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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для підвищення ефективності роботи програмістів»

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

_____ Михайло СКОВПЕНЬ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42
Михайло СКОВПЕНЬ

Керівник: Андрій КОЛОДЮК
ассистент кафедри ІПЗ

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

_____Сковпень Михайлу Анатолійовичу_____

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для підвищення ефективності роботи програмістів»
керівник кваліфікаційної роботи асистент кафедри ІПЗ Андрій КОЛОДЮК, 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: оптимізація особистої продуктивності користувача за рахунок використання методів тайм-менеджменту (Pomodoro) та зберігання короткострокових нотаток.

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

1. Огляд та аналіз існуючих підходів до цифрової самоорганізації та вивчити ефективність методики Pomodoro в контексті особистої продуктивності.
2. Проектування web-застосунку для підвищення ефективності роботи програмістів.

3. Програмна реалізація та опис функціонування web-застосунку для підвищення ефективності роботи програмістів
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів.
6. Діаграма діяльності.
7. Діаграма послідовності.
8. Діаграма компонентів.
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	05.03-12.03.2025	
3	Огляд існуючих рішень застосунків для підвищення продуктивності	13.03-17.03.2025	
4	Проектування веб-застосунку для підвищення ефективності роботи програмістів	18.03-23.03.2025	
5	Програмна реалізація веб-застосунку	24.03-24.04.2025	
6	Тестування застосунку	25.04-28.04.2025	
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 _____ Михайло СКОВПЕНЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Андрій КОЛОДЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 1 табл., 20 рис., 20 джерел.

Мета роботи – підвищення продуктивності програмістів шляхом ефективного управління часом, ведення нотаток та візуального відстеження прогресу.

Об'єкт дослідження – процес управління робочим часом за допомогою цифрових засобів.

Предмет дослідження – веб-застосунок для організації робочого часу на основі методології Pomodoro з додаванням функціоналу для підвищення мотивації, контролю за результатами та нотатками.

Короткий зміст роботи: В роботі проаналізовано методики організації робочого часу та підвищення ефективності роботи програмістів з використанням Pomodoro-техніки. Проаналізовано інструментальні засоби для реалізації Pomodoro-систем: Forest, Focus To-Do, Pomofocus, Minimalist Pomodoro Timer.

Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: запуск Pomodoro-таймера, керування його станом, ведення нотаток, перегляд мотиваційних цитат, отримання статистики завершених сесій.Проведено функціональне та модульне тестування додатку.

В роботі використано FastAPI для створення API, SQLite для зберігання даних, React і Tailwind CSS для створення користувацького інтерфейсу.

Сферою використання застосунку є підвищення ефективності робочого часу програмістів та допомога в плануванні і контролі особистої продуктивності

КЛЮЧОВІ СЛОВА: POMODORO, ТАЙМ-МЕНЕДЖМЕНТ, ПРОДУКТИВНІСТЬ, FASTAPI, REACT, НОТАТКИ, МОТИВАЦІЯ, ВЕБ-ЗАСТОСУНОК, ТАЙМЕР, СТАТИСТИКА СЕСІЙ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

JWT – JSON Web Token

PWA – Progressive Web Application

UI – User Interface

UX – User Experience

API – Application Programming Interface

REST – Representational State Transfer

CRUD – Create, Read, Update, Delete

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

SPA – Single Page Application

SQL – Structured Query Language

CI/CD – Continuous Integration / Continuous Deployment

SPA – Single Page Application

JSON – JavaScript Object Notation

TS – TypeScript

DOM – Document Object Model

ВСТУП

Актуальність: У сучасному світі програмування та інформаційних технологій ефективне управління часом є критичним фактором успіху для фахівців багатьох галузей. Через високу інтенсивність розумової праці програмісти часто стикаються з труднощами концентрації та втратою продуктивності. Тому використання таких методів управління часом, як метод Pomodoro, у поєднанні з сучасними цифровими інструментами допомагає покращити концентрацію, підвищити ефективність та зменшити вигорання.

З огляду на це, створення веб-додатку, який поєднує таймер Pomodoro, зручний інтерфейс для створення нотаток, візуалізацію статистики та супровід у вигляді музики та мотиваційних цитат, є надзвичайно актуальним. Такий інструмент може стати незамінним помічником для програмістів, які хочуть раціонально організувати свій робочий процес.

Об'єктом дослідження є ефективність процесу управління робочим часом програмістів з допомогою цифрових засобів.

Предметом дослідження є програмне забезпечення, котре допомагатиме в організації робочого часу програмістам і буде побудовано на основі методології Pomodoro з додаванням функціоналу для підвищення мотивації, контролю за результатами та нотатками.

Метою цієї кваліфікаційної роботи є створення веб-застосунку для поліпшення продуктивності програмістів, який надає можливості для ефективного управління часом, ведення нотаток, візуального відстеження прогресу та створює комфортне і просте в розумінні середовище для користувача завдяки вбудованій музиці й цитатам.

Методи дослідження: в процесі написання даної роботи було проаналізовано аналогічні рішення, а саме інших веб-застосунків для підвищення продуктивності які використовують методику Pomodoro. Також проведено формулювання загального уявлення про саму Pomodoro техніку і визначення початкових

функціональних і нефункціональних вимог до програмного забезпечення. Наступним кроком проведено проектування архітектури клієнт-серверної взаємодії і реалізація функціональності за допомогою стеку технологій: React.js, Flask, SQLite та Chart.js. Під час реалізації проводились тестування клієнтської та серверної частин разом з візуалізацією архітектури веб-застосунку.

Наукова новизна роботи: Розроблений застосунок поєднує класичний підхід тайм-менеджменту з сучасним користувацьким досвідом, включаючи графічну аналітику за сесіями, мотиваційні підказки, аудіосупровід та зручну роботу з нотатками. Програмне забезпечення створено як SPA (Single Page Application), що дозволяє досягти високої швидкості взаємодії користувача з системою.

Практична значущість полягає в тому, що розроблений застосунок підійде для більшості фахівців з різних професій. Завдяки його мультиплатформенності та візуальній простоті він може стати частиною щоденної практики.

1 АНАЛІЗ ВЕБ-ЗАСТОСУНКУ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОБОТИ З ВИКОРИСТАННЯМ POMODORO-ТАЙМЕРА ТА НОТАТОК

1.1 Веб-застосунки для ефективної організації робочого часу.

У сучасному цифровому суспільстві з дедлайнами, що стискають час, та багатозадачністю, що стала нормою, гостро постає запит на інструменти для організації робочого дня в структурований спосіб.

Одним із найефективніших способів підвищення продуктивності вважається використання спеціалізованих веб-застосунків. Ці застосунки створюють умови для кращого планування діяльності користувачів, забезпечують запобігання перевантаження, адекватне переключення між різними завданнями та зменшення негативного впливу від прокрастинації. Через це ідеально підійдуть інструменти для менеджменту часу, інтеграція з календарями, всебічні замітки, віджети візуалізації статистики зробленої роботи.

Застосунки, в яких закладена методика Pomodoro, займають особливе місце в контексті цього опису. Вони допомагають користувачу сфокусувати увагу на одному завданні протягом строго відведеного часу. Цей підхід ґрунтується на циклічному чергуванні коротких робочих сесій із суворо обов’язковими перервами для збереження концентрації уваги, запобігання втоми та підвищення рівня продуктивності.

Основні переваги веб-застосунків для організації часу:

– Доступність: можливість використання з будь-якого пристрою, що підтримує веб-браузер.

– Гнучкість: індивідуальне налаштування інтерфейсу та функціоналу відповідно до потреб користувача.

– Інтерактивність: інтеграція різноманітних інструментів (таймер, нотатки, музика) в єдиному середовищі.

– Мотивація: відстеження прогресу та візуалізація досягнень для підтримки залученості користувача.

Недоліки таких застосунків можуть включати залежність від інтернет-з’єднання та обмежену функціональність у порівнянні зі спеціалізованими десктопними програмами. Однак розроблений застосунок компенсує ці недоліки за рахунок локального збереження даних (у випадку авторизації) та оптимізованого інтерфейсу.

Враховуючи популярність самої методики, в Інтернеті існує чимало онлайн-інструментів, що реалізують Pomodoro-таймери, і деякі пропонують додаткові функції, такі як облік часу, гейміфікація, інтеграція з іншими сервісами або збереження задач.

Проте більшість систем має недостатній набір функцій: у більшості відсутні
– гнучкі налаштування сесій.

– індивідуалізація процесу, через відсутність якої іноді складно аналізувати результати.

– системи мотивації або геймифікації, які в свою чергу могли б замотивувати користувача.

Ці недоліки власне й стали передумовою до розробки власного застосунку.

1.2 Особливості методики Pomodoro та її мета

Pomodoro - найпопулярніша техніка управління часом у світі. Її запропонував наприкінці 1980-х років італійський розробник Франческо Чірілло. Техніка названа на честь червоного кухонного таймера, який Чірілло використовував під час навчання [1]. Сьогодні техніка Помодоро вважається ефективною як для самостійного навчання, так і для професійної роботи, зокрема в IT. Здатність швидко переключати фокус є важливою для багатозадачності.

При використанні цього методу, багатоденна робота над фокусом включає в себе 25-хвилинні робочі інтервали з 5-хвилинними перервами. Після чотирьох таких сесій зробіть тривалу перерву приблизно на 15-30 хвилин. Завдяки такому

підходу можна уникнути втоми, забезпечити максимальну концентрацію, обмежити багатозадачність і підвищити продуктивність.

Ключові переваги техніки Pomodoro:

– Зменшення стресу. Встановлення чітких меж часу дозволяє не перевантажуватись і уникати виснаження.

– Фокусування на одній задачі. Користувач зосереджується лише на одному завданні впродовж сесії.

– Полегшення початку роботи. Короткий проміжок роботи (25 хвилин) здається легшим і мотивує до старту.

– Оцінка результативності. Кількість завершених “помідорок” слугує показником досягнень протягом дня.

– Розвиток самодисципліни. Впровадження Pomodoro допомагає користувачу структурувати день без зовнішнього контролю.

Мета методики полягає в підвищенні концентрації та покращенні якості робочого процесу. Використання таймера Pomodoro у веб-застосунку дозволяє автоматизувати цей процес, надаючи користувачу зручний інструмент для слідування методиці [2].

У розробленому застосунку реалізовано класичний таймер Pomodoro з налаштуванням тривалості робочих фаз та перерв, а також лічильником завершених циклів. Це допомагає користувачам ефективно організовувати свій час і контролювати продуктивність.

1.3 Цільова аудиторія веб-застосунку

Будь-який розробник програмного забезпечення повинен враховувати певну групу користувачів для яких буде корисний його проєкт. Для даного веб-додатку цільовою аудиторією є:

– Програмісти та ІТ-фахівці. Часто працюють над складними проєктами, що вимагають глибокої концентрації, а тому Pomodoro — ідеальне рішення для підвищення ефективності.

– Студенти. Особливо ті, хто готується до іспитів, працює над дипломами чи виконує лабораторні завдання. Вони потребують системи, яка стимулює до навчання.

– Фрилансери. Особи, які працюють віддалено, часто відчувають складнощі з самоорганізацією та дисципліною.

– Творчі спеціалісти. Дизайнери, копірайтери, маркетологи, які чергують натхнення з продуктивною роботою.

– Люди, які займаються саморозвитком, наприклад, вивчають мови, розвивають навички або проходять онлайн-курси.

Однією з особливостей додатку є його всеохоплюючий характер - не обмежуючи користувача простим таймером, він також дозволяє створювати прості і зручні нотатки, відстежувати прогрес та отримувати мотиваційні повідомлення і доступ до музики, щоб підбадборювати користувача. Таким чином, він є надзвичайно корисним ресурсом як для досвідчених професіоналів, так і для новачків в особистій самоорганізації.

1.4 Аналіз існуючих рішень

У сучасному цифровому просторі функціонує велика кількість веб-застосунків, створених для реалізації методики Pomodoro. Ці застосунки істотно відрізняються між собою як за функціональністю, так і за дизайном, масштабованістю, підтримкою аналітики, наявністю додаткових сервісів тощо.

У межах цього дослідження було здійснено ґрунтовний аналіз найвідоміших рішень, що поєднують або спеціалізуються на техніці Pomodoro. Далі детально розглянуто кожен з них.

1.4.1 Forest

"Forest" — це мобільний застосунок, який став одним із найвідоміших прикладів гейміфікації Pomodoro-методу. Його основна концепція полягає у тому, що під час фокусованої роботи користувач «висаджує дерево». Якщо таймер

обривається достроково, дерево гине. Користувач таким чином мотивується не переривати сесію.

Функції Forest:

– Вбудований таймер Pomodoro з автоматичним перемиканням між роботою та перервами;

– Гейміфікована система нагород (віртуальний ліс);

– Історія фокус-сесій, щоденна статистика;

– Синхронізація з обліковим записом користувача;

– Можливість «вирощування» реальних дерев через партнерську програму.

Переваги:

– Мотивація через гейміфікацію: унікальний механізм вирощування дерев стимулює користувача не переривати сесію, що особливо ефективно для людей, які схильні відволікатися;

– Візуальна привабливість: приємна графіка та звуковий супровід роблять застосунок емоційно привабливим;

– Екологічна ініціатива: за накопичені «монети» користувач може сприяти висадженню реальних дерев;

– Доступність на iOS та Android: можливість використовувати додаток на будь-якому мобільному пристрої.

Недоліки:

– Відсутність веб-версії: для користувачів, які працюють на ПК, Forest не надає нативної браузерної підтримки;

– Відсутність підтримки нотаток або інтеграції з Task Manager: не можна прив'язати сесію до конкретного завдання;

– Обмежені можливості персоналізації: тривалість сесій не можна гнучко змінити;

– Обмеження у безкоштовній версії: частина функцій недоступна без придбання повної версії.

Рис. 1.1 Приклад інтерфейсу додатку Forest на платформі Ios

1.4.2 Focus To-Do

Focus To-Do — це багатоплатформний застосунок, який поєднує Pomodoro-таймер і повноцінний планувальник завдань. Його структура базується на списках справ, які можна групувати за проєктами. Кожне завдання може бути призначене на певний день, мати пріоритет, коментарі та нагадування.

Функції Focus To-Do:

– Pomodoro-таймер із налаштовуваними інтервалами;
– Створення та управління завданнями та підзадачами;
– Система тегів, пріоритетів, календарних прив’язок;
– Детальна статистика: звіти, графіки, аналітика;
– Синхронізація між пристроями, хмарне збереження даних.

Переваги:

– Інтеграція таймера та завдань: користувач працює безпосередньо з задачами, відстежуючи час для кожної з них;

– Аналіз продуктивності: можливість оцінити, скільки часу витрачено на конкретні завдання або проєкти;

– Гнучке налаштування таймера: користувач може змінювати тривалість роботи, коротких та довгих перерв;

– Доступність на мобільних і десктопних платформах.

Недоліки:

– Складний інтерфейс для новачків: велика кількість функцій ускладнює освоєння застосунку;

– Відсутність мотиваційних елементів: жодних цитат, музики чи геймифікації не передбачено;

– Відсутність підтримки нотаток у форматі текстових блоків: нотатки до завдань обмежені текстовим полем без форматування;

– Частина функцій доступна лише у платній версії.

Рис. 1.2 Приклад інтерфейсу таймеру додатку Focus To-Do

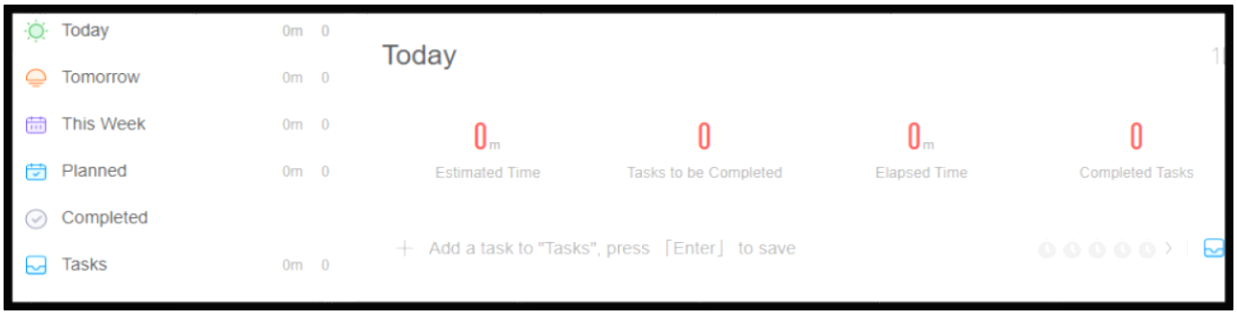


Рис. 1.3 Приклад інтерфейсу статистики додатку Focus To-Do

1.4.3 Pomofocus

Pomofocus — це веб-застосунок із лаконічним дизайном, що забезпечує базову реалізацію Pomodoro-методу. Він не потребує реєстрації та відкривається у будь-якому браузері.

Функції Pomofocus:

– Таймер із налаштовуваною тривалістю роботи та перерв;

– Вибір типу сесії: "Робота", "Коротка перерва", "Довга перерва";

– Список завдань з назвами та відмітками виконання;

– Темна та світла тема інтерфейсу.

Переваги:

– Мінімалізм: дуже простий у використанні, підходить для початківців та людей які шукають не складний веб-застосунок;

– Безкоштовність: не потребує оплати чи підписок;

– Доступність без реєстрації: миттєвий запуск у браузері.

Недоліки:

– Відсутність збереження даних: при оновленні сторінки втрачається історія виконаних завдань;

– Немає мотиваційних функцій: жодних аудіо, цитат чи візуального супроводу;

– Не підтримує синхронізацію між пристроями;

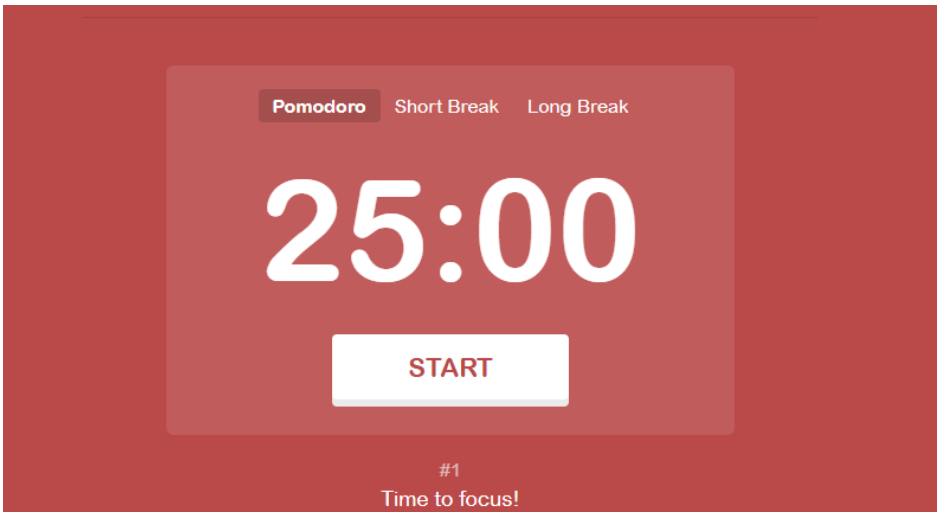


Рис. 1.4 Приклад інтерфейсу сторінки Pomofocus

1.4.4 Minimalist Pomodoro Timer

Minimalist Pomodoro Timer – є веб-застосунком який подає дуже простий але в свою чергу доволі функціональний Pomodoro метод. На відміну від багатьох своїх більш популярних аналогів, цей інструмент не має жодних додаткових функцій, таких як електронні листи завдань чи візуалізація продуктивності.

Натомість він цілком сфокусований на підтримці виконання тайм-циклів Pomodoro. Цей підхід знаходить відгук у користувачів, що цінують простоту, швидкодію й абсолютний мінімалізм інтерфейсу.

Функції Minimalist Pomodoro Timer:

– Прості помодоро-таймери з класичними інтервалами 25/5/15;

– Аудіо-повідомлення про закінчення сесії;

– Інтерфейс без реєстрації, реклами чи відволікаючих елементів.

Переваги:

– Максимальна простота: програма запускається миттєво, і до її інтерфейсу не потрібно вчитися;

– Швидкодія: сторінка відкривається за кілька секунд;

– Приватність: відсутність облікових записів та хмарного збереження гарантує збереження всіх даних лише у руках користувача;

– Гнучке налаштування таймера: хоч і реалізовано все у вигляді мінімалізму, користувач може за бажанням налаштувати тривалість усіх видів сесій.

Недоліки:

– Відсутність додаткових функцій: Не можна додавати або структурувати завдання, записувати замітки, дивитися статистику чи історію;

– Відсутність мотивації та гейміфікації: ніяких цитат, нагород;

– Немає синхронізації чи адаптації до мобільних платформ: інтерфейс хоч і адаптивний, не оптимізований для професійного використання на смартфонах.

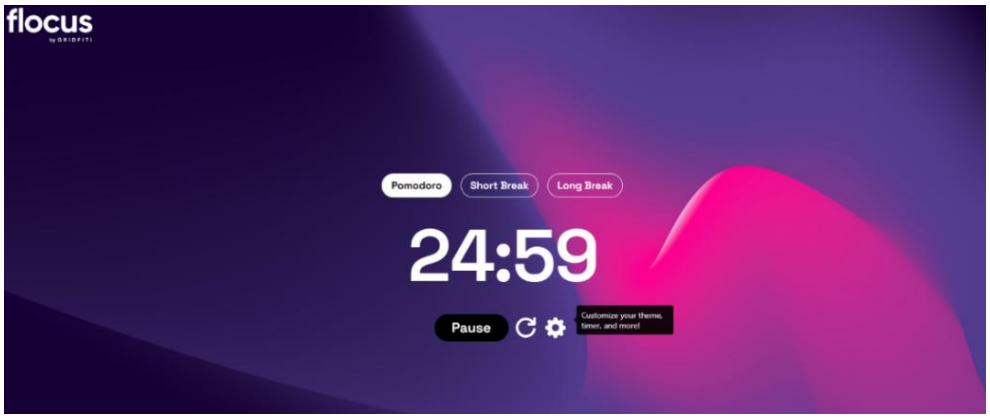


Рис. 1.5 Приклад сторінки таймеру Minimalist Pomodoro Timer

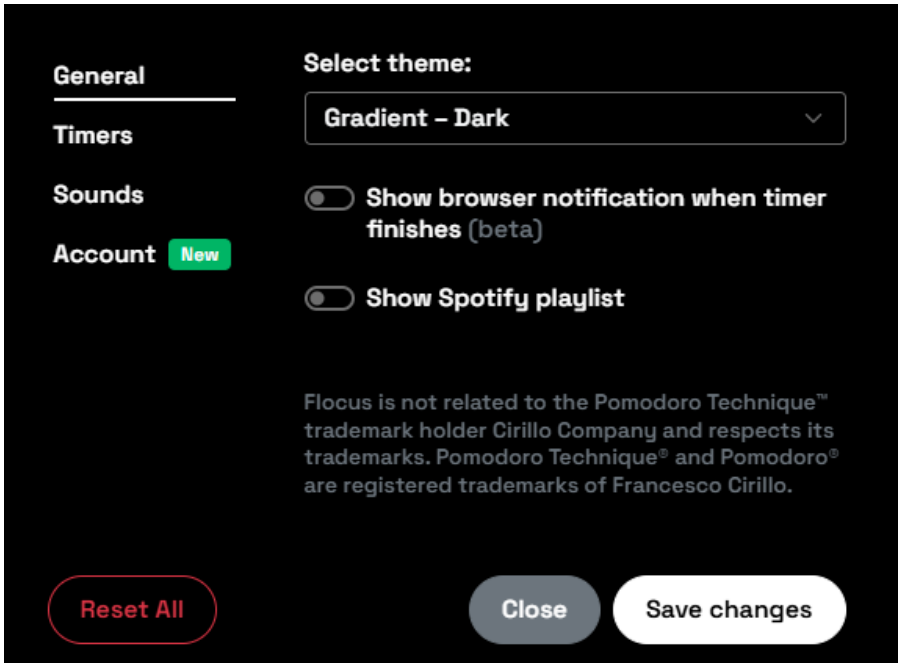


Рис. 1.6 Приклад сторінки персоналізації Minimalist Pomodoro Timer

В таблиці 1.1 буде наведено визначені рельтати аналізу існуючих аналогів веб-застосунків, спрямованих на роботу з Pomodoro-методом.

Таблиця 1.1

Зведена таблиця аналізу існуючих аналогів веб-застосунків, спрямованих на роботу з Pomodoro-методом.

Показник	**Forest**	**Focus To-Do**	**Pomofocus**	**Minimalist Pomodoro**
Таймер	Фази 25/5 без налаштувань. Візуалізація дерева	Налаштування інтервалів циклів	Звичайний таймер з сесіями	Фази 25/5 з довгою перервою додатково
Додання задач/нотаток	Відсутні задачі	Керування задачами та нотатками	Лише назви завдань	Відсутні
Мотиваційні елементи	Віртуальні дерева та візуальне заохочення	Відсутні	Відсутні	Відсутні
Аудіо	Відсутнє	Відсутнє	Відсутнє	Є інтеграція з Spotify та звукові сповіщення
Аналітика	Щоденна статистика у вигляді графіка.	Статистика завдань, трекінг часу.	Відсутня	Відсутня
Інтерфейс	Мобільний, візуально детальний	Складний інтерфейс з великою кількістю вкладок	Трохи урізаний але зручний	Мінімалістичний та базовий але зрозумілий користувачу

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих аналогів веб-застосунків, спрямованих на роботу з Pomodoro-методом.

Показник	**Forest**	**Focus To-Do**	**Pomofocus**	**Minimalist Pomodoro**
Платформа	Мобільні пристрої (iOS, Android)	Веб, десктоп, мобільні додатки	Веб	Веб
Персоналіза-ція	Відсутня	Присутні налаштування як таймера так і завдань (нотаток)	Мінімальні налаштування інтервалу	Таймер можна налаштувати індивідуально
Цільова аудиторія	Студенти, користувачі мобільних пристроїв	Програмісти, студенти, підходить широкому колу користувачів.	Користувачі, які хочуть швидкий веб-інструмент	Користувачі, яким потрібен максимальний мінімалізм
Особливості	Гейміфікація, еко-ініціатива	Інтеграція календаря	Доступність без реєстрації.	Відсутність жодних відволікаючих елементів
Недоліки	Відсутність браузерної версії, неможливість додавання завдань	Надмірна складність, реклама в безкоштовній версії	Відсутність збереження, історії, жодних додаткових функцій	Відсутність статистики та обмеження безкоштовного функціоналу.

1.5 Визначення вимог до продукту

Після аналізу наявних рішень, особливостей цільової аудиторії та методології Pomodoro можна сформулювати комплекс вимог до функціоналу, архітектури та інтерфейсу веб-застосунку PomodoroStickNotes. Ці вимоги відображають очікування користувачів, враховують практики UX/UI, а також відповідають сучасним технічним стандартам.

Функціональні вимоги:

– Pomodoro-таймер з чітко визначеними фазами роботи (25 хв), короткої перерви (5 хв) та довгої перерви (15 хв), з можливістю налаштування тривалості кожного інтервалу;

– Автоматичне перемикання між фазами (робота → коротка перерва → робота → довга перерва) для зменшення залученості користувача в технічне керування процесом;

– Система збереження нотаток, яка дозволяє створювати, редагувати та видаляти нотатки, з можливістю збереження їх на сервері;

– Реалізація реєстрації та логіну користувача для забезпечення безпеки особистих даних;

– Мотиваційна функція у вигляді цитат;

– Аудіоплеєр, що дає змогу відтворювати музику під час сесій;

– Додання функції зміни теми сайту з тесної на світлу для забезпечення комфорту користувача;

– Адаптивний дизайн інтерфейсу, який забезпечує коректне відображення елементів на пристроях з різними розмірами екранів (ПК, планшет, смартфон);

Нефункціональні вимоги:

– Час відгуку інтерфейсу не повинен перевищувати 2 секунд на всіх етапах взаємодії;

– Висока доступність — застосунок повинен працювати стабільно у сучасних браузерах (Chrome, Edge, Safari);

– Безпека збереження даних, зокрема нотаток користувача, без втрати інформації після перезавантаження сторінки;

Саме ці визначені вимоги будуть відповідати запитам цільової аудиторії та конкурувати з наявними аналогами.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Вибір середовища розробки

Правильно підібране середовище розробки значною мірою визначає комфорт і ефективність роботи над програмним проєктом. Воно забезпечує зручність написання, перевірки, відлагодження і запуску програмного коду в єдиному середовищі.

Для розробки веб-застосунку було використано кілька середовищ, зокрема Visual Studio Code — як універсальне середовище для обох частин проєкту (frontend та backend).

2.1.1 Visual Studio Code

Visual Studio Code - це легкий, швидкий і розширюваний редактор коду, який став стандартом де-факто серед веб-розробників. Незважаючи на те, що формально він не є повноцінним IDE, функціональність VS Code можна розширити до потрібного рівня за допомогою плагінів. Для розробляємого проекту було використано велику кількість таких розширень, включаючи підтримку TypeScript, React, Python, FastAPI, TailwindCSS та багато іншого.

Рішення працювати з VS Code було прийнято з кількох причин:

– підтримка одночасної роботи з декількома мовами (TypeScript, Python);

– інтеграція з Git і терміналом, що зручно для контролю версій і запуску серверної частини;

– кросплатформеність - можна працювати на Windows, Linux або macOS;

– гнучке налаштування середовища під специфіку проекту.

В результаті VS Code став прекрасним рішенням: легкий, швидкий і достатньо потужний, щоб реалізувати повний цикл розробки веб-додатку з клієнтом, сервером і базою даних.

2.2 Мови програмування

Успішна реалізація проєкту вимагала поєднання декількох мов програмування, кожна з яких відповідала за свій рівень логіки та обробки даних. Для цього були використані Python на стороні сервера та TypeScript на стороні клієнта. Такий розподіл був обраний не випадково, а виходячи з практичних міркувань: Python ідеально підходить для побудови REST API завдяки своїй простоті та великій кількості бібліотек, а TypeScript значно підвищує надійність клієнтського коду в динамічних інтерфейсах.

Таке розділення дозволило забезпечити чітку архітектуру застосунку з незалежною логікою та відповідальністю на кожному рівні.

2.2.1 Python

Python — це високорівнева, інтерпретована мова програмування, яка відзначається простим синтаксисом, читабельністю та широкими можливостями. Створена у 1991 році, вона сьогодні є однією з найпопулярніших мов у світі. Python підтримує кілька парадигм — процедурну, об'єктно-орієнтовану, функціональну — що дозволяє ефективно вирішувати завдання різного типу. Python використовувався для реалізації серверної частини, і його вибір був зумовлений багатьма практичними перевагами [5].

Основною перевагою стале те, що ця мова дозволяє дуже швидко створити прототип API, особливо при використанні сучасного фреймворку FastAPI, який підтримує асинхронність, автоматичну генерацію документації та просту інтеграцію з базою даних через ORM [6].

Основні завдання, що були вирішені за допомогою Python:

- обробка HTTP-запитів (GET, POST, PUT, DELETE);
- взаємодія з базою даних через ORM SQLAlchemy;
- підвантаження мотиваційних цитат;
- серіалізація об'єктів у JSON;
- підтримка асинхронних викликів для покращення продуктивності.

Завдяки своїй простоті та широкому екосередовищу Python дозволив реалізувати гнучку, розширювану і масштабовану серверну частину, яка може легко адаптуватися до нових функцій у майбутньому.

2.2.2 TypeScript

Клієнтська частина веб-застосунку була реалізована на основі *TypeScript* — мови, що є типізованим надбудовою над JavaScript. Використання саме TypeScript, а не класичного JavaScript, було принциповим рішенням для проєкту, який має масштабуватися, містити багато станів і взаємодій.

Переваги TypeScript в ході розробки стало те, що його типізація значно зменшила ризик помилок проєкту, автодоповнення покращувало процес паписання коду і допомагала з його читабельністю.До того ж інтеграція з React дозволила легко та зручно створювати типізовані компоненти.

Таким чином, TypeScript дозволив не тільки створити гнучкий та інтерактивний інтерфейс користувача, але й забезпечив високу якість та стабільність клієнтського коду в усьому проєкті.

2.2.3 HTML та CSS

Окрім мов програмування, при розробці веб-застосунків критично важливу роль відіграють мови розмітки та стилізації — *HTML (HyperText Markup Language)* та *CSS (Cascading Style Sheets).* Хоча вони технічно не є мовами програмування в повному розумінні цього терміну, їхнє застосування є обов'язковим у будь-якому сучасному веб-проєкті. Саме ці технології відповідають за формування структури та зовнішнього вигляду інтерфейсу користувача, що робить їх невід'ємною частиною реалізації клієнтської частини PomodoroStickNotes.

У проєкті HTML використовується у вигляді JSX — синтаксису, що поєднує можливості HTML і JavaScript та застосовується в компонентах React. Завдяки цьому підходу, розмітка створюється не в окремих HTML-файлах, а безпосередньо в коді компонентів, що спрощує логіку взаємодії між структурою, станом і

поведінкою елементів. JSX дозволяє описувати структуру інтерфейсу в більш декларативний спосіб, що робить код зрозумілішим і компактнішим.

2.3 Веб-фреймворки для клієнта та сервера

Веб-фреймворки є ключовими інструментами, які значно спрощують і прискорюють процес створення веб-застосунків. Вони надають готову архітектуру, стандарти, структури даних і механізми обробки запитів, які дозволяють зосередитися на реалізації логіки самого застосунку, а не на розв’язанні типових технічних завдань із нуля.

У проєкті PomodoroStickNotes були використані сучасні веб-фреймворки як на стороні клієнта, так і на сервері: React, Tailwind CSS та FastAPI. Їх поєднання забезпечило швидку, зручну й масштабовану розробку.

2.3.1 React

React — це популярна JavaScript-бібліотека, яка використовується для створення інтерфейсів користувача. Хоча React часто відносять до фреймворків, офіційно він є саме бібліотекою, однак його можливості, розширення та екосистема дозволяють використовувати його як повноцінний фронтенд-фреймворк. Створений компанією Facebook у 2013 році, React став одним із найпоширеніших інструментів для побудови односторінкових застосунків (SPA).

Однією з основних переваг React є його компонентний підхід: інтерфейс розбивається на незалежні блоки (компоненти), які мають власну логіку, стиль і стан. Це дозволяє розробникам легко організовувати код, повторно використовувати частини інтерфейсу та швидко масштабувати застосунок. У PomodoroStickNotes було реалізовано окремі компоненти для Pomodoro-таймера, системи нотаток, панелі цитат та музичного плеєра.

Ще однією сильною стороною React є віртуальний DOM, який дозволяє оновлювати тільки ті частини сторінки, які змінилися, що значно підвищує продуктивність застосунку. Також React добре інтегрується з TypeScript, що було

важливо для нас, оскільки ми прагнули до максимальної передбачуваності та безпеки коду [8,9].

Завдяки своїй гнучкості, модульності та активній спільноті, React забезпечив ефективну розробку інтерактивного, динамічного та зручного інтерфейсу для кінцевого користувача.

2.3.2 Tailwind CSS

Tailwind CSS — це сучасний CSS-фреймворк, який базується на утилітарному підході до стилізації. На відміну від класичних CSS-фреймворків, які пропонують готові компоненти (наприклад, кнопки чи панелі), Tailwind надає набір низькорівневих класів, що дозволяють будувати інтерфейс за принципом "знизу вгору".

У PomodoroStickNotes Tailwind був використаний для оформлення інтерфейсу без створення окремих CSS-файлів. Це дозволило розробляти компоненти швидко й узгоджено, прямо в рамках JSX-розмітки React. Tailwind забезпечує високу швидкість стилізації завдяки своїй передбачуваності, великій кількості готових класів та широким можливостям кастомізації — кольори, відступи, шрифти, адаптивність і багато іншого.

Перевагою Tailwind є також його відмінна інтеграція з редактором коду VS Code, що забезпечує автодоповнення та підсвічування класів, що в свою чергу знижує кількість помилок і пришвидшує верстку.

Такий підхід до стилізації став оптимальним для нашого застосунку, оскільки дозволив забезпечити одночасно простоту, узгодженість і естетичність інтерфейсу без додаткових залежностей або дизайнерських шаблонів [10].

2.3.3 FastAPI

FastAPI — це сучасний високопродуктивний веб-фреймворк для створення REST API на мові Python. Він був обраний для реалізації серверної частини PomodoroStickNotes завдяки своїй простоті у використанні, відмінній продуктивності та підтримці асинхронного програмування [5].

FastAPI має низку переваг, які були ключовими для нашого проєкту:

– автоматична генерація документації до API (Swagger UI);
– підтримка валідації запитів та відповідей завдяки Pydantic;
– повна типізація, яка добре поєднується з редакторами коду;
– висока продуктивність, близька до Node.js і Go;
– зручна інтеграція з ORM (у нашому випадку — SQLAlchemy);
– асинхронність, яка дозволяє обробляти запити ефективніше.

В розробленому веб-додатку FastAPI відповідав за обробку всіх клієнтських запитів, збереження нотаток, повернення мотиваційних цитат, а також підтримку взаємодії з базою даних. Завдяки компактному й зрозумілому синтаксису розробка серверної частини зайняла мінімум часу при збереженні високої якості реалізації.

2.3.4 JWT (JSON Web Token)

Також в проєкті реалізована базова система аутентифікації користувачів, що використовує механізм JSON Web Token (JWT). Це сучасний і безпечний метод передачі авторизаційної інформації між клієнтом і сервером у вигляді компактного, самодостатнього токена.

JWT складається з трьох частин: заголовка, тіла (payload) та підпису. Така структура дозволяє шифрувати ідентифікатор користувача, строк дії токена, а також додаткову інформацію у форматі JSON, яку потім сервер може перевірити без потреби у повторному зверненні до бази даних при кожному запиті.

У проєкті JWT використовується під час логіну користувача: після введення логіну та пароля сервер формує токен, який надсилається клієнту й зберігається в локальному сховищі. При подальших зверненнях клієнт надсилає цей токен у заголовку Authorization, що дозволяє серверу ідентифікувати користувача без повторного входу.

FastAPI має вбудовану підтримку для роботи з JWT через модулі fastapi.security та jose, що значно спростило реалізацію. Також це забезпечило збереження приватності даних та обмеження доступу до персоналізованих нотаток лише для відповідного користувача.

2.4 Система збереження та обробки даних

Кожен сучасний веб-додаток покладається на добре організовану систему зберігання та обробки даних, оскільки вона забезпечує стабільне та безпечне функціонування. У випадку з PomodoroStickNotes було важливо вибрати рішення для зберігання даних, яке було б легким, простим в обслуговуванні та повністю сумісним з FastAPI. Для цього використано SQLite у поєднанні з SQLAlchemy ORM.

SQLite - це файлова реляційна база даних, яка не потребує окремого сервера, що робить її ідеальним вибором для малих та середніх проектів. Її простота і портативність значно скорочують час налаштування і дозволяють швидко створювати прототипи без додаткових накладних витрат. Оскільки всі дані зберігаються в одному файлі, міграція та розгортання є простими, що робить SQLite особливо придатним для низько концентрованих систем, таких як PomodoroStickNotes.

Взаємодія з базою даних була реалізована за допомогою *SQLAlchemy,* потужної бібліотеки об'єктно-реляційного відображення, яка заповнює прогалину між об'єктами Python та реляційними таблицями. Вона дозволяє визначати моделі даних як класи Python, роблячи код бекенду чистішим і зручнішим для супроводу.

У PomodoroStickNotes в базі даних зберігаються такі типи даних як облікові дані користувача та дані сесії, нотатки користувача з позначками часу створення,мотиваційні цитати,метадані сесії Pomodoro.

Ця структура забезпечує надійну основу для серверної логіки, з чітким поділом сутностей і мінімальними залежностями. Використання SQLite у поєднанні з SQLAlchemy дозволило створити надійний рівень даних, зберігаючи при цьому легкість системи.

2.5 Система контролю версій

Контроль версій є фундаментальним аспектом сучасної розробки програмного забезпечення, що дозволяє командам та окремим розробникам керувати змінами коду, співпрацювати та ефективно відстежувати прогрес.

Git - це розподілена система контролю версій, яка дозволяє розробникам відстежувати кожну модифікацію кодової бази. Однією з головних переваг Git є його здатність до розгалуження та злиття, що дозволяє легко експериментувати з новими функціями, не впливаючи на основну версію проекту. Для PomodoroStickNotes Git використовувався не лише для управління вихідним кодом, але й для ведення структурованої історії коммітів, що допомагає відстежувати етапи розробки та налагоджувати проблеми за потреби.

GitHub, як широко розповсюджена платформа для хостингу коду та спільної роботи, надав необхідну інфраструктуру для віддаленого зберігання та доступу.

3 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ

3.1 Загальна архітектура системи

Загальна архітектура PomodoroStickNotes базується на класичній клієнт-серверній моделі, яка широко застосовується в сучасній веб-розробці завдяки своїй чіткості, модульності та масштабованості. Ця модель дозволяє чітко розділити завдання: клієнтська сторона відповідає за взаємодію з користувачем, тоді як серверна сторона обробляє бізнес-логіку, зберігання даних та обробку відповідей.

Застосунок структурований як *односторінковий застосунок (SPA),* що означає, що інтерфейс динамічно оновлюється без повного перезавантаження сторінки. Такий підхід покращує взаємодію з користувачем та продуктивність. Фронтенд розроблено за допомогою React, компонентної бібліотеки JavaScript, і він взаємодіє з бекендом через HTTP-запити до RESTful API. Ці взаємодії здійснюються асинхронно та повертають відповіді у форматі JSON.

Бекенд працює на FastAPI, високопродуктивному веб-фреймворку Python, який підтримує асинхронну обробку та забезпечує автоматичну документацію API. Бекенд відповідає за керування сеансами користувачів, перевірку токенів JWT, обробку операцій з даними для нотаток та цитат, а також забезпечення безпечного зв'язку з базою даних SQLite.

Обрана архітектура забезпечує кілька суттєвих для проєкту переваг:

– Масштабованість: Нові функції можна додавати до клієнта або сервера з мінімальним втручанням.

– Модульність: систему можна підтримувати та налагоджувати окремо.

– Гнучкість: перемикання технологій з будь-якої сторони можливе без переписування всієї системи.

– Продуктивність: Швидкий час відгуку забезпечується легким дизайном API та асинхронною обробкою.

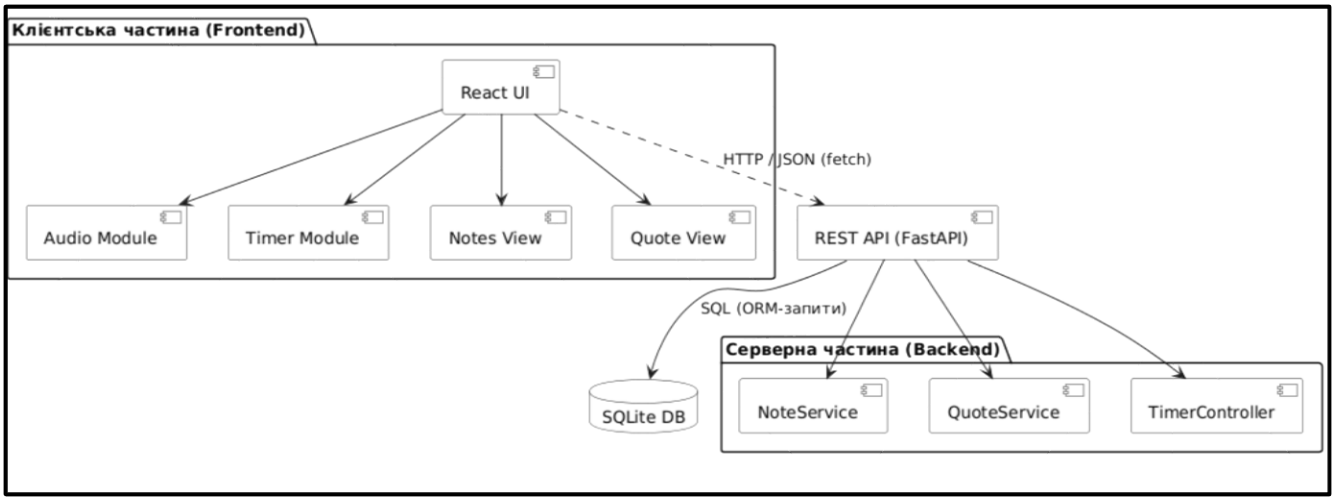


Рис. 3.1 Діаграма компонентів

Для ілюстрації додано діаграму компонентів, котра показує логічну структуру взаємодії між клієнтською частиною, API, сервісами та базою даних

3.2 Структура клієнтської частини

Фронтенд реалізовано за допомогою бібліотеки React, покращеної TypeScript для безпеки типів та кращої підтримки. Ця частина системи розроблена відповідно до компонентно-орієнтованої парадигми, яка заохочує модульність, повторне використання коду та чіткий розподіл обов'язків. Це дозволяє створювати інтерактивні та адаптивні інтерфейси, які динамічно реагують на дії користувача без необхідності перезавантажувати всю сторінку.

Кожна логічна частина інтерфейсу розроблена як незалежний компонент. Це означає, що різні функції застосунку, такі як таймер Pomodoro, функція створення нотаток, відображення мотиваційних цитат та автентифікація користувача, розділені у відповідних модулях. Ці модулі безперешкодно взаємодіють, залишаючись самодостатніми, що дозволяє легко налагоджувати, оновлювати та розвивати програму.

Логіка таймера обробляється в спеціальному компоненті, відповідальному за керування робочими сеансами та перервами, функцію зворотного відліку, звукові сповіщення та переходи. Функція створення нотаток інтегрована через окремий компонент, який надає форми для створення, редагування та видалення нотаток, а також асинхронну взаємодію з серверною частиною для збереження даних користувача. Мотиваційні цитати отримуються та відображаються в окремому ізольованому модулі, який автоматично оновлюється між робочими сеансами, щоб покращити зосередженість та мотивацію користувача.

Процеси автентифікації, включаючи вхід та реєстрацію, керуються за допомогою спеціального набору компонентів, які взаємодіють з кінцевими точками автентифікації бекенду. Ці компоненти відповідають за перевірку введених користувачем даних, обробку помилок та безпечне зберігання токена JWT після успішного входу. Крім того, фронтенд широко використовує React та допоміжні функції, які абстрагують повторення, такі як керування станами таймера, доступ до локального сховища та отримання даних API [12, 13].

Загалом, ця клієнтська архітектура забезпечує масштабований, зручний у підтримці та адаптивний досвід для кінцевих користувачів. Її модульна природа спрощує майбутні вдосконалення, полегшує тестування та дозволяє розробникам працювати незалежно над різними частинами інтерфейсу без перешкод.

Наведена діаграма (див. рисунок 3.2) варіантів використання демонструє всі можливі основні сценарії взаємодії між користувачем та клієнтською частиною застосунку.

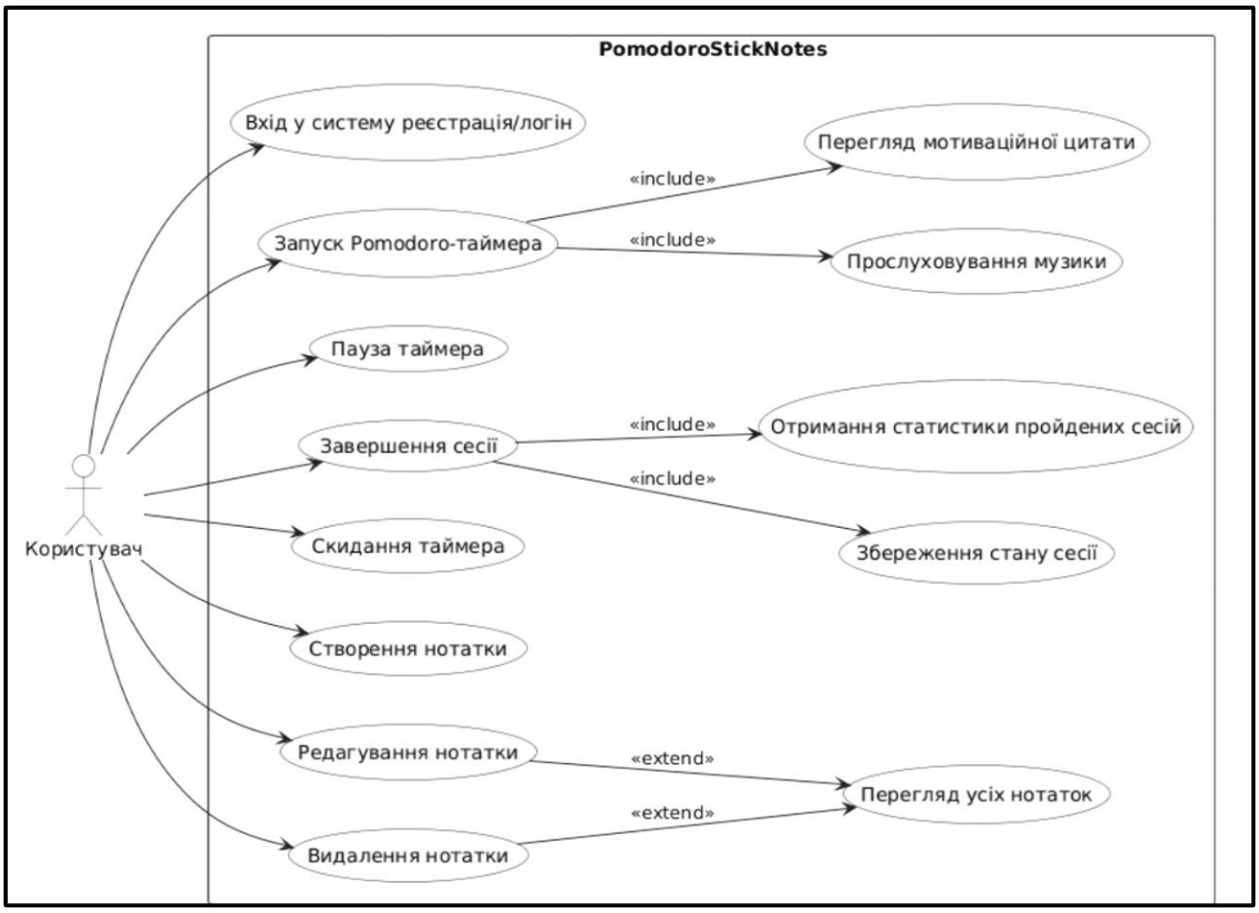


Рис. 3.2 Діаграма варіантів використання

3.3 Структура серверної частини

Бекенд побудовано за допомогою FastAPI, сучасного, швидкого (високопродуктивного) веб-фреймворку для створення API за допомогою Python. Серверна структура застосунку розроблена з чітким розподілом логічних обов'язків, що відповідає принципам модульної та підтримуваної архітектури програмного забезпечення.

Кожна основна функція застосунку — керування нотатками, пошук мотиваційних цитат, координація таймера та автентифікація користувачів — інкапсульована у власному модулі або сервісі.

Ці сервіси надають чітко визначені кінцеві точки та працюють незалежно, зменшуючи зв'язки та роблячи тестування та налагодження більш керованими.

Вхідні запити обробляються обробниками маршрутів FastAPI, де вхідні дані перевіряються, логіка обробляється, а структуровані відповіді JSON генеруються та повертаються клієнту.

Підтримка FastAPI для впровадження залежностей дозволяє застосунку функціонально масштабуватися, а такі компоненти, як проміжне програмне забезпечення автентифікації, обробка сеансів бази даних та валідатори даних, узгоджено керуються в межах проекту. Логіка, пов'язана з безпекою, зокрема використання веб-токенів JSON (JWT), реалізується через виділені кінцеві точки, які видають токени після успішного входу та перевіряють токени в захищених маршрутах [6].

Усі взаємодії з базою даних виконуються за допомогою SQLAlchemy ORM. Моделі, визначені за допомогою SQLAlchemy, описують форму даних та пов'язують бізнес-логіку з рівнем персистентності. Кожен маршрут API взаємодіє з цими моделями безпосередньо або через сервісні рівні, які абстрагують загальні операції з базою даних, забезпечуючи чисту та зручну в обслуговуванні кодову базу.

Завдяки вбудованій підтримці FastAPI документації OpenAPI та обробки асинхронних запитів, бекенд залишається легким та ефективним, пропонуючи водночас багатий досвід розробника. Він забезпечує баланс між продуктивністю та читабельністю, що робить його придатною основою як для поточних потреб, так і для майбутньої масштабованості.

На рисунку 3.3 наведено діаграму класів, котра ілюструє головні сутності на сервері та атрибути і зв’язки між ними, зокрема користувача, нотаток, JWT-токену і активної сесії таймеру.

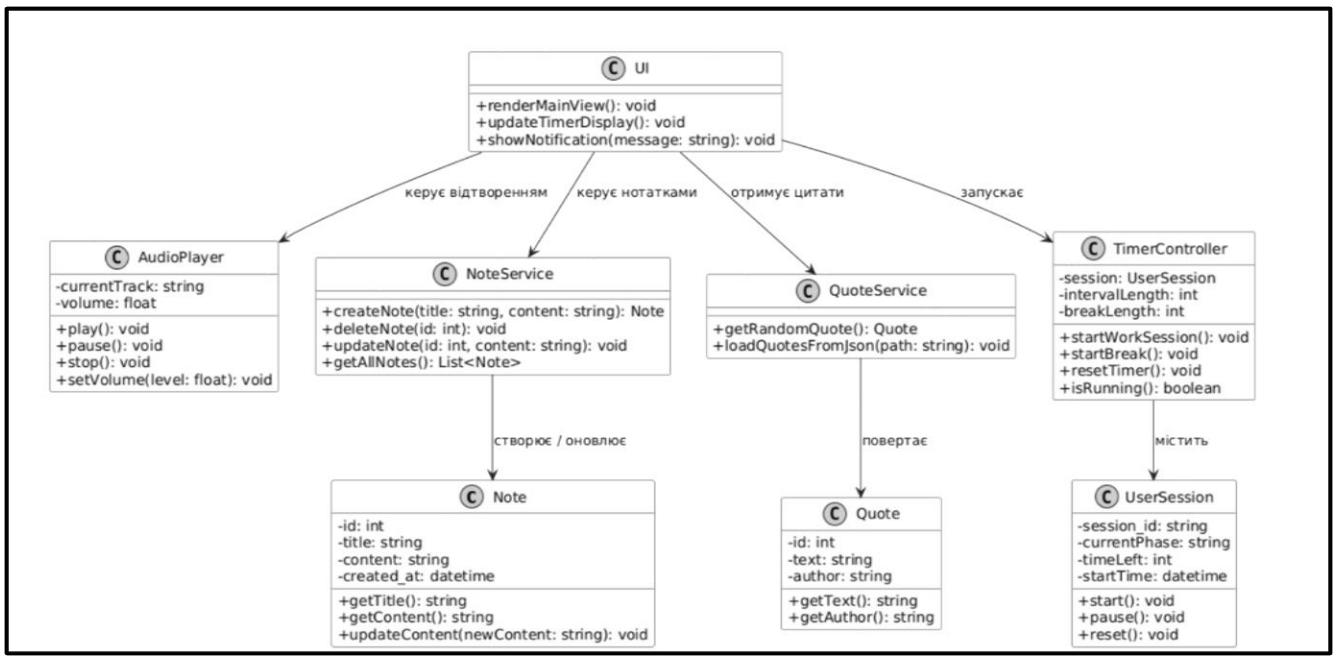


Рис. 3.3 Діаграма класів

3.4 Модель даних та база даних

Модель даних PomodoroStickNotes структурована як реляційна схема, реалізована з використанням SQLite як базового механізму баз даних. SQLite було обрано завдяки його простоті, портативності та мінімальним вимогам до конфігурації, що робить його ідеальним вибором для легких програм або середовищ розробки.

Програма визначає невеликий набір ключових сутностей: користувачів, нотатки та цитати. Кожна з цих сутностей відображається в таблицю в базі даних за допомогою SQLAlchemy, яка забезпечує об'єктно-орієнтований інтерфейс до реляційних даних. Такий підхід спрощує процес запитів, вставки, оновлення та видалення даних, зберігаючи при цьому узгодженість між бізнес-логікою та рівнем сховища.

Кожна таблиця визначена за допомогою первинного ключа, відповідних зовнішніх ключів, позначок часу для відстеження створення та оновлень, а також обмежень для забезпечення цілісності даних. Таблиця користувачів зберігає облікові дані та дані профілю, включаючи зашифровані паролі. Таблиця нотаток зберігає текстові нотатки, пов'язані з певними користувачами. Таблиця цитат зберігає мотиваційні фрази, які випадковим чином вибираються під час сеансів Pomodoro.

FastAPI інтегрується з SQLAlchemy, забезпечуючи асинхронний доступ до бази даних та використовуючи систему типізації Python для визначення та перевірки схем запитів і відповідей. Вся комунікація між серверною частиною та базою даних абстрагується через моделі та сервіси, що забезпечує чисті та безпечні операції з даними.

Використання реляційної моделі даних забезпечує основу для таких функцій, як користувацький контент, управління сеансами та персоналізація на основі даних. Це також відкриває можливості для майбутніх розширень, таких як статистика використання, звіти про відстеження часу та розширена фільтрація даних.

ER-діаграма, зображена на рисунку 3.4, відображає логічну модель даних застосунку, реалізовану у вигляді реляційної бази даних. Вона включає чотири основні сутності: Користувач, Нотатка, Сесія та Цитата. Зв’язки між таблицями демонструють реальні залежності в системі: один користувач може мати багато нотаток та багато сесій. Кожна нотатка або сесія належить одному конкретному користувачу.

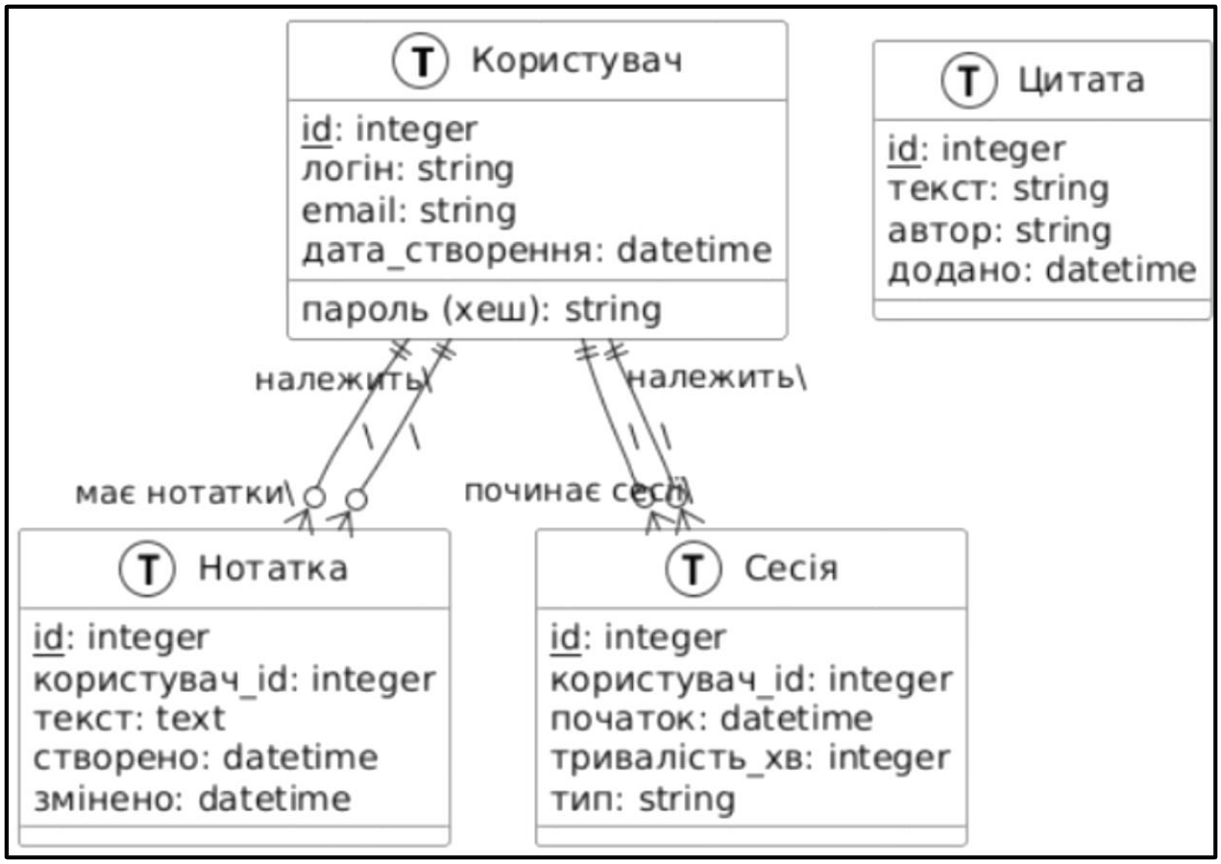


Рис. 3.4 ER-діаграма

4 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Реалізація Pomodoro-таймера

Функціональність Pomodoro-таймера є основною частиною веб-застосунку, адже саме вона забезпечує реалізацію методики Pomodoro — техніки управління часом, яка передбачає чергування періодів сфокусованої роботи з короткими і довгими перервами. Основна мета таймера — допомогти користувачеві залишатись продуктивним, контролюючи час фокусування та відпочинку, щоб уникнути вигорання та надати комфортне середовище для побудови персонального тайм менеджменту робочого часу.

На серверному боці логіка таймера реалізована з використанням Flask. У файлі routes/timer.py створено окремий blueprint — timer_bp, який відповідає за обробку трьох основних маршрутів: отримання поточної фази таймера (/api/timer/), запуск таймера (/api/timer/start) та його скидання (/api/timer/reset). Усі маршрути використовують глобальні змінні для збереження поточного стану таймера. Таймер перемикається між двома фазами: "focus" (робота) і "break" (перерва), тривалість яких складає 25 і 5 хвилин відповідно. У момент, коли час поточної фази вичерпується, таймер автоматично перемикається на наступну фазу.

```
@timer_bp.route('/api/timer/', methods=['GET'])
def get_timer():
    global start_time, current_phase
    if start_time:
        elapsed = time.time() - start_time
        if elapsed >= PHASE_DURATIONS[TIMER_PHASES[current_phase]]:
            current_phase = (current_phase + 1) % len(TIMER_PHASES)
```

```
        start_time = time.time()
    return jsonify({"phase": TIMER_PHASES[current_phase]})
```

Цей механізм був реалізований досить просто, без використання сторонніх бібліотек — завдяки модулю time та функціям для розрахунку пройденого часу. Відповіді маршрути повертають у форматі JSON, що дозволяє легко обробляти їх у клієнтській частині.

На клієнті, в компоненті PomodoroTimer, побудованому за допомогою React, таймер реалізований окремим функціональним компонентом. Тут використано вже описаний хук useState для керування такими станами, як залишок часу, активна фаза, прогрес-бар, статус запуску та кількість завершених сесій.

За допомогою useEffect реалізовано головну логіку зворотного відліку: при запуску таймера створюється інтервал, який кожну секунду зменшує час. Після досягнення нуля таймер перемикає фазу на "перерву" або "довгу перерву" після кожних чотирьох робочих циклів.

```
useEffect(() => {
  if (!isRunning) return;
  const timer = setInterval(() => {
    setTimeLeft(prev => prev - 1);
  }, 1000);
  return () => clearInterval(timer);
}, [isRunning]);
```

Важливою деталлю стало візуальне оформлення. Користувач бачить стильну анімовану панель з таймером у центрі, де колір рамки та іконки змінюється залежно від активної фази. Таймер супроводжується кнопками запуску, зупинки та перемикання на довгу перерву. Після завершення кожної сесії користувач також бачить мотиваційну цитату, обрану випадковим чином із заздалегідь підготовленого масиву.

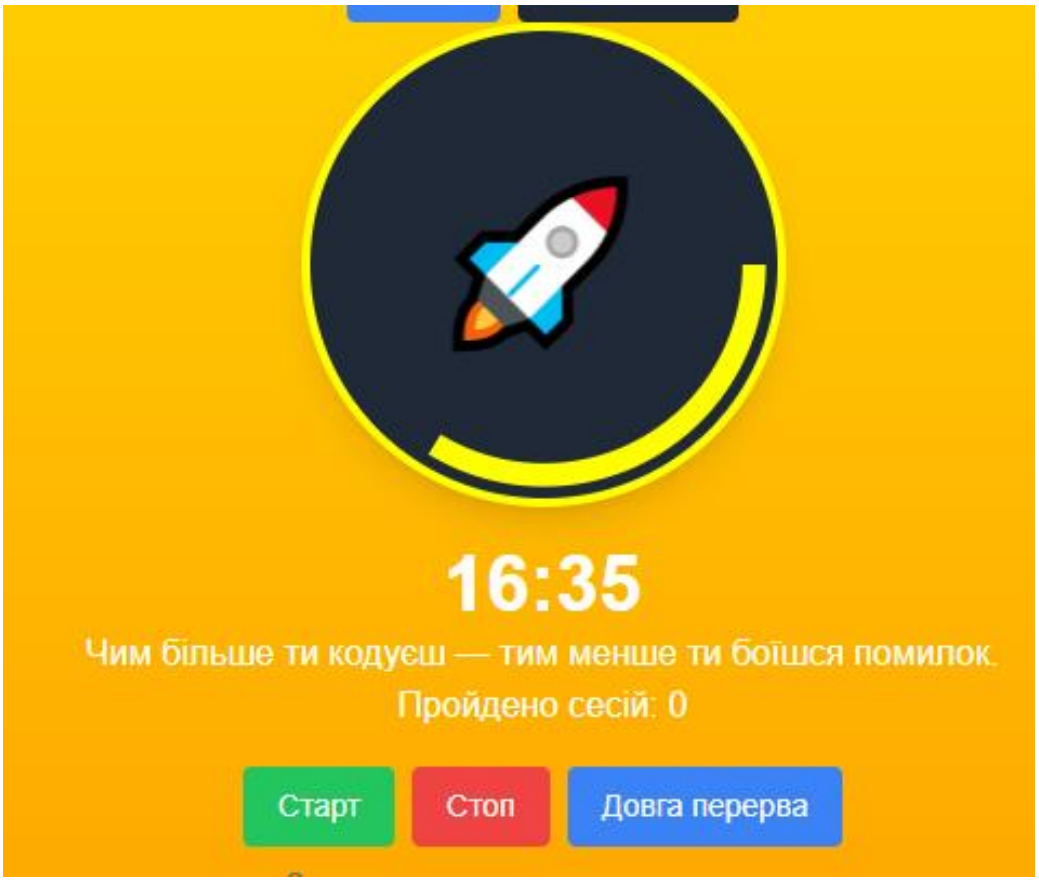


Рис. 4.1 Екранна форма таймеру з активною робочою фазою та кнопками керування

Крім того, реалізовано систему сповіщень — під час зміни фази користувач отримує повідомлення через Notification API.

Таким чином, навіть якщо він переключив вкладку, система нагадує про завершення поточної сесії.

На завершення, система також відображає просту статистику у вигляді діаграми: користувач може бачити, скільки часу він провів у фазі фокусування, а скільки — у перервах.

Для цього використовується компонент Bar з бібліотеки react-chartjs-2, який візуалізує дані у вигляді стовпчастої діаграми.

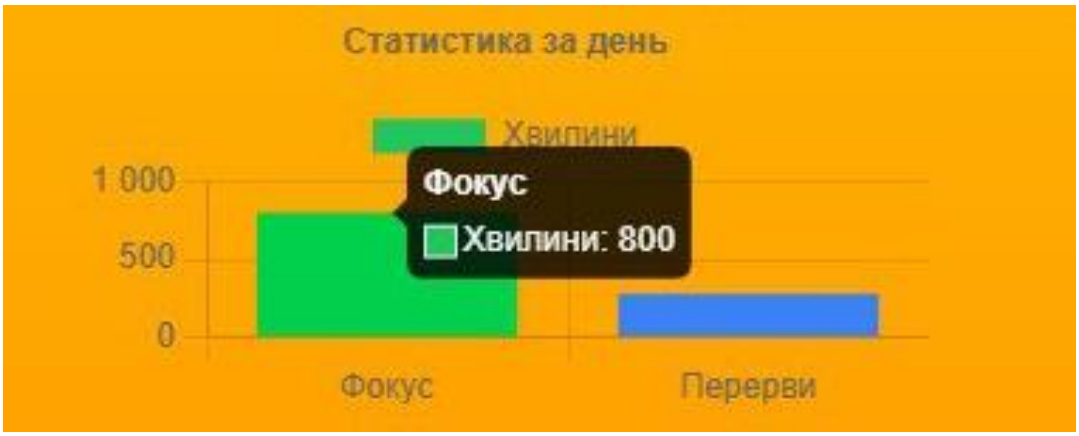


Рис. 4.2 Екранна форма діаграми статистики сесій Pomodoro

4.2 Реалізація системи мотиваційних цитат та плеєру

Щоб збагатити користувацький досвід та заохотити продуктивність, додаток інтегрує систему мотиваційних цитат і мінімалістичний аудіо плеєр. Ці елементи спрямовані не лише на зменшення монотонності, але й на створення заспокійливого та підтримуючого середовища, яке доповнює структурований робочий ритм, що сприяє техніка Pomodoro.

Функція мотиваційних цитат працює за простим механізмом: коли компонент таймера ініціалізується або починається новий сеанс, він надсилає запит надання цитати до серверної частини для отримання випадкової мотиваційної цитати. Після отримання відповіді цитата відображається на видному місці, одразу під таймером.

Наводжу приклад логіки запиту:

```
const handleStart = () => {
   const random = Math.floor(Math.random() * QUOTES.length);
   setQuote(QUOTES[random]);
   setIsRunning(true);
 };
```

Простота цієї реалізації забезпечує безперешкодну інтеграцію з існуючою логікою таймера. Користувач отримує нову цитату на початку

кожного робочого сеансу, що забезпечує розумову стимуляцію або заохочення.

На стороні сервера цитати зберігаються в базі даних. Кінцева точка вибирає випадковий запис за допомогою легкого запиту, що забезпечує ефективність. Це спрощує масштабування або заміну набору даних у майбутньому.

Паралельно із системою цитат, вбудований HTML5 <audio> програвач забезпечує функціональність фонової музики. Цей програвач завантажує локальний або віддалений аудіофайл і пропонує користувачам базове керування відтворенням — відтворення, паузу, регулювання гучності — через зручний інтерфейс, розроблений на Tailwind CSS. Відтворення аудіо запускається вручну або автоматично синхронно з початком сеансу Pomodoro.

Це приклад компонента програвача:

```
const MusicPlayer = () => {
  const audioRef = useRef(null);
  const [isPlaying, setIsPlaying] = useState(false);
  const [volume, setVolume] = useState(0.5);
  const togglePlay = () => {
    if (isPlaying) {
      audioRef.current.pause();
    } else {
      audioRef.current.play();
    }
    setIsPlaying(!isPlaying);
  };
…
<audio
      ref={audioRef}
```

```
        loop
        src="https://streams.fluxfm.de/Chillhop/mp3-128/streams.fluxfm.de/"
      />
```

Обидві додані функції спрямовані на одну мету — покращити робоче середовище, враховуючи психологічні та емоційні аспекти зосередженості. У той час як таймер Pomodoro нав'язує структуру, цитати та музика забезпечують внутрішнє підкріплення та покращують настрій.

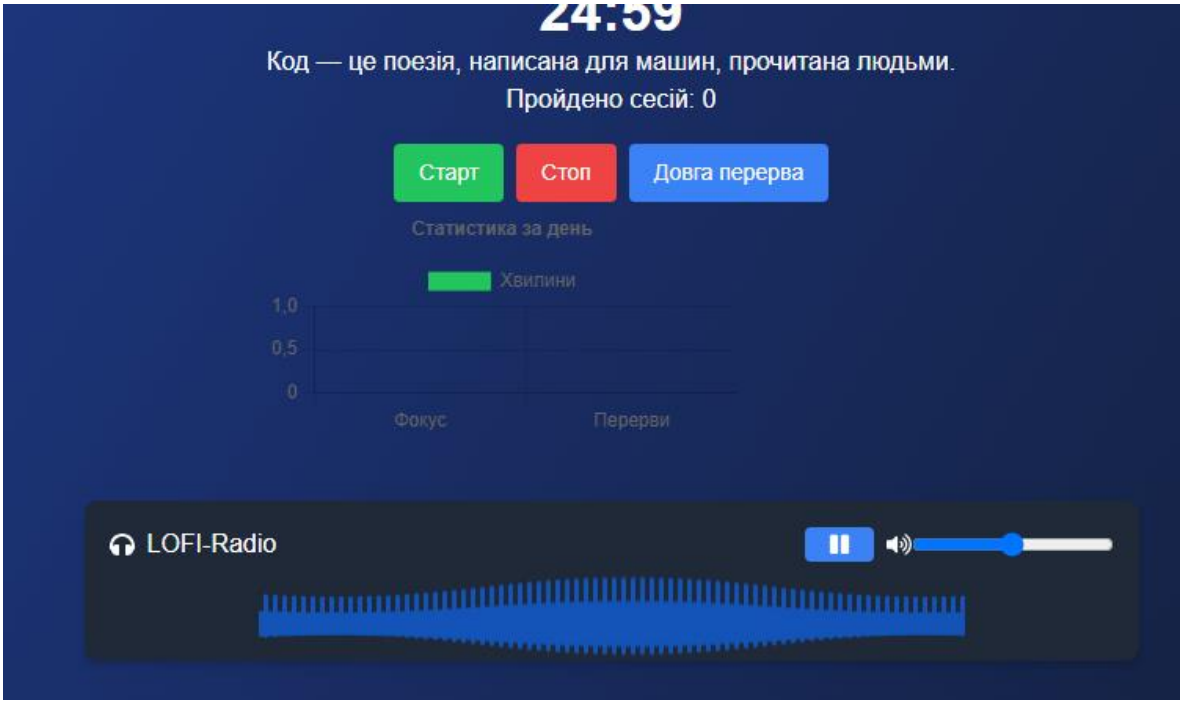


Рис. 4.3 Екранна форма сторінки, де знаходяться цитати та аудіо плеєр

4.3 Реалізація функціоналу нотаток

Одним із найважливіших аспектів програми PomodoroStickNotes є функція ведення нотаток. Ця функція дозволяє користувачам швидко записувати думки, завдання чи ідеї, що виникають під час зосереджених сесій Pomodoro.

Вона гарантує, що швидкоплинні ідеї фіксуються без порушення концентрації, що підсилює мету підтримки глибокої зосередженості та безперебійної продуктивності.

У серверній частині основний функціонал нотаток міститься у таких файлах:

– backend/database/models/py— визначення моделі Note (таблиця в базі даних);

– backend/routers/notes.py — ендпоінти для створення, оновлення, отримання та видалення нотаток та логіка доступу до бази даних (create, read, update, delete);

– backend/app.py — підключення маршруту notes до FastAPI-застосунку.

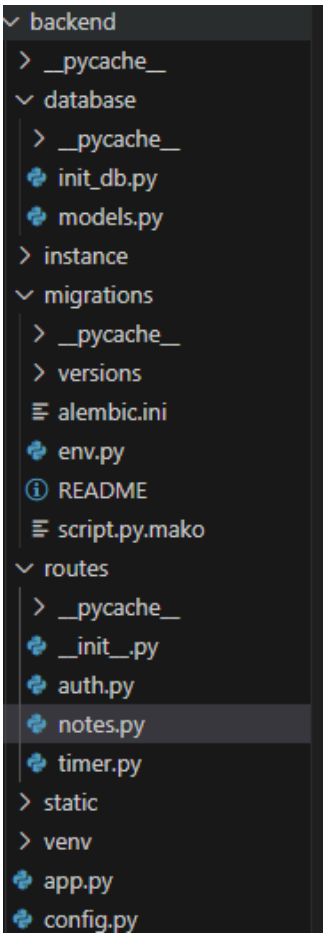


Рис.4.4 Структура бекенду проєкту

Таке розділення відповідає модульній архітектурі в адаптації до FastAPI.

Усі дані нотаток отримуються, зберігаються та оновлюються через взаємодію з API серверної частини через HTTP-запити.

Коли користувач створює або редагує нотатку, вхідні дані надсилаються до бекенду за допомогою запитів POST або PUT. Кожна операція обробляється асинхронно для підтримки швидкості реагування інтерфейсу та забезпечення візуального зворотного зв'язку. Залежно від операції відображається спіннер завантаження або тимчасове повідомлення про успіх.

На бекенді управління нотатками побудовано за допомогою FastAPI. Основні кінцеві точки включають /notes/, /notes/{id} та відповідні методи GET, POST, PUT та DELETE.

Система впровадження залежностей FastAPI використовується для перевірки токенів JWT та авторизації користувача перед обробкою запитів на нотатки.

Спрощена схема моделі Note з backend/database/models.py виглядає наступним чином:

```
class Note(db.Model):
    __tablename__ = 'notes'
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.String(255), nullable=False)
    def __repr__(self):
        return f"<Note {self.id}: {self.text}>"
```

Формат реалізації є класичним прикладом CRUD (Create, Read, Delete), де користувач взаємодіє з інтерфейсом, а дії над нотатками зберігаються на сервері. На сервері, у файлі routes/notes.py, знаходиться логіка для обробки трьох основних запитів:

– GET /notes — повертає список усіх збережених нотаток:

```
@router.get("/notes")
def get_notes():
    return notes
```

– POST /notes — створює нову нотатку на основі JSON-запиту:

```
@router.post("/notes")
def create_note(note: dict):
    note["id"] = len(notes) + 1
    notes.append(note)
    return note
```

– DELETE /notes/{id} — видаляє нотатку з вказаним ID:

```
@router.delete("/notes/{note_id}")
def delete_note(note_id: int):
    global notes
    notes = [n for n in notes if n["id"] != note_id]
    return {"message": "Note deleted"}
```

Ця модель визначає основні поля, що використовуються для кожної нотатки, та забезпечує належне відстеження оновлення.

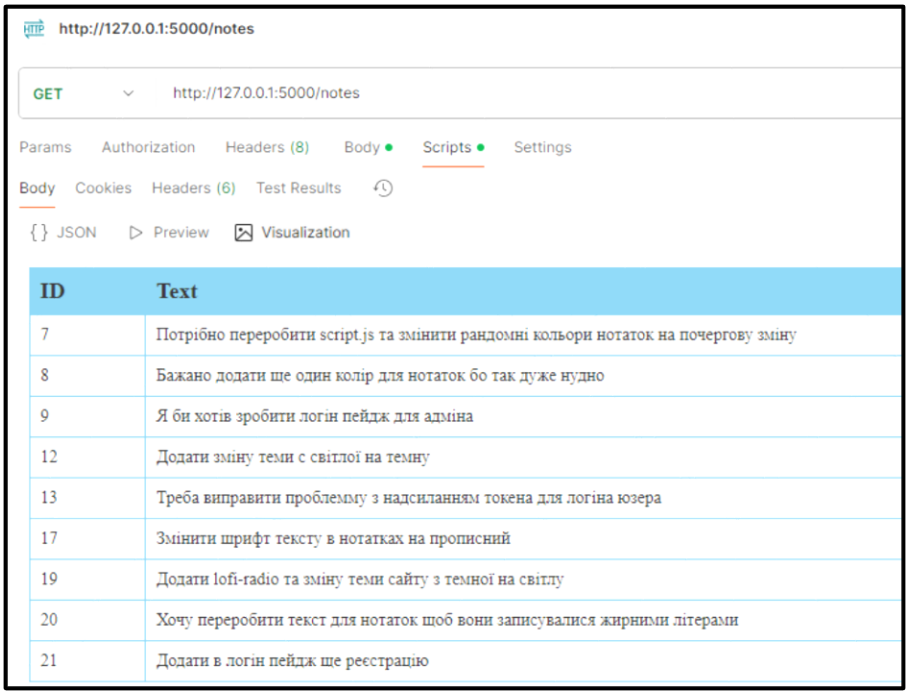


Рис.4.5 Виведення GET-запиту в Postman

Зі сторони інтерфейсу користувач бачить текстове поле для вводу нотатки, кнопку додавання, а також список вже створених нотаток. Кожна нотатка має кнопку видалення.

Додатково реалізовано функцію toggleCheck, яка дозволяє візуально відмічати нотатку як виконану (відзначення зроблене у стані компоненту). Це дає змогу користувачу бачити, які нотатки вже виконано, хоч і без збереження цього стану на сервері.

Рис. 4.6 Екранна форма сторінки, де знаходиться блок нотаток та текстове поле для їх введення

4.4 Інтерфейс користувача

Інтерфейс користувача створений із урахуванням принципів мінімалізму, доступності та візуального задоволення. Однією з ключових цілей дизайну було забезпечити зручну, логічну і приємну у користуванні структуру, яка не відволікає користувача від основного завдання — концентрації та ефективної роботи.

Інтерфейс зібраний у компоненті Dashboard, який об'єднує три основні модулі: таймер, музичний плеєр та нотатки. Цей підхід дозволяє зберігати цілісність взаємодії та уникнути зайвих переходів між сторінками. Завдяки цьому користувач отримує повноцінне робоче середовище на одній панелі.

Колірна гама інтерфейсу змінюється залежно від обраної теми. Було реалізовано перемикач між світлою та темною темами, що дозволяє адаптувати застосунок до умов освітлення або особистих вподобань користувача. Перемикання теми реалізовано через useState та збереження вибору в localStorage, завдяки чому вибраний режим зберігається між сесіями:

```
const toggleTheme = () => {
  setIsDarkTheme((prev) => {
    const next = !prev;
    localStorage.setItem("theme", next ? "dark" : "light");
    return next;
  });
};
```

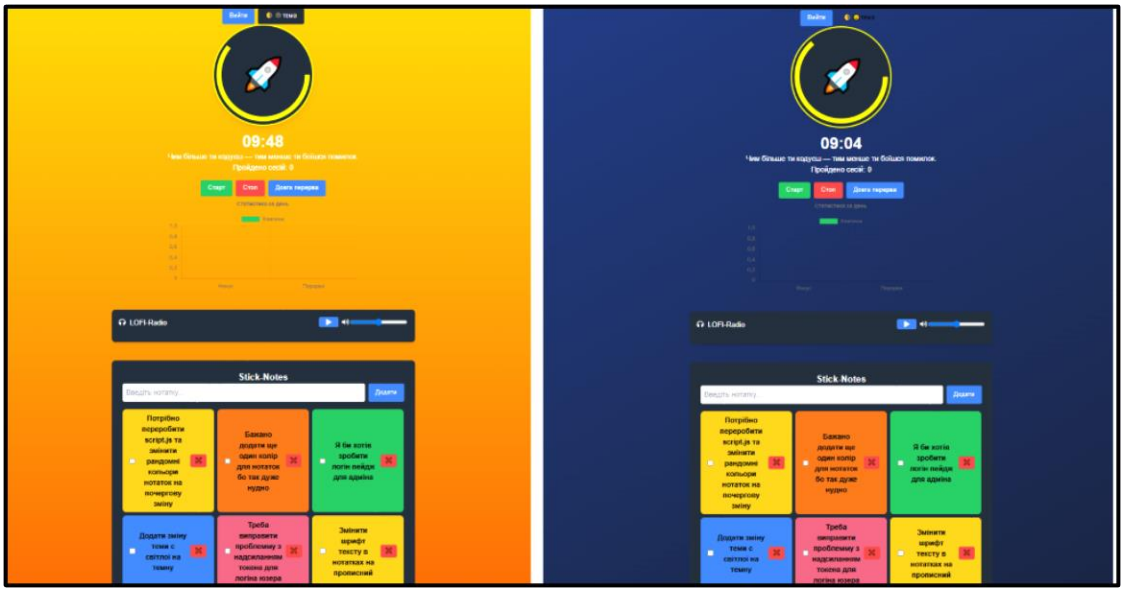


Рис. 4.7 Порівняння світлої та темної кольорової гами інтерфейсу

Інтерфейс також містить просту, але функціональну шапку з кнопкою для виходу з облікового запису та перемикачем теми. Розміщення цих елементів у верхньому правому куті забезпечує швидкий доступ без зайвих рухів миші.

Сторінка оформлена за допомогою Tailwind CSS, що дозволило забезпечити сучасний вигляд, адаптивність до різних екранів і швидку стилізацію. Анімації іконки ракети під час активної фази, створюють динаміку, яка мотивує користувача, не перевантажуючи при цьому візуальний стиль.

Особливу увагу приділено мобільній адаптивності. Всі елементи інтерфейсу використовують Flexbox і Grid, що дозволяє їм підлаштовуватися до ширини вікна без втрати функціональності чи естетики.

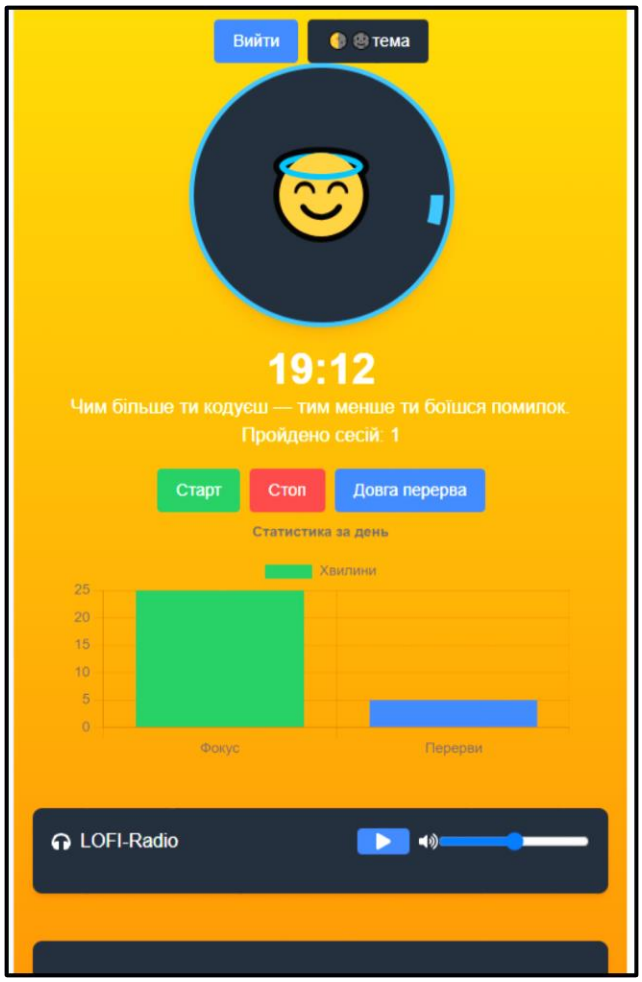


Рис. 4.8 Приклад інтерфейсу на мобільних пристроях

4.5 Реалізація реєстрації та логіну

Додатково в проєкті було реалізовану систему аутентифікації користувача, яка дозволяє захистити дані та забезпечити доступ лише авторизованим особам. Цей функціонал реалізовано за допомогою JSON Web Token (JWT), що забезпечує безпечну авторизацію без необхідності зберігати сесію на сервері. Вся логіка реалізована у файлі routes/auth.py на сервері (Flask) та у компоненті Login.jsx на клієнті (React).

На стороні клієнта користувач може як зареєструватися, так і увійти в обліковий запис через єдину форму. Перемикання між режимами входу та реєстрації реалізоване за допомогою стану isRegister. У разі успішного входу отриманий токен зберігається в localStorage, що дозволяє зберігати авторизацію між перезавантаженнями сторінки.

```
localStorage.setItem("token", data.token);
navigate("/");
```

На сервері реалізовано три основні маршрути:

– /register — для створення нового користувача. При реєстрації пароль хешується з використанням werkzeug.security.

– /login — для входу користувача, який у разі успіху отримує JWT-токен.

– /user — маршрут, що повертає дані поточного користувача (використовується для перевірки автентичності).

Для захисту приватних маршрутів використовується декоратор token_required, який перевіряє валідність токена:

```
@wraps(f)
def decorated(*args, **kwargs):
    token = request.headers.get("x-access-token")
```

Рис. 4.9 Інтерфейс сторінки аутентифікації користувача

На клієнтському інтерфейсі форма авторизації оформлена у приємній жовто-помаранчевій гамі, що створює дружню атмосферу з першого погляду. Після реєстрації користувач отримує сповіщення про успіх і може одразу виконати вхід у систему.

4.6 Взаємодія з сервером та збереження даних

У PomodoroStickNotes, фронтенд взаємодіє з API бекенду за допомогою протоколу HTTP, використовуючи запити на вибірку для надсилання та отримання даних у форматі JSON. Ці взаємодії відповідають за динамічні операції з даними, такі як створення та видалення нотаток, керування сеансами користувачів, отримання фаз таймера та оновлення статистики прогресу.

Усі кінцеві точки API доступні через RESTful архітектуру, реалізовану на бекенді за допомогою креслень FastAPI та Flask. Фронтенд, написаний на React, використовує асинхронні операції (синтаксис async/await та .then()) для обробки відповідей API та відповідного оновлення інтерфейсу користувача.

Наприклад, коли користувач створює нову нотатку на панелі інструментів, спрацьовує функція addNote(), яка надсилає POST-запит до кінцевої точки /notes:

```
fetch("http://127.0.0.1:5000/notes", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ text: newNote })
});
```

Збереження даних наразі реалізовано в пам'яті для простоти та цілей тестування. На серверній частині нотатки зберігаються в глобальному списку Python, який скидається після перезавантаження сервера. Хоча цей метод не рекомендується для виробничих середовищ, його достатньо на етапах розробки та створення прототипів, де пріоритет надається простоті та швидкості.

Ще одним важливим аспектом зв'язку клієнт-сервер є синхронізація станів. Фронтенд використовує useEffect та useState React для синхронізації

інтерфейсу користувача з даними на стороні сервера. Щоразу, коли вносяться зміни, наприклад, видалення нотатки, зміна негайно відображається в локальному стані, покращуючи сприйняту швидкість реагування, ще до того, як запит буде повністю підтверджено сервером.

Загалом, взаємодія клієнт-сервер у цьому проєкті демонструє збалансований підхід між простотою, зручністю використання та сучасними веб-стандартами. Це створює адаптивний та інтерактивний досвід для користувачів, зберігаючи при цьому чітку та модульну структуру серверної частини.

4.7 Тестування роботи застосунку

Для забезпечення стабільної та коректної роботи розробленого веб-застосунку було проведено базове тестування основних компонентів як на серверному, так і на клієнтському боці. Було використано як ручне тестування для перевірки візуального функціоналу, так і автоматизоване тестування на основі модульних сценаріїв з використанням бібліотеки pytest для FastAPI.

З метою перевірки основних REST API-ендпоінтів застосунку були створені та виконані два модулі тестування: test_auth.py та test_notes.py. Тестування проводилося з використанням TestClient із FastAPI, що дозволило ізольовано запускати запити до API без необхідності запуску реального сервера.

Файл test_auth.py включає наступні перевірки:

– успішну реєстрацію нового користувача;
– успішний вхід зареєстрованого користувача та отримання JWT-токена;
– обробку запитів з некоректними даними.

Приклад одного з тестів:

```
def test_login():
    response = client.post("/auth/login", json={"username": "test", "password": "1234"})
    assert response.status_code == 200
    assert "token" in response.json()
```

Файл test_notes.py в свою чергу включає:

– створення нової нотатки з валідацією вмісту;
– отримання списку нотаток у форматі JSON;
– видалення нотатки за ID.

Фрагмент з test_notes.py:

```
def test_delete_note():
    client.post("/notes", json={"text": "Temp"})
    notes = client.get("/notes").json()
    note_id = notes[-1]["id"]
    delete_response = client.delete(f"/notes/{note_id}")
    assert delete_response.status_code == 200
```

Ці тести дозволили переконатись, що взаємодія з моделлю Note відбувається правильно, відповіді приходять у коректному форматі, а операції виконуються без помилок.

Окрім модульних перевірок, було проведено ручне тестування клієнтської частини застосунку у браузері, а також за допомогою інструментів розробника:

– перевірено перемикання між фазами Pomodoro-таймера, зокрема фокусування, короткі та довгі перерви;
– протестовано відображення та видалення нотаток з використанням вбудованого інтерфейсу;
– перевірено коректність авторизації та збереження токена в localStorage;

– успішно виконано доступ до захищених маршрутів лише після логіну;

– візуально перевірено динамічні елементи інтерфейсу, такі як діаграми, таймер, плеєр, темна/світла тема.

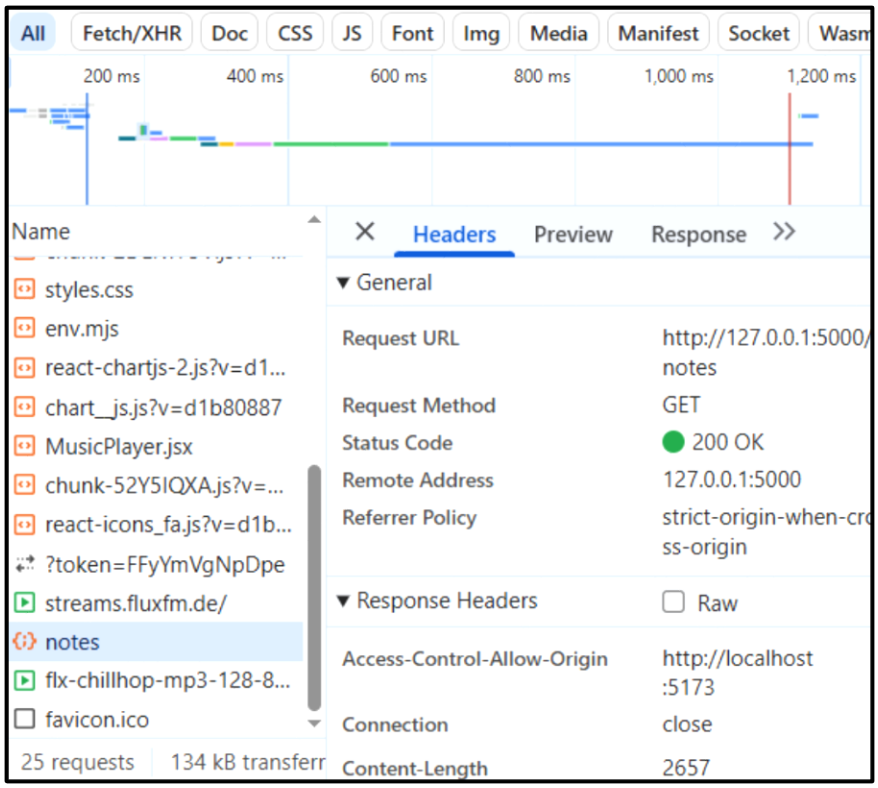


Рис. 4.10 Вкладка Network в браузері з відповідями на GET-запити

Під час тестування було підтверджено стабільну роботу всіх основних маршрутів та інтерфейсів. Запити до API виконуються без помилок, стан клієнта оновлюється коректно, а користувацький досвід — плавний та інтуїтивно зрозумілий.

ВИСНОВКИ

В результаті роботи було розроблено веб-додаток, орієнтований на продуктивність, з використанням Python та фреймворку FastAPI на бекенді, а також React з Tailwind CSS на фронтенді.

Було виконано такі завдання:

– Проаналізовано предметну область: визначено концепцію додатку для продуктивності на основі техніки Pomodoro; описано цільову аудиторію; проведено огляд існуючих рішень, а також окреслено їх переваги та недоліки. На основі цього аналізу визначено основні функціональні та нефункціональні вимоги, зокрема: автентифікацію користувачів, реалізацію таймера Pomodoro, систему мотиваційних цитат, музичний плеєр та легкий інтерфейс для створення нотаток.

– Обрано інструменти реалізації: Visual Studio Code та PyCharm як IDE, TypeScript та Python як мови програмування, React та FastAPI як фреймворки розробки, SQLite як система керування базами даних, GitHub як система контролю версій та Figma для прототипування інтерфейсу користувача.

– Прототип робочої системи розроблено з використанням діаграм UML, включаючи діаграми варіантів використання, діаграми класів, діаграми компонентів, діаграми послідовностей та діаграми активності.

– Веб-застосунок було реалізовано з використанням обраних технологій: прототипування інтерфейсу у Figma, ініціалізація та структурування проєкту React, налаштування бекенду FastAPI з базою даних SQLite, реалізація автентифікації за допомогою JWT, успішний обмін даними між клієнтом та сервером, публікація готової системи на GitHub. Було проведено ручне та модульне тестування для забезпечення коректності основного функціоналу.

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

1. Сковпень М.А., Колодюк А.В. Визначення вимог до веб-системи “Pomodoro & notes” для підтримки ефективності в роботі. *Матеріали конференції “Застосування програмного забезпечення в ІКТ”.* 24 квітня 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025 (с. 123-126)
2. Сковпень М.А., Колодюк А.В. Ігрофікація в управлінні часом програміста: Pomodoro-система візуальної мотивації. // Матеріали V Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К: ДУІКТ, 2025 (подано до друку)

ПЕРЕЛІК ПОСИЛАНЬ

1. Cirillo F. The Pomodoro Technique. 2025. URL: https://www.pomodorotechnique.com/ (date of access: 03.03.2025).

2. Costales J. A Learning Assessment Applying Pomodoro Technique as A Productivity Tool for Online Learning. 2021. URL: https://www.researchgate.net/publication/355185097 (date of access: 07.03.2025).

3. Understanding effort regulation: Comparing 'Pomodoro' breaks and self-regulated breaks. 2023. URL: https://pubmed.ncbi.nlm.nih.gov/36859717/ (date of access: 22.04.2025).

4. Focus To-Do: Pomodoro & Tasks. URL: https://play.google.com/store/apps/details?id=com.superelement.pomodoro (date of access: 23.04.2025).

5. Pomofocus: Best web app (with a built-in to-do list). URL: https://www.getclockwise.com/blog/pomodoro-timer (date of access: 24.04.2025).

6. Flow - Focus & Pomodoro Timer. URL: https://apps.apple.com/us/app/flow-focus-pomodoro-timer/id1423210932 (date of access: 25.04.2025).

7. Todoist. The Pomodoro Technique — Why it works & how to do it. URL: https://www.todoist.com/productivity-methods/pomodoro-technique (date of access: 10.03.2025).

8. Zapier. The 6 best Pomodoro timer apps in 2025. URL: https://zapier.com/blog/best-pomodoro-apps/ (date of access: 12.03.2025).

9. Lubanovic B. FastAPI: Modern Python Web Development. O'Reilly Media, 2023. 277 p.

10. FastAPI Documentation. URL: https://fastapi.tiangolo.com (date of access: 29.03.2025)..

11. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. O'Reilly Media, 2018. 258 p.

12. Learn React with TypeScript: A beginner's guide to reactive web development. Packt Publishing, 2023. 350 p.

13. Boduch A. React Material-UI Cookbook. Packt Publishing, 2019. 350 p.

14. Accomazzo M. React Design Patterns and Best Practices. Packt Publishing, 2016. 300 p.

15. Rappin N. Modern CSS with Tailwind. 2nd ed. Pragmatic Bookshelf, 2023. 200 p.

16. Tailwind CSS Basics for Beginners. Daily.dev, 2024. URL: https://daily.dev/blog/tailwind-css-basics-for-beginners (date of access: 26.04.2025).

17. Study Smarter, Not Harder: The Pomodoro Technique. Herzing University, 2023. URL: https://www.herzing.edu/blog/study-smarter-not-harder-pomodoro-technique (date of access: 27.04.2025).

18. Tailwind CSS Documentation. URL: https://v3.tailwindcss.com/docs (date of access: 10.04.2025).

19. Wathan A., Schoger S. Refactoring UI. 2019. URL: https://www.refactoringui.com/ (date of access: 14.04.2025).

20. UIDesignz. Top 10 User Interface Design Principles in 2024. Medium, 2024. URL: https://medium.com/@uidesign0005/top-10-user-interface-design-principles-in-2024-e18702a5d395 (date of access: 15.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для підвищення ефективності роботи програмістів

Виконав студент 4 курсу
групи ПД-42
Сковпень Михайло Анатолійович
Керівник роботи
асистент кафедри ІПЗ Колодюк Андрій Васильович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - підвищення продуктивності програмістів шляхом ефективного управління часом, ведення нотаток та візуального відстеження прогресу.
- **Об’єкт дослідження -** процес управління робочим часом за допомогою цифрових засобів.
- **Предмет дослідження -** веб-застосунок для організації робочого часу на основі методології Pomodoro з додаванням функціоналу для підвищення мотивації, контролю за результатами та нотатками.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати сучасні підходи до цифрової самоорганізації та вивчити ефективність методики Pomodoro в контексті особистої продуктивності.
2. Дослідити існуючі реалізації Pomodoro-застосунків, визначити їх недоліки та потенціал для покращення функціоналу.
3. Визначити функціональні та нефункціональні вимоги до функціоналу веб-застосунку.
4. Визначити програмні засоби для розробки проєкту.
5. Спроєктувати архітектуру веб-застосунку, що поєднує Pomodoro-таймер, систему нотаток та мотиваційний контент.
6. Розробити серверну частину використовуючи Python та FastAPI і клієнтську частину з використанням React та TypeScript.
7. Провести тестування прототипу та зафіксувати результати щодо стабільності та взаємодії модулів.

3

АНАЛІЗ АНАЛОГІВ

Показник	Forest	Focus To-Do	Pomofocus	Minimalist	PomodoroStickNotes
Pomodoro Timer	+	+	+	+	+
Нотатки	-	+	-	-	+
Мотиваційні елементи / геймифікація	+ (геймифікація)	-	-	-	+ (цитати)
Аудіо	-	-	-	+	+
Аналітика	-	+	-	+ (рахунок сесій)	+ (графік з пройденими хвилинами кожної з сесій та їх загальний рахунок)
Налаштування фаз таймеру	-	+	+	+	+
Платформа	Мобільна	Кроссплатформенна	Веб	Веб	Веб
Доступність	Обмежена безкоштовна версія	Обмежена безкоштовна версія	Безкоштовна	Безкоштовна	Безкоштовна

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Pomodoro-таймер з чітко визначеними фазами роботи, перерви та довгої перерви;
2. Автоматичне перемикання між фазами (робота → коротка перерва → робота → довга перерва);
3. Система нотаток, яка дозволяє створювати, редагувати та видаляти нотатки, з можливістю збереження їх на сервері;
4. Мотиваційна функція у вигляді цитат що будуть надаватись користувачу при старті фукус сесії;
5. Аудіоплеєр, що дає змогу відтворювати музику під час сесій.

Нефункціональні вимоги:

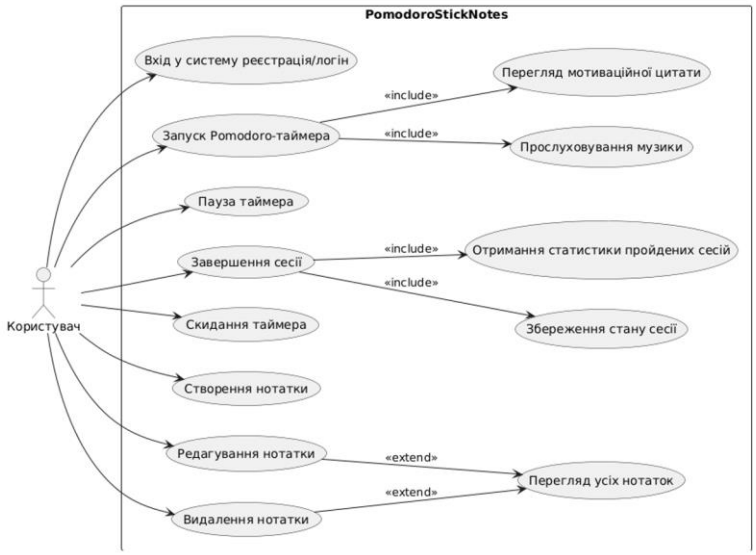
1. Час відгуку інтерфейсу не повинен перевищувати 2 секунд на всіх етапах взаємодії;
2. Висока доступність — застосунок повинен працювати стабільно у сучасних браузерах (Chrome, Edge, Safari);
3. Безпека збереження даних, зокрема нотаток користувача, без втрати інформації після перезавантаження сторінки;
4. Адаптивний дизайн інтерфейсу, який забезпечує коректне відображення елементів на пристроях з різними розмірами екранів (ПК, планшет, смартфон).

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

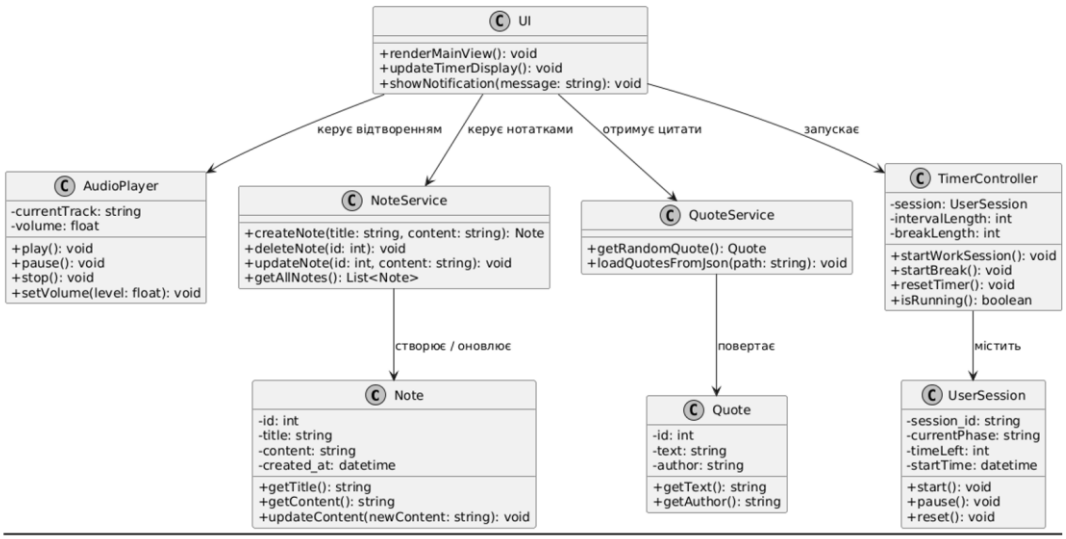
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



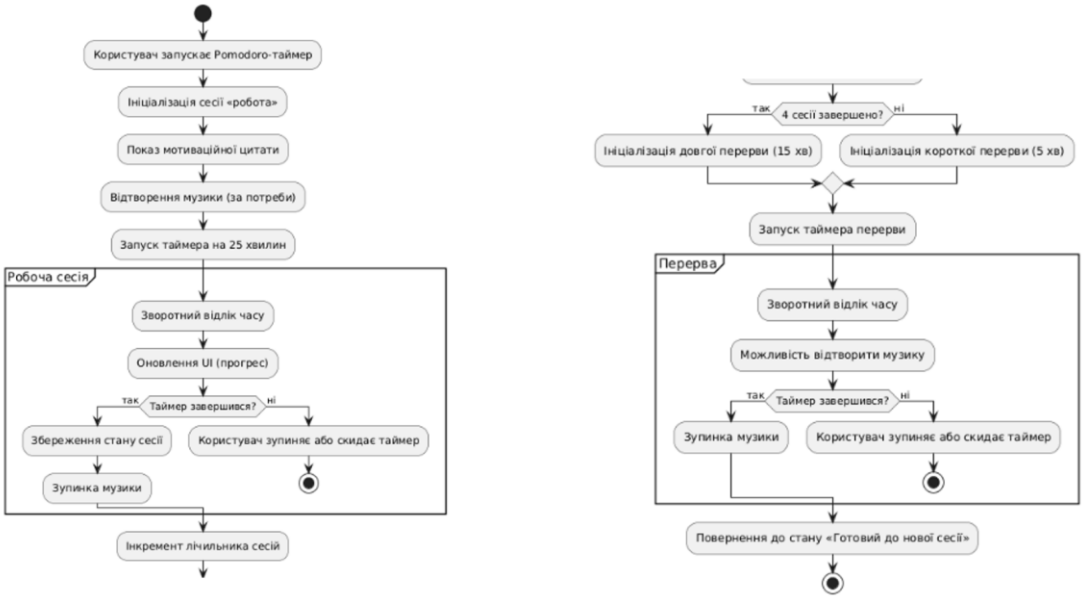
7

ДІАГРАМА КЛАСІВ



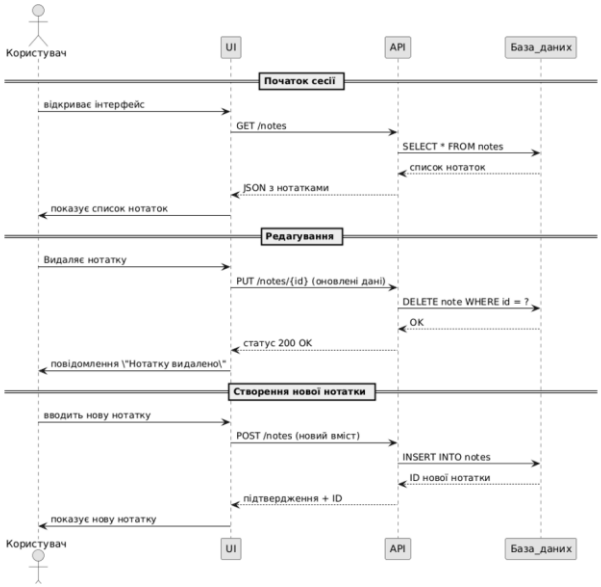
8

ДІАГРАМА ДІЯЛЬНОСТІ Pomodoro-таймера



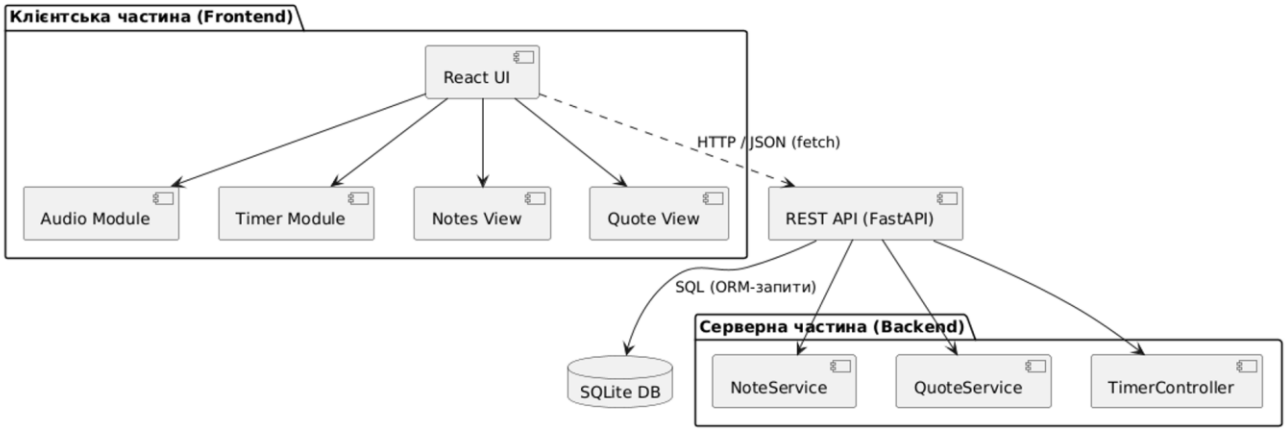
9

ДІАГРАМА ПОСЛІДОВНОСТІ



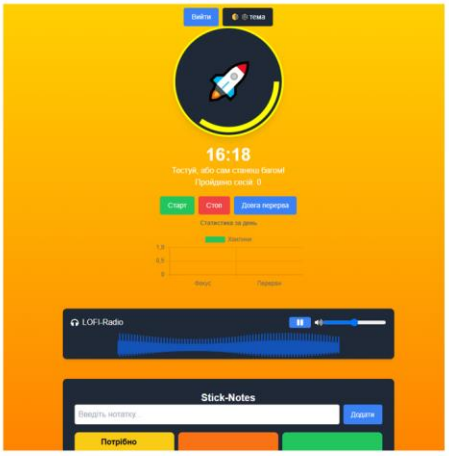
10

ДІАГРАМА КОМПОНЕНТІВ

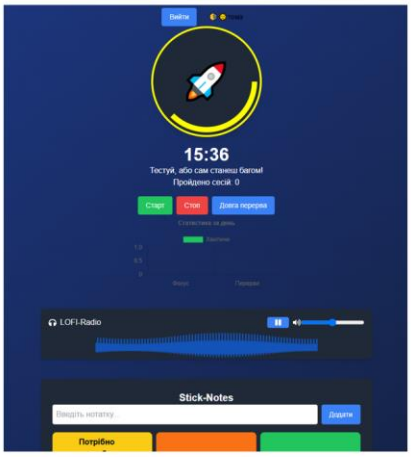


11

ЕКРАННІ ФОРМИ



Основна сторінка веб додатку в світлій темі

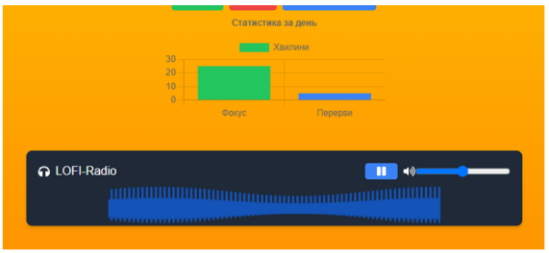


Основна сторінка веб додатку в темній темі

12

ЕКРАННІ ФОРМИ

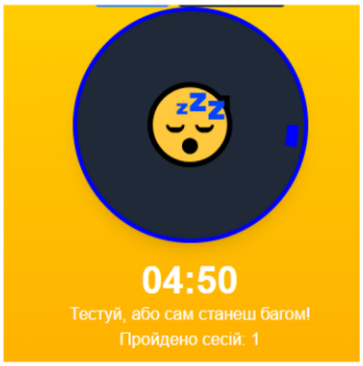
Частина сторінки з функціоналом модулю нотаток у вигляді Sticky-Notes



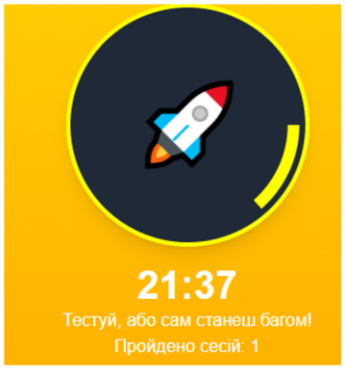
Частина сторінки з функціоналом модулю графіку продуктивності та музичним плеером

13

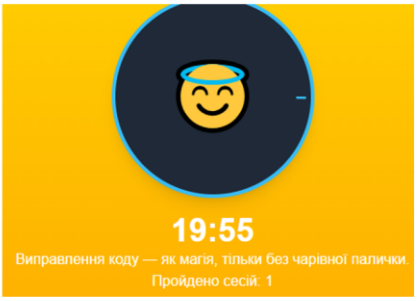
ЕКРАННІ ФОРМИ



Вигляд таймеру під час короткої перерви



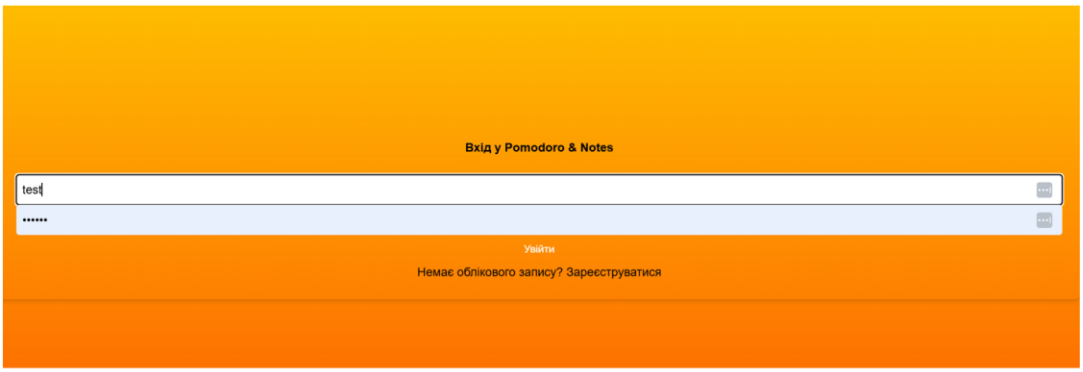
Вигляд таймеру під час робочої “фокус” сесії



Вигляд таймеру під час довгої перерви

14

ЕКРАННІ ФОРМИ



Сторінка з функціоналом логіну та реєстрації

15

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Сковпень М.А., Колодюк А.В. Визначення вимог до веб-системи “Pomodoro & notes” для підтримки ефективності в роботі. // Матеріали конференції “Застосування програмного забезпечення в ІКТ”. 24 квітня 2025 року, ДУІКТ, м. Київ, Україна . К: ДУІКТ, 2025 (с. 123-126)
2. Сковпень М.А., Колодюк А.В. ІГРОФІКАЦІЯ В УПРАВЛІННІ ЧАСОМ ПРОГРАМІСТА: POMODORO-СИСТЕМА ВІЗУАЛЬНОЇ МОТИВАЦІЇ. // Матеріали V Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025 року, ДУІКТ, м. Київ, Україна . К: ДУІКТ, 2025 (подано до друку)

16

ВИСНОВКИ

1. Проведено аналіз сучасних підходів до цифрової самоорганізації та досліджено ефективність методики Pomodoro, завдяки чому було обґрунтовано доцільність застосування Pomodoro-таймера для підвищення ефективності робочого часу.

2. Проаналізовано популярні Pomodoro-застосунки (Forest, Focus To-Do, Pomofocus, Minimalist Pomodoro Timer), що дозволило виявити їх ключові недоліки — відсутність системи нотаток, слабку персоналізацію, брак мотиваційного контенту та обмежений функціонал у безкоштовних версіях.

3. Визначено функціональні та нефункціональні вимоги до веб-застосунку, включаючи підтримку Pomodoro-циклів, систему створення й збереження нотаток, статистику сесій, адаптивний інтерфейс, підтримку мотиваційних елементів і безпечну авторизацію.

4. Проведено аналіз програмних засобів, за рахунок якого обрано оптимальні інструменти для розробки веб-застосунку, який підходив би під всі вимоги.

5. Спроєктовано архітектуру веб-застосунку, що складається з незалежних модулів таймера, нотаток, системи мотиваційних цитат, та захищеної авторизації.

6. Реалізовано повноцінний веб-застосунок із розмежуванням клієнтської та серверної логіки. Увесь функціонал було протестовано як вручну, так і шляхом автоматизованого тестування, що підтвердило коректність взаємодії компонентів та стабільність роботи застосунку.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Код з models.py

```
from flask_sqlalchemy import SQLAlchemy # type: ignore

db = SQLAlchemy()

class Note(db.Model):
    __tablename__ = 'notes'
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.String(255), nullable=False)

    def __repr__(self):
        return f"<Note {self.id}: {self.text}>"
class Pomodoro(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    status = db.Column(db.String(20), nullable=False,
default="stopped")  # stopped, running, paused
    start_time = db.Column(db.DateTime, nullable=True)
    end_time = db.Column(db.DateTime, nullable=True)
    duration = db.Column(db.Integer, nullable=False,
default=25)  # У хвилинах

    def to_dict(self):
        return {
            "id": self.id,
            "status": self.status,
            "start_time": self.start_time.isoformat() if
self.start_time else None,
            "end_time": self.end_time.isoformat() if self.end_time
else None,
            "duration": self.duration
        }
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True,
nullable=False)
    password = db.Column(db.String(255), nullable=False)  #
Хешований пароль
```

Код auth.py

```
from flask import Blueprint, request, jsonify
from werkzeug.security import generate_password_hash,
check_password_hash
import jwt
import datetime
from functools import wraps
from database.models import db, User

auth_bp = Blueprint("auth", __name__)

SECRET_KEY =
"3fe0fe102961d2f1184470406b6fe51f7d7b97e8e21ae315f2392
3d82a09e539"


# Декоратор для перевірки JWT-токена
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get("x-access-token")
        if not token:
            return jsonify({"message": "Токен відсутній!"}), 401
        try:
            data = jwt.decode(token, SECRET_KEY,
algorithms=["HS256"])
            current_user =
User.query.filter_by(id=data["user_id"]).first()
        except:
            return jsonify({"message": "Невірний токен!"}), 401
        return f(current_user, *args, **kwargs)
    return decorated


# Реєстрація користувача
@auth_bp.route("/register", methods=["POST"])
def register():
    data = request.get_json()
    username = data.get("username")
    password = data.get("password")

    if not username or not password:
        return jsonify({"message": "Введіть ім'я користувача та
пароль!"}), 400

    existing_user =
User.query.filter_by(username=username).first()
    if existing_user:
        return jsonify({"message": "Користувач з таким ім'ям
вже існує!"}), 400

    hashed_password = generate_password_hash(password,
method="pbkdf2:sha256")
    new_user = User(username=username,
password=hashed_password)
    db.session.add(new_user)
    db.session.commit()

    return jsonify({"message": "Користувач зареєстрований
успішно!"})
```

```
# Вхід користувача
@auth_bp.route("/login", methods=["POST"])
def login():
    data = request.get_json()
    username = data.get("username")
    password = data.get("password")

    user = User.query.filter_by(username=username).first()
    if not user or not check_password_hash(user.password,
password):
        return jsonify({"message": "Невірне ім'я користувача
або пароль!"}), 401

    token = jwt.encode(
        {"user_id": user.id, "exp": datetime.datetime.utcnow() +
datetime.timedelta(hours=1)},
        SECRET_KEY,
        algorithm="HS256"
    )

    return jsonify({"token": token})


# Отримати інформацію про поточного користувача
@auth_bp.route("/user", methods=["GET"])
@token_required
def get_user(current_user):
    return jsonify({"id": current_user.id, "username":
current_user.username})
```

Код notes.py

```
from flask import Blueprint, request, jsonify
from database.models import db, Note

notes_bp = Blueprint('notes', __name__)
@notes_bp.route('', methods=['GET'])
def get_notes():
    notes = Note.query.all()
    return jsonify([{'id': note.id, 'text': note.text} for note in
notes])

@notes_bp.route('/<int:note_id>', methods=['DELETE'])
def delete_note(note_id):
    note = Note.query.get(note_id)
    if not note:
        return jsonify({'error': 'Note not found'}), 404
    db.session.delete(note)
    db.session.commit()
    return jsonify({'message': 'Note deleted'})


@notes_bp.route('', methods=['POST'])
def add_note():
    data = request.get_json()
    if not data or 'text' not in data:
        return jsonify({'error': 'Invalid request'}), 400
    new_note = Note(text=data['text'])
    db.session.add(new_note)
    db.session.commit()
    return jsonify({'id': new_note.id, 'text': new_note.text})
```

Код timer.py

```
from flask import Blueprint, jsonify
import time

# Ініціалізація Blueprint
timer_bp = Blueprint('timer', __name__)

# Глобальні змінні для таймера
TIMER_PHASES = ["focus", "break"]  # Фази: фокусування
(25 хв) -> перерва (5 хв)
PHASE_DURATIONS = {"focus": 25 * 60, "break": 5 * 60}  #
Тривалість у секундах
current_phase = 0
start_time = None

@timer_bp.route('/api/timer/', methods=['GET'])
def get_timer():
    """Повертає поточну фазу таймера."""
    global start_time, current_phase
    if start_time:
        elapsed = time.time() - start_time
        if elapsed >=
PHASE_DURATIONS[TIMER_PHASES[current_phase]]:
            current_phase = (current_phase + 1) %
len(TIMER_PHASES)
            start_time = time.time()
    return jsonify({"phase": TIMER_PHASES[current_phase]})

@timer_bp.route('/api/timer/start', methods=['POST'])
def start_timer():
    """Запускає таймер."""
    global start_time
    start_time = time.time()
    return jsonify({"message": "Таймер запущено"})

@timer_bp.route('/api/timer/reset', methods=['POST'])
def reset_timer():
    """Скидає таймер до початкового стану."""
    global start_time, current_phase
    start_time = None
    current_phase = 0
    return jsonify({"message": "Таймер скинуто"})
```

Код app.py

```
from flask import Flask
from flask_cors import CORS
from database.models import db  # Імпортуємо базу даних
from routes.notes import notes_bp
from routes.timer import timer_bp
from routes.auth import auth_bp

app = Flask(__name__)

# Налаштування бази даних
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] =
False

# Ініціалізуємо базу
db.init_app(app)

# Включаємо CORS для всіх маршрутів
#CORS(app, resources={r"/*": {"origins": "*"}},
supports_credentials=True)
CORS(app, resources={r"/notes/*": {"origins": "*"}})

# Підключення маршрутів
app.register_blueprint(notes_bp, url_prefix='/notes')

app.register_blueprint(timer_bp, url_prefix='/timer')

# Створення таблиць
with app.app_context():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True, port=5000)

app.register_blueprint(auth_bp, url_prefix="/auth")

CORS(app, resources={r"/*": {"origins":
"http://localhost:5173"}})
```

Код pages/Dashboard.jsx

```
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import MusicPlayer from "./MusicPlayer";
import { Bar } from "react-chartjs-2";
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  BarElement,
  Title,
  Tooltip,
  Legend,
} from "chart.js";

ChartJS.register(CategoryScale, LinearScale, BarElement,
Title, Tooltip, Legend);

const WORK_TIME = 25 * 60;
const BREAK_TIME = 5 * 60;
const LONG_BREAK_TIME = 20 * 60;
const COLORS = ["#facc15", "#f97316", "#22c55e",
"#3b82f6", "#eb607e"];
const QUOTES = [
  "Код — це поезія, написана для машин, прочитана
людьми.",
  "Тестуй, або сам станеш багом!",
  "Слава тим, хто дебажив без кави!",
  "У кожному багу живе нова фіча.",
  "Виправлення коду — як магія, тільки без чарівної
палички.",
  "Чим більше ти кодуєш — тим менше ти боїшся
помилок.",
  "Push — pray — profit.",
  "Стартуєш Pomodoro — стартуєш перемогу."
];

const PomodoroTimer = () => {
  const [timeLeft, setTimeLeft] = useState(WORK_TIME);
  const [isRunning, setIsRunning] = useState(false);
  const [phase, setPhase] = useState("work");
  const [progress, setProgress] = useState(283);
  const [quote, setQuote] = useState("");
  const [sessionCount, setSessionCount] = useState(0);
  const [focusTime, setFocusTime] = useState(0);
  const [breakTime, setBreakTime] = useState(0);

  const getPhaseDuration = () => {
    switch (phase) {
      case "work": return WORK_TIME;
      case "break": return BREAK_TIME;
      case "longBreak": return LONG_BREAK_TIME;
      default: return WORK_TIME;
    }
  };

  useEffect(() => {
    if (!isRunning) return;

    const timer = setInterval(() => {
      setTimeLeft((prev) => {
        if (prev <= 0) {
          clearInterval(timer);

          if (phase === "work") {
            const newSessionCount = sessionCount + 1;
            setSessionCount(newSessionCount);
            setFocusTime((prev) => prev + WORK_TIME);
            const nextPhase = newSessionCount % 4 === 0 ?
"longBreak" : "break";
            setPhase(nextPhase);
```

```
          const nextDuration = nextPhase === "longBreak" ?
LONG_BREAK_TIME : BREAK_TIME;
          setTimeLeft(nextDuration);
          showNotification(nextPhase === "longBreak" ? "Час
на довгу перерву!" : "Час для перерви!");
        } else {
          setBreakTime((prev) => prev + getPhaseDuration());
          setPhase("work");
          setTimeLeft(WORK_TIME);
          showNotification("Час повернутися до роботи!");
        }

        setProgress(283);
        return getPhaseDuration();
      }

      setProgress((prevProgress) => Math.max(prevProgress -
283 / getPhaseDuration(), 0));
      return prev - 1;
    });
  }, 1000);

  return () => clearInterval(timer);
 }, [isRunning, phase, sessionCount]);

 const showNotification = (message) => {
  if (Notification.permission === "granted") {
   new Notification(message);
  } else if (Notification.permission !== "denied") {
   Notification.requestPermission().then((permission) => {
    if (permission === "granted") {
     new Notification(message);
    }
   });
  }
 };

 const handleStart = () => {
  const random = Math.floor(Math.random() *
QUOTES.length);
  setQuote(QUOTES[random]);
  setIsRunning(true);
 };

 const startLongBreak = () => {
  setPhase("longBreak");
  setTimeLeft(LONG_BREAK_TIME);
  setProgress(283);
  setIsRunning(true);
 };

 const borderColor = phase === "break" ? "blue" : phase ===
"longBreak" ? "#38bdf8" : "yellow";
 const icon = phase === "break" ? "😌" : phase ===
"longBreak" ? "😳" : "🚀";

 const data = {
  labels: ["Фокус", "Перерви"],
  datasets: [
   {
    label: "Хвилини",
    data: [Math.floor(focusTime / 60), Math.floor(breakTime /
60)],
    backgroundColor: ["#22c55e", "#3b82f6"],
   },
  ],
 };

 const options = {
  responsive: true,
  plugins: {
   legend: { position: "top" },
   title: { display: true, text: "Статистика за день" },
  },
 };

 return (
  <div className="flex flex-col items-center mb-10 relative
z-10">
   <div className="relative w-56 h-56 flex items-center
justify-center rounded-full bg-gray-800 shadow-lg border-4"
style={{ borderColor }}>
    <span className={`text-7xl ${phase === "work" ?
"shake" : ""}`}>{icon}</span>
    <svg className="absolute w-full h-full" viewBox="0 0
100 100">
     <circle cx="50" cy="50" r="45" fill="none"
stroke={borderColor} strokeWidth="5"
strokeDasharray="283" strokeDashoffset={progress} />
    </svg>
   </div>
   <p className="text-4xl font-bold mt-4">
    {String(Math.floor(timeLeft / 60)).padStart(2,
"0")}:{String(timeLeft % 60).padStart(2, "0")}
   </p>
   <p className="mt-2 italic text-sm text-center max-w-
xs">{quote}</p>
   <p className="mt-1 text-sm">Пройдено сесій:
{sessionCount}</p>
   <div className="mt-4 space-x-2">
    <button className="bg-green-500 px-4 py-2 rounded
text-white" onClick={handleStart}>Старт</button>
    <button className="bg-red-500 px-4 py-2 rounded text-
white" onClick={() => { setIsRunning(false);
setTimeLeft(WORK_TIME); setPhase("work");
setProgress(283); setQuote(""); }}>Стоп</button>
    <button className="bg-blue-500 px-4 py-2 rounded text-
white" onClick={startLongBreak}>Довга перерва</button>
   </div>
   <div className="w-full max-w-md mt-6">
    <Bar data={data} options={options} />
   </div>
  </div>
 );
};

const Notes = () => {
 const [notes, setNotes] = useState([]);
 const [newNote, setNewNote] = useState("");
 const [checked, setChecked] = useState({});

 useEffect(() => {
  fetch("http://127.0.0.1:5000/notes")
   .then((res) => res.json())
   .then((data) => setNotes(data))
   .catch((error) => console.error("Помилка при
завантаженні нотаток:", error));
 }, []);

 const addNote = () => {
  if (!newNote.trim()) return;
  fetch("http://127.0.0.1:5000/notes", {
   method: "POST",
   headers: { "Content-Type": "application/json" },
   body: JSON.stringify({ text: newNote }),
  })
   .then((res) => res.json())
   .then((data) => {
```

```
      setNotes([...notes, data]);
      setNewNote("");
    })
    .catch((error) => console.error("Помилка при додаванні нотатки:", error));
 };

 const deleteNote = (id) => {
  fetch(`http://127.0.0.1:5000/notes/${id}`, { method: "DELETE" })
    .then(() => setNotes(notes.filter((note) => note.id !== id)))
    .catch((error) => console.error("Помилка при видаленні нотатки:", error));
 };

 const toggleCheck = (id) => {
  setChecked((prev) => ({ ...prev, [id]: !prev[id] }));
 };

 return (
  <div className="w-full max-w-2xl bg-gray-800 p-6 rounded-lg shadow-md">
    <h2 className="text-xl font-bold mb-1">Stick-Notes</h2>
    <div className="flex space-x-2 mb-4">
     <input className="w-full p-2 rounded text-black" type="text" placeholder="Введіть нотатку..." value={newNote} onChange={(e) => setNewNote(e.target.value)} />
     <button className="bg-blue-500 px-4 py-2 rounded text-white" onClick={addNote}>Додати</button>
    </div>
    <div className="grid grid-cols-3 gap-2">
     {notes.map((note, index) => (
      <div key={note.id} className="p-3 rounded-lg text-black uppercase break-words flex justify-between items-center w-full min-h-[50px] px-4 py-2" style={{ backgroundColor: COLORS[index % COLORS.length] }}>
       <div className="flex items-center gap-2">
        <input type="checkbox" checked={checked[note.id] || false} onChange={() => toggleCheck(note.id)} />
        <p className="flex-1 font-bold uppercase">{note.text}</p>
       </div>
       <button className="ml-2 px-2 py-1 text-white bg-red-500 rounded" onClick={() => deleteNote(note.id)}>✕</button>
      </div>
     ))}
    </div>
  </div>
 );
};

const Dashboard = () => {
 const navigate = useNavigate();
 const [isDarkTheme, setIsDarkTheme] = useState(true);

 useEffect(() => {
  const storedTheme = localStorage.getItem("theme");
  if (storedTheme === "light") setIsDarkTheme(false);
 }, []);

 const toggleTheme = () => {
  setIsDarkTheme((prev) => {
   const next = !prev;
   localStorage.setItem("theme", next ? "dark" : "light");
   return next;
  });
 };

 const handleLogout = () => {
  localStorage.removeItem("token");
  navigate("/login");
 };

 return (
  <div className={`flex flex-col items-center text-white p-4 relative min-h-screen transition-all duration-500 ${isDarkTheme ? "bg-gradient-to-br from-blue-900 to-gray-900" : "bg-gradient-to-br from-yellow-300 to-orange-400 text-white"}`}>
    <div className="top-4 right-4 flex space-x-2">
     <button className="bg-blue-500 px-4 py-2 rounded text-white" onClick={handleLogout}>Вийти</button>
     <button className={`px-4 py-2 rounded ${isDarkTheme ? "bg-yellow-300 text-black" : "bg-gray-800 text-white"}`} onClick={toggleTheme}>
      {isDarkTheme ? "◑⚙тема" : "◑●тема"}
     </button>
    </div>
    <PomodoroTimer />
    <MusicPlayer />
    <Notes />
  </div>
 );
};

export default Dashboard;
```

Код pages/MusicPlayer.jsx

```
import React, { useRef, useState } from "react";
import { FaHeadphones, FaPlay, FaPause, FaVolumeUp } from "react-icons/fa";
const MusicPlayer = () => {
 const audioRef = useRef(null);
 const [isPlaying, setIsPlaying] = useState(false);
 const [volume, setVolume] = useState(0.5);
 const togglePlay = () => {
  if (isPlaying) {
   audioRef.current.pause();
  } else {
   audioRef.current.play();
  }
  setIsPlaying(!isPlaying);
 };
 const handleVolumeChange = (e) => {
  const newVolume = parseFloat(e.target.value);
  setVolume(newVolume);
```

```
    audioRef.current.volume = newVolume;
  };
  return (
    <div className="w-full max-w-2xl bg-gray-800 p-4
rounded-lg shadow-md mb-10">
      <div className="flex justify-between items-center mb-4">
        <div className="flex items-center space-x-2 text-white">
          <FaHeadphones />
          <span className="font-semibold">LOFI-Radio</span>
        </div>
        <div className="flex items-center space-x-2">
          <button
            className="bg-blue-500 hover:bg-blue-600 text-white
px-3 py-1 rounded"
            onClick={togglePlay}
          >
            {isPlaying ? <FaPause /> : <FaPlay />}
          </button>
          <div className="flex items-center">
            <FaVolumeUp className="mr-1 text-white" />
            <input
              type="range"
              min="0"
              max="1"
              step="0.01"
              value={volume}
              onChange={handleVolumeChange}
              className="w-20"
            />
          </div>
        </div>
      </div>
      {/* Хвилі */}
      {isPlaying && (
        <div className="wave-wrapper">
          {[...Array(150)].map((_, i) => (
            <div key={i} className="wave" style={{
animationDelay: `${i * 0.02}s` }}></div>
          ))}
        </div>
      )}
      <audio
        ref={audioRef}
        loop
        src="https://streams.fluxfm.de/Chillhop/mp3-
128/streams.fluxfm.de/"
      />
    </div>
  );
};
export default MusicPlayer;
```

Код pages/Login.jsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";

function Login() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [isRegister, setIsRegister] = useState(false);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();

    const url = isRegister
      ? "http://127.0.0.1:5000/auth/register"
      : "http://127.0.0.1:5000/auth/login";

    try {
      const res = await fetch(url, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ username, password }),
      });

      const data = await res.json();

      if (res.ok) {
        if (isRegister) {
          alert("Реєстрація успішна! Тепер увійдіть.");
          setIsRegister(false);
        } else if (data.token) {
          localStorage.setItem("token", data.token);
          navigate("/");
        } else {
          alert("Невірне ім’я користувача або пароль.");
        }
      } else {
        alert(data.message || "Помилка");
      }
    } catch (err) {
      console.error("Помилка:", err);
      alert("Помилка. Спробуйте пізніше.");
    }
  };

  return (
    <div className="flex flex-col items-center justify-center
min-h-screen bg-gradient-to-br from-yellow-300 to-orange-400
text-black">
      <h1 className="text-3xl font-bold mb-6">
        {isRegister ? "Реєстрація" : "Вхід у Pomodoro &
Notes"}
      </h1>
      <form
        onSubmit={handleSubmit}
        className="w-full max-w-xs bg-white p-6 rounded-lg
shadow-md space-y-4"
      >
        <input
          type="text"
          placeholder="Ім’я користувача"
          className="w-full p-2 border rounded"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
        <input
          type="password"
          placeholder="Пароль"
          className="w-full p-2 border rounded"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <button
          type="submit"
          className="w-full bg-yellow-500 text-white py-2
rounded hover:bg-yellow-600"
        >
          {isRegister ? "Зареєструватися" : "Увійти"}
        </button>
        <p
```

```
        className="text-sm text-center text-blue-600 cursor-
pointer"
        onClick={() => setIsRegister(!isRegister)}
       >
        {isRegister ? "Вже є обліковий запис? Увійти" :
"Немає облікового запису? Зареєструватися"}
       </p>
     </form>
   </div>
  );
}

export default Login;
```