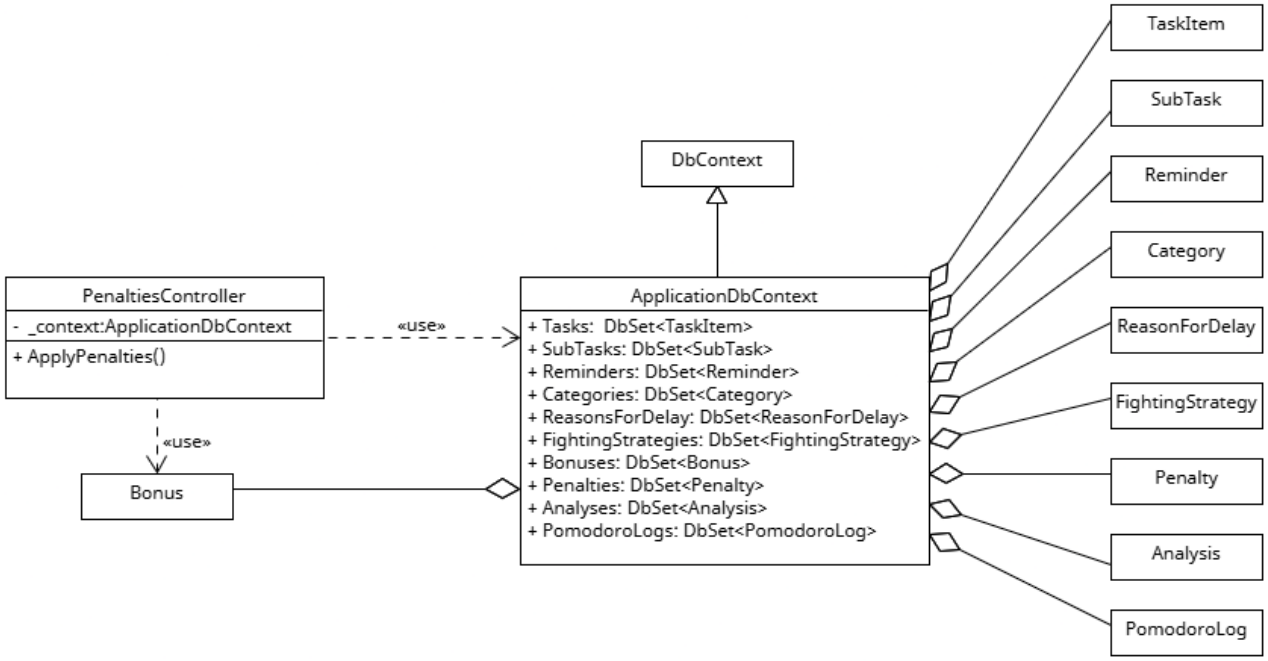


Рис.3.8 Діаграма класів контексту даних

- Додатково сервер підтримує глобальне оброблення помилок і централізоване логування через Serilog, JWT-аутентифікацію з перевіркою токенів на кожному запиті та CORS-політику для безпечної взаємодії з клієнтами.
-

Завдяки такому розділенню шарів рішення легко масштабувати: можна додавати нові контролери, сервіси чи оновлювати схему БД без втручання в інші частини проєкту; забезпечується високий рівень повторного використання коду в різних клієнтських або мікросервісних рішеннях.

3.4 Архітектура бази даних

Архітектура бази даних реалізована за реляційною моделлю з використанням SQLite у поєднанні з Entity Framework Core (Code-First). Це рішення забезпечує просте розгортання без додаткових СУБД, автоматичне створення та оновлення схеми через міграції, а також достатню продуктивність для більшості персональних

та корпоративних сценаріїв.

Сутності бази даних відображаються такими таблицями:

– User (UserId, Username, PasswordHash, CreatedAt) з інформацією про облікові записи;

– TaskItem (TaskItemId, UserId, Title, Description, Priority, CreatedAt, DueDate, IsCompleted, CategoryId) — завдання користувача;

– SubTask (SubTaskId, TaskItemId, Title, IsCompleted, Deadline) — дрібніші кроки в межах одного завдання;

– PomodoroLog (PomodoroLogId, TaskItemId, StartTime, EndTime, Duration, IsBreak) — записи Pomodoro-сесій;

– Reminder (ReminderId, TaskItemId, IntervalMinutes, LastSentAt, IsActive) — налаштовані нагадування;

– Bonus (BonusId, TaskItemId, Points, AwardedAt, Type) — нараховані бали за виконання;

– FightingStrategy (FightingStrategyId, Name, Description) — стратегії боротьби з прокрастинацією;

– ReasonForDelay (ReasonForDelayId, TaskItemId, Reason) — причини прострочення;

– Category (CategoryId, Name) — категорії завдань;

– Analysis (AnalysisId, TaskItemId, RecordedAt, FocusLevel, TasksCompleted, TasksDelayed) — результати аналітики.

Зв’язки реалізовано через зовнішні ключі: один-до-одного між TaskItem → Reminder, один-до-багатьох між User → TaskItem, TaskItem → SubTask, TaskItem → PomodoroLog, TaskItem → ReasonForDelay, Category → TaskItem.

Fluent API в EF Core налаштовує каскадні правила видалення та індекси для швидких запитів за UserId, TaskItemId та RecordedAt.

Щоб наочно продемонструвати предметну галузь та всі ключові сутності з їхніми зв’язками, наведено ER-діаграму.

Рис. 3.9 Діаграма предметної галузі

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Дизайн інтерфейсу

У цьому проєкті макети інтерфейсу не створювалися у зовнішніх редакторах — весь дизайн опрацьовано безпосередньо в коді з використанням можливостей Blazor WebAssembly та CSS. Такий підхід дозволив швидко ітеративно вносити зміни, відразу бачити результат у браузері та гарантувати, що стилі і компоненти будуть точно відповідати кінцевій реалізації.

– Razor-компоненти як основа UI

– Усі екрани оформлено як окремі .razor-файли: MainLayout.razor, Tasks.razor, Pomodoro.razor, Statistics.razor тощо.

– Вбудована CSS-ізоляція (файли ComponentName.razor.css) забезпечує локальні стилі для кожного компонента без конфліктів.

– CSS-фреймворк і власні змінні

– Використовувався частково Bootstrap для базових сіток і утиліт (контейнер, row/col, кнопки).

– Додано набір CSS-змінних у `wwwroot/css:

css :root { --primary-color: #ffbb33; --secondary-color: #3366ff; --font-family: 'Segoe UI', sans-serif; --border-radius: 8px; }

– Стилі Flexbox і CSS Grid застосовані для адаптивного розташування панелей і карток завдань.

– Компонентна бібліотека у коді

– TaskCard (TaskCard.razor) — відображає заголовок, статус, дату виконання та кнопку “Edit”/“Done”.

– TimerPanel (PomodoroTimer.razor) — показує зворотний відлік, кнопки “Start/Stop” і “Skip Break”, а також індикатор прогресу.

– StatsChart (StatisticsChart.razor) — обгортає бібліотеку chart.js через JS-інтероп

для побудови гістограм та лінійних графіків.

– Адаптивний дизайн

– Використано медіа-запити у site.css:

css @media (max-width: 768px) { .task-grid { grid-template-columns: 1fr; } .sidebar { display: none; } }

– Інтерфейс автоматично перелаштовується під мобільні (≤480 px), планшети (≤768 px) і десктопи (>768 px).

– UX-оптимізації

– Завантаження списку завдань виконується через асинхронний виклик TaskService.GetAllAsync(), під час якого показується “Loading…” — реалізовано в LoadingIndicator.razor.

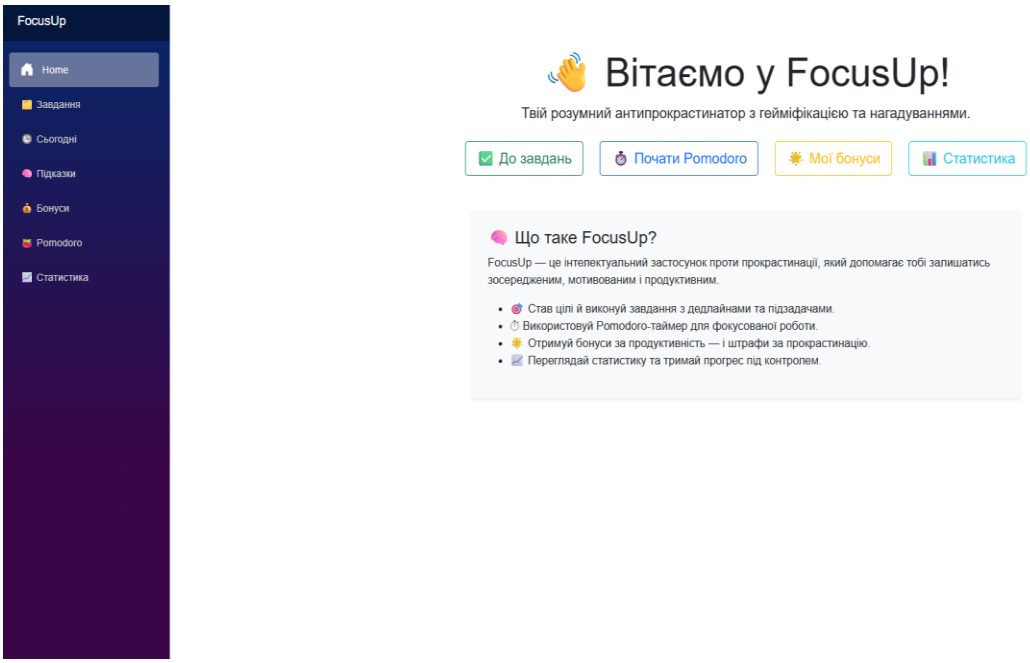


Рис.4.1 Головна сторінка застосунку з переліком завдань і Pomodoro-таймером

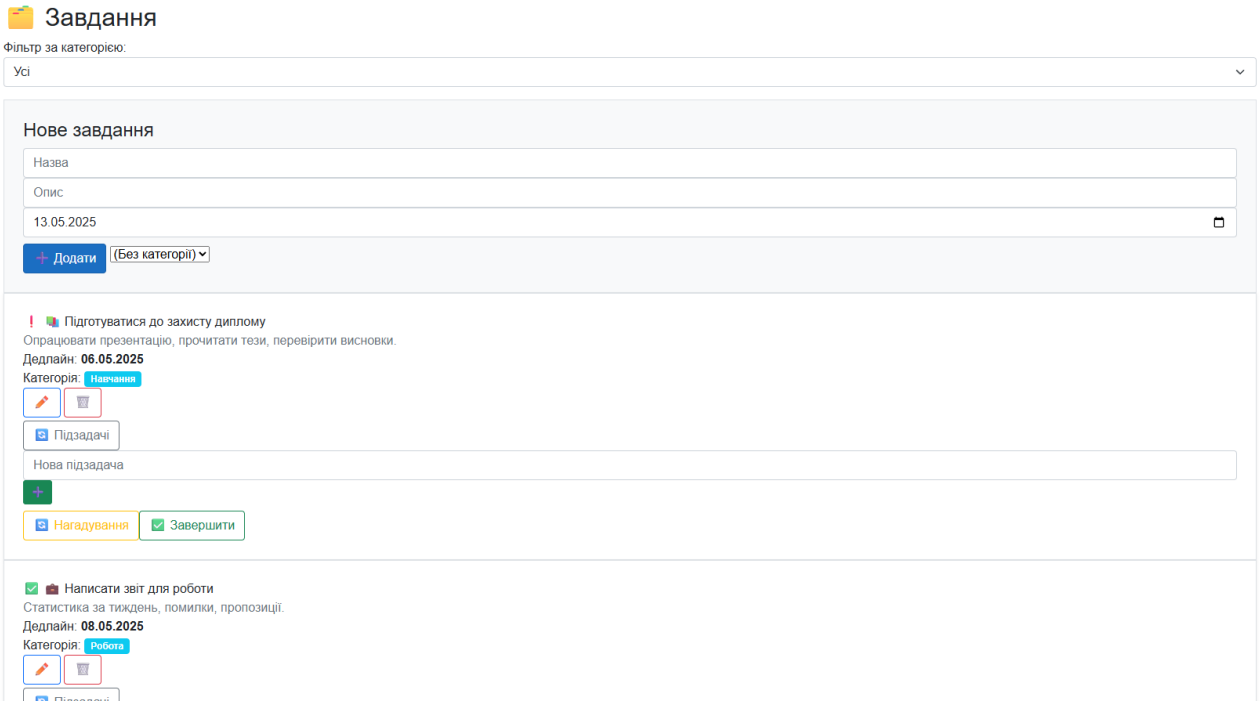


Рис.4.2 Форма створення/редагування завдання

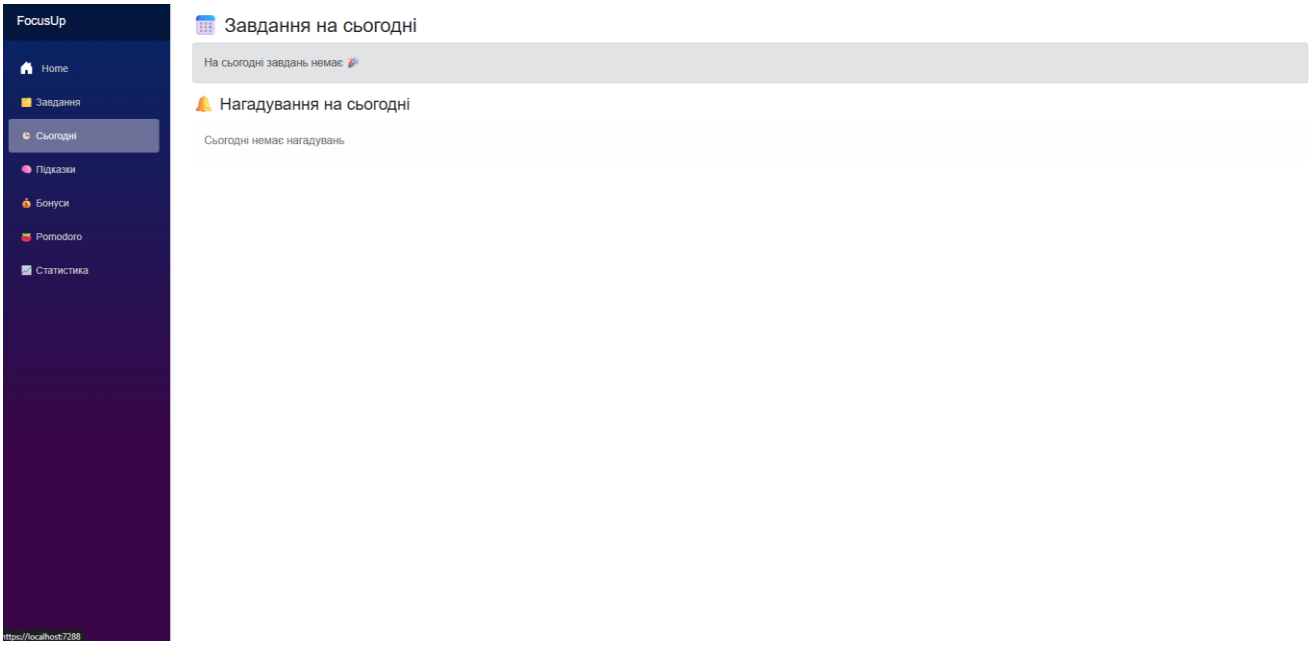


Рис.4.3 Екран «Сьогодні» з переліком завдань та нагадувань

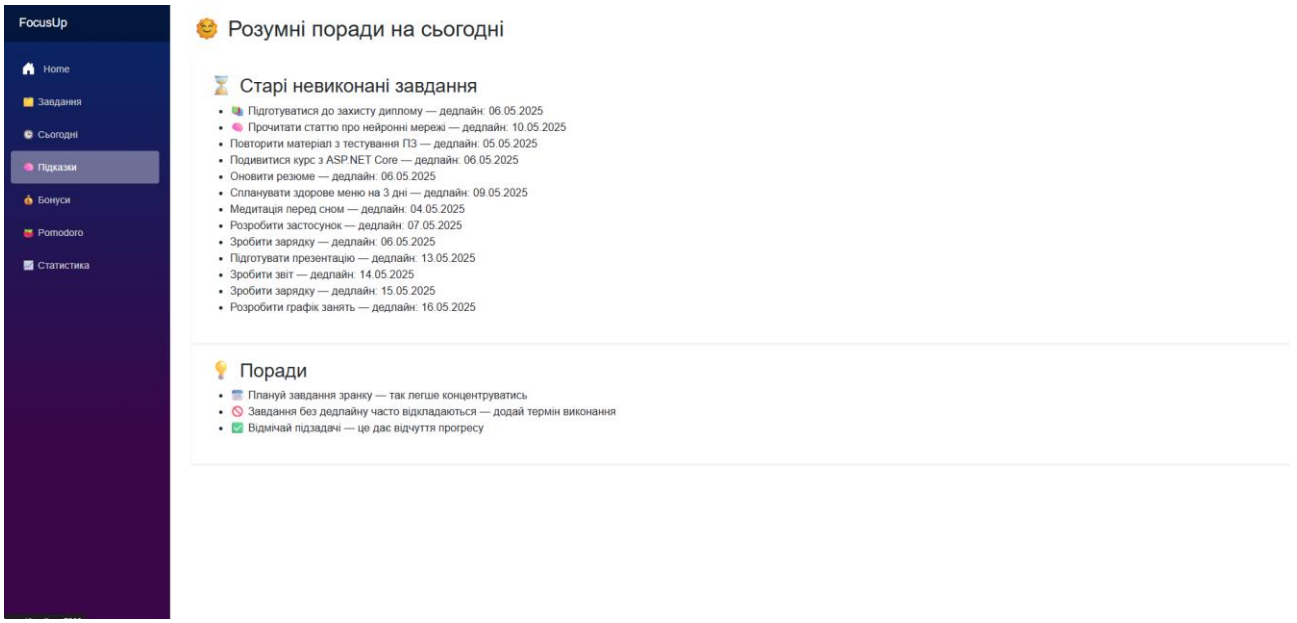


Рис.4.4 Екран «Підказки» з розумними порадами та списком невиконаних завдань

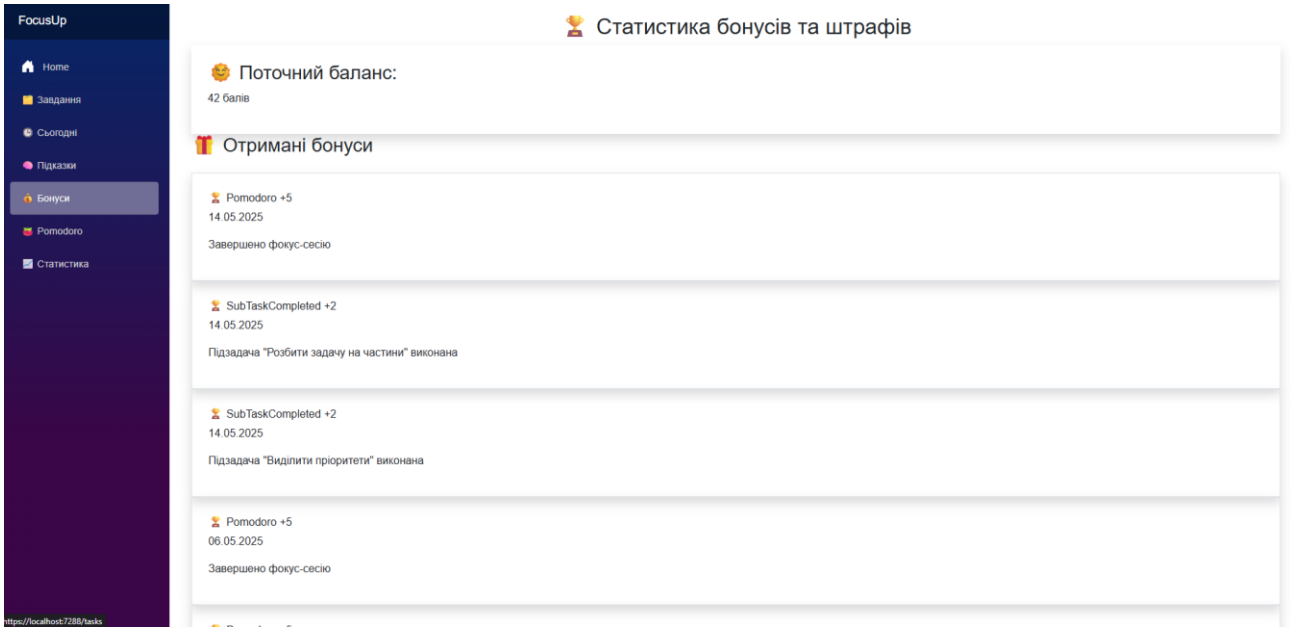


Рис.4.5 Екран «Бонуси» з поточним балансом і деталями отриманих нагород

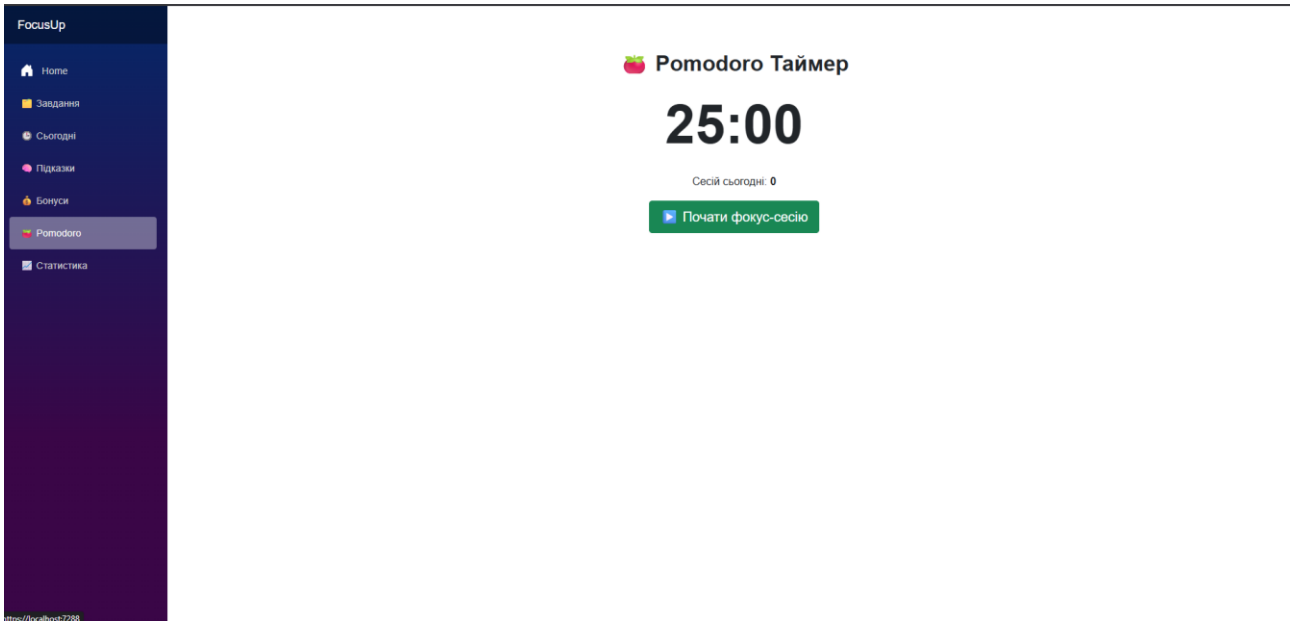


Рис.4.6 Екран «Pomodoro» із таймером фокус-сесії та лічильником сесій за день

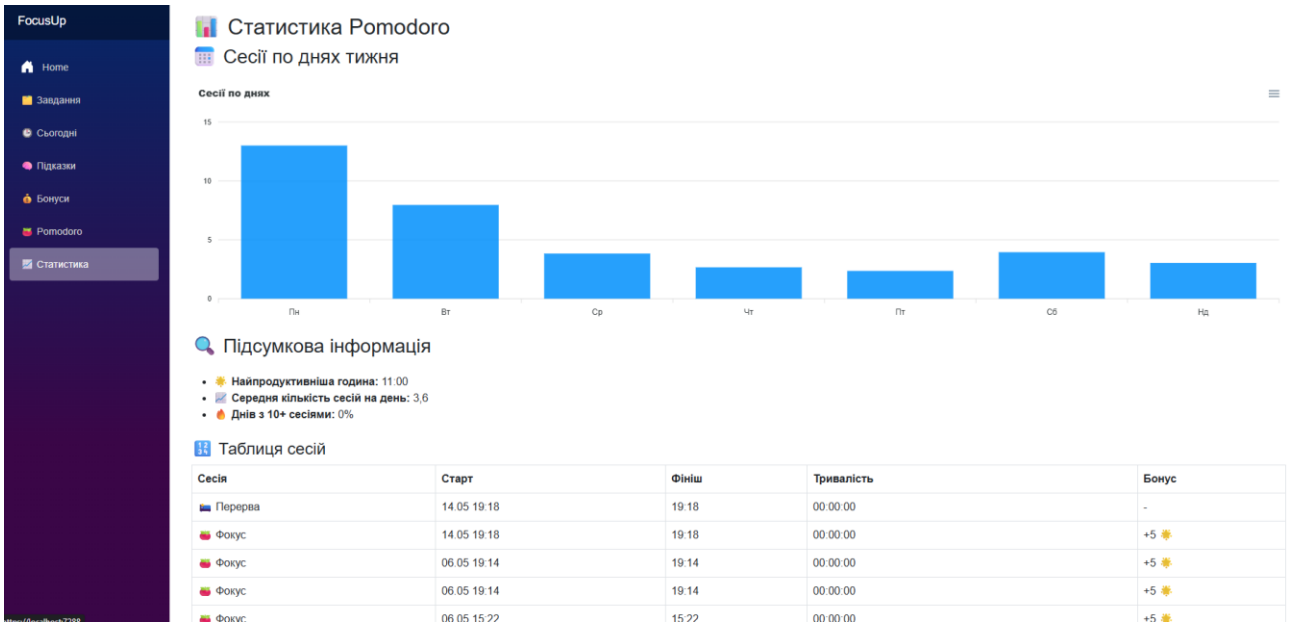


Рис.4.7 Екран «Статистика» з гістограмою Pomodoro-сесій за тиждень і таблицею сесій

4.2 Реалізація клієнтської частини

Клієнтська частина веб-застосунку побудована на Blazor WebAssembly. Нижче наведено основні етапи її реалізації.

Створення проєкту

Створити новий Blazor WASM-проєкт можна командою:

dotnet new blazorwasm -o FocusUp.Client

У результаті з’являється стандартна структура:

Program.cs та Startup.cs (конфігурація DI і маршрутів)

wwwroot/ – статичні ресурси (CSS, JS, зображення)

Pages/ – Razor-сторінки (наприклад, Index.razor, Tasks.razor)

Shared/ – компоненти, спільні між сторінками (наприклад, NavMenu.razor, MainLayout.razor)

Services/ – сервіси бізнес-логіки (TaskService, PomodoroService тощо)

Models/ – DTO та моделі для передачі даних

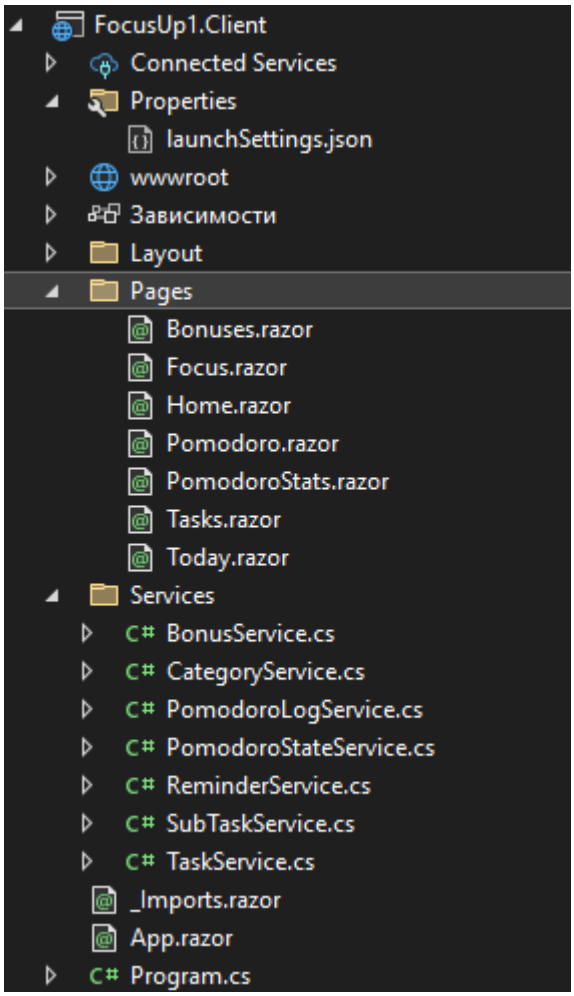


Рис. 4.8 Приклад структури клієнтської частини

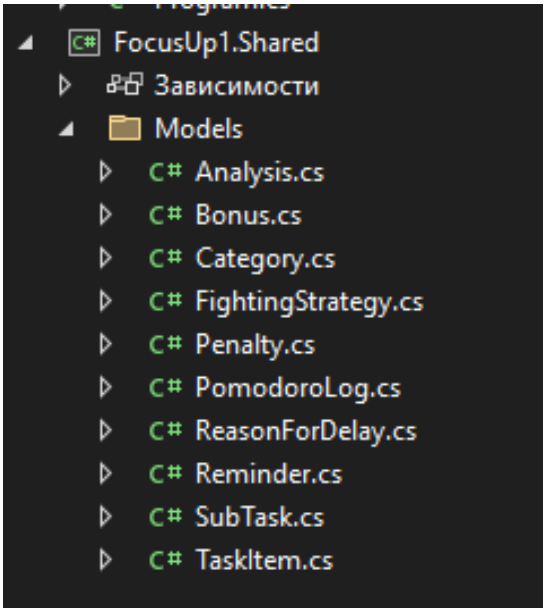


Рис.4.9 Приклад структури Shared

Налаштування DI та HttpClient

В Program.cs зареєстровано сервіси та HTTP-клієнт:

```
builder.Services.AddScoped(sp ⇒
  new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });
builder.Services.AddScoped<TaskService>();
builder.Services.AddScoped<PomodoroService>();
builder.Services.AddScoped<ReminderService>();
builder.Services.AddScoped<BonusService>();
```

Компоненти інтерфейсу

У директорії Pages/ реалізовано:

– Tasks.razor – головний список завдань із прив’язкою до TaskService.GetAllAsync()

– Pomodoro.razor – Pomodoro-таймер із кнопками «Start/Stop» та індикатором прогресу

– Statistics.razor – графіки продуктивності (гістограма та лінійний графік через chart.js)

Кожний компонент супроводжується власним файлом стилів Component.razor.css.

Приклад коду компонента

У Components/TaskCard.razor відображається одна картка завдання:

```
<div class="task-card">
  <h3>@TaskItem.Title</h3>
  <p>Due: @TaskItem.DueDate.ToShortDateString()</p>
  <button @onclick="() ⇒ OnComplete(TaskItem.Id)"></button>
</div>
@code {
  [Parameter] public TaskItem TaskItem { get; set; }
```

```
  [Parameter] public EventCallback<Guid> OnComplete { get; set; }
}
```

Сервіси бізнес-логіки

В директорії Services/:

– TaskService.cs – методи GetAllAsync(), CreateAsync(), UpdateAsync(), DeleteAsync()

– PomodoroService.cs – StartSessionAsync(), StopSessionAsync(), GetHistoryAsync()

– ReminderService.cs – ScheduleAsync(), CancelAsync()

– BonusService.cs – CalculateAsync(), GetBalanceAsync()

– Графіки статистики

У Statistics.razor підключено chart.js через JS-інтероп. Дані передаються у вигляді:

```
var config = new {
  type = "bar",
  data = new { labels, datasets }
};
await JSRuntime.InvokeVoidAsync("renderChart", "chartCanvas", config);
```

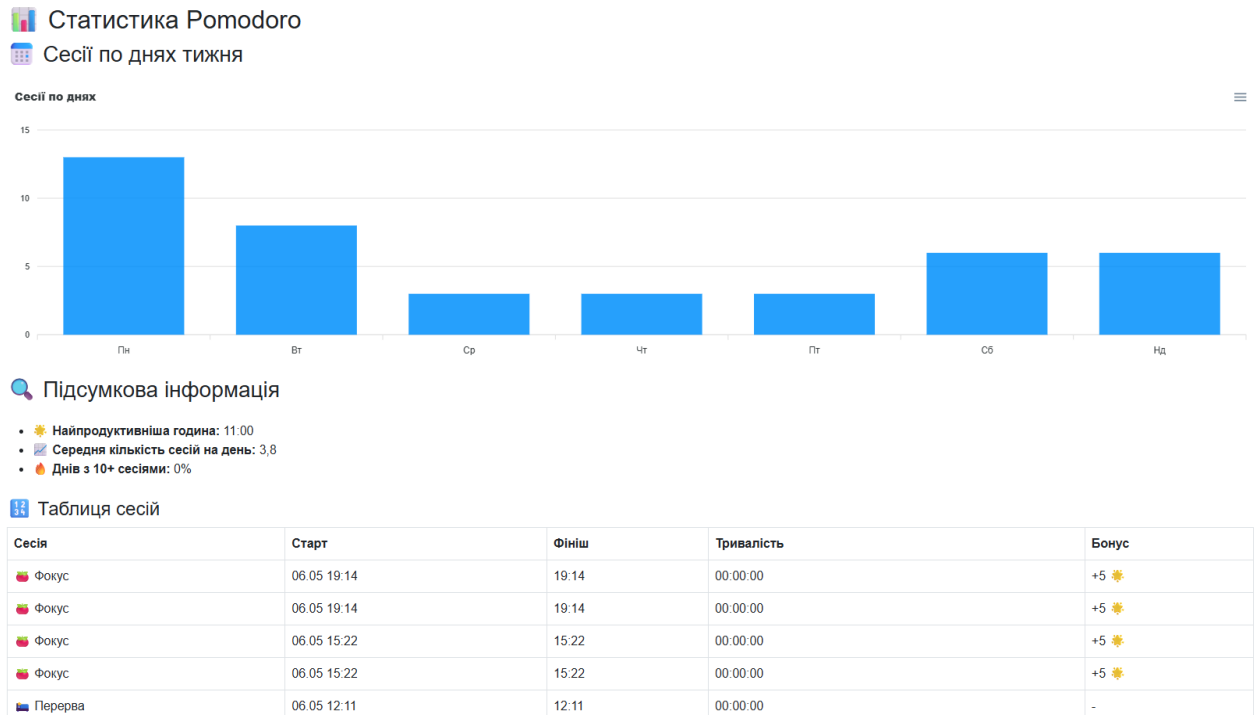


Рис. 4.10 Графік продуктивності Pomodoro-сесій у браузері

Запуск і відлагодження

Для локального запуску:

```
cd FocusUp.Client
dotnet run
```

Застосунок стане доступним за адресою https://localhost:5001. Під час розробки зміни в Razor-файлах миттєво відображаються у браузері без перезавантаження.

4.3 Реалізація серверної частини

Серверна частина застосунку реалізована на базі платформи ASP.NET Core із чітким поділом на конфігурацію хоста, інтеграцію зі сховищем даних, реалізацію HTTP-інтерфейсів, впровадження бізнес-логіки та забезпечення перехресних аспектів (логування, безпека, документація).

4.3.1 Конфігурація хоста та середовища виконання

У файлі Program.cs формується конвеєр обробки запитів:

- утворення об’єкта WebApplicationBuilder із підвантаженням налаштувань із конфігураційних файлів та змінних середовища;
- реєстрація контексту даних (ApplicationDbContext) із викликом UseSqlite(...) для підключення до локальної бази SQLite;
- підключення набору контролерів через AddControllers(), що забезпечує роботу MVC-моделі API;
- увімкнення генерації специфікації OpenAPI/Swagger (AddSwaggerGen()), завдяки чому автоматично формується інтерактивна документація;
- налаштування політики CORS, що дозволяє фронтенд-застосунку звертатися за API з будь-якого походження (у продуктиві рекомендується звуження до конкретних доменів);

- додавання проміжних програм (middleware) для перехоплення виключень, логування запитів та відповідей, а також обробки статичних файлів у разі спільного хостингу SPA.

Після побудови об’єкта WebApplication за допомогою builder.Build(), конфігурація конвеєру завершується викликами:

```
app.UseSwagger();
app.UseSwaggerUI();
app.UseHttpsRedirection();
app.UseCors("DefaultPolicy");
app.UseAuthorization();
app.MapControllers();
```

Така схема гарантує безпечне перенаправлення на HTTPS, коректне обходження політик безпеки, централізовану обробку помилок і доступ до інтерактивної документації.

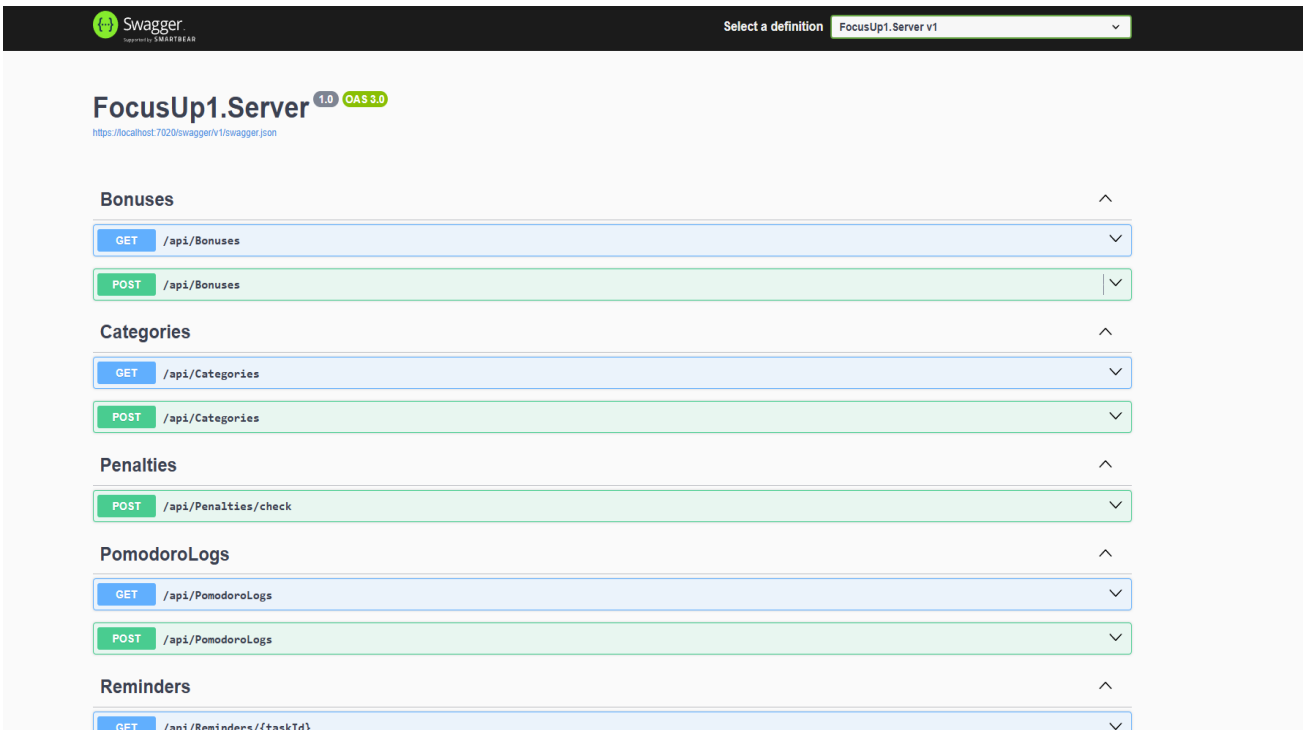


Рис.4.11 Інтерфейс Swagger UI із доступними маршрутами API

4.3.2 Інтеграція зі сховищем даних та EF Core

Схема зберігання даних побудована за підходом Code-First.

- Клас ApplicationDbContext успадковується від DbContext і містить DbSet<TEntity> для кожної сутності (TaskItem, SubTask, PomodoroLog, Reminder, Bonus, Category, Penalty).
- У методі OnModelCreating виконуються конфігурації через Fluent API:
  - визначення зовнішніх ключів та каскадних правил видалення;
  - індексування полів UserId, TaskItemId, RecordedAt для оптимізації вибірок за користувачем та датою;
  - налаштування складних типів даних (DateTime, текстових полів, булевих прапорців).
- Створення початкової міграції (dotnet ef migrations add Initial) і застосування її до бази (dotnet ef database update) призводить до автоматичної генерації необхідних таблиць.
- Для наповнення довідкових даних (категорій, демонстраційних завдань) використовуються механізми сідінгу, які виконуються при старті програми, якщо відповідні записи відсутні.

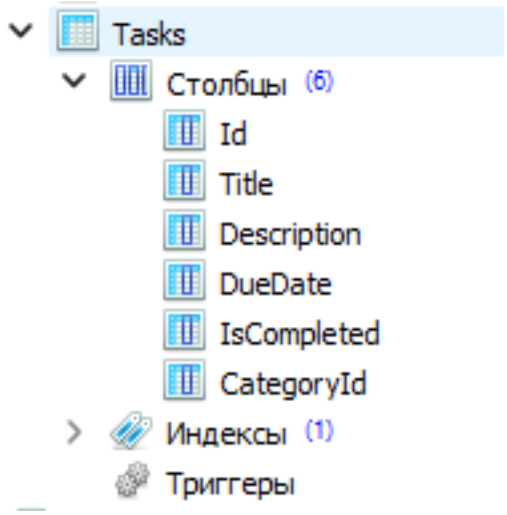


Рис.4.12 Структура таблиць у SQLite

4.3.3 Архітектура контролерів та HTTP-шари

Контролери організовані за функціональними зонами та відповідають REST-принципам:

- TasksController — CRUD-операції над завданнями, пошук за фільтрами, маркування виконання;
- SubTasksController — керування підзадачами, змінення статусу окремої підзадачі;
- PomodoroLogsController — створення та перегляд записів Pomodoro, включаючи тривалість та чи була сесія перервою;
- RemindersController — налаштування персональних нагадувань із адаптивними інтервалами;
- BonusesController та PenaltiesController — нарахування та списання балів залежно від продуктивності та порушень дедлайнів;
- CategoriesController — керування категоріями завдань.

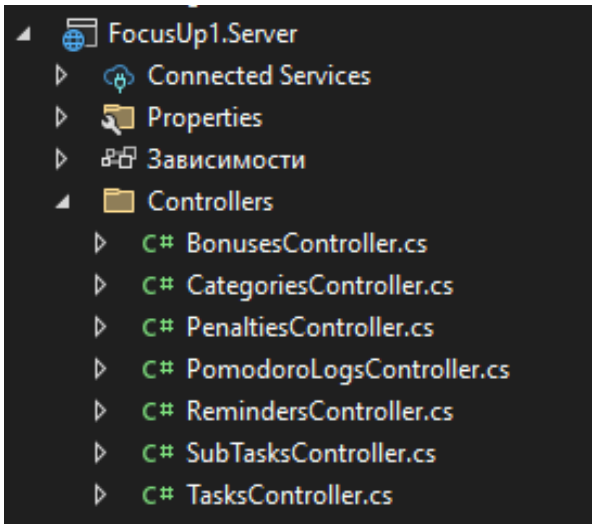


Рис.4.13 Приклад структури контролерів

Кожний контролер позначено атрибутом [ApiController] та маршрутом "/api/[controller]". Валідація моделей здійснюється через атрибути [Required], [Range] тощо, а помилки автоматично конвертуються в зрозумілі клієнту відповіді з кодом 400.

4.3.4 Реалізація бізнес-логіки в окремих службах

Щоб уникнути надмірного завантаження контролерів, алгоритми винесено в окремі служби, що ін’єктуються через DI:

- TaskService — фільтрація, сортування, оновлення статусів завдань і підзадач;
- PomodoroService — забезпечення роботи Pomodoro-таймера, розрахунок оптимальних інтервалів і збереження результатів;
- ReminderService — обчислення часу відправлення наступного нагадування на основі історії продуктивності;
- BonusService — алгоритм нарахування балів або штрафів залежно від точності дотримання інтервалів та завершення завдань.

Служби оформлено як інтерфейси з чітко визначеними контрактами (ITaskService, PomodoroStateService тощо), що спрощує модульне тестування.

4.3.5 Перехресна функціональність: логування, безпека та документація

- CORS-політика забезпечує безпечний обмін даними між клієнтською частиною та API.
- OpenAPI/Swagger автоматично генерує документацію з прикладами запитів і відповідей, що значно полегшує інтеграцію сторонніх клієнтів та подальшу підтримку.
-

4.3.6 Наповнення початкових даних (Data Seeding) і валідація

Оскільки в проєкті не передбачено автоматизованого юніт-тестування, для перевірки коректності роботи API та демонстраційного наповнення бази було реалізовано три методи seed-даних у класі ApplicationDbContext:

public static void SeedTestData(IServiceProvider services) { … }

public static void SeedSubTasksForExistingTasks(IServiceProvider services) { … }

public static void SeedPomodoroLogs(IServiceProvider services) { … }

Ці методи викликаються одразу після застосування міграцій у Program.cs:

```
using var app = builder.Build();
ApplicationDbContext.SeedTestData(app.Services);
ApplicationDbContext.SeedSubTasksForExistingTasks(app.Services);
ApplicationDbContext.SeedPomodoroLogs(app.Services);


app.Run();
```

1. Категорії

У SeedTestData спочатку перевіряється, чи в таблиці Categories вже є записи. Якщо ні, додаються три вбудовані категорії:

- “Робота”
- “Навчання”

“Особисте”

```
if (!context.Categories.Any()) {
    context.Categories.AddRange(
        new Category { Name = "Робота" },
        new Category { Name = "Навчання" },
        new Category { Name = "Особисте" }
    );
    context.SaveChanges();
}
```

	Id	Name
1	1	Робота
2	2	Навчання
3	3	Особисте

Рис. 4.14 Seed-дані Categories

2. Завдання (TaskItem)

Ті самі методи додають чотири приклади завдань, кожне з яких пов’язане з однією з трьох категорій:

- “Підготуватися до захисту диплому” (Категорія “Навчання”, дедлайн — завтра)
- “Написати звіт для роботи” (Категорія “Робота”, дедлайн — через 3 дні)
- “Пробіжка у парку” (Категорія “Особисте”, дедлайн — сьогодні, вже виконане)
- “Прочитати статтю про нейронні мережі” (Категорія “Навчання”, дедлайн — через 5 днів)
-

```
if (!context.Tasks.Any()) {
   var categories = context.Categories.ToList();
   context.Tasks.AddRange(
      new TaskItem { Title = " Підготуватися до захисту диплому", CategoryId = categories.Single(c => c.Name=="Навчання").Id, DueDate = DateTime.Today.AddDays(1), … },
      …
   );
   context.SaveChanges();
}
```

Отфильтровать… Всего загружено строк: 15

	Id	Title	Description	DueDate	IsComp	Catego
1	43	Підготуватися до захисту диплому	Опрацювати презентацію, прочитати тези, перевірити висновки.	2025-05-06 00:00:00	0	2
2	44	Написати звіт для роботи	Статистика за тиждень, помилки, пропозиції.	2025-05-08 00:00:00	1	1
3	45	Пробіжка у парку	30 хв на свіжому повітрі після обіду.	2025-05-05 00:00:00	1	3
4	46	Прочитати статтю про нейронні мережі	Для розширення знань у сфері ШІ.	2025-05-10 00:00:00	0	2
5	47	Повторити матеріал з тестування ПЗ	Пройтися по конспекту, повторити типи тестування, зробити 10 …	2025-05-05 00:00:00	0	2
6	48	Подивитися курс з ASP.NET Core	Відео №7–9 + записати нотатки.	2025-05-06 00:00:00	0	NULL
7	49	Оновити резюме	Додати останній досвід, перевірити помилки, експортувати в PDF.	2025-05-06 00:00:00	0	1
8	50	Спланувати здорове меню на 3 дні	Сніданки, обіди, вечері без фастфуду.	2025-05-09 00:00:00	0	3

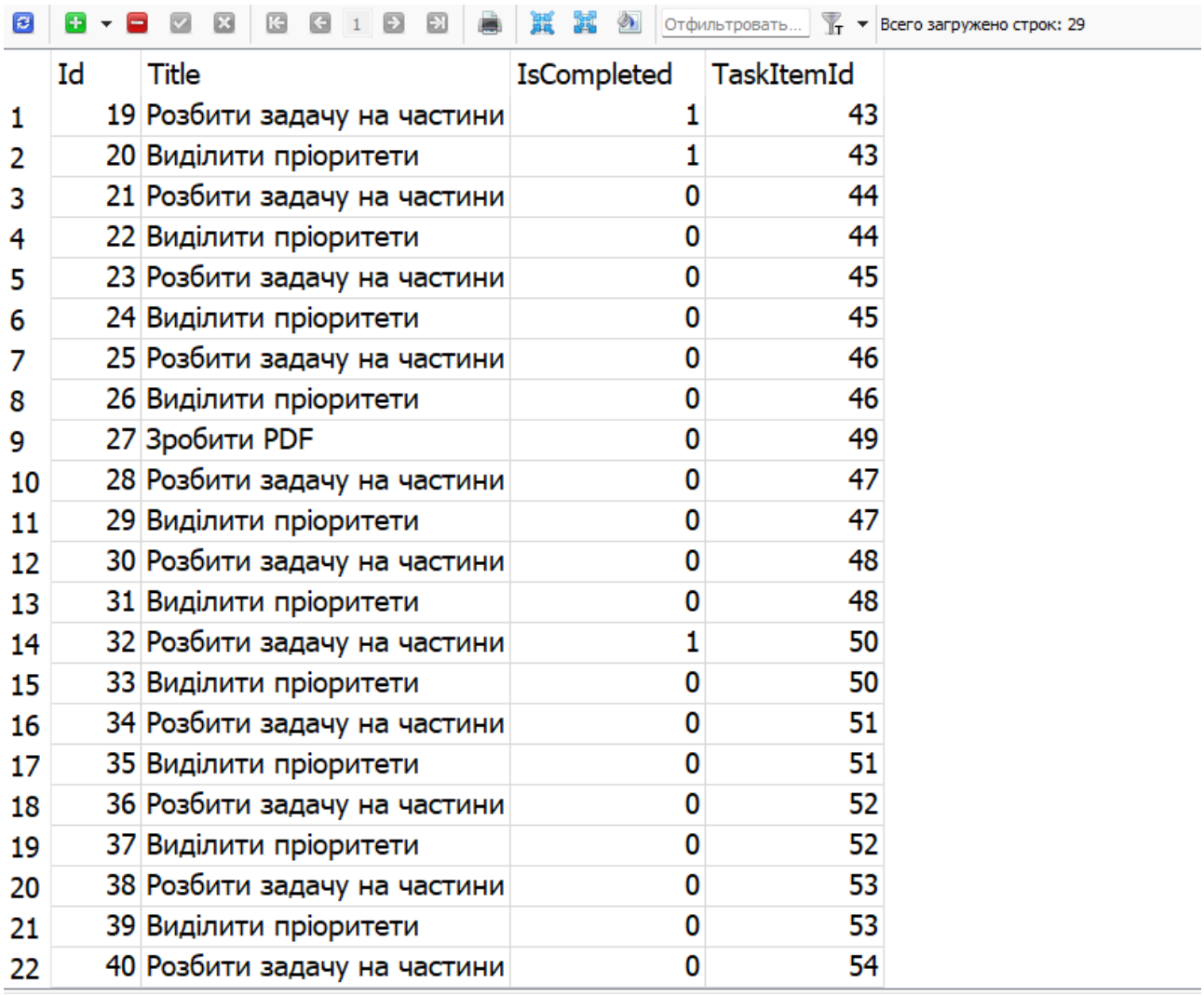
Рис. 4.15 Seed-дані TaskItems

3. Підзадачі (SubTask)

У методі SeedSubTasksForExistingTasks для кожного завдання, що не має жодної підзадачі, додаються дві стандартні підзадачі:

- “Розбити задачу на частини”
-
- “Виділити пріоритети”

```
foreach (var task in context.Tasks) {
   if (!context.SubTasks.Any(s => s.TaskItemId == task.Id)) {
      context.SubTasks.AddRange(
         new SubTask { TaskItemId = task.Id, Title = "Розбити задачу на частини" },
         new SubTask { TaskItemId = task.Id, Title = "Виділити пріоритети" }
      );
   }
}
context.SaveChanges();
```



	Id	Title	IsCompleted	TaskItemId
1	19	Розбити задачу на частини	1	43
2	20	Виділити пріоритети	1	43
3	21	Розбити задачу на частини	0	44
4	22	Виділити пріоритети	0	44
5	23	Розбити задачу на частини	0	45
6	24	Виділити пріоритети	0	45
7	25	Розбити задачу на частини	0	46
8	26	Виділити пріоритети	0	46
9	27	Зробити PDF	0	49
10	28	Розбити задачу на частини	0	47
11	29	Виділити пріоритети	0	47
12	30	Розбити задачу на частини	0	48
13	31	Виділити пріоритети	0	48
14	32	Розбити задачу на частини	1	50
15	33	Виділити пріоритети	0	50
16	34	Розбити задачу на частини	0	51
17	35	Виділити пріоритети	0	51
18	36	Розбити задачу на частини	0	52
19	37	Виділити пріоритети	0	52
20	38	Розбити задачу на частини	0	53
21	39	Виділити пріоритети	0	53
22	40	Розбити задачу на частини	0	54

Рис. 4.16 Seed-дані SubTasks

4. Pomodoro-логи

SeedPomodoroLogs заповнює 3 Pomodoro-сесії на кожен із останніх 10 днів. Кожен запис містить час початку, час завершення, прапорець IsBreak = false та BonusGiven = true:

```
for (int dayOffset = 0; dayOffset < 10; dayOffset++) {
    var day = DateTime.Today.AddDays(-dayOffset);
    logs.Add(new PomodoroLog { StartTime = day.AddHours(9), EndTime = day.AddHours(9).AddMinutes(25), IsBreak = false, BonusGiven = true });
    …
}
context.PomodoroLogs.AddRange(logs);
```

context.SaveChanges();

Всего загружено строк: 47

	Id	StartTime	EndTime	IsBreak	BonusGiven
26	88	2025-04-27 10:00:00	2025-04-27 10:25:00	0	1
27	89	2025-04-27 11:00:00	2025-04-27 11:25:00	0	1
28	90	2025-04-26 09:00:00	2025-04-26 09:25:00	0	1
29	91	2025-04-26 10:00:00	2025-04-26 10:25:00	0	1
30	92	2025-04-26 11:00:00	2025-04-26 11:25:00	0	1
31	93	2025-05-05 15:42:57.213	2025-05-05 15:42:57.213	0	1
32	94	2025-05-05 15:43:01.262	2025-05-05 15:43:01.262	1	0
33	95	2025-05-05 15:43:05.29	2025-05-05 15:43:05.29	0	1
34	96	2025-05-05 15:43:10.612	2025-05-05 15:43:10.612	0	1
35	97	2025-05-05 15:43:41.811	2025-05-05 15:43:41.811	0	1
36	98	2025-05-05 15:43:45.844	2025-05-05 15:43:45.845	1	0
37	99	2025-05-05 15:43:49.866	2025-05-05 15:43:49.866	0	1
38	100	2025-05-05 15:43:54.534	2025-05-05 15:43:54.534	0	1
39	101	2025-05-05 15:57:56.156	2025-05-05 15:57:56.156	0	1
40	102	2025-05-06 12:11:30.157	2025-05-06 12:11:30.157	0	1
41	103	2025-05-06 12:11:34.191	2025-05-06 12:11:34.191	1	0
42	104	2025-05-06 15:22:39.701	2025-05-06 15:22:39.701	0	1
43	105	2025-05-06 15:22:46.05	2025-05-06 15:22:46.05	0	1
44	106	2025-05-06 19:14:29.954	2025-05-06 19:14:29.954	0	1
45	107	2025-05-06 19:14:36.971	2025-05-06 19:14:36.971	0	1
46	108	2025-05-14 19:18:27.448	2025-05-14 19:18:27.448	0	1
47	109	2025-05-14 19:18:31.488	2025-05-14 19:18:31.488	1	0

Рис. 4.17 Seed-дані PomodoroLogs

4.4 Завантаження проєкту на GitHub

Застосування системи контролю версій у процесі розробки є надзвичайно важливим для забезпечення відстежуваності змін, можливості відкотитися до попередніх станів коду та організації колективної роботи. У цьому випадку весь локальний проєкт — як серверна частина на ASP.NET Core та Entity Framework Core, так і клієнтська на Blazor WebAssembly — був підготовлений до публікації на GitHub згідно з найкращими практиками індустрії.

Перш ніж почати безпосередньо завантаження, слід упевнитися, що в кореневому каталозі проєкту присутній файл .gitignore. У нього включені типові

шаблони для .NET-проєктів та Node-місту:

- bin/ і obj/ для всіх .NET-проектів;
- .vs/, .user, .suo та інші тимчасові файли Visual Studio;
- node_modules/, dist/ та package-lock.json (за потреби) для клієнтської частини;
- appsettings.Development.json і будь-які інші конфіденційні налаштування, які не слід публікувати.

Після того, як .gitignore налаштовано, виконуємо ініціалізацію нового Git-репозиторію в корені проєкту:

- git init

Ця команда створює прихований каталог .git/, у якому зберігатиметься вся історія змін.

Далі перші зміни додаємо в індекс та фіксуємо перший коміт:

- git add .
- git commit -m "Initial commit: базова структура застосунку на .NET 8 і Blazor"

Важливо давати коміт-повідомленням зміст, щоб їх легко було читати в історії — наприклад, «Додано контролери і моделі», «Реалізовано Pokédex-сервіс» тощо.

Наступним кроком створюємо віддалений репозиторій на GitHub. Після авторизації на сайті слід:

- натиснути “New repository”;
- задати ім’я (наприклад, FocusUp1-server чи єдине FocusUp1);
- переконатися, що репозиторій порожній і не містить шаблонного README.

Після створення на GitHub отримуємо URL у форматі https://github.com/EduardMartynenko/FocusUp1.git і додаємо його як віддалений «origin»:

- git remote add origin https://github.com/EduardMartynenko/FocusUp1.git

З’єднавши локальний та віддалений репозиторії, виконуємо команду пушу:

- git push -u origin master

Прапорець -u встановлює відстеження локальної гілки master (або main) гілкою origin/master. Відтепер при виконанні git push або git pull не потрібно

вказувати origin master кожного разу.

Для подальшої розробки рекомендується використовувати модель гілок:

- Основна гілка main/master містить завжди стабільний код, готовий до релізу.
- Для кожної нової функції чи виправлення створюється окрема гілка, наприклад, feature/pomodoro-reminder або fix/reminder-bug.
- Після завершення роботи над гілкою слід створити Pull Request у веб-інтерфейсі GitHub, до якого можна додати рев'ю колег, обговорення змін та автоматичні перевірки.

За бажанням можна налаштувати GitHub Actions для автоматичного складання та базового тестування при кожному Pull Request. Це забезпечує додатковий рівень захисту від помилок і дозволяє виявляти проблеми ще до злиття коду в основну гілку.

Окрім цього, GitHub пропонує інструменти для керування завданнями:

- Issues — для фіксації багів і запитів на нову функціональність;
- Projects — для візуального планування спринтів та організації завдань;
- Wiki — для документації архітектури, API та процесів розгортання.

Публікація проєкту на GitHub надає такі переваги:

- Співпраця. Інші розробники можуть легко клонувати репозиторій, створювати свої форки та пропонувати зміни.
- Прозорість. Історія комітів зберігає весь хід розробки, дозволяє відслідковувати, хто і коли вносив зміни.
- Безпека. Віддалений репозиторій надійно зберігається у хмарі GitHub, з можливістю налаштувати багатофакторну автентифікацію для доступу.
- Інтеграції. GitHub легко підключити до хмарних сервісів CI/CD, моніторингу та інших зовнішніх інструментів.

Таким чином, комплексний підхід до завантаження застосунку на GitHub — від коректного налаштування .gitignore, структурованих комітів і моделі гілок до впровадження CI/CD та менеджменту завдань через Issues/Projects — забезпечує надійний конвеєр розробки, який полегшує як індивідуальну, так і командну роботу над проєктом.

ВИСНОВКИ

Результатами виконаної роботи стало зменшення рівня прокрастинації шляхом розробки веб-застосунку анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями.

Було виконано наступні задачі:

– Проаналізовано предметну галузь та існуючі рішення: визначено природу прокрастинації та методи її подолання, описано цільову аудиторію, проведено огляд аналогів (Forest, RescueTime, Pomodoro To-Do, Focus Keeper) із виокремленням їхніх сильних і слабких сторін; на основі цього сформульовано вимоги до застосунку, зокрема: управління завданнями й підзадачами, Pomodoro-таймер із кнопками «Зупинити» та «Пропустити перерву», система бонусів і штрафів, адаптивні нагадування, модуль візуалізації статистики та швидкий інтерфейс.

– Обрано технічний стек: C# і .NET 8 як основну платформу, ASP.NET Core для реалізації RESTful API, Blazor WebAssembly для клієнтського UI, Entity Framework Core зі SQLite як ORM і вбудована БД, GitHub для контролю версій.

– Запроектовано багаторівневу N-tier архітектуру: Presentation Layer на Blazor WebAssembly, Business Logic Layer із сервісами TaskService, PomodoroService, ReminderService та SolverService, Data Access Layer із застосуванням патернів Repository і Unit of Work та Fluent API.

– Розроблено UML-діаграми: діаграма рівневої архітектури, діаграми прецедентів для ключових функцій, діаграма діяльності життєвого циклу завдання, діаграма станів, діаграма класів ApplicationDbContext і сутностей, діаграми послідовностей.

– Реалізовано клієнтську частину: створено Razor-компоненти (.razor) із локальною CSS-ізоляцією, організовано адаптивний дизайн за допомогою Flexbox і CSS Grid, інтегровано chart.js через JS-інтероп для побудови гістограм і лінійних

графіків.

– Реалізовано серверну частину: створено контролери для CRUD-операцій (TasksController, SubTasksController, PomodoroLogsController, RemindersController, BonusesController, PenaltiesController), налаштовано JWT-автентифікацію, глобальне логування Serilog і політику CORS; налаштовано ApplicationDbContext із Code-First міграціями та Seed-методами для наповнення тестових даних.

– Проведено функціональне та модульне тестування: перевірено коректність роботи контролерів і сервісів, валідацію даних, обробку виключень і транзакцій.

– Завантажено проєкт на GitHub: ініціалізовано Git-репозиторій, встановлено віддалений origin, виконано команди git add, git commit та git push; налаштовано базовий CI/CD для автоматизації побудови та тестування.

Отриманий застосунок демонструє ефективність комбінування гейміфікації, адаптивних нагадувань і візуалізації прогресу для боротьби з прокрастинацією та підвищення продуктивності користувачів.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Мартиненко Е.В., Гаманюк І.М. Використання діаграми предметної галузі для моделювання системи анти-прокрастинатора: Матеріали VI Міжнародної науково-технічної конференції «Сучасний стан та перспективи розвитку IoT», 15.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій (подано до друку).
2. Мартиненко Е.В. Гаманюк І.М. Використання діаграм прецедентів для моделювання функціональності веб-застосунку «Анти-прокрастинатор» // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез К.: ДУІКТ, 2025. С.43-45.

ПЕРЕЛІК ПОСИЛАНЬ

1. APA PsycNet. *APA PsycNet*. URL: https://psycnet.apa.org/doiLanding?doi=10.1037/bul0000221

2. Sirois F., Melia-Gordon M. L., & Pychyl T. A. (2021). Procrastination, stress, and chronic health conditions: A scoping review. *Stress and Health*, 37(2), 175–187. URL: https://doi.org/10.1002/smi.2998

3. Gamification and learning: Effects on motivation and engagement (Computers in Human Behavior, 2020) URL: https://www.sciencedirect.com/science/article/pii/S0747563220301977

4. Arm waving in stylophoran echinoderms: three-dimensional mobility analysis illuminates cornute locomotion - PMC. *PMC Home*. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7353985/

5. Behavior Change Support Systems for Self-Treating Procrastination: Systematic Search in App Stores and Analysis of Motivational Design Archetypes. *Journal of Medical Internet Research*. URL: https://www.jmir.org/2025/1/e65214

6. How does the Pomodoro Technique enhance productivity in remote working environments, and what studies back its effectiveness? Consider referencing research from psychology journals and productivity blogs like Harvard Business Review or Smarter Living. *Psicosmart- Software para recursos humanos, en la nube para empresas*. URL: https://psico-smart.com/en/blogs/blog-how-does-the-pomodoro-technique-enhance-productivity-in-remote-working-185621

7. Investigating Gamification to Reduce Procrastination - Systematic Literature Review | Journal on Interactive Systems. *Digital Open Library of the Brazilian Computing Society | Journals*. URL: https://journals-sol.sbc.org.br/index.php/jis/article/view/5412

8. Implementation Intention and Reminder Effects on Behavior Change in a Mobile Health System: A Predictive Cognitive Model. *Journal of Medical Internet Research*. URL: https://www.jmir.org/2017/11/e397/

9. View of StudyTracker Self-Tracking App and its Relationship to University Student Procrastination | Canadian Journal of Learning and Technology. *Canadian Journal of Learning and Technology*. URL: https://cjlt.ca/index.php/cjlt/article/view/28644/21158

10. Research on Impact of Work Gamification on Employee Performance | Francis Academic Press. *Francis Academic Press*. URL: https://francis-press.com/papers/14026

11. Blazor | Build client web apps with C# | .NET. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

12. ASP.NET Core Blazor. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0

13. Overview of Entity Framework Core - EF Core. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/ef/core/

14. Get started with Bootstrap. *Bootstrap · The most popular HTML, CSS, and JS library in the world*. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/

15. CSS grid layout - CSS: Cascading Style Sheets | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout

16. Basic concepts of flexbox - CSS: Cascading Style Sheets | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox

17. Dependency injection in ASP.NET Core. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-9.0

18. JSON Web Token Introduction - jwt.io. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction/

19. Grafiati: Оформити списки використаних джерел онлайн. *Grafiati: Оформити списки використаних джерел онлайн*. URL: https://www.grafiati.com/uk/

20. W3Schools.com. *W3Schools Online Web Tutorials*. URL: https://www.w3schools.com/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка розумного анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями

Виконав студент 4 курсу
групи ПД-42
Мартиненко Едуард Вікторович
Керівник роботи
Старший викладач кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – покращення боротьби з прокрастинацією шляхом впровадження розумного веб-застосунку анти-прокрастинатора з гейміфікацією та динамічними нагадуваннями.
- **Об’єкт дослідження** – процес прокрастинації та методи її подолання.
- **Предмет дослідження** – веб-застосунок для боротьби з прокрастинацією.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз існуючих методів боротьби з прокрастинацією та визначити їхні переваги та недоліки.
2. Описати основні підходи до розробки розумних нагадувань і гейміфікації у застосунках для підвищення продуктивності.
3. Здійснити огляд інструментів для розробки програмного забезпечення веб-застосунків на базі C# та Blazor.
4. Сформулювати вимоги до розробки клієнт-серверного застосунку для боротьби з прокрастинацією.
5. Спроєктувати архітектуру веб-застосунку для взаємодії з користувачем, що включає динамічні нагадування, таймери та гейміфікацію.
6. Розробити веб-застосунок з інтерактивними нагадуваннями, таймером Помодоро та елементами гейміфікації.
7. Провести тестування розробленого веб-застосунку для оцінки його ефективності в боротьбі з прокрастинацією.

3

АНАЛІЗ АНАЛОГІВ

Показник	Focus Booster	RescueTime	Trello	FocusUp
Наявність веб-платформи:	+	+	+	+
Персоналізовані нагадування	-	+	-	+
Інтерактивна мотивація	-	-	-	+
Можливість використання таймера Pomodoro	+	-	-	+
Аналіз продуктивності	-	+	+	+
Система бонусів та штрафів	-	-	-	+
Основна цільова аудиторія	Студенти, фрілансери	Люди, що аналізують продуктивність	Команди, менеджери	Студенти, фрілансери, працівники

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Додавання завдань та встановлення дедлайнів.
2. Нагадування про завдання через персоналізовані сповіщення.
3. Інтерактивна взаємодія з користувачем для мотивації.
4. Використання таймера Pomodoro для фокусування.
5. Система гейміфікації (бонуси, штрафи за виконання завдань).
6. Аналіз виконаних завдань та продуктивності.

Нефункціональні вимоги:

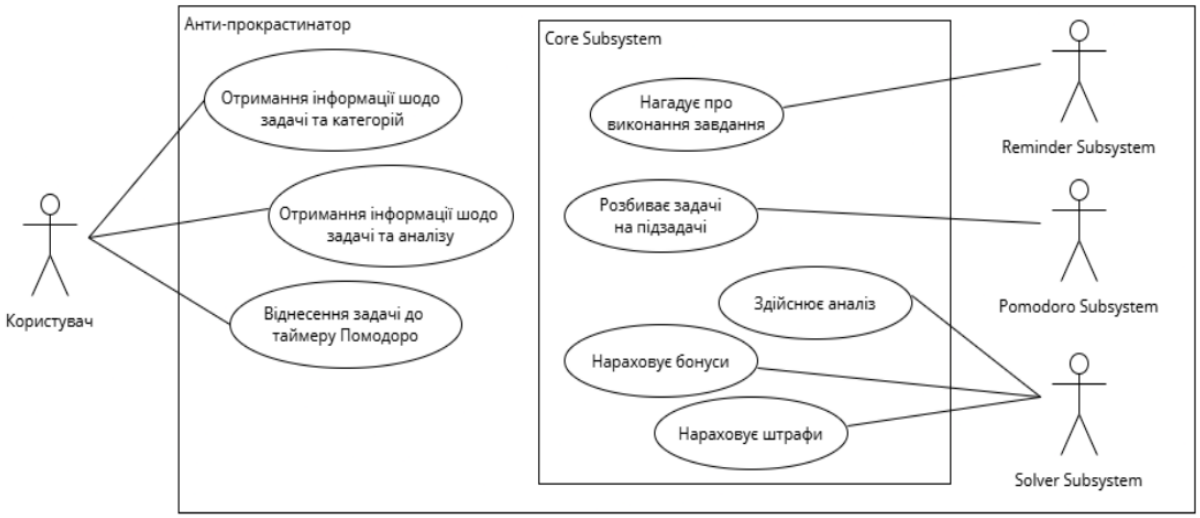
1. Мова інтерфейсу: українська
2. Швидкість роботи (продуктивність): вебсторінка повинна забезпечувати час відповіді не більше 3 секунд.
3. Портативність: Програма, що працює в Windows 10, має без змін функціонувати й в Windows 11.
4. Підтримка мобільних версій.
5. Захист даних користувачів, відповідність стандартам безпеки.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

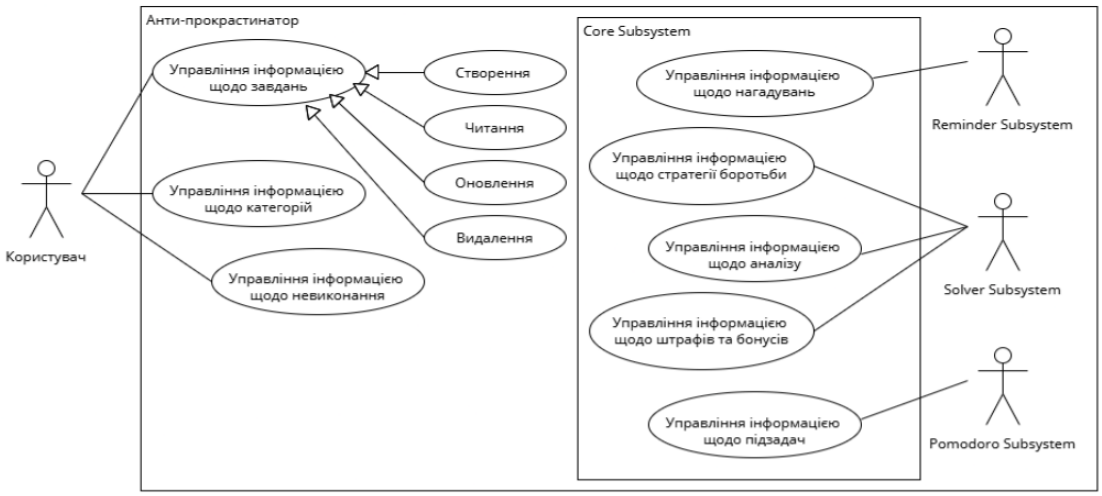
МОДЕЛЮВАННЯ ОСНОВНИХ ФУНКЦІЙ СИСТЕМИ



ДІАГРАМА ПРЕЦЕДЕНТІВ

7

МОДЕЛЮВАННЯ УПРАВЛІННЯ ІНФОРМАЦІЄЮ ТА ОПЕРАЦІЇ CRUD



ДІАГРАМА ПРЕЦЕДЕНТІВ

8

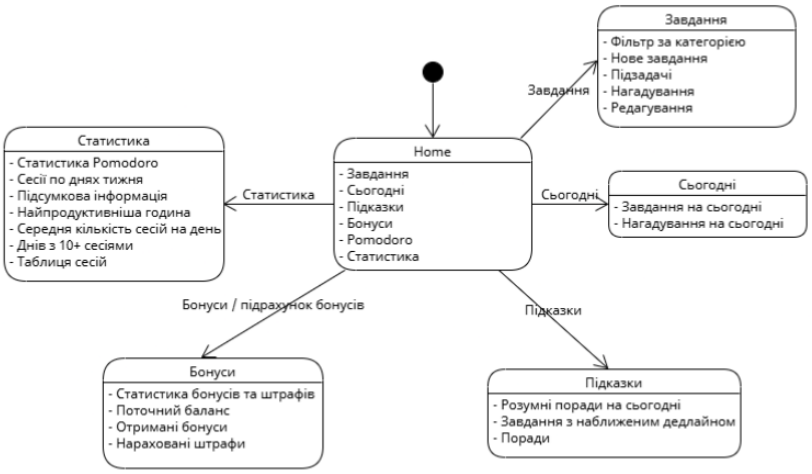
БІЗНЕС МОДЕЛЬ

ДІАГРАМА ПРЕДМЕТНОЇ ГАЛУЗІ

УПРАВЛІННЯ ЖИТТЄВИМ ЦИКЛОМ

ДІАГРАМА ДІЯЛЬНОСТІ

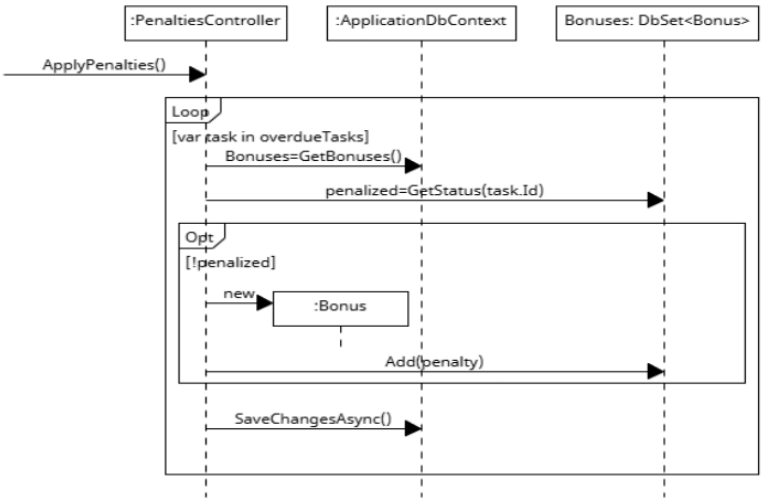
МОДЕЛЮВАННЯ ГОЛОВНОГО ІНТЕРФЕЙСУ ЗАСТОСУНКУ



ДІАГРАМА СТАНІВ

11

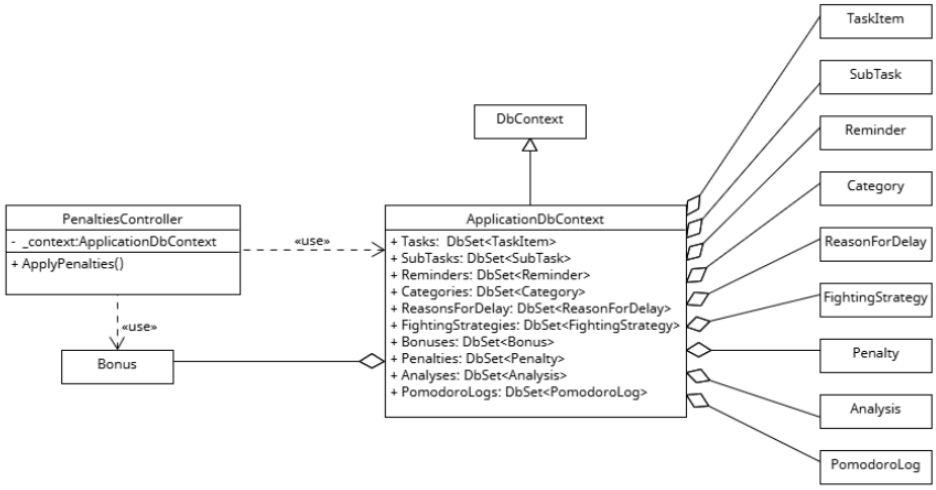
ПРОЄКТУВАННЯ МЕТОДУ СТВОРЕННЯ ШТРАФУ



ДІАГРАМА ПОСЛІДОВНОСТЕЙ

12

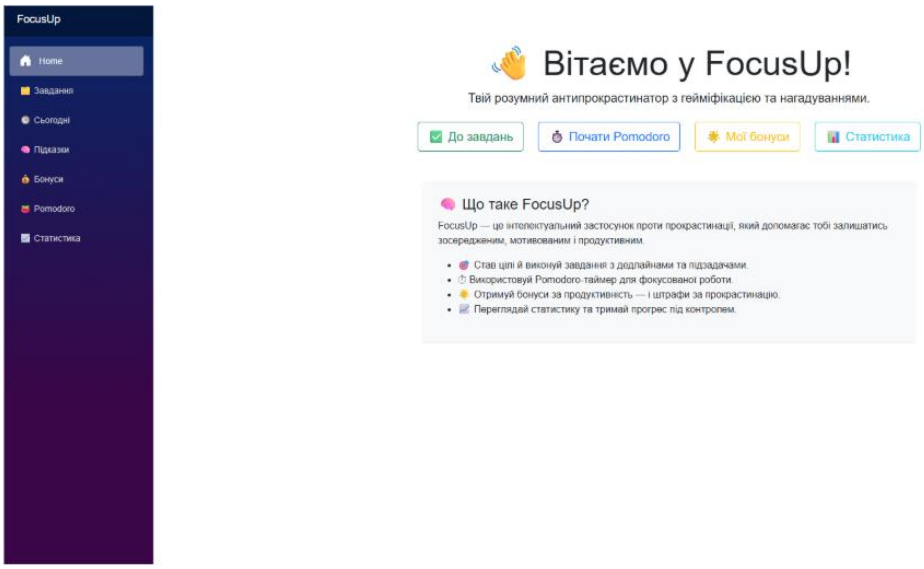
ПРОЄКТУВАННЯ ВЗАЄМОВІДНОШЕННЯ КЛАСІВ



ДІАГРАМА КЛАСІВ

13

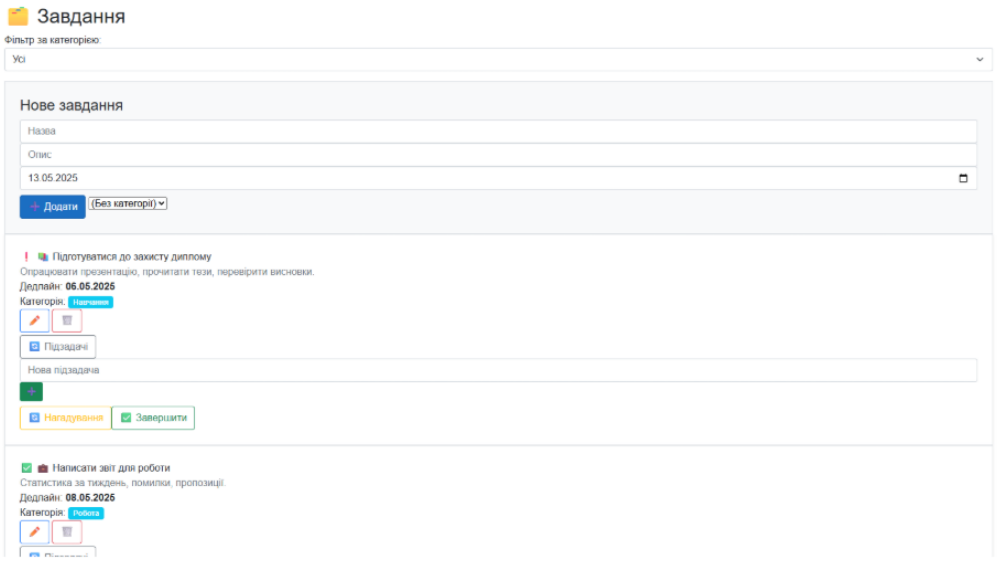
ЕКРАННІ ФОРМИ



Головна сторінка

14

ЕКРАННІ ФОРМИ



Сторінка із завданнями

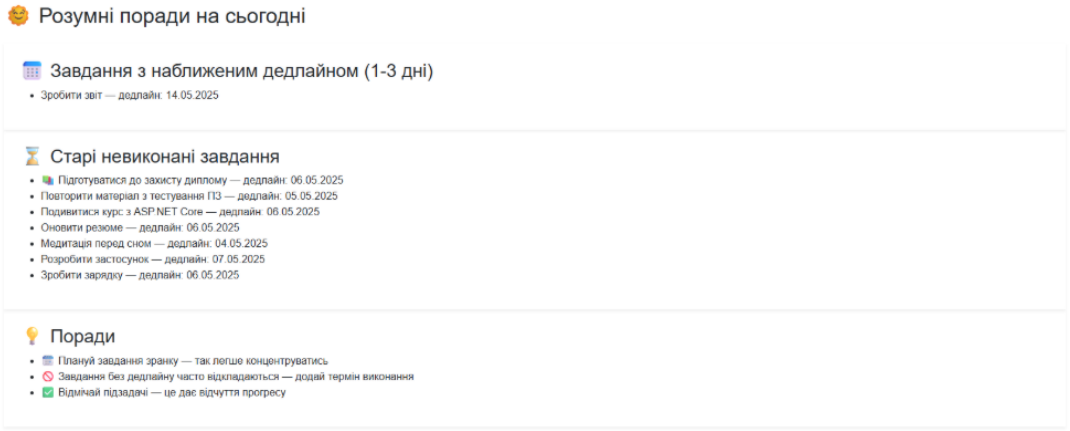
15

ЕКРАННІ ФОРМИ

Сторінка “Сьогодні”

16

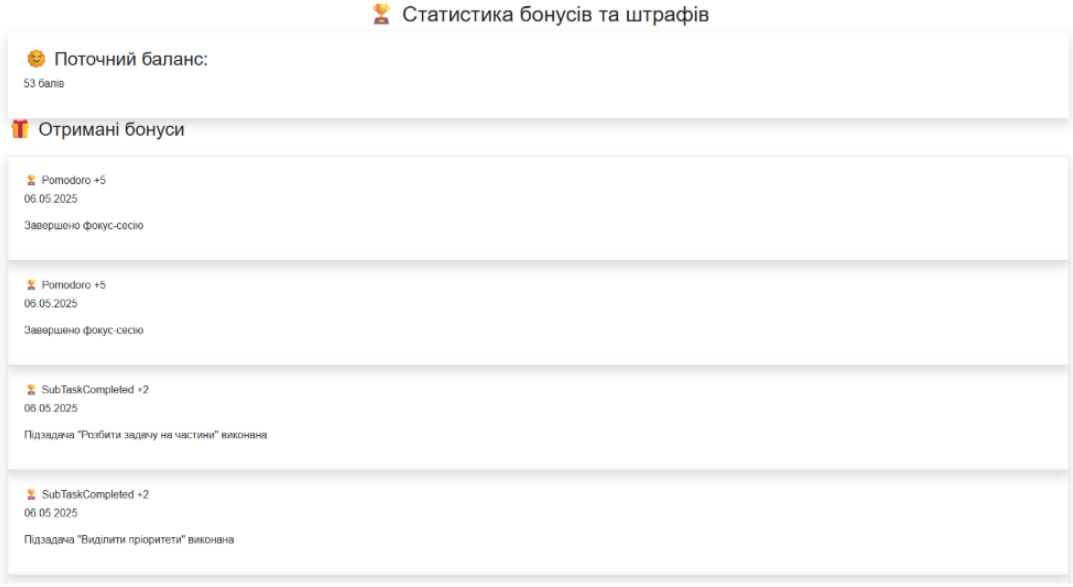
ЕКРАННІ ФОРМИ



Сторінка “Підказки”

17

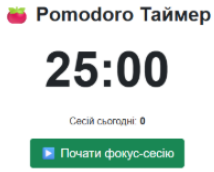
ЕКРАННІ ФОРМИ



Сторінка “Бонуси”

18

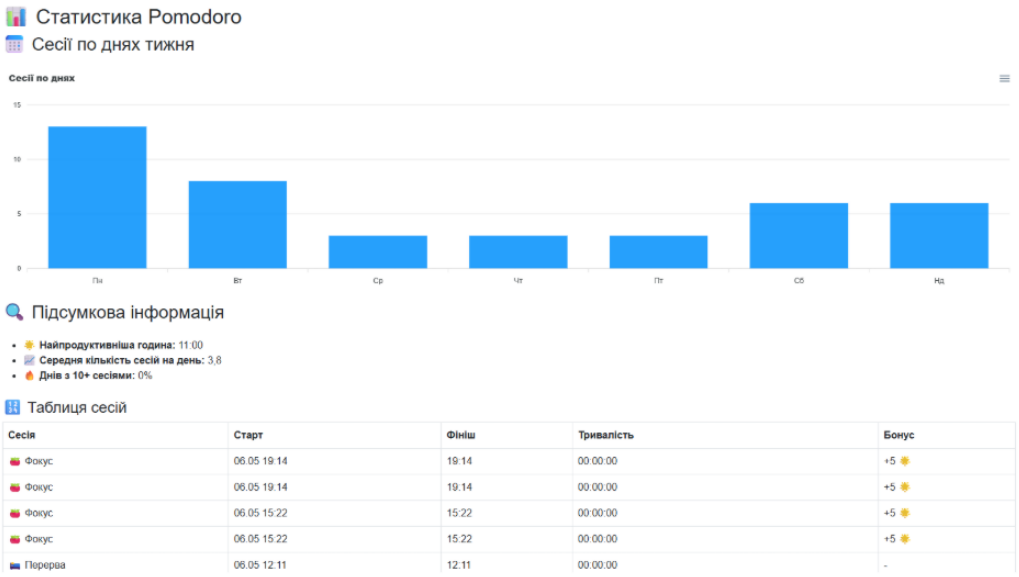
ЕКРАННІ ФОРМИ



Сторінка “Pomodoro”

19

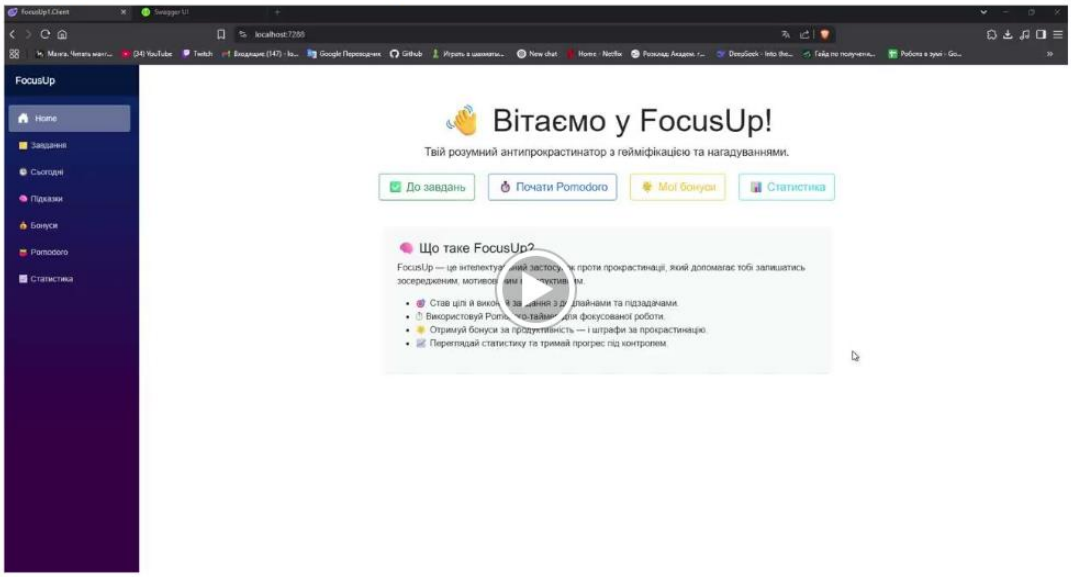
ЕКРАННІ ФОРМИ



Сторінка “Статистика”

20

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ



21

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Мартиненко Е.В., Гаманюк І.М. Використання діаграми предметної галузі для моделювання системи анти-прокрастинатора: Матеріали VI Міжнародної науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». 15.04.25. Державний університет інформаційно-комунікаційних технологій, Київ, Україна, 2025 (подано до друку).

2. Мартиненко Е.В. Використання діаграм прецедентів для моделювання функціональності веб-застосунку «Анти-прокрастинатор»: Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.25. Державний університет інформаційно-комунікаційних технологій, Київ, Україна, 2025. Збірник тез К.: ДУІКТ, С.42-44.

22

ВИСНОВКИ

1. Проведено аналіз аналогів. Визначено функціональність яку необхідно реалізувати.
2. Відпрацьовано функціональні та нефункціональні вимоги до застосунку.
3. Визначено інструменти для розробки застосунку: C#, ASP.NET Core, Blazor, SQlite, Visual Studio, Bootstrap.
4. Відпрацьовано моделі прецедентів та предметної галузі. Визначено бізнес модель.
5. Відпрацьовано моделі діяльності та станів. Визначено поведінку системи.
6. Здійснено апробацію досліджень з моделювання системи анти-прокрастинатора.
7. Реалізовано систему гейміфікації за допомогою бонусів і штрафів, що забезпечує додаткову мотивацію до виконання завдань.
8. Під час тестування застосунку було підтверджено його відповідність вимогам, зокрема щодо ефективності боротьби з прокрастинацією.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ СЕРВЕРНОЇ ЧАСТИНИ

Контролер завдань TasksController.cs

```
using Microsoft.AspNetCore.Mvc;
using FocusUp1.Server.Services;
using FocusUp1.Shared.Models;

namespace FocusUp1.Server.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class TasksController : ControllerBase
    {
        private readonly TaskService _taskService;

        public TasksController(TaskService taskService)
        {
            _taskService = taskService;
        }

        [HttpGet]
        public async Task<ActionResult<List<TaskItem>>> GetAll()
        {
            var tasks = await _taskService.GetAllAsync();
            return Ok(tasks);
        }

        [HttpPost]
        public async Task<ActionResult> Create(TaskItem task)
        {
            await _taskService.CreateAsync(task);
            return Ok();
        }

        [HttpPut("{id}")]
        public async Task<ActionResult> Update(int id, TaskItem task)
        {
            if (id != task.Id)
                return BadRequest();

            await _taskService.UpdateAsync(task);
            return NoContent();
        }

        [HttpDelete("{id}")]
        public async Task<ActionResult> Delete(int id)
        {
            await _taskService.DeleteAsync(id);
            return NoContent();
        }
    }
}
```

PomodoroLogsController.cs

```
using Microsoft.AspNetCore.Mvc;
```

```
using FocusUp1.Server.Services;
using FocusUp1.Shared.Models;
using System.Threading.Tasks;

namespace FocusUp1.Server.Controllers
{
    [ApiController]
    [Route("api/pomodorologs")]
    public class PomodoroLogsController : ControllerBase
    {
        private readonly PomodoroLogService _pomodoroLogService;

        public PomodoroLogsController(PomodoroLogService pomodoroLogService)
        {
            _pomodoroLogService = pomodoroLogService;
        }

        [HttpGet]
        public async Task<IActionResult> GetAllLogs()
        {
            var logs = await _pomodoroLogService.GetAllAsync();
            return Ok(logs);
        }

        [HttpPost]
        public async Task<IActionResult> AddLog(PomodoroLog log)
        {
            await _pomodoroLogService.AddAsync(log);
            return Ok();
        }
    }
}
```

RemindersController.cs

```
using Microsoft.AspNetCore.Mvc;
using FocusUp1.Server.Services;
using FocusUp1.Shared.Models;
using System.Threading.Tasks;

namespace FocusUp1.Server.Controllers
{
    [ApiController]
    [Route("api/reminders")]
    public class RemindersController : ControllerBase
    {
        private readonly ReminderService _reminderService;

        public RemindersController(ReminderService reminderService)
        {
            _reminderService = reminderService;
        }

        [HttpGet]
        public async Task<IActionResult> GetAllReminders()
        {
            var reminders = await _reminderService.GetAllAsync();
            return Ok(reminders);
        }

        [HttpPost]
        public async Task<IActionResult> CreateReminder(Reminder reminder)
```

```
        {
            await _reminderService.CreateAsync(reminder);
            return Ok();
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteReminder(int id)
        {
            await _reminderService.DeleteAsync(id);
            return NoContent();
        }
    }
}
```

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<TaskService>();
builder.Services.AddScoped<PomodoroLogService>();
builder.Services.AddScoped<ReminderService>();
builder.Services.AddScoped<BonusService>();
builder.Services.AddScoped<PenaltyService>();
builder.Services.AddScoped<SubTaskService>();

var app = builder.Build();

app.MapControllers();

app.Run();
```