Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВЕБЗАСТОСУНОК ДЛЯ ПЛАНУВАННЯ ЗАВДАНЬ З ІГРОВИМ ІНТЕРФЕЙСОМ»

08-54.БКР.019.00.000 ПЗ

Виконав: студент 4 курсу, групи 1КІ-21б
спеціальності
123 Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
освітня програма «Комп'ютерна інженерія»
Шум Ю. Р.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ОТ
Савицька Л. А.
(прізвище та ініціали)

Рецензент: к.т.н., доцент каф.ПЗ
Хошаба О. М.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«13» червня 2025 р.

Вінниця 2025

Вінницький національний технічний університет

Факультет Інформаційних технологій та комп'ютерної інженерії

Кафедра Обчислювальної техніки

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 - Інформаційні технології

Спеціальність 123 - «Комп'ютерна інженерія»

Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н.проф. _____Азаров О. Д.
«21» березня 2025 року

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

Шуму Юрію Романовичу

1 Тема роботи «Вебзастосунок для планування завдань з ігровим інтерфейсом», керівник роботи к.т.н., доц. Савицька Л. А., затверджені наказом ВНТУ від «20» березня 2025 року №97.

2 Термін подання студентом роботи: 13.05.2025 року

3 Вихідні дані до роботи: вибір технологій фронтенду та бекенду, параметри тестування модулів і навчальні матеріали з веброзробки і UX.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз предметної області, аналіз аналогічних систем; вибір та обгрунтування інструментів, розробка вебзастосунку, персоналізація, тестування.

5 Перелік графічного матеріалу: схема логічної структури бази даних (ERD), схема логіки API-запитів між клієнтом і сервером. Перелік ілюстративного матеріалу: сторінка зі завданням, сторінка профілю користувача, сторінка входу і регестрації.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Спеціальна частина	Савицька Л. А., доцент кафедри ОТ	21.03.25	21.03.25
Нормоконтроль	Швець С.І., Асис. каф. ОТ	14.05.25	30.05.25

7 Дата видачі завдання «21» березня 2025 р.

8 Календарний план виконання БКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Постановка задачі	21.03.25	27.03.25	виконано
2	Огляд і класифікація мов програмування	28.03.25	04.04.25	виконано
3	База даних та середовище розробки	05.04.25	12.04.25	виконано
4	Особливості роботи аналогових таксменеджерів	13.04.25	27.04.25	виконано
5	Тестування вебзастонку	28.04.25	09.05.25	виконано
6	Оформлення пояснювальної записки та ілюстративного матеріалу	10.05.25	13.05.25	виконано
7	Попередній захист БКР	13.05.25	13.05.25	виконано

Студент _____ (підпис) Шум Ю. Р.

Керівник роботи _____ (підпис) Савицька Л. А.

АНОТАЦІЯ

УДК 004.382.4:621.382

Шум Ю. Р. Вебзастосунок для планування завдань з елементами гейміфікації. Бакалаврський проєкт зі спеціальності 123 — Комп’ютерна інженерія, освітня програма — Комп’ютерна інженерія. Вінниця: ВНТУ, 2025. 69 с.

На укр. мові. Бібліогр.: 20 назв; рис.: 19; таблиць: 1

У бакалаврській кваліфікаційній роботі розроблено сучасний вебзастосунок для планування щоденних завдань із гейміфікованим інтерфейсом, орієнтованим на покращення мотивації та продуктивності користувача. Реалізація здійснена з використанням Nuxt 3 (фреймворк JavaScript), Prisma як ORM для роботи з базами даних, а також сучасних засобів CSS для забезпечення адаптивного дизайну.

Система підтримує персоналізацію інтерфейсу відповідно до потреб користувача, включає динамічне керування завданнями, нагороди за виконані дії, прогрес-бари та інші гейміфіковані елементи. Описано архітектуру застосунку, структуру даних, принципи маршрутизації та компонування. Продемонстровано практичну реалізацію з урахуванням потреб реальних користувачів.

Проєкт поєднує як теоретичне обґрунтування, так і практичну реалізацію, що робить його корисним як у навчальному процесі, так і при створенні сучасних інтерактивних вебсервісів.

Ключові слова: вебзастосунок, JavaScript, Nuxt 3, Prisma, CSS, адаптивний дизайн, персоналізація, гейміфікація, планування завдань.

ABSTRAT

UDC 004.382.4:621.382

Shum Y. R. Web application for task planning with elements of gamification. Bachelor's project in specialty 123 - Computer Engineering, educational program - Computer Engineering. Vinnytsia: VNTU, 2025. 69 c.

In Ukrainian. Bibliogr.: 20 titles; figs: 19; tables: 1

In the bachelor's qualification work, a modern web application for planning daily tasks with a gamified interface aimed at improving user motivation and productivity was developed. The implementation was carried out using Nuxt 3 (JavaScript framework), Prisma as an ORM for working with databases, and modern CSS tools to ensure responsive design.

The system supports the personalization of the interface according to the user's needs, includes dynamic task management, rewards for completed actions, progress bars, and other gamified elements. The application architecture, data structure, routing and layout principles are described. The practical implementation is demonstrated, taking into account the needs of real users.

The project combines both theoretical justification and practical implementation, which makes it useful both in the educational process and in the creation of modern interactive web services.

Keywords: web application, JavaScript, Nuxt 3, Prisma, CSS, responsive design, personalization, gamification, task scheduling.

ЗМІСТ

ВСТУП

У час, коли ритм життя змушує нас постійно адаптуватися, приймати десятки рішень щодня та виконувати безліч завдань, планування перестає бути просто корисною навичкою — воно перетворюється на необхідність. Людина, яка вміє керувати своїм часом, має значно більше шансів досягти цілей, зберігаючи при цьому емоційний баланс і продуктивність. Цифрові інструменти самоорганізації стали невід'ємною частиною життя. Поширення отримали різноманітні застосунки для ведення списків справ, планування дня, тижня або навіть року. Проте багато з них виконують лише технічну функцію — дають змогу фіксувати завдання, але не сприяють внутрішній мотивації. Саме тому виникає потреба у засобах, які поєднують корисність із емоційною залученістю.

Одним із таких напрямів стала гейміфікація — тобто впровадження ігрових елементів у неігрові процеси. Ідея полягає в тому, щоб перенести механіки, які роблять ігри захопливими (наприклад, накопичення балів, рівні, нагороди), у сферу реального життя — зокрема, у планування завдань. Це дозволяє користувачеві не просто «закрити справу», а відчути задоволення від досягнення, що підсилює бажання рухатися далі. У наш час особливо актуально створювати саме такі платформи, які не лише зберігають інформацію, а й формують звичку до самоорганізації. Це особливо важливо для молоді, яка часто шукає нові підходи до власної ефективності, не сприймаючи класичних методів у чистому вигляді. Тому поєднання сучасних технологій веброзробки з елементами гейміфікації — це не просто функціональний виклик, а водночас спроба змінити підхід до взаємодії людини з її завданнями.

Актуальність обраної теми зумовлена потребою в ефективних цифрових засобах самоорганізації, які поєднують планування із психологічною підтримкою користувача. У світі, де інформаційне перевантаження та прокрастинація є типовими проблемами, впровадження гейміфікованих елементів у повсякденні системи планування допомагає не лише

організовувати справи, а й підтримувати мотивацію, створювати відчуття прогресу та досягнення. Сучасний користувач очікує від застосунку не лише функціональності, а й емоційного залучення, гнучкості та персоналізації — саме цим вимогам відповідає розроблений у межах цієї роботи вебзастосунок.

Метою роботи є створення вебзастосунку для планування завдань із гейміфікованим інтерфейсом, який би поєднував зручність та інтуїтивність класичних планерів із ігровими механіками (досвід, рівні, монети, нагороди), здатними підтримувати мотивацію користувача у довгостроковій перспективі.

Об’єктом роботи є процес цифрового планування особистих завдань і організації часу користувача.

Предметом роботи є принципи розробки вебзастосунку, що поєднує класичне планування завдань із гейміфікаційними елементами, та особливості їх реалізації у контексті покращення користувацького досвіду.

Апробація результату бакалаврської кваліфікаційної роботи здійснена у доповіді на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН- 2025)» [1]:

Шум Ю. Р., Савицька Л. А. Вебзастосунок для планування завдань з ігровим інтерфейсом // Матеріали всеукраїнської науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців — Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25398

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ГЕЙМІФІКОВАНИХ РІШЕНЬ У ПЛАНУВАННІ ЗАВДАНЬ

1.1 Що таке системи планування завдань і навіщо вони потрібні?

Сучасне життя вимагає від людини вміння керувати часом, контролювати власні дії та швидко адаптуватися до змін. Усе більше людей у повсякденній діяльності стикаються з необхідністю вести облік справ, подій, зустрічей, цілей, дедлайнів. У такому контексті системи планування завдань стають не просто допоміжними інструментами, а ключовими елементами в організації особистого та професійного простору.

Системи планування завдань — це програмні рішення, що дозволяють користувачам створювати, редагувати, сортувати та контролювати перелік справ. Вони виконують роль цифрового асистента, який не лише нагадує про заплановане, а й дозволяє аналізувати продуктивність, визначати пріоритети та краще розуміти структуру навантаження. Завдяки таким інструментам користувачі можуть приймати більш зважені рішення, уникати перевантажень і краще планувати свій час.

Класичні планувальники здебільшого реалізують базовий функціонал: створення завдань, встановлення термінів, пріоритизацію, категоризацію, нагадування. Деякі підтримують інтеграцію з календарем, сповіщеннями, командною роботою. Однак із розвитком цифрових технологій очікування користувачів значно зросли — тепер їм недостатньо простої таблиці з датами.

У контексті сьогодення система планування має бути динамічною, гнучкою, інтерактивною. Вона повинна відповідати не лише утилітарним потребам, а й психологічним. Наприклад, підтримка регулярності виконання завдань, мотивація користувача повертатись щодня, візуалізація прогресу — усе це формує емоційний зв’язок із застосунком.

Особливо важливими ці функції стають в умовах віддаленої роботи, навчання, фрилансу, де користувач сам повинен структурувати свій час. У

таких випадках планер стає не просто списком справ, а опорою, яка допомагає підтримувати ритм, не втрачати фокус і досягати поставлених цілей.

Зокрема, молоді користувачі — школярі, студенти, початківці у сфері IT або творчих професій — часто стикаються з проблемою відкладення справ («прокрастинація»). У цьому випадку класичні системи не завжди ефективні, оскільки не створюють позитивного підкріплення або відчуття досягнення. Саме тому все більше уваги приділяється гейміфікації — впровадженню ігрових механік у неігрові системи.

Гейміфіковане планування — це наступний етап розвитку цифрових інструментів організації. Воно базується на простих, але потужних психологічних тригерах: прогрес, винагорода, персоналізація, досягнення. Коли користувач виконує завдання — він отримує віртуальну нагороду, досвід, переходить на новий рівень. Це створює відчуття зростання та мотивує не кидати використання системи.

Таким чином, сучасна система планування — це не просто цифрова таблиця чи нотатник. Це інтерактивне, адаптивне середовище, яке має бути водночас функціональним, візуально привабливим і мотивуючим. Поєднання цих характеристик і стало основою для створення вебзастосунку, розробленого в межах цієї роботи.

1.2 Огляд існуючих систем планування завдань.

Щоб зрозуміти, як саме можна поєднати зручність, ефективність і мотивацію в одному інструменті, варто поглянути на ті рішення, які вже існують на ринку. Їх багато, і кожен застосунок має свою філософію, функціонал та підхід до взаємодії з користувачем. Але водночас можна виділити певні закономірності та недоліки, що повторюються.

Одним із найвідоміших інструментів для організації завдань є Todoist. Це простий, але функціональний планувальник, який дозволяє створювати списки справ, ділити їх на проєкти, встановлювати дедлайни та пріоритети. Він легкий у користуванні та має інтуїтивний інтерфейс. Втім, його головний акцент —

саме на ефективності. Гейміфікація в ньому реалізована мінімально: є система «карми» — балів за активність, але вона мало впливає на поведінку користувача. Ось вигляд Todoist на рисунку 1.1.

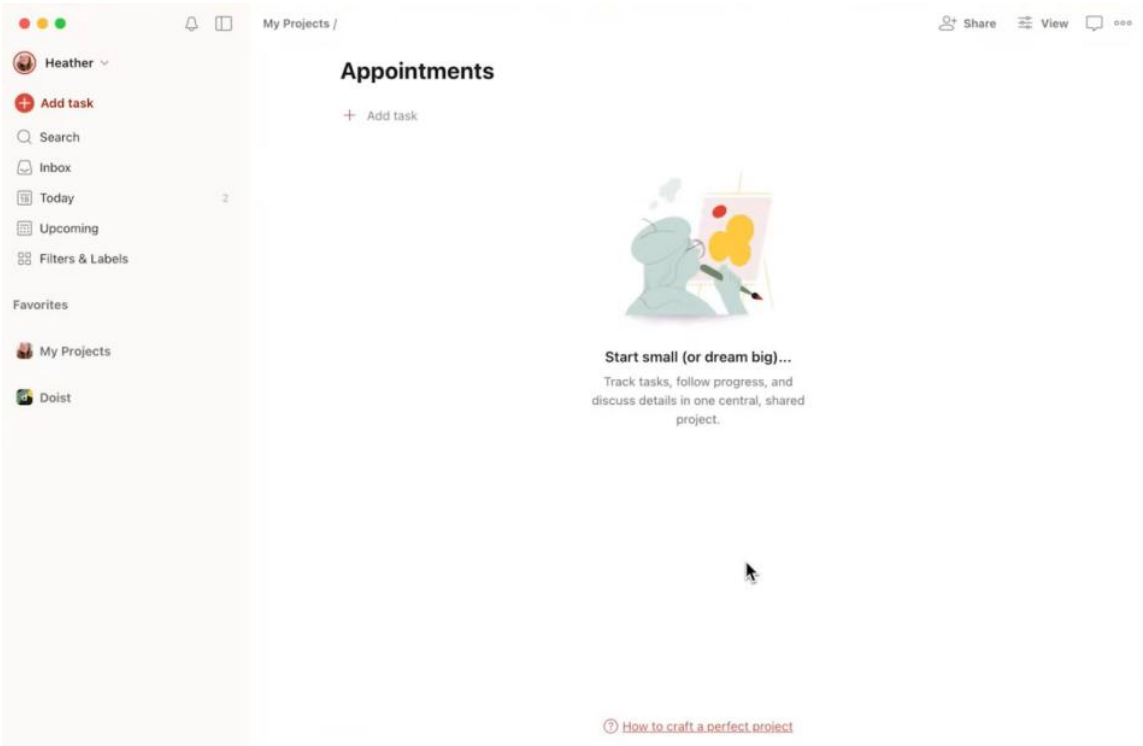

Рисунок 1.1 — Інтернет таскменеджер Todoist

Інший приклад — Trello. Його головна сила — візуальне представлення завдань. Канбан-дошка з картками — це те, що подобається багатьом, особливо в командній роботі. Тут можна призначати завдання, додавати теги, прикріплювати файли, працювати з командою в реальному часі. Але, як і у випадку з Todoist, гейміфікація тут не є основною ідеєю. Це інструмент для менеджменту, а не для залучення. Можна побачити на рисунку 1.2 вигляд Trello.

Є й такі рішення, як Habitica (рис. 1.3), які зосереджені саме на гейміфікації. Цей застосунок перетворює щоденні справи на RPG-гру. Кожне завдання — це «ворог», якого треба перемогти, щоб прокачати свого персонажа. Користувач отримує золото, досвід, предмети. Інтерфейс далеко не

ідеальний і потребує звикання, але саме принцип гри зробив його популярним серед тих, кому звичайні списки не допомагають.

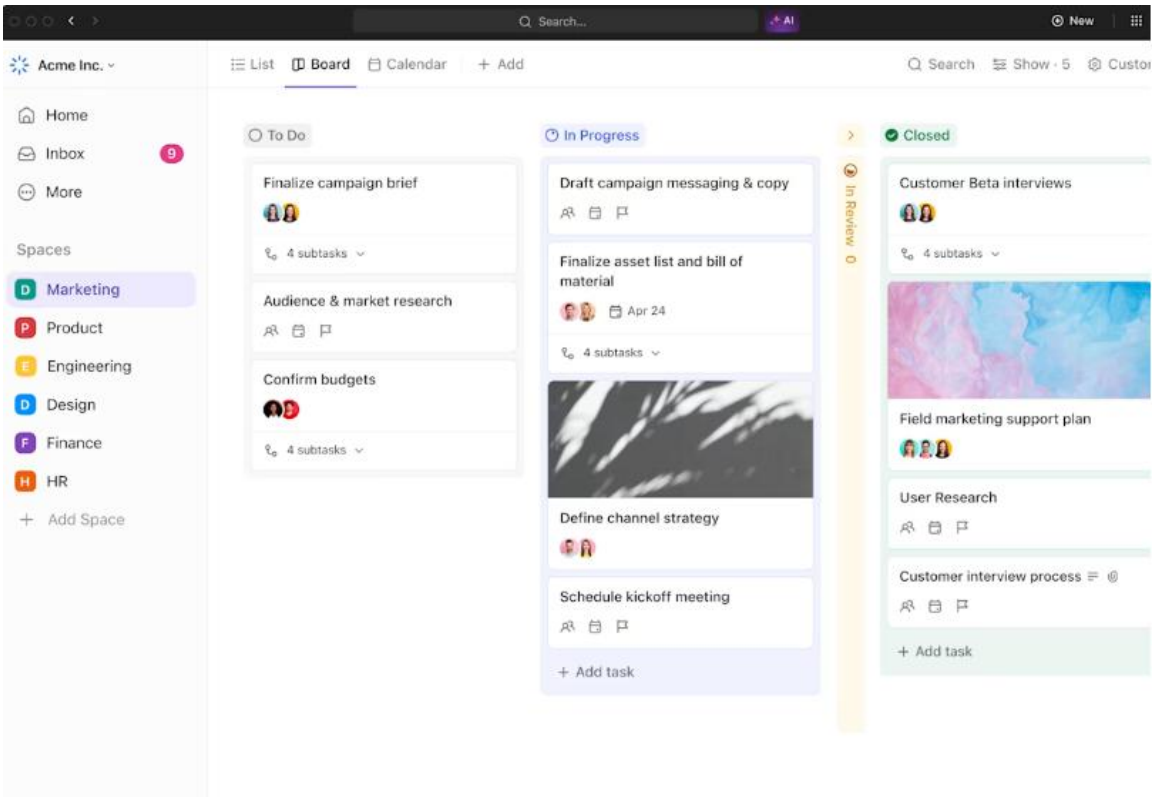

Рисунок 1.2 — Інтерфейс управління проєктами Trello

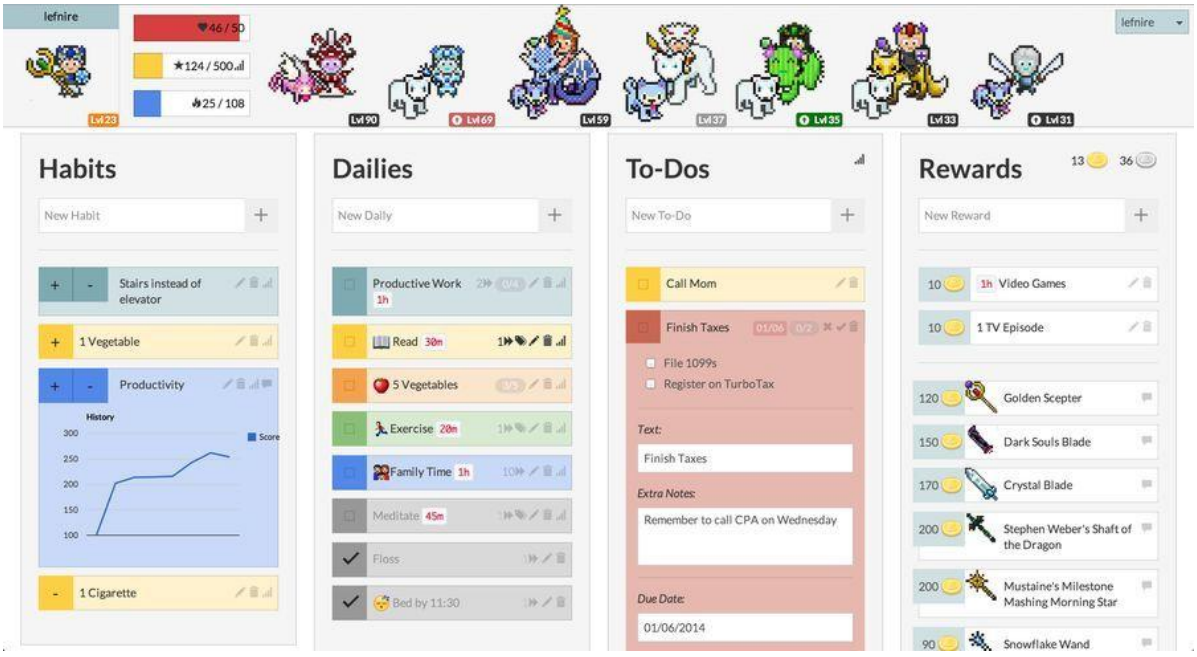

Рисунок 1.3 — Онлайн-менеджер задач з ігровою механікою Habitica

Ще один яскравий приклад — Notion (рис. 1.4), універсальний блокнот із функцією баз даних, який дає змогу створювати свої шаблони, дошки, таблиці, ідеально налаштовуючи систему під себе. Він потужний, але потребує багато часу на освоєння. Для користувача, якому потрібно просто «додати справу і виконати її», це може бути надто складним рішенням.

На основі аналізу можна зробити висновок: або застосунок зручний, але нудний, або цікавий, але надто специфічний. Більшість продуктів обирають щось одне — або ефективність, або залученість. Рідко коли ці два аспекти поєднуються рівноважно. Саме тому і виникає потреба у новому рішенні — такому, що буде водночас зрозумілим, зручним, і водночас підтримуватиме мотивацію через елементи гри. Не обов'язково перетворювати планер на повноцінну гру. Достатньо дрібних деталей: рівні, бали досвіду, нагороди, аватари, тематичні оформлення. Все це вже здатне зробити рутину менш монотонною.

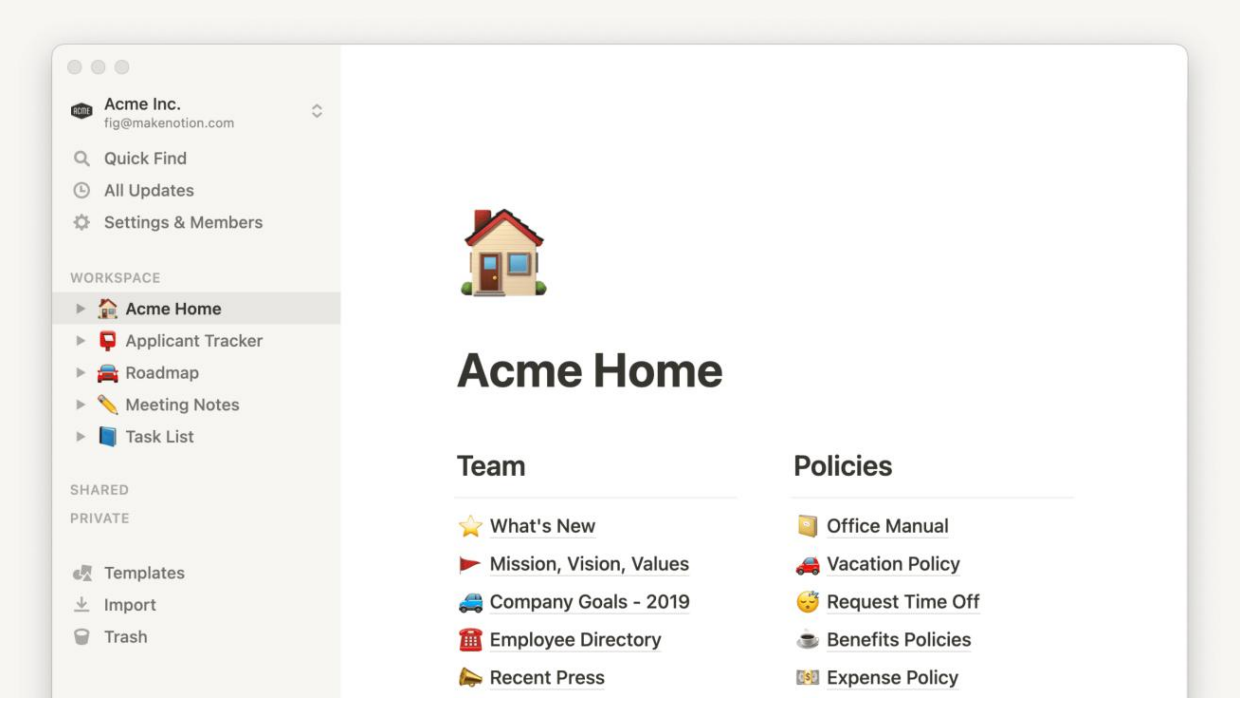

Рисунок 1.4 — універсальний блокнот Notion.

1.3 Гейміфікація як мотиваційний інструмент у цифрових продуктах

Гейміфікація (англ. gamification) — це процес інтеграції ігрових механік у неігровий контекст з метою підвищення залученості, мотивації та утримання уваги користувачів. Застосування гейміфікаційних підходів у цифрових продуктах, зокрема в системах планування, стало помітним трендом останніх років. Цей підхід ґрунтується на психологічних механізмах внутрішнього стимулювання: змагання, досягнення, нагороди, прогрес, персоналізація.

Суть гейміфікації полягає не в тому, щоб зробити продукт грою, а в тому, щоб використати мотиваційні механізми з ігор у неігровому середовищі. Люди природно прагнуть розвитку, визнання та винагород. Коли ці елементи додаються до звичайних функцій, користувачі починають взаємодіяти з системою частіше, довше і з більшою зацікавленістю.

Система досягнень (achievement system) — це набір цілей або рубежів, яких користувач має досягти. Наприклад, виконати 5 завдань поспіль, досягти рівня 10, вперше скористатися магазином. Досягнення можуть мати візуальне представлення у вигляді бейджів, медалей, титулів.

Рівні та досвід (leveling, XP) — кожна дія в системі приносить бали досвіду (XP), які накопичуються. При досягненні певної кількості XP користувач підвищує свій рівень. Це дозволяє відчути зростання, навіть якщо досягнення малопомітні в реальному житті.

Винагороди (rewards) — за активність користувач отримує віртуальні монети, нові теми оформлення, доступ до аватарів чи розблокованих функцій. Нагороди можуть бути миттєвими або відкладеними (наприклад, щотижневі бонуси).

Персоналізація — це можливість налаштувати інтерфейс, обрати аватар, тему або фон, формує відчуття контролю й причетності до системи. Користувач відчуває, що інтерфейс “його”, а не типовий шаблон.

Прогрес-бар або шкала досягнення. Графічна візуалізація прогресу — потужний стимул. Заповнений на 70% прогрес-бар психологічно спонукає завершити дію, щоби «довести до кінця».

Змагання або соціальне порівняння (опційно). У складніших системах можуть бути рейтинги, лідери, командні досягнення. Це стимулює активність за рахунок конкуренції або кооперації.

Ці елементи активують внутрішню мотивацію (intrinsic motivation) — бажання виконати дію не заради зовнішньої користі, а заради відчуття прогресу, завершеності, естетичного задоволення.

У контексті планування завдань гейміфікація виконує дуже конкретну функцію: вона бореться з прокрастинацією, тобто відкладанням справ на потім. Коли виконання завдання приносить не лише викреслений пункт, а й XP, рівень, новий бейдж або монети — користувач сприймає цю дію як досягнення. Це стимулює регулярність.

Варто відзначити, що гейміфікація працює найбільш ефективно тоді, коли вона не є нав’язливою, а делікатно вплетена в інтерфейс. Якщо елементи гри не заважають основному функціоналу, але підсилюють задоволення — це оптимальний сценарій.

У межах цієї роботи всі описані принципи було адаптовано до контексту вебзастосунку для планування завдань. Реалізовано XP, рівні, монети, систему покупок, теми, аватари та базові нагороди. Ці елементи не перевантажують систему, а органічно підтримують мотивацію до щоденного використання.

1.4 Проблеми типових систем та обґрунтування створення нового застосунку

Попри велику кількість існуючих рішень для планування завдань, жодна з доступних систем не задовольняє одночасно всі ключові потреби користувача — функціональність, простота, мотивація, персоналізація та емоційне залучення. Проведений аналіз дозволяє виокремити низку проблем, які

спостерігаються в сучасних інструментах, і на основі цього обґрунтувати доцільність створення нового, збалансованого рішення.

Передусім, багато популярних систем планування фокусуються лише на технічному аспекті — створити завдання, задати дедлайн, встановити нагадування. Такий підхід підходить для людей із високим рівнем самодисципліни, але абсолютно не ефективний для користувачів, які потребують підтримки, стимулів і візуального підкріплення. Відсутність відчуття прогресу чи винагороди знижує бажання повертатися до планера знову.

Інша проблема полягає у відсутності емоційної взаємодії. Класичні списки завдань виглядають суворо, часто одноманітно, що психологічно асоціюється з обов’язком або тиском. Замість того, щоб допомагати, система може викликати відчуття провини за незавершене або перенесене завдання. Це, своєю чергою, призводить до уникання взаємодії з інструментом і, зрештою, до його забуття.

Велика частина систем орієнтована на командну роботу, керування проєктами або офісне середовище. Вони мають розширений функціонал, який для особистого використання не тільки надмірний, а й складний. Користувач змушений витрачати час на вивчення системи, що суперечить самій ідеї — спрощення організації життя.

Гейміфіковані планери, навпаки, часто мають складний інтерфейс, перенасичений термінами з рольових ігор, елементами фентезі чи символікою, зрозумілою лише вузькій аудиторії. Для користувача, який не захоплюється іграми або хоче лише «легке» підкріплення — такі системи є надмірними.

Також варто зазначити, що майже жоден із доступних сервісів не пропонує збалансованої системи персоналізації: вибору теми, аватара, інтерфейсних елементів тощо. Можливість зробити застосунок «своїм», підлаштувати його під настрій чи стиль — одна з базових потреб сучасного користувача.

На основі цих спостережень можна зробити висновок: на ринку бракує легкого, приємного у використанні, але водночас мотивуючого вебзастосунку, який:

— не перевантажує функціями;

— має простий інтерфейс;

— надає базову гейміфікацію (XP, рівні, монети);

— дозволяє персоналізувати вигляд;

— надає швидкий візуальний відгук на дії користувача.

Запропонований у цій роботі застосунок є спробою закрити саме цю нішу. Його головна ідея — перетворити щоденні завдання на гру з простими механіками, що забезпечує природну мотивацію без відчуття примусу. Такий підхід дозволяє користувачу не лише виконувати плани, а й відчувати задоволення від процесу. Це критично важливо в умовах, коли самодисципліна часто підривається інформаційним навантаженням, стресом і втомою.

Таким чином, створення нового вебзастосунку є не просто експериментом, а відповіддю на конкретні запити користувачів, яким бракує мотиваційно-підсиленого інструменту для організації повсякденних справ у доступній, емоційно комфортній та функціональній формі.

2 РОЗГЛЯД І ЗАСТОСУВАННЯ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СУЧАСНОГО ВЕБЗАСТОСУНКУ

2.1 Поняття вебзастосунку та його особливості

Вебзастосунок — це програмне забезпечення, яке працює у веббраузері та забезпечує користувача певним функціоналом або сервісами через інтернет. Його особливість полягає в тому, що він не потребує встановлення на комп’ютер чи мобільний пристрій — достатньо мати доступ до браузера і мережі. На відміну від статичних вебсайтів, які лише відображають інформацію, вебзастосунки мають інтерактивність і часто вимагають авторизації, дозволяють вводити й зберігати дані, взаємодіють із сервером у реальному часі, надають користувачу особистий досвід, схожий на роботу з традиційною програмою.

Сучасні вебзастосунки можуть повністю замінити десктопне або мобільне програмне забезпечення. Приклади вебзастосунків — це поштові сервіси (як-от Gmail), офісні платформи (Google Docs), інтернет-магазини (Rozetka, Amazon), банківські сервіси, системи керування проєктами (Trello, Asana), соціальні мережі, стримінгові платформи, а також внутрішні корпоративні портали.

З технічної точки зору, вебзастосунок складається з клієнтської та серверної частини. Клієнтська частина (frontend) — це те, що бачить користувач у браузері: інтерфейс, кнопки, анімації, таблиці тощо. Вона створюється з використанням таких технологій, як HTML, CSS, JavaScript і сучасні JavaScript-фреймворки на зразок React, Vue, Angular або Svelte. Серверна частина (backend) працює на віддаленому сервері й відповідає за обробку запитів, зберігання та передачу даних, автентифікацію, логіку бізнес-процесів. Вона може бути написана на мовах програмування, як-от Node.js, Python, PHP, Java або Ruby. Зберігання інформації зазвичай відбувається в базах даних — наприклад, PostgreSQL, MySQL, MongoDB.

Вебзастосунки можуть бути односторінковими (SPA — Single Page Application), де сторінка не перезавантажується повністю при переходах, а лише оновлюється потрібний контент за допомогою JavaScript. Це робить застосунок більш динамічним і схожим на нативні програми. Іншим типом є багатосторінкові застосунки (MPA — Multi Page Application), де кожна дія веде до повного завантаження нової сторінки. SPA вважаються ефективнішими для сучасних сервісів із великою кількістю інтерактивності.

Ще однією важливою рисою вебзастосунків є те, що вони кросплатформені — працюють на будь-якому пристрої з браузером, незалежно від операційної системи. Це робить їх зручними для бізнесу, адже не потрібно розробляти окремі версії для Windows, macOS, Android чи iOS. Водночас існують так звані прогресивні вебзастосунки (PWA — Progressive Web Apps), які можна встановити на пристрій, як звичайний додаток, і використовувати навіть офлайн. Вони поєднують зручність вебу з функціональністю нативних програм.

Вебзастосунки можуть бути як відкритими для всіх користувачів (наприклад, онлайн-магазин), так і приватними, з доступом лише для зареєстрованих користувачів або працівників компанії. Вони дозволяють автоматизувати процеси, аналізувати дані, здійснювати комунікацію, вести документацію, обслуговувати клієнтів — залежно від конкретної мети.

З погляду користувача, вебзастосунок — це зручний, доступний і швидкий спосіб взаємодіяти з сервісами. З погляду розробника, це складна система, що потребує знання інтерфейсного дизайну, архітектури даних, безпеки, масштабованості, UX, UI та інтеграції з API. Безпечна авторизація, захист від атак, обробка помилок, оптимізація швидкодії — усе це обов’язкові складові розробки.

Безпека є ключовим аспектом вебзастосунків, особливо з урахуванням відкритості мережі Інтернет. Системи повинні захищати дані користувачів, контролювати доступ і протидіяти поширеним загрозам, таким як SQL-ін’єкції, CSRF та XSS атаки.

З технічної точки зору, фронтенд вебзастосунку часто реалізується за допомогою сучасних JavaScript-фреймворків (React, Vue.js, Angular), які дозволяють створювати ефективний, адаптивний та реактивний інтерфейс. Серверна частина може бути реалізована на базі Node.js, Python (Django, Flask), Ruby on Rails або інших платформах. Взаємодія між фронтендом і бекендом здійснюється через API — найчастіше RESTful або GraphQL.

Для збереження і керування даними використовуються бази даних: реляційні (PostgreSQL, MySQL) або нереляційні (MongoDB). Розгортання та підтримка сучасних вебзастосунків значно спрощуються за допомогою технологій контейнеризації (Docker), а також практик безперервної інтеграції та доставки (CI/CD). На рисунку 2.1 зображенна схема для кращого розуміння як клієнт (браузер) взаємодіє з фронтендом, який обробляє запити через API сервер. Серверна логіка працює з базою даних..

Рисунок 2.1 — Загальна архітектура вебзастосунку.

2.2 Обґрунтування вибору інструментів і технологій

Вибір технологічного стеку для розробки вебзастосунку є одним із найважливіших етапів проєктування інформаційної системи. Він впливає на продуктивність, стабільність, зручність підтримки, масштабованість і безпеку майбутнього продукту.

Для реалізації клієнтської частини обрана платформа Nuxt 3, що базується на Vue.js.

Nuxt 3 — це сучасний фреймворк для створення вебзастосунків на базі Vue 3, який поєднує зручність, гнучкість та потужність новітніх вебтехнологій. Повністю переписаний з нуля, він пропонує розробникам ще більше можливостей завдяки підтримці Composition API, серверного рендерингу,

генерації статичних сайтів, інтеграції з TypeScript та використанню нового серверного рушія Nitro. Завдяки цьому Nuxt 3 адаптується до будь-якого сценарію — від класичних SSR-застосунків до JAMstack-сайтів або серверлес-архітектур.

Основна ідея Nuxt 3 полягає у тому, щоб максимально спростити створення потужних і продуктивних вебдодатків, надаючи розробникам чітку структуру проєкту, автоматичну маршрутизацію, реєстрацію компонентів та інтеграцію серверної логіки через API-файли. Завдяки використанню Vite Nuxt забезпечує блискавичну швидкість розробки з гарячим оновленням, миттєвим перезапуском сервера та швидкою збіркою, хоча також підтримується Webpack для тих, кому це потрібно.

Одним із найбільших переваг фреймворку є те, що він дозволяє використовувати різні стратегії рендерингу на рівні всього застосунку або окремих сторінок: це може бути SSR, коли HTML формується на сервері; CSR — коли рендеринг виконується на боці клієнта; SSG — коли сайт збирається статично під час компіляції; або ISG — коли поєднується попередня генерація з динамічним оновленням. Така гнучкість дозволяє створювати сайти, які ідеально відповідають потребам продуктивності, SEO та взаємодії з користувачем.

Nuxt 3 також повністю підтримує TypeScript без потреби додаткових налаштувань, автоматично генерує типи, забезпечуючи глибоку інтеграцію з редакторами коду. Розробники мають доступ до composables — повторно використовуваних логічних блоків, які допомагають спростити роботу з реактивністю, фетчингом даних і станом програми. Зокрема, функція useAsyncData дозволяє ефективно працювати з асинхронними запитами як на сервері, так і на клієнті, забезпечуючи кешування, повторне використання даних і автоматичне оновлення при зміні маршруту.

Екосистема Nuxt 3 включає велику кількість модулів, що дозволяють швидко підключити сторонні інструменти, як-от оптимізація зображень,

підтримка Markdown-контенту, SEO-налаштування, автентифікація, i18n та інші.

Також у фреймворку реалізовано Nuxt DevTools — інтерактивний інтерфейс для розробників, який дозволяє в реальному часі аналізувати сторінки, модулі, performance, кеш і маршрути. З точки зору SEO та продуктивності Nuxt 3 має всі необхідні засоби: підтримку мета-тегів, Open Graph, генерацію sitemap, оптимізацію зображень, лейзі-лоадинг, tree-shaking і мінімізацію JavaScript. Завдяки універсальному рушію Nitro застосунок легко розгортається на різних платформах — Vercel, Netlify, Cloudflare, AWS Lambda, Docker або навіть у вигляді статичного сайту без сервера. Nuxt 3 не просто інструмент, а ціла платформа для створення високоякісних Vue-застосунків, яка дозволяє розробникам зосередитися на логіці, не витрачаючи час на зайву конфігурацію. Його продумана архітектура, продуктивність і простота роблять його ідеальним вибором для створення сучасних, масштабованих і персоналізованих вебдодатків.

Вона поєднує переваги server-side rendering (SSR) і одночасно дозволяє створювати SPA (односторінкові додатки). SSR забезпечує швидке завантаження сторінок та покращує SEO, що є важливим для залучення користувачів і ранжування в пошукових системах. Nuxt 3 має модульну структуру, що сприяє масштабованості та легкій підтримці коду.

Для стилізації застосовано Tailwind CSS — це сучасний CSS-фреймворк, заснований на утилітарному підході до стилізації. Замість створення окремих класів для кожного елемента, розробник безпосередньо в HTML додає десятки маленьких класів, кожен з яких відповідає за окремий стиль — від кольору до відступу, розміру, розміщення, тіней, шрифтів тощо. Такий підхід дозволяє будувати інтерфейси швидко, гнучко та без написання кастомного CSS. Tailwind не нав’язує жодного дизайну — він не має готових компонентів у стилі Bootstrap, зате дає повну свободу дизайну з нуля. Класи мають зрозумілу і передбачувану структуру: наприклад, bg-blue-500 означає синій фон певного

насичення, p-4 — паддінг, text-xl — великий розмір шрифту, rounded-lg — заокруглені кути.

Tailwind ідеально підходить для роботи в екосистемі компонентних фреймворків типу React, Vue чи Next.js, де кожен компонент має ізольовану структуру. Він також чудово працює з системами збірки на кшталт Vite, Webpack або Parcel. Окрему роль у фреймворку відіграє файл tailwind.config.js, який дозволяє повністю налаштовувати палітру кольорів, розміри, брейкпоінти, шрифти та додавати свої кастомні утиліти. Завдяки вбудованій системі purge, у фінальний CSS потрапляють тільки ті стилі, які реально використовуються в проєкті, тому навіть великі сайти з Tailwind можуть мати мінімальний об’єм CSS.

Одна з найсильніших сторін Tailwind — гнучка робота з адаптивністю, темною темою та станами на кшталт hover, focus або active. Наприклад, можна легко задати стиль, який діє лише на мобільних або тільки при наведенні. Tailwind також підтримує анімації, transitions, grid, flexbox, псевдокласи і кастомні плагіни. За потреби можна використовувати директиву @apply, яка дозволяє створювати свої класи на базі наявних утиліт. Хоча початково Tailwind виглядає як безлад з класів у HTML, з часом структура стає логічною і читається, як опис дизайну.

Це значно пришвидшує процес розробки і забезпечує консистентність візуального оформлення.

Серверна частина реалізована у вигляді API-ендпоінтів, організованих у рамках Nuxt, що дозволяє підтримувати логіку бізнес-процесів і керування даними. Для взаємодії з базою даних застосовується Prisma ORM — інструмент, який полегшує роботу з реляційними базами даних, надаючи зручний та типізований API.

Для зберігання даних обрана реляційна база даних PostgreSQL, відома своєю надійністю, гнучкістю та підтримкою складних SQL-запитів. PostgreSQL — це потужна, об'єктно-реляційна система керування базами даних з відкритим вихідним кодом, яка відома своєю надійністю, масштабованістю,

гнучкістю та відповідністю стандартам SQL. Вона активно розвивається з 1986 року, спочатку як академічний проєкт (Ingres, потім Postgres) у Каліфорнійському університеті в Берклі, а згодом перетворилася на один із найпопулярніших СКБД у світі.

PostgreSQL підтримує всі основні функції реляційних баз даних — таблиці, зв’язки, транзакції, перевірку цілісності даних, індексацію, запити SQL, — і при цьому має розширені можливості, які відрізняють її від багатьох конкурентів. Вона дозволяє створювати складні типи даних, використовувати користувацькі функції, зберігати JSON-структури, виконувати повнотекстовий пошук, створювати тригери, представлення (views), зберігати дані у вигляді масивів і навіть будувати запити до графових структур завдяки підтримці рекурсивних запитів. Усе це дає змогу поєднувати класичний підхід до зберігання даних із сучасними потребами веб- і бізнес-застосунків.

Однією з ключових переваг PostgreSQL є її розширюваність — розробники можуть створювати власні типи даних, оператори, агрегатні функції або навіть мови процедурного програмування (наприклад, PL/pgSQL, PL/Python, PL/Perl). Це дозволяє адаптувати базу даних під конкретні потреби проєкту без втрати продуктивності. Система також підтримує складні транзакції з повною ACID-гарантією, що робить її надзвичайно стабільною для фінансових, логістичних або аналітичних систем, де важлива точність і цілісність даних.

PostgreSQL добре масштабується як вертикально, так і горизонтально, завдяки можливості роботи з великими обсягами даних, використанню паралельних запитів, реплікації та кластеризації. Також існують численні інструменти та розширення для PostgreSQL, такі як PostGIS (робота з геоданими), TimescaleDB (серії часу), pgAdmin (графічний інтерфейс), які ще більше розширюють її функціональність.

Для розгортання і підтримки середовища розробки використовується Docker — це платформа для створення, розгортання та запуску програм у контейнерах. Контейнери дозволяють упакувати застосунок разом із усіма його

залежностями (бібліотеками, середовищем виконання, конфігураціями) в єдиний ізольований об’єкт, який можна легко переносити та запускати на будь-якому середовищі — незалежно від операційної системи чи інфраструктури. Це вирішує класичну проблему: "У мене все працює, а в тебе — ні", яка виникає через відмінності в середовищах розробки, тестування та продакшн.

У традиційній моделі розгортання розробник пише код, який працює в певному середовищі, а при перенесенні на інший сервер виникають помилки через різні версії бібліотек або налаштувань. Docker усуває цю проблему шляхом створення так званих образів (images), які є повністю самодостатніми одиницями з усім необхідним. Ці образи створюються на основі інструкцій у файлі Dockerfile, де покроково описано, як налаштувати середовище для запуску застосунку.

Коли образ створено, його можна запустити як контейнер — це легкий, ізольований процес, який працює на базі ядра операційної системи хост-машини. Контейнери не потребують окремої віртуалізації, як у випадку з віртуальними машинами, тому вони набагато швидші, легші й ефективніші за використанням ресурсів.

Docker дозволяє керувати численними контейнерами одночасно, і саме завдяки цьому став популярним у світі DevOps, CI/CD, хмарних технологій та мікросервісної архітектури. Кожна частина складної системи може бути упакована в окремий контейнер — наприклад, фронтенд, бекенд, база даних — і всі ці частини працюють незалежно, взаємодіючи через мережу. Таке розділення полегшує масштабування, оновлення та тестування.

Docker також має власну екосистему, зокрема Docker Hub — публічний реєстр, де зберігаються тисячі готових до використання образів (наприклад, PostgreSQL, Node.js, Nginx, Redis, MongoDB та інші). Це дає змогу швидко налаштувати середовище для розробки або продакшн без необхідності конфігурувати все вручну.

Завдяки Docker можна забезпечити однаковість середовищ на всіх етапах розробки — від локального комп’ютера до хмарної інфраструктури. Наприклад,

розробник запускає застосунок у контейнері локально, тестувальник перевіряє його у тому ж контейнері, а DevOps-фахівець розгортає ідентичний контейнер на сервері. Цей підхід значно зменшує кількість помилок, прискорює процес розробки та полегшує обслуговування застосунків.

Контейнеризація гарантує однаковість середовища на різних машинах, що знижує кількість помилок, пов’язаних з несумісністю, і спрощує процес CI/CD (безперервної інтеграції і доставки).

Обраний стек технологій відповідає сучасним стандартам розробки, забезпечує високу продуктивність, безпеку і зручність подальшого розвитку і підтримки вебзастосунку.

Vue.js — це сучасний JavaScript-фреймворк для створення інтерфейсів користувача та односторінкових вебзастосунків. Його розробив Еван Ю у 2014 році з метою створити легкий, гнучкий і водночас потужний інструмент для розробки вебінтерфейсів. Vue вирізняється своєю простотою у використанні, низьким порогом входження та можливістю поступової інтеграції у будь-який проєкт. Це означає, що ви можете почати з використання Vue для невеликого фрагмента сторінки і поступово нарощувати функціональність до повноцінного SPA (Single Page Application) без необхідності перебудовувати все з нуля.

Основою Vue є реактивна система, яка автоматично відстежує зміни в даних і оновлює DOM без прямого втручання розробника. Компонентна структура дозволяє будувати інтерфейс із незалежних блоків, кожен із яких має власну логіку, шаблон і стилі, що полегшує масштабування й обслуговування проєктів. Розмітка в Vue базується на HTML-шаблонах, які розширюються спеціальними директивами для динамічного зв’язування даних і керування логікою відображення. Це створює зрозумілий і декларативний стиль (рис. 2.2) написання інтерфейсів, який легко читати і підтримувати.

Із появою Vue 3 фреймворк отримав значне технічне оновлення, включаючи нову архітектуру, оптимізовану продуктивність, повну підтримку TypeScript та впровадження Composition API.

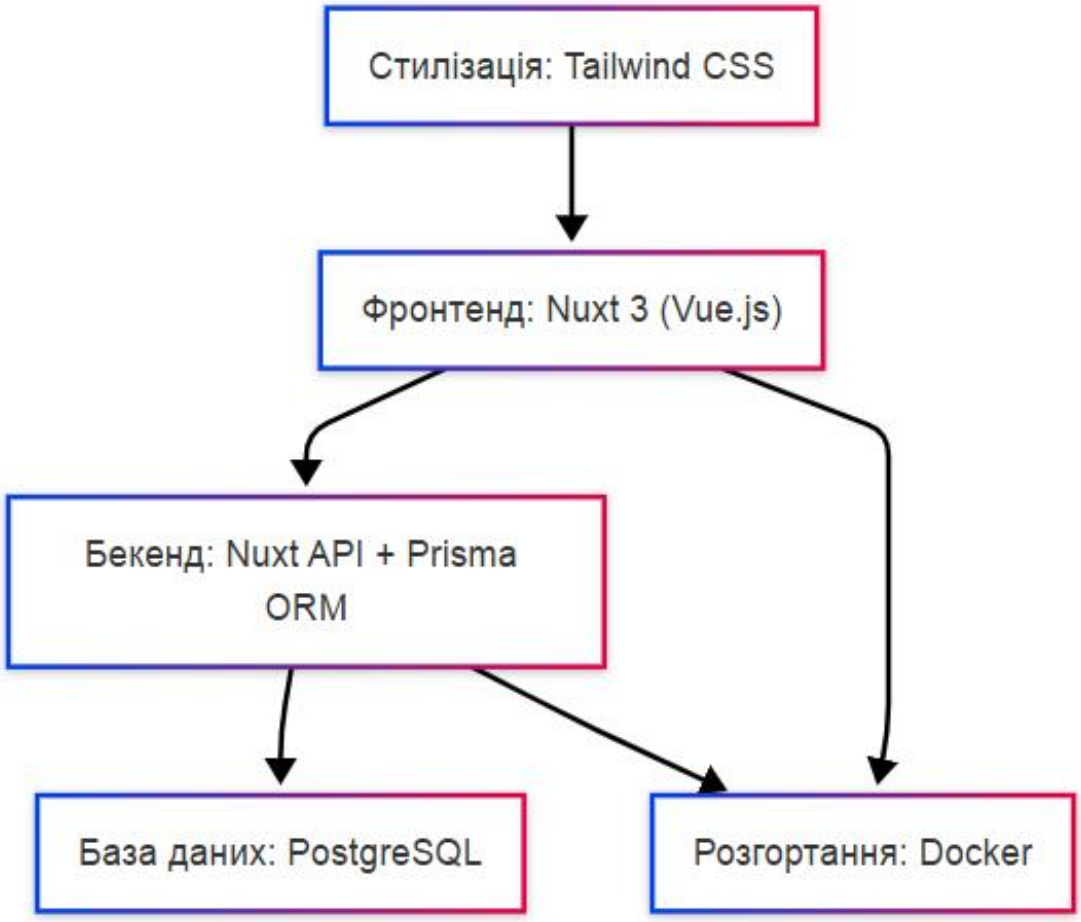

Рисунок 2.2 — Обґрунтування вибору інструментів

Останній дозволяє писати логіку компонентів у більш модульний і зрозумілий спосіб, що особливо корисно у великих застосунках. Vue 3 також краще працює з сучасними інструментами розробки, такими як Vite, який забезпечує надзвичайно швидке збирання та гаряче оновлення під час розробки.

Велика перевага Vue полягає у його екосистемі, яка включає офіційні бібліотеки для маршрутизації, управління станом, серверного рендерингу, роботи з формами, тестування та іншого. Завдяки цьому Vue може бути використаний як у невеликих вебпроєктах, так і в складних корпоративних застосунках. Крім того, Vue має активну спільноту, розвинену документацію та постійну підтримку з боку розробників, що сприяє його стабільному розвитку й популярності серед фронтенд-розробників у всьому світі.

2.3 Опис архітектури вебзастосунку та взаємодії компонентів

Архітектура вебзастосунку визначає структуру системи, способи взаємодії її складових і розподіл відповідальностей між ними. Від правильного архітектурного рішення залежить продуктивність, безпека, масштабованість і зручність підтримки продукту.

У сучасних вебзастосунках переважає багаторівнева клієнт-серверна архітектура, що включає такі основні компоненти: клієнтський інтерфейс (frontend), серверну логіку (backend), базу даних (database) і проміжний API шар, що забезпечує комунікацію між фронтендом і бекендом.

Клієнтська частина відповідає за відображення даних, збір і обробку дій користувача, підтримку інтерактивності і адаптивності інтерфейсу. Вона реалізується за допомогою сучасних фреймворків і технологій, що забезпечують високу продуктивність та гнучкість.

Серверна частина — це центр обробки даних, який реалізує бізнес-логіку, керує доступом, обробляє запити і відповідає за цілісність даних. Вона працює у взаємодії з базою даних, що зберігає всі необхідні інформаційні об’єкти.

API шар виступає посередником між фронтендом і бекендом, визначаючи стандартизовані інтерфейси для запитів і відповідей. Це дозволяє чітко розділити функціональні області, спростити тестування і подальший розвиток.

Взаємодія між компонентами (рис. 2.3) відбувається поетапно: користувач через браузер надсилає дії, які обробляються фронтендом, далі запити спрямовуються до API, обробляються серверною логікою з подальшим доступом до бази даних, і результати повертаються назад до користувача.

Таке розділення дозволяє легко підтримувати, масштабувати і розвивати вебзастосунок, розподіляючи навантаження і оптимізуючи кожен з компонентів.

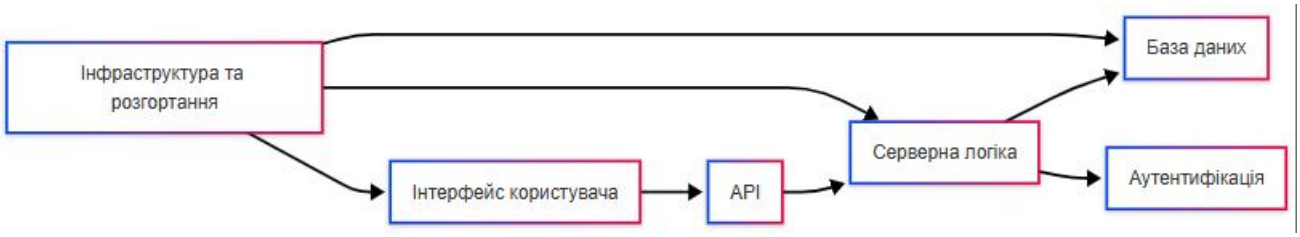

Рисунок 2.3 — Архітектура та взаємодія компонентів вебзастосунку

2.4 Технологічна структура власного вебзастосунку

У процесі розробки вебзастосунку велике значення має правильний вибір інструментів і технологічних стеків. На сучасному ринку існує багато різноманітних рішень, і кожне має свої переваги і недоліки.

Для фронтенду широко використовуються такі фреймворки (рис. 2.4), як React, Vue.js та Angular. React відзначається великою екосистемою і підтримкою Facebook, що забезпечує стабільність і швидкий розвиток. Vue.js популярний через простоту освоєння і гнучкість, а Angular — через повноцінність і стандартизацію, зручно підходить для великих корпоративних проєктів.

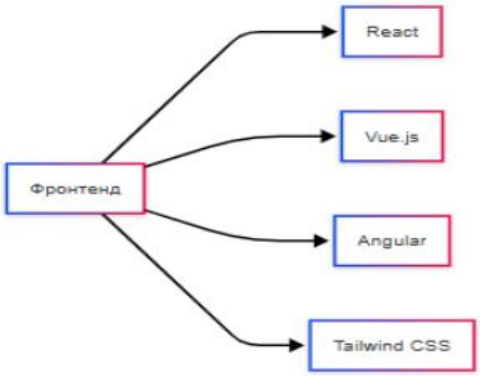

Рисунок 2.4 — Огляд фронтенду і інструментів

У сфері стилізації все частіше застосовують утилітарні CSS-фреймворки (наприклад, Tailwind CSS), які дозволяють швидко і ефективно створювати адаптивні дизайни без надмірного кодування.

На бекенді популярні Node.js (рис. 2.5), що дозволяє використовувати одну мову — JavaScript — на клієнті і сервері, а також Python з фреймворками Django і Flask, які характеризуються зручністю і швидкістю розробки. Ruby on Rails відомий своєю концепцією «конвенція понад конфігурацію», що прискорює розробку.

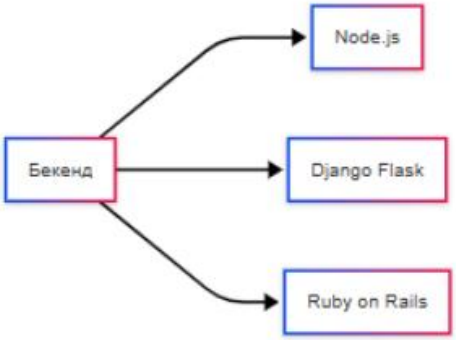

Рисунок 2.5 — Огляд бекенду і інструментів

Вибір систем управління базами даних (рис. 2.6) теж великий: реляційні СУБД (PostgreSQL, MySQL) забезпечують потужні засоби для складних транзакцій і аналітики, в той час як нереляційні бази (MongoDB, Redis) підходять для гнучких схем і високопродуктивних застосунків.

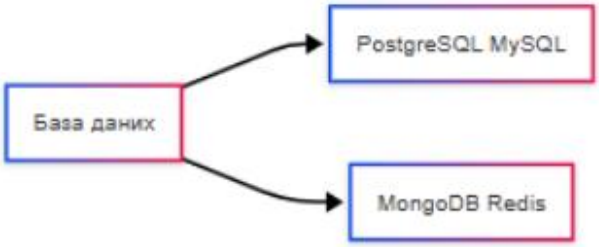

Рисунок 2.6 — Огляд бази даних і інструментів

Для зв’язку між компонентами застосовують API (рис. 2.7), зокрема REST та GraphQL. REST — це перевірений часом стиль, простий у реалізації, тоді як GraphQL дає змогу клієнту гнучко формувати запити, зменшуючи надмірність даних.

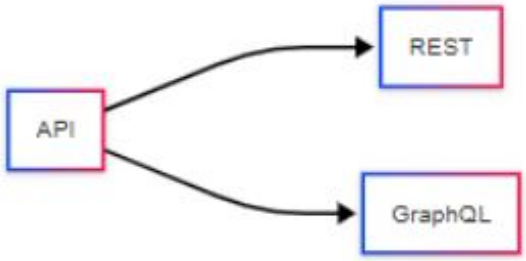

Рисунок 2.7 — Огляд API і технологій

Сучасні практики DevOps і контейнери (Docker, Kubernetes) дозволяють автоматизувати процеси розробки, тестування, розгортання і масштабування.

Вибір кожного інструменту має базуватися на конкретних вимогах проєкту, кваліфікації команди, а також наявності підтримки і документації.

2.5 Гейміфікація як інструмент підтримки користувацької активності

Гейміфікація — це процес впровадження ігрових елементів у неігрові середовища з метою підвищення залученості, мотивації та продуктивності користувачів. Вона базується на механіці, яка традиційно використовується в комп’ютерних чи настільних іграх, але адаптується до таких сфер як освіта, бізнес, охорона здоров’я, маркетинг, соціальні мережі, особистий розвиток тощо. Йдеться не про створення повноцінної гри, а про перетворення повсякденних чи рутинних дій на цікаві, структуровані та мотивуючі завдання.

Основна суть гейміфікації полягає у викликанні позитивної поведінкової реакції — бажання брати участь, повертатися до задач, виконувати завдання та досягати цілей. Вона активує системи винагороди в мозку, стимулюючи як внутрішню мотивацію (інтерес, прагнення саморозвитку), так і зовнішню (нагороди, змагання, визнання). Візуальні індикатори прогресу, як-от бали, бейджі, рівні, статуси або таблиці лідерів, сприяють тому, щоб людина бачила свою динаміку та відчувала успіх навіть у малих досягненнях.

Гейміфікація ефективно застосовується в освіті, зокрема у мовних додатках, навчальних платформах і навіть в університетських курсах, де замість звичайного тестування використовуються ігрові місії, балова система

або командні змагання. У маркетингу гейміфіковані програми лояльності стимулюють клієнтів повертатися до бренду. У бізнесі це може бути мотивація для працівників виконувати завдання, брати участь у внутрішніх ініціативах або підвищувати кваліфікацію через систему нагород і визнання.

Разом з перевагами існують і ризики. Якщо гейміфікація впроваджена формально або поверхово, вона може швидко втратити ефективність і навіть викликати роздратування. Надмірне використання зовнішніх стимулів може знижувати внутрішню мотивацію, особливо якщо користувачі сприймають механіку як маніпуляцію. Крім того, не всі люди реагують однаково на змагання або системи нагород — одних це надихає, інших демотивує.

Щоб гейміфікація була ефективною, вона має бути доречною, гнучкою, адаптованою до потреб цільової аудиторії та спрямованою на досягнення реальних цілей. Вона повинна посилювати, а не підміняти сам зміст діяльності. Ідеальна гейміфікація не просто розважає, а створює глибше залучення, підтримує сталі звички та допомагає людині рухатися вперед — у навчанні, кар’єрі, здоров’ї чи особистісному розвитку.

У контексті нашого вебзастосунку для планування завдань, гейміфікація стає центральним елементом, який покликаний підвищити мотивацію користувачів виконувати завдання регулярно, покращувати їхній досвід і забезпечувати зворотній зв'язок у вигляді винагород. Вона здійснюється через систему накопичення XP (досвід), досягнення рівнів, заробіток монет та отримання нагрудних значків (бейджів), а також відкриття нових можливостей для персоналізації інтерфейсу, таких як темами оформлення та аватарами.

Механіка XP і рівнів. Основним елементом гейміфікації в нашому вебзастосунку є система XP (досвід). Кожне виконане завдання приносить користувачеві певну кількість XP, що відповідає за його прогрес в системі. Таким чином, користувач має чітке уявлення про те, як він просувається у виконанні завдань і як наближається до наступного рівня.

Кожен рівень у системі має прогресивно збільшену складність для досягнення, що забезпечує постійну мотивацію користувача виконувати нові

завдання. Наприклад, для досягнення 1-го рівня може бути потрібно лише кілька завдань, а для 10-го — вже набагато більше XP. Це створює відчуття досягнень і дає користувачеві відчуття, що він постійно розвивається і прогресує.

Прогрес до нового рівня можна побачити через графічну шкалу, що візуально відображає кількість XP, яку користувач набрав. Кожен новий рівень дає можливість отримати додаткові бонуси: монети, нові теми для оформлення інтерфейсу або нові аватари. Оскільки кожен новий рівень потребує більшої кількості XP для досягнення, система поступово стає більш складною, що сприяє утриманню уваги користувача та збереженню його мотивації.

Винагороди: бейджі, монети і персоналізація. Однією з головних мотиваційних складових є винагороди. В нашому застосунку користувач отримує бейджі за досягнення певних віх або за виконання складних завдань, таких як:

— початківець (за перше завдання),

— наполегливий (за виконання 5 завдань),

— профі (за 10 виконаних завдань).

Такі бейджі не лише виступають як елементи гейміфікації, але й є соціальними доказами досягнень користувача, що підвищує його статус серед інших.

Крім бейджів, монети є основним засобом для покупок у внутрішньому магазині застосунку. Користувач отримує монети за виконані завдання та за досягнення нових рівнів. Монети можна витратити на покупку тем оформлення для змінення зовнішнього вигляду інтерфейсу або аватарів, що персоналізують користувацький профіль. Цей механізм підтримує постійну зацікавленість користувача, оскільки нові покупки надають йому ще більше можливостей для самовираження.

Логіка оновлення XP, рівнів та монет на бекенді. Вся логіка оновлення XP, монет та рівнів розміщена на сервері (бекенді) застосунку. Коли користувач завершує завдання, на сервер надсилається запит, який оновлює

його поточний досвід, монети та рівень. Використання бекенду для обробки цих операцій гарантує контроль даних, запобігає шахрайству та дозволяє користувачам зберігати їх досягнення в реальному часі.

Використовуються таблиці бази даних для збереження таких даних, як XP, level, coins (монети), badges (бейджі). Кожен новий рівень додає нові можливості для користувача, наприклад, можливість придбати нові предмети в магазині.

Підтримка мотивації через регулярність використання. Гейміфікація значно підвищує регулярність використання застосунку, оскільки кожен новий рівень чи винагорода мотивує користувача повертатися і виконувати нові завдання. У свою чергу, це покращує користувацький досвід і збільшує час взаємодії з додатком.

Для цього в нашому застосунку реалізовано систему повсякденних завдань (наприклад, кожен день — нове завдання), що стимулює користувачів активно взаємодіяти з системою і щодня виконувати хоча б одне завдання.

Психологічний ефект від гейміфікації. Гейміфікація має великий психологічний ефект. Вона активує внутрішні механізми мотивації, такі як прагнення досягнень, конкуренції і визнання. Користувачі, які бачать свій прогрес через XP, рівні та бейджі, відчувають більше задоволення від виконання завдань, оскільки це створює відчуття досягнень та особистого прогресу.

Ігрові механіки, такі як монети та аватари, також викликають відчуття нагороди (рис. 2.8), що робить процес планування завдань більш захоплюючим і мотивуючим.

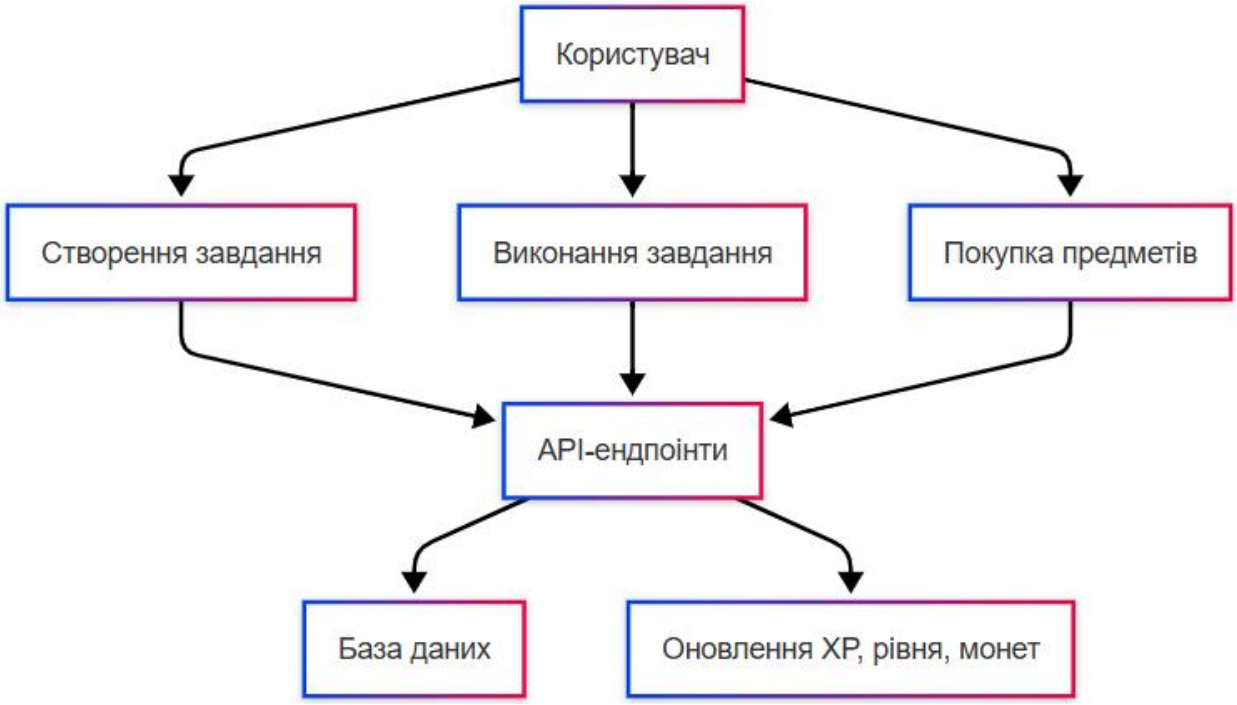

Рисунок 2.8 — Концептуальна модель гейміфікації у вебзастосунку

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ТА ЙОГО ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ

3.1 Підготовка середовища розробки

Розробка вебзастосунку починається з налаштування зручного та стабільного середовища, яке дозволяє ефективно тестувати, змінювати та масштабувати проєкт. У даній роботі для реалізації клієнтської частини було обрано фреймворк Nuxt 3, який працює на основі Vue.js та забезпечує зручну структуру проєкту, серверний рендеринг, автоматичне маршрутизування і хорошу підтримку TypeScript та Composition API.

Уся логіка додатку була реалізована мовою JavaScript, без використання TypeScript, що зробило структуру коду максимально простою та зручною для навчальних цілей. Стилізацію забезпечує Tailwind CSS — сучасна утилітарна CSS-бібліотека, що дозволяє швидко створювати адаптивні та чисті інтерфейси без написання окремих CSS-файлів.

Для роботи із базою даних використовується PostgreSQL — потужна, надійна і відкрита реляційна система управління базами даних. Доступ до неї реалізується через Prisma ORM, яка дозволяє описувати структуру таблиць у вигляді моделей у файлі schema.prisma, а потім автоматично створювати відповідні таблиці в базі через команду міграції. На рисунку 3.1 представлено структуру проєкту в середовищі розробки Cursor.

Cursor — це інтегроване середовище розробки нового покоління, орієнтоване на глибоку взаємодію між розробником і штучним інтелектом. Створене на базі Visual Studio Code, Cursor не просто копіює його можливості, а значно розширює функціональність за рахунок повноцінної AI-інтеграції, що змінює підхід до програмування. Ця платформа дає змогу писати, рефакторити, аналізувати й тестувати код набагато швидше завдяки підтримці природної мови, контекстному розумінню проєкту й інструментам на основі великих мовних моделей.

Cursor побудований так, щоб зекономити час розробника. Наприклад, якщо потрібно змінити логіку функції або додати нову функціональність, достатньо написати короткий запит звичайною мовою — наприклад, “зроби так, щоб ця функція працювала з масивами замість рядків”, — і AI сам перепише код, не порушуючи структури проєкту. Це значно знижує навантаження при великих оновленнях та підвищує зручність підтримки legacy-коду. Завдяки вбудованим інструментам генерації тестів, користувач може швидко покрити нові фрагменти коду юніт-тестами, не відволікаючись на шаблони або повторювані конструкції.

AI у Cursor працює не як окремий помічник, а як повноцінна частина IDE. Він розуміє контекст відкритого проєкту: структуру папок, залежності, змінні, класи, логику викликів — і може працювати не лише з поточним файлом, а й на рівні всієї програми. Якщо виникає помилка, AI може пояснити її, запропонувати виправлення або навіть знайти причину помилки у пов’язаних модулях. Для цього достатньо задати запит у відповідному режимі — наприклад, “чому ця функція не повертає очікуване значення?” — і AI проведе аналіз.

Cursor також підтримує режим “Agent”, в якому можна делегувати більш складні завдання. Наприклад, користувач пише: “додай автентифікацію через Google OAuth у цей застосунок”, — і AI сам знаходить відповідну бібліотеку, підключає її, генерує відповідні конфігурації та вносить необхідні зміни в код. Це не просто автодоповнення — це повноцінне делегування задачі, що раніше займала години ручної роботи. У цьому режимі Cursor може також згенерувати документацію до коду, допомогти з розділенням великих файлів на менші модулі, а також оптимізувати структуру проєкту.

Ще одна сильна сторона Cursor — інтеграція з Git. IDE не лише підтримує стандартні команди (commit, pull, push), а й може пояснювати відмінності між гілками, описувати зміни у pull request, а також попереджати про потенційні конфлікти. Це зручно під час командної роботи, особливо коли потрібно швидко зорієнтуватися в чужому коді або описати зміни для рев’ю.

Платформа також приділяє увагу безпеці та конфіденційності. Розробники можуть працювати в локальному режимі, коли всі AI-обчислення відбуваються без відправки коду на сервери. Це критично важливо для роботи з приватними або корпоративними репозиторіями. Також Cursor відповідає вимогам безпеки за стандартом SOC 2, що робить його придатним для використання в комерційних та фінансових компаніях.

Cursor підтримує понад 20 мов програмування, включаючи найпопулярніші серед веб- і бекенд-розробників: JavaScript, TypeScript, Python, Java, C++, Go, Rust, Ruby та інші. Програма доступна на Windows, macOS і Linux, має швидкий інсталятор і мінімальні системні вимоги. Почати роботу можна одразу після встановлення: користувача зустрічає інтерфейс, звичний для VS Code, з додатковими панелями для AI-команд, історії змін, результатів генерації тощо.

Завдяки цим можливостям Cursor уже завоював популярність серед фахівців, які працюють у стартапах, продуктових компаніях, а також серед фрилансерів і студентів. Його активно використовують для розробки нових фіч, написання технічної документації, генерації boilerplate-коду, інтеграції API, оптимізації SQL-запитів та багатьох інших рутинних завдань, які тепер можна виконувати у кілька разів швидше. На рисунку 3.1 зображено вигляд середовища розробки.

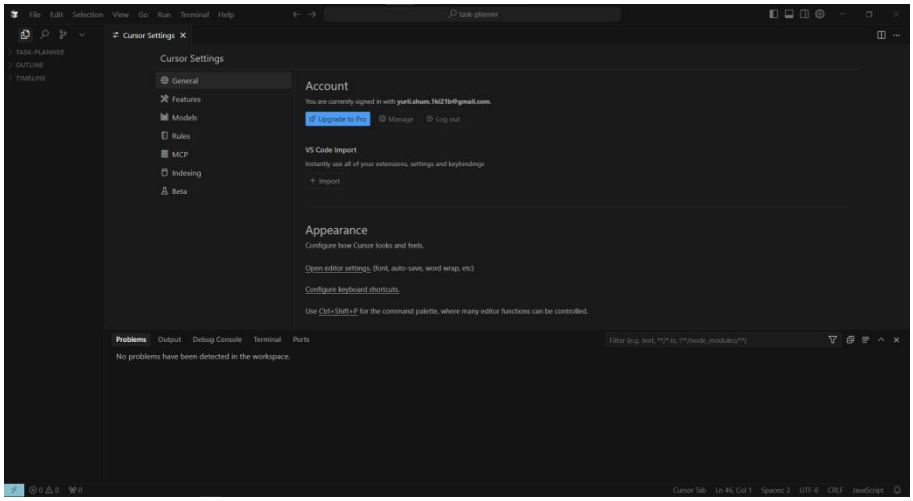

Рисунок 3.1 — Вигляд середовища Cursor

Окрім локальної розробки, було створено контейнеризоване середовище з використанням Docker. На рисунку 3.2 зображено приклад запуску застосунку за допомогою Docker Compose, що демонструє структуру контейнерів.

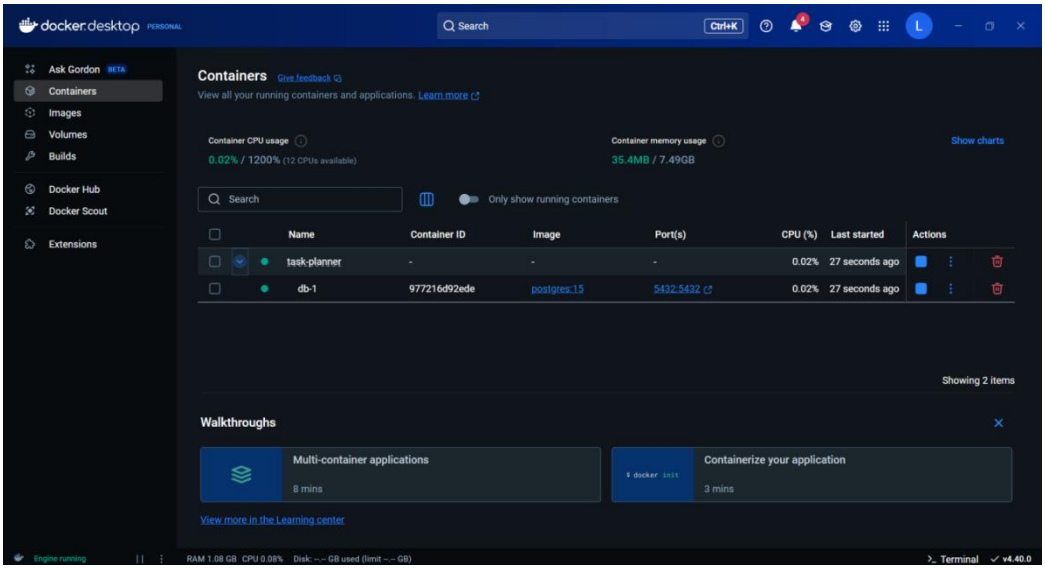

Рисунок 3.2 —Запуск середовища розробки через Docker Compose

Це дозволяє запускати вебзастосунок, серверну частину та базу даних у ізольованих контейнерах з однаковими параметрами, незалежно від операційної системи розробника. У файлі docker-compose.yml описано сервіси:

— web — контейнер для Nuxt 3 застосунку;

— db — контейнер для PostgreSQL;

— adminer (опціонально) — веб-інтерфейс для перегляду даних у базі.

Після налаштування середовища структура проєкту виглядала наступним чином:

— /pages — сторінки інтерфейсу користувача;

— /components — окремі візуальні елементи (карточки, кнопки, форми);

— /server/api — серверні ендпоінти для обробки запитів;

— /prisma/schema.prisma — опис моделей бази даних;

— .env — змінні середовища (наприклад, URL бази);

— docker-compose.yml — конфігурація контейнерів.

Завдяки такому підходу середовище вийшло стабільним, масштабованим і зручним у супроводі. Подальші етапи розробки реалізуються вже безпосередньо в межах цього структурального каркасу.

3.2 Реалізація користувацького інтерфейсу

Користувацький інтерфейс — це те, з чим взаємодіє кожен користувач, тому його реалізація потребувала особливої уваги. Основними критеріями стали: простота, логіка, адаптивність і візуальна мотивація. Інтерфейс був реалізований за допомогою Tailwind CSS, що дозволило швидко створити чисту й адаптивну візуальну структуру без зайвих стилів.

Весь фронтенд побудовано в рамках Nuxt 3, який забезпечує автоматичне маршрутизування на основі структури папки /pages. Наприклад:

—/pages/index.vue — головна сторінка;

—/pages/tasks.vue — список завдань;

—/pages/profile.vue — профіль користувача;

—/pages/shop.vue — магазин.

Компоненти, що використовуються на кількох сторінках (наприклад, заголовок, кнопки, картки завдань), були винесені до окремої папки /components.

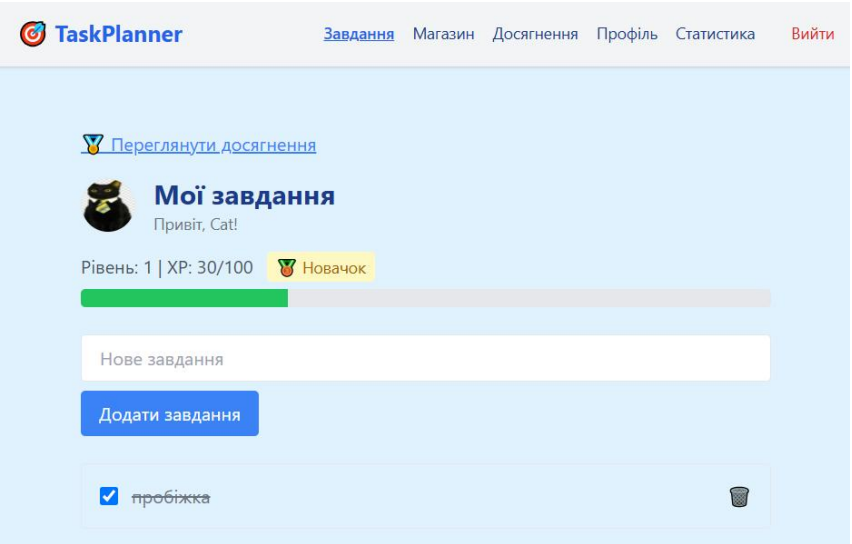

Рисунок 3.3 – Інтерфейс сторінки завдань з діями користувача

Сторінка завдань дозволяє створювати нові записи, редагувати їх (рис. 3.3), відмічати виконаними або видаляти. Інтерфейс підтримує миттєве оновлення списку після дії користувача, що створює ефект живої взаємодії. При завершенні завдання автоматично нараховується XP і монети.

На сторінці профілю користувач бачить свій рівень, кількість XP, кількість монет, обраний аватар і тему. Також передбачено графічну шкалу прогресу до наступного рівня. На рисунку 3.4 зображено приклад сторінки профілю.

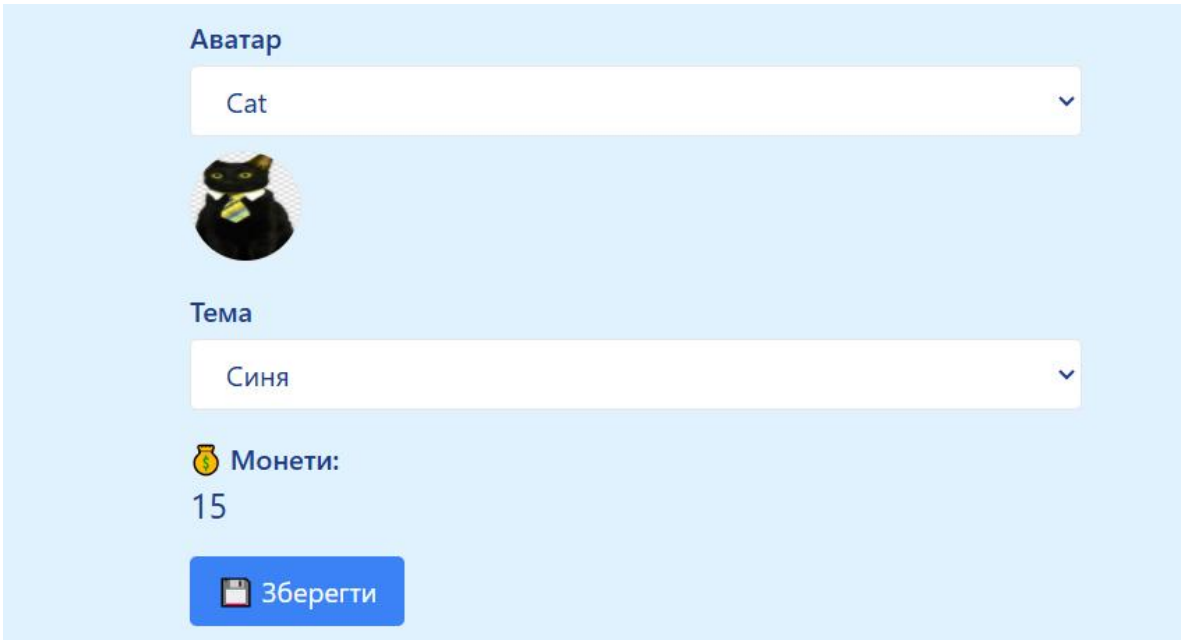

Рисунок 3.4 – Сторінка профілю з візуалізацією прогресу користувача

У магазині відображаються товари, доступні для покупки за монети: теми оформлення та аватари. Після придбання предмет автоматично додається до колекції користувача (рис. 3.5) й стає доступним у налаштуваннях профілю. Придбані елементи відмічені як «розблоковані» або недоступні повторно.

Навігація між сторінками реалізована через внутрішній маршрутизатор Nuxt 3. Для захисту приватних сторінок (наприклад, завдань або профілю) використовується middleware, яке перевіряє, чи користувач авторизований — інакше його перенаправляє на сторінку входу.

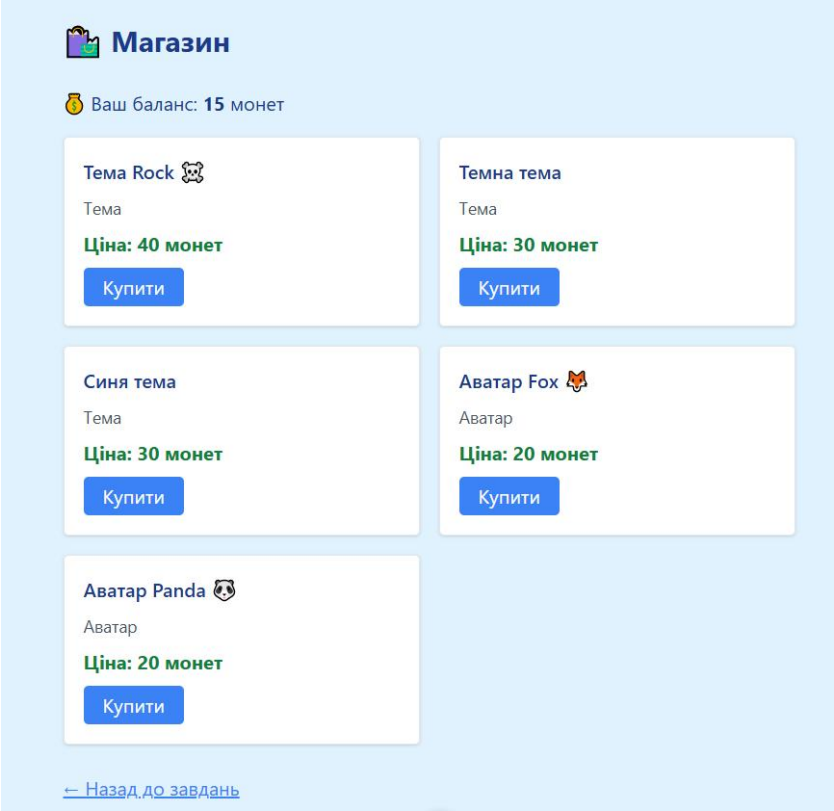

Рисунок 3.5 – Сторінка магазину з прикладами доступних тем та аватарів

Завдяки Tailwind CSS усі інтерфейсні елементи мають єдиний стиль, чіткі відступи та анімації, які створюють ефект приємного взаємодійного досвіду. У застосунку також реалізовано адаптивність: інтерфейс коректно відображається як на комп'ютері, так і на мобільних пристроях.

Таким чином, інтерфейс не лише виконує технічну функцію, а й формує загальне враження від застосунку, заохочуючи користувача повертатися знову.

3.3 Реєстрація, авторизація та захист маршруту

Оскільки застосунок є персоналізованим і прив'язаний до конкретного користувача, першим кроком у взаємодії з системою є авторизація або реєстрація. Для цього були реалізовані окремі маршрути /login і /register (рис. 3.6), кожен з яких містить просту форму введення даних — електронна пошта, ім'я (для реєстрації) і пароль.

Після успішного введення даних форма відправляє запит на сервер (через /server/api/auth/login або /auth/register). Сервер перевіряє дані у базі (через

Prisma), і якщо все коректно — видає токен сесії, який зберігається у cookie. Надалі цей токен автоматично перевіряється при кожному запиті користувача до захищених маршрутів.

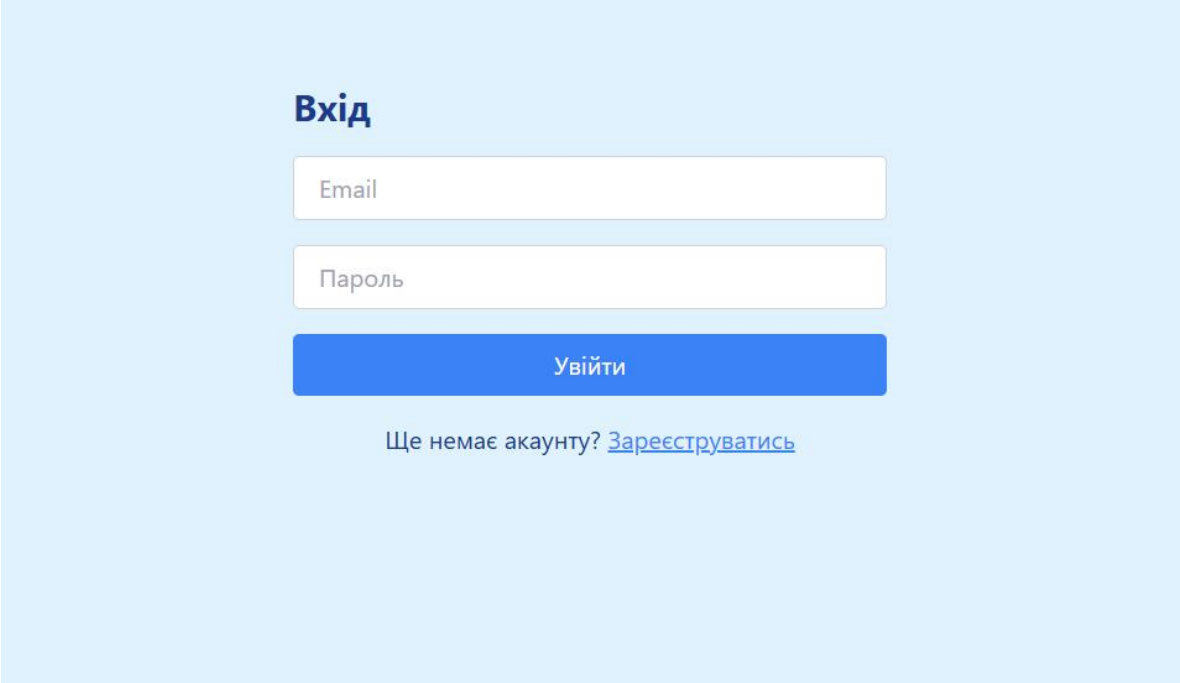

Рисунок 3.6 – Інтерфейс форми авторизації з валідацією введених даних

Захист сторінок реалізовано через Nuxt middleware. Створено глобальний файл auth.global.ts у директорії /middleware, який перевіряє, чи у користувача є валідний cookie. Якщо ні — його автоматично перенаправляє на сторінку входу. Такий підхід дозволяє захистити всі сторінки без потреби дублювати перевірки в кожному компоненті.

У випадку успішної авторизації користувача автоматично перенаправляє на сторінку завдань (/tasks). У разі невдалої спроби (неправильний пароль або відсутній акаунт) система показує повідомлення про помилку.

Для зручності та безпеки використовується хешування паролів перед збереженням у базу. Таким чином, навіть якщо база буде скомпрометована, зловмисник не зможе прочитати паролі у відкритому вигляді.

Завдяки такій реалізації забезпечується як захист персональних даних, так і контроль доступу до внутрішніх функцій застосунку. Це дозволяє

працювати з профілем, завданнями, монетами й гейміфікацією в межах безпечного середовища.

3.4 Завдання: створення, виконання, збереження

Робота із завданнями — це центральна функція вебзастосунку (рис. 3.7). Усі інші елементи, включаючи гейміфікацію, побудовані довкола цього процесу. Основна логіка взаємодії з завданнями реалізована на сторінці /tasks, де користувач може створювати, переглядати, виконувати та видаляти свої задачі.

Створення нового завдання відбувається за допомогою простої форми з одним текстовим полем. Після натискання кнопки «Додати» відправляється POST-запит на сервер (/api/tasks/create), який створює новий запис у базі даних із полем completed = false. Одразу після цього оновлений список завдань виводиться на екран.

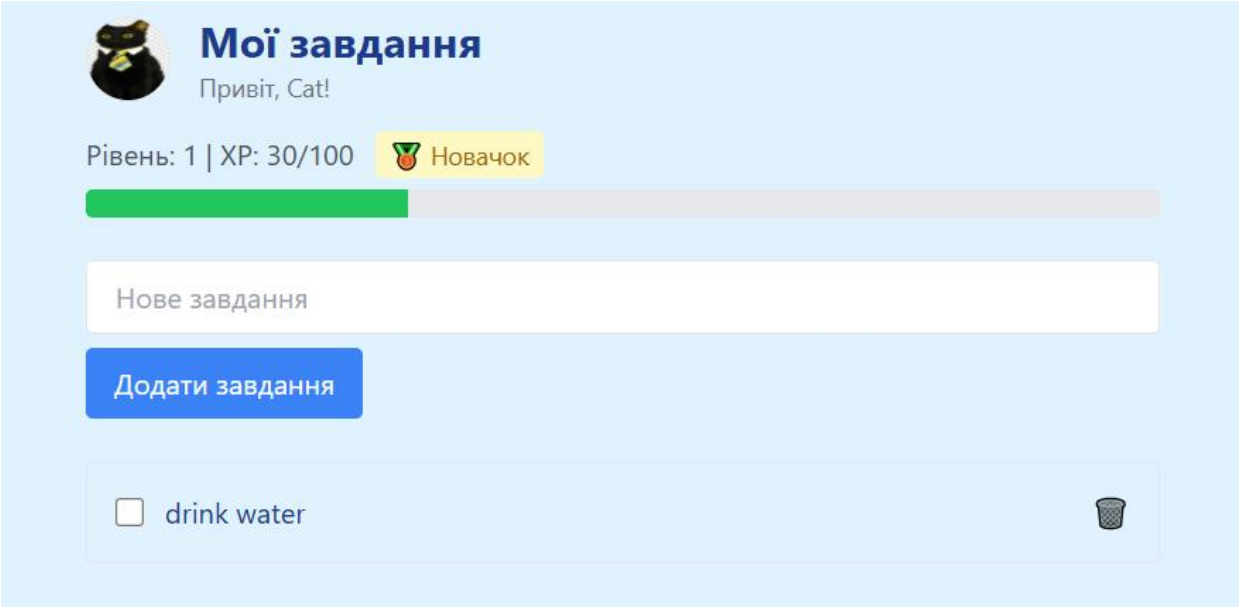

Рисунок 3.7 – Створення нового завдання у вебінтерфейсі

Всі завдання відображаються у вигляді списку, кожен елемент якого містить:

— назву завдання;

— чекбокс або кнопку для відмітки виконання;

— кнопку видалення.

При виконанні завдання змінюється його статус у базі (completed = true), і користувачу нараховуються XP та монети. Це виконується через PATCH-запит до /api/tasks/complete. Одразу після цього система автоматично оновлює рівень, якщо перевищено поріг досвіду, а також зберігає нові значення у таблиці User.

У разі видалення завдання (через DELETE-запит) відповідний запис вилучається з бази, і список оновлюється без перезавантаження сторінки. Усі ці операції обгорнуті в повідомлення (наприклад, «Завдання виконано!»), що дає користувачу миттєвий зворотний зв’язок.

Додатково реалізовано сортування списку — спочатку відображаються незавершені завдання, а вже потім — виконані (візуально затінені або з перекресленим текстом).

Вся логіка завдань реалізована в рамках API-файлів /server/api/tasks/*.ts і пов'язана з моделлю Task у базі даних, де кожне завдання має id, title, userId, createdAt, completed.

Таким чином, процес планування виглядає просто і швидко: додати → виконати → отримати результат. Це відповідає основній ідеї проєкту — зробити продуктивність максимально доступною й водночас мотивуючою.

3.5 Гейміфікація: XP, рівні, монети, нагороди

Щоб підтримати мотивацію користувача та зробити процес виконання завдань цікавішим, у застосунку реалізовано гейміфікаційну систему. Вона базується на чотирьох ключових компонентах: досвід (XP), рівень (Level), внутрішня валюта (монети) та нагороди (бейджі).

За кожне виконане завдання користувач отримує фіксовану кількість XP. Коли загальна кількість досвіду перевищує певний поріг (рис. 3.8), рівень автоматично підвищується. Формула зростання рівня нелінійна, тобто кожен наступний рівень потребує більше XP, що стимулює активність.

Монети також нараховуються автоматично:

— за завершення завдання;

— за досягнення нового рівня;

— за певні досягнення (наприклад, 10 виконаних завдань поспіль).

Рисунок 3.8 – Відображення рівня, XP і монет у профілі користувача

У профілі користувача візуально представлено поточний рівень, прогрес до наступного, загальну кількість монет, а також обраний аватар або тема оформлення. Прогрес рівня реалізовано у вигляді шкали, яка плавно заповнюється при накопиченні XP.

Окремим елементом є система нагород (бейджів). Наприклад:

— перший рівень — бейдж «Початківець»;

— 5-й рівень — «Наполегливий»;

— 10-й рівень — «Профі».

Ці нагороди відображаються у профілі користувача у вигляді значків. Їх реалізовано як окремі поля у базі або логіку на клієнтській стороні, що перевіряє level і динамічно показує відповідний бейдж.

Вся гейміфікаційна логіка реалізована на сервері: при виконанні завдання оновлюються поля xp, level, coins у таблиці User, а при досягненні порогових значень — додатково видаються нагороди.

Завдяки цьому користувач бачить не лише результат завдань, а й свій особистий прогрес, що формує бажання повертатись до застосунку щодня.

3.6 Магазин і система покупок

Однією з важливих складових гейміфікації є можливість витрачати зароблені монети на віртуальні бонуси. У застосунку реалізовано магазин, де

користувач може придбати візуальні покращення — такі як теми інтерфейсу або аватари.

Сторінка магазину /shop відображає всі доступні товари, кожен із яких має:

— назву;

— тип (тема або аватар);

— ціну в монетах;

— статус (доступний / куплений).

Дані про товари зберігаються у таблиці ShopItem, а про покупки — у таблиці UserItem. Це дозволяє реалізувати зв’язок «багато до багатьох», оскільки один товар можуть мати багато користувачів, і навпаки.

Після натискання кнопки «Придбати» виконується POST-запит до /api/shop/unlock, який:

— перевіряє баланс монет користувача;

— списує необхідну кількість;

— додає запис у таблицю UserItem;

— повертає оновлену інформацію для оновлення інтерфейсу.

Придбані предмети відображаються як «розблоковані» — кнопка покупки стає неактивною або змінюється на напис «Отримано».

У профілі користувача є можливість активувати куплену тему або аватар, що дозволяє персоналізувати вигляд застосунку. Активна тема змінює фонові кольори інтерфейсу, а вибраний аватар — зображення користувача у профілі.

Усі дії з покупками виконуються в рамках авторизованої сесії — користувач не може витратити більше монет, ніж має, або придбати один і той самий товар кілька разів.

Цей механізм підтримує користувачів не лише отримувати нагороди за свої досягнення, а й відчувати вплив власної активності на зовнішній вигляд застосунку, що робить взаємодію з ним більш цікавою і приємною. Завдяки цьому гейміфікаційний механізм не лише підтримує інтерес до планування

завдань, а й створює відчуття прогресу і персоналізації, що сприяє довготривалій залученості користувачів.

3.7 Сторінка профілю користувача

Сторінка профілю дозволяє користувачу побачити базову інформацію про себе та змінити зовнішній вигляд застосунку (рис. 3.9). Це простий, але важливий елемент системи, який відповідає за персоналізацію досвіду.

На сторінці /profile відображається:

— ім’я або email користувача;

— кількість зароблених монет;

— обраний аватар;

— активна тема оформлення;

— кнопки для зміни теми або аватара.

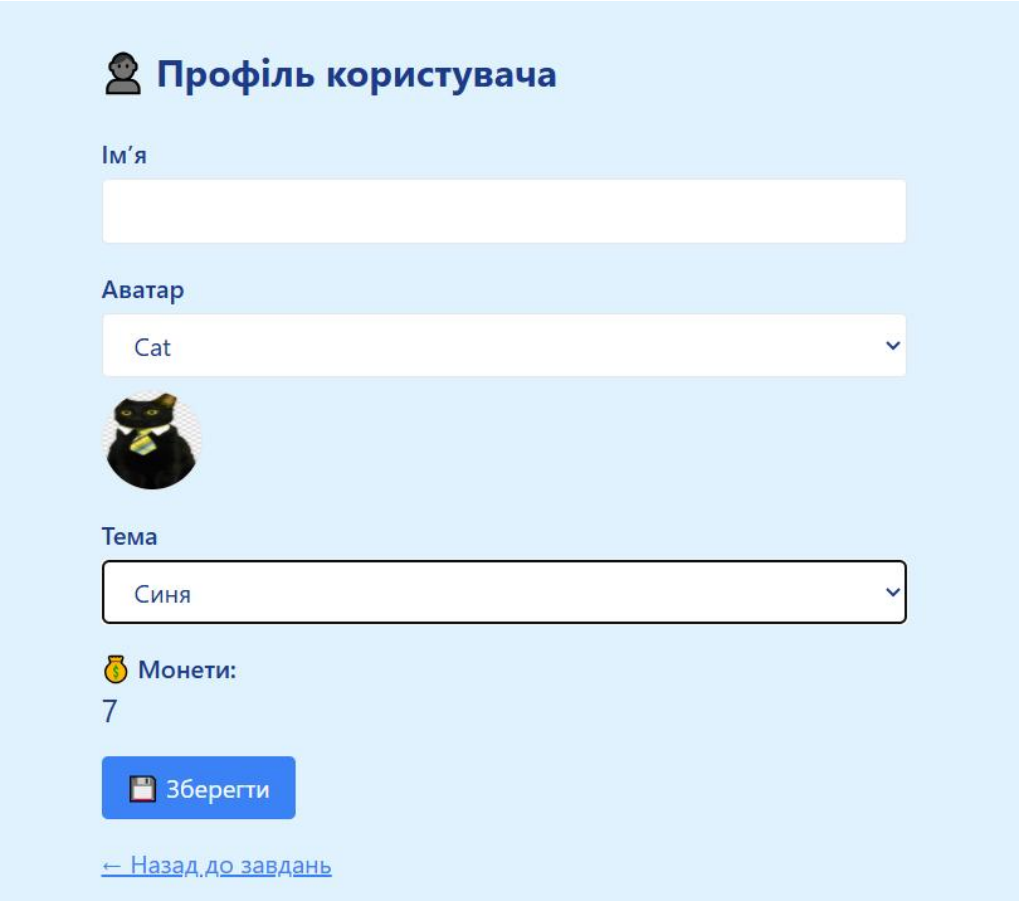

Рисунок 3.9 – Профіль користувача з персональними налаштуваннями

При натисканні на кнопку «Змінити тему» або «Змінити аватар» відкривається перелік придбаних користувачем предметів. Після вибору нової теми інтерфейс застосунку змінюється автоматично — кольорова схема оновлюється без перезавантаження сторінки. Аватар також змінюється одразу і відображається поруч з ім’ям користувача або у заголовку профілю.

Ці функції реалізовано на клієнтському рівні через прості форми, а на сервері — за допомогою оновлення запису користувача в базі (User.currentThemeId та інше). Зміни зберігаються, і при наступному вході відображаються автоматично.

У подальших версіях планується додати:

— відображення рівня;

— шкалу XP;

— бейджі за досягнення.

Наразі ж сторінка профілю виконує свою головну функцію — дозволяє користувачу адаптувати вигляд застосунку під власні вподобання.

3.8 Статистика активності

Статистика у застосунку відіграє важливу роль у підтриманні мотивації та формуванні звичок до регулярного планування. Вона не лише фіксує досягнення користувача, а й надає візуальний зворотний зв’язок про його активність, що дозволяє краще усвідомлювати власну динаміку продуктивності. Користувач може бачити, скільки завдань він виконав за певний період — день, тиждень чи місяць — і, таким чином, оцінювати свій прогрес у часі. Це сприяє формуванню позитивного підкріплення: що частіше користувач завершує завдання, то більше він бачить результат своєї праці, що створює ефект задоволення від досягнень і стимулює продовжувати у тому ж темпі. На окремій сторінці або у секції профілю (залежно від реалізації) відображаються такі показники:

— загальна кількість створених завдань;

— кількість виконаних завдань;

— кількість активних завдань (незавершених);

— (опційно) відсоток виконаних завдань.

Окрім цього, реалізовано графік активності за днями тижня або датами (рис. 3.10), який будується на основі дати створення/виконання завдань. Такий графік дозволяє побачити, в які дні користувач найбільш активний.

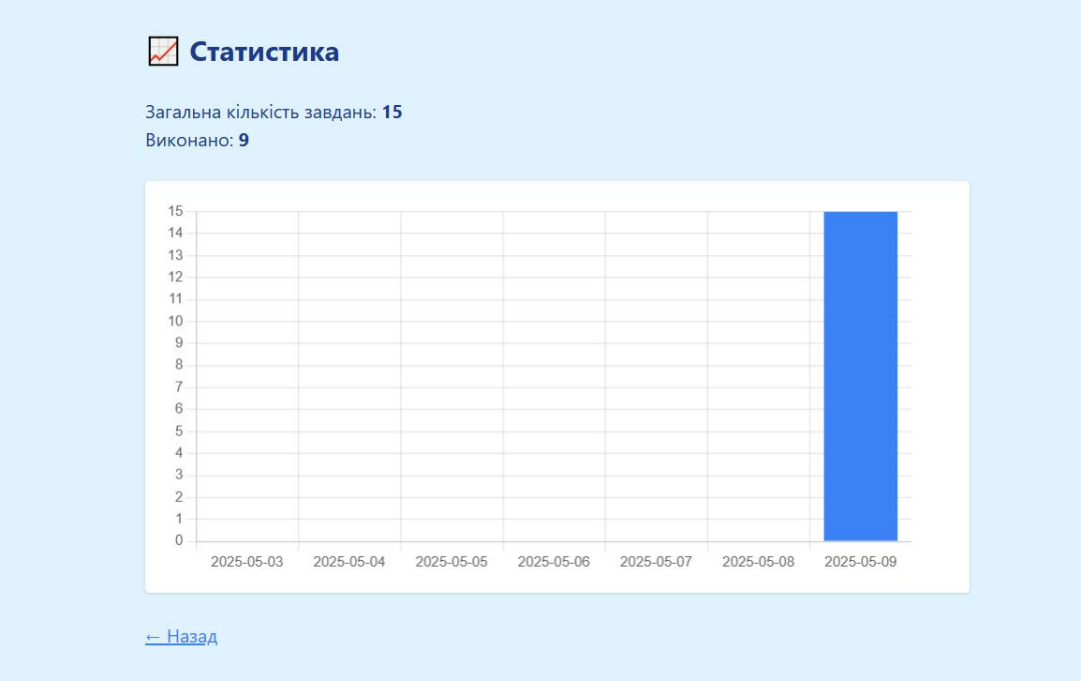

Рисунок 3.10 – Графік активності користувача за останній тиждень

Візуалізацію реалізовано за допомогою бібліотеки типу Chart.js або аналогічного простого інструмента (якщо використано SVG/Canvas вручну). Дані агрегуються на сервері або обробляються безпосередньо на клієнті з усіх завдань, що мають мітку completed = true.

Графік і числова статистика оновлюються в реальному часі — при створенні або виконанні завдання користувач одразу бачить зміну на графіку чи в лічильниках. Це створює ефект “живої аналітики” й стимулює не переривати активність.

Таким чином, функція статистики виконує як аналітичну, так і мотиваційну роль. Вона дозволяє не лише відстежувати, а й оцінювати свою

динаміку, а також ставити собі мікроцілі — наприклад, виконувати завдання кожного дня.

ВИСНОВОК

У ході виконання бакалаврської роботи було проаналізовано сучасні підходи до цифрового планування завдань та принципи гейміфікації, які сприяють підвищенню мотивації користувача. Встановлено, що більшість наявних рішень мають суттєві обмеження: або зосереджені виключно на функціональності, або надмірно ускладнені, що унеможливлює їх ефективне використання для широкого кола користувачів. У той самий час існує реальна потреба в простому, інтуїтивному і водночас захоплюючому інструменті для планування особистих завдань.

На основі проведеного аналізу було спроєктовано та реалізовано вебзастосунок для планування завдань з елементами гейміфікації. Архітектура системи передбачає чіткий поділ на клієнтську та серверну частини, з використанням сучасних технологій: Nuxt 3, Tailwind CSS, Prisma ORM, PostgreSQL та Docker. Це забезпечило стабільну роботу, масштабованість і можливість подальшого розвитку.

Функціональність застосунку охоплює повний цикл роботи з завданнями: створення, виконання, видалення, а також накопичення XP, підвищення рівня, отримання монет і придбання віртуальних предметів у магазині. Уся гейміфікаційна логіка реалізована як на рівні інтерфейсу, так і на стороні сервера, що гарантує коректне оновлення стану користувача.

Особливу увагу приділено персоналізації — користувач може обрати тему оформлення та аватар, що підвищує емоційну залученість. Також реалізовано сторінку профілю з базовою статистикою, яка візуалізує результати роботи.

Результати розробки демонструють, що поєднання простої системи планування з легкими гейміфікаційними елементами здатне значно підвищити мотивацію користувача до регулярного використання. Запропоноване рішення є адаптивним, доступним та придатним для розширення — як у функціональному, так і в візуальному вимірах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шум Ю. Р., Савицька Л. А. Вебзастосунок для планування завдань з ігровим інтерфейсом // Матеріали всеукраїнської науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців — Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25398

2. Крижановський Є. М., Богачук А. Р. Інформаційно-аналітична система інтернет-продажу // Науково-технічна конференція факультету комп'ютерних систем і автоматики. Вінниця: ВНТУ, 2021. [Електронний ресурс]. — Режим доступу: https://conferences.vntu.edu.ua (дата звернення: 01.05.2025).

3. Deterding S., Dixon D., Khaled R., Nacke L. From Game Design Elements to Gamefulness: Defining "Gamification" // Proceedings of the 15th International Academic MindTrek Conference. – 2011. – С. 9–15.

4. Robson K., Plangger K., Kietzmann J. H., McCarthy I., Pitt L. Gamification: Motivations and design principles for marketing applications // Journal of Strategic Marketing. – 2015. – № 3. – С. 1–17.

5. Todoist. Офіційний сайт. — Режим доступу: https://todoist.com (дата звернення: 11.04.2025).

6. Trello. Офіційний сайт. — Режим доступу: https://trello.com (дата звернення: 15.04.2025).

7. Habitica. Gamify Your Life. — Режим доступу: https://habitica.com (дата звернення: 15.04.2025).

8. Tailwind CSS Documentation. — Режим доступу: https://tailwindcss.com/docs (дата звернення: 15.04.2025).

9. Nuxt 3 Official Docs . — Режим доступу: https://nuxt.com/docs (дата звернення: 20.04.2025).

10. Prisma ORM Documentation. — Режим доступу: https://www.prisma.io/docs (дата звернення: 20.04.2025).

11. PostgreSQL. Офіційна документація. — Режим доступу: https://www.postgresql.org/docs (дата звернення: 21.04.2025).

12. McGonigal J. Reality is Broken: Why Games Make Us Better and How They Can Change the World. – Penguin Press, 2011.

13. Fogg B.J. Persuasive Technology: Using Computers to Change What We Think and Do. – Morgan Kaufmann, 2003.

14. Nielsen J. Usability Engineering. – Morgan Kaufmann, 1993.

15. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

16. Стенфордський курс “Gamification” (Kevin Werbach). – Coursera. — Режим доступу: https://www.coursera.org/learn/gamification (дата звернення: 05.05.2025).

17. Build a Full-Stack E-Commerce Website with Nuxt 3, Vue.js and Tailwind CSS [Електронний ресурс]. – Режим доступу: https://morioh.com/a/11765e05fa54 – Назва з екрана. – Дата звернення: 30.05.2025.

18. Full-Stack Social Media App with Nuxt 3, TailwindCSS, and Prisma [Електронний ресурс]. – Режим доступу: https://morioh.com/a/0fbbad9b8091 – Назва з екрана. – Дата звернення: 30.05.2025.

19. Nuxt 3 E-commerce Admin Dashboard with Prisma & Stripe [Електронний ресурс] // Reddit. – Режим доступу: https://www.reddit.com/r/Nuxt/comments/1h0vgme – Назва з екрана. – Дата звернення: 30.05.2025.

20. Nuxt 3 Bootcamp: Full-Stack Guide with Real-World Projects [Електронний ресурс] // Udemy. – Режим доступу: https://www.udemy.com/course/nuxtjs-fluency-the-premier-nuxt-3-masterclass – Назва з екрана. – Дата звернення: 30.05.2025.

21. Master Nuxt 3 – Full-Stack Complete Guide [Електронний ресурс] // CoderProg. – Режим доступу: https://coderprog.com/master-nuxt-3-full-stack-course – Назва з екрана. – Дата звернення: 30.05.2025.

22. Nuxt Awesome – Projects Using Nuxt.js [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/snewmanri/nuxt-awesome – Назва з екрана. – Дата звернення: 30.05.2025.

ДОДАТОК А — Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

10.03.2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
«Вебзастосунок для планування завдань з ігровим інтерфейсом »
08-54.БКР.0019.00.000 ТЗ

Керівник роботи к.т.н. доц. каф. ОТ

Савицька Л. А.

Студент групи 1КІ–21б

Шум Ю. Р.

ВНТУ 2025

1 Підстава для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність теми полягає в тому, що вебзастосунок для планування завдань з ігровим інтерфейсом стане у нагоді для багатьох людей з різних сфер у яких проблеми з плануванням свого часу.

1.2 Наказ ВНТУ №97 про затвердження теми БКР від 20.03.2025.

2 Мета БКР і призначення розробки

2.1 Мета роботи — створення вебзастосунку для планування завдань із гейміфікованим інтерфейсом, який би поєднував зручність та інтуїтивність класичних планерів із ігровими механіками (досвід, рівні, монети, нагороди), здатними підтримувати мотивацію користувача у довгостроковій перспективі.

2.2 Призначення розробки — набуття звички планування і оптимізації вільного часу.

3 Вихідні дані для виконання БКР

3.1 Набір базової інформації.

3.2 Принципи технічних вимог.

3.3 Методи підключення бібліотек та фреймворків.

3.4 Середовище розробки програмного забезпечення.

4 Вимоги до виконання БКР

4.1 Провести аналіз предметної області для інформаційної системи.

4.2 Розробити веб застосунок.

4.3 Провести тестування системи в кросплатформному середовищі.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А. 1

№ з/п	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз задачі планування завдань з використанням гейміфікації	22.03.25	25.03.25	Огляд джерел, висновки
2	Огляд предметної області вебзастосунків	28.03.25	01.04.25	Розділ 1
3	Обґрунтування обраних інструментів для розробки гейміфікованого вебзастосунку	04.04.25	15.04.25	Розділ 2
4	Особливості розробки вебінтерфейсу та організації бази даних для системи планування	16.04.25	28.04.25	Розділ 3
6	Підготовка пояснювальної записки	9.05.24	13.05.25	ПЗ, презентація, додатки

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні та ілюстративні матеріали, протокол попереднього захисту на кафедрі, відгуки наукового керівника та рецензента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 Вимоги до оформлення:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»; документи на які посилаються у вище вказаних.

8.2 Порядок виконання БКР

Порядок виконання викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21.

ДОДАТОК Б — ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Вебзастосунок для планування завдань з ігровим інтерфейсом

Тип роботи: бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 99% Схожість 1%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

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

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Шум Ю. Р.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Савицька Л. А.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

ВЕБЗАСТОСУНОК ДЛЯ ПЛАНУВАННЯ ЗАВДАНЬ З ІГРОВИМ ІНТЕРФЕЙСОМ

СХЕМА ЛОГІЧНОЇ СТРУКТУРИ БАЗИ ДАНИХ (ERD) ТА API ЗАПИТІВ

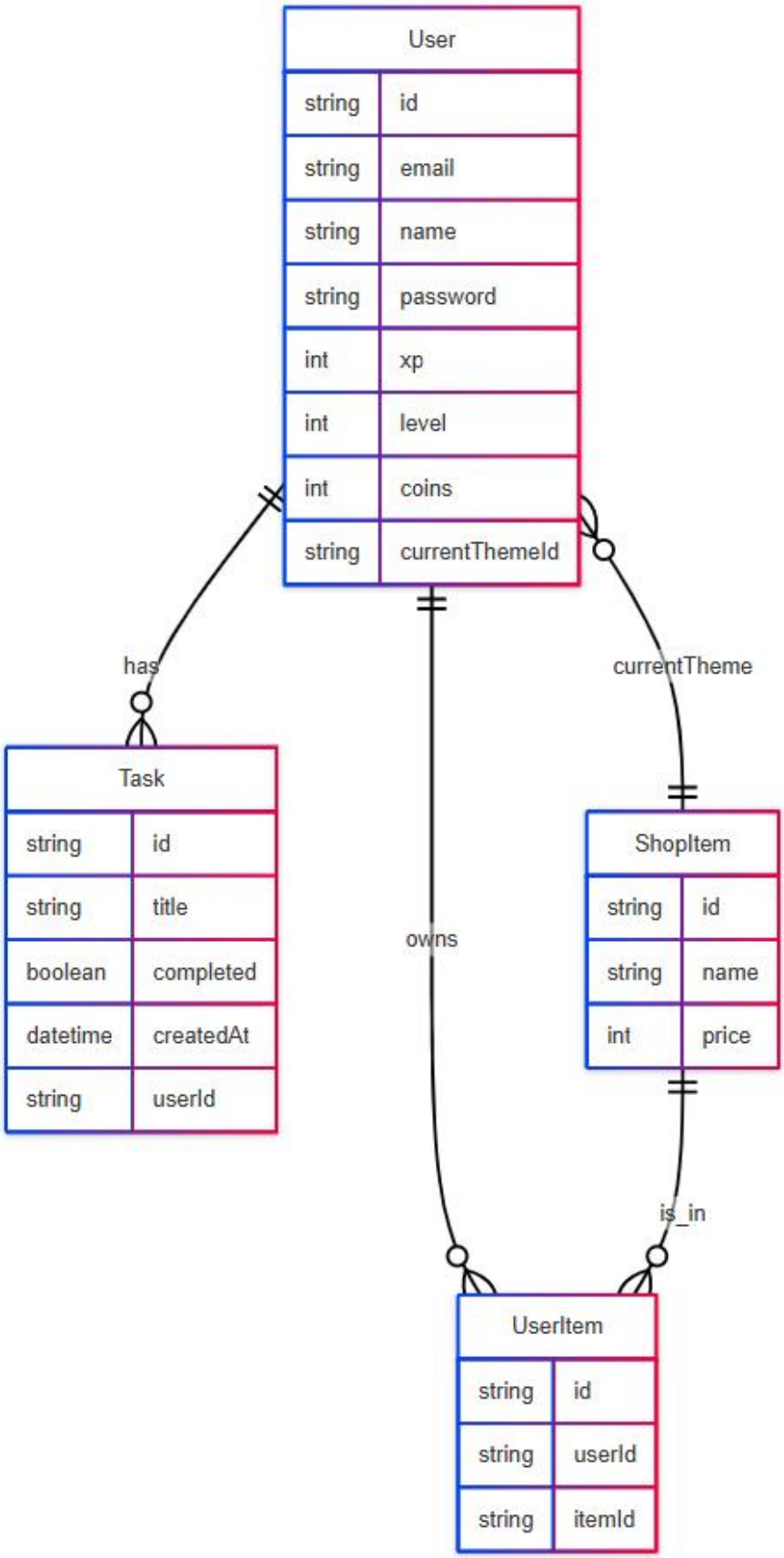

Рисунок В.1 — Схема бази даних

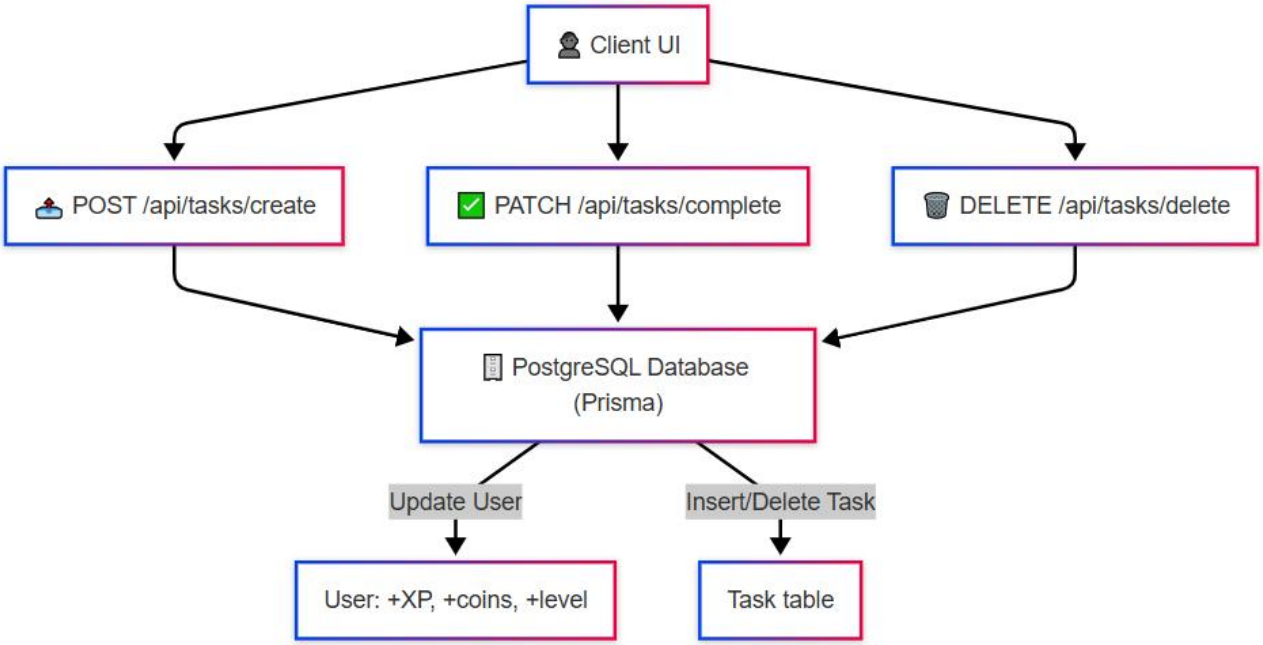

Рисунок В.2 — Схема API запитів між клієнтом і сервером

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА СИСТЕМА ОНЛАЙН-СЕРВІСУ ПОШУКУ ЗНАЙОМСТВ

ЗОВНІШНІЙ ВИГЛЯД ОСНОВНИХ СТОРІНОК

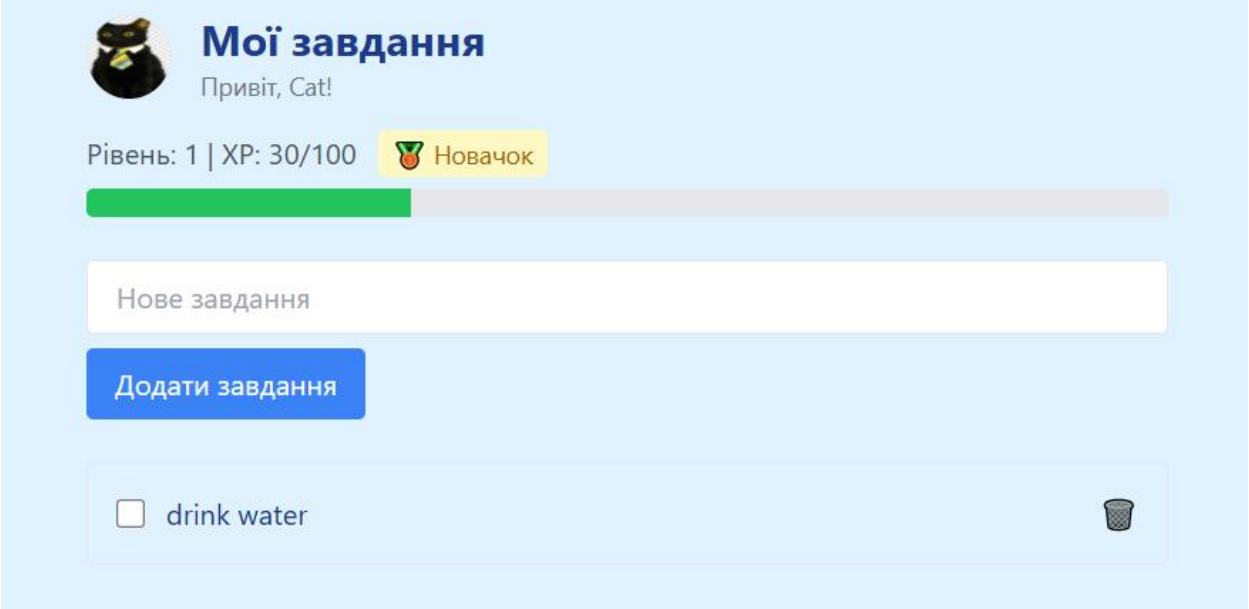

Рисунок Г.1 — Сторінка зі завданнями

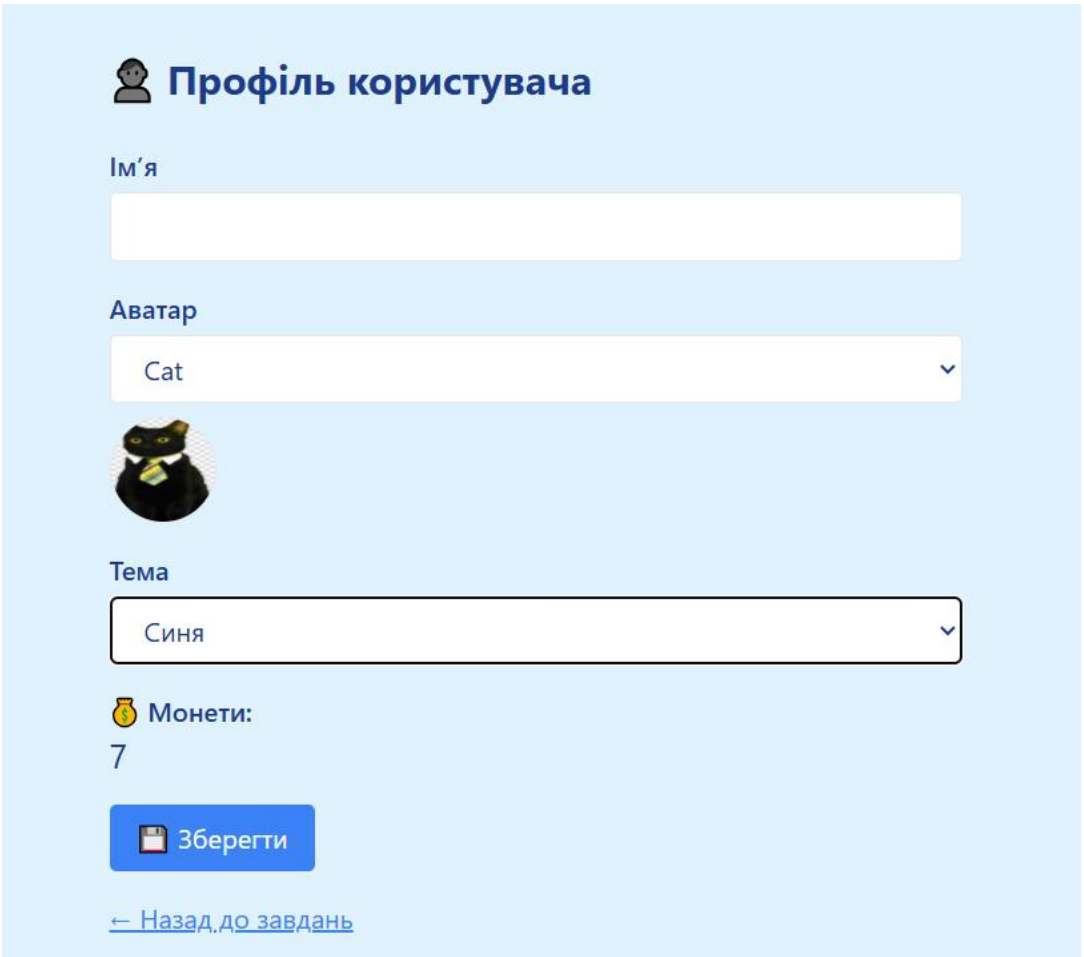

Рисунок Г.2 — Сторінка профілю користувача

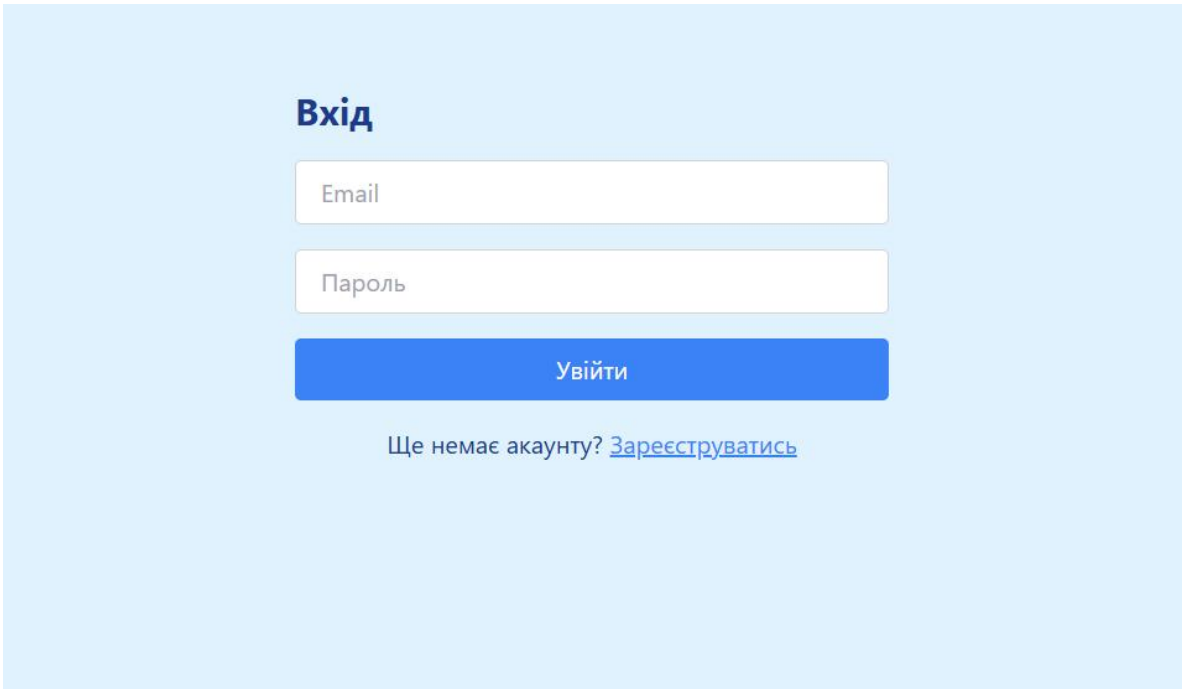

Рисунок Г.3 — Сторінка входу і регестрації

ДОДАТОК Д

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ІНФОРМАЦІЙНА СИСТЕМА ОНЛАЙН-СЕРВІСУ ПОШУКУ ЗНАЙОМСТВ

ДОДАТОК Д

Лістинг програмного забезпечення

docker-compose.yml

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - '3000:3000'
    env_file:
      - .env
    depends_on:
      - db
  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: planner
    ports:
      - '5432:5432'
    volumes:
      - postgres_data:/var/lib/postgresql/data
volumes:
  postgres_data:
```

Ключові компоненти інтерфейсу

```
<!-- TaskCard.vue -->
<template>
  <div class="flex justify-between items-center bg-white shadow p-4 rounded">
    <p :class="{ 'line-through': task.completed }">{{ task.title }}</p>
    <div class="flex gap-2">
      <button @click="completeTask(task.id)" class="text-green-600">✓</button>
      <button @click="deleteTask(task.id)" class="text-red-600">  </button>
    </div>
  </div>
</template>
```

Фрагмент логіки геймфіфікації (нарахування XP, монет)

```
const xpPerTask = 10
const coinsPerTask = 5
function applyRewards(user) {
    user.xp += xpPerTask
    user.coins += coinsPerTask
    if (user.xp >= user.level * 100) {
        user.level += 1
        user.xp = 0
    }
    return user
}
```