Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКИ WEB-ДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ ОСОБИСТОГО ЧАСУ

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Федоров А. Ю.
(прізвище та ініціали)
Керівник: доктор філософії асистент каф. КН
Перепелиця В.І.
(прізвище та ініціали)
«04» грудня 2025 р.

Рецензент: к.т.н., доцент каф. САІТ
Варчук І. В.
(прізвище та ініціали)
«05» грудня 2025 р.

Допущено до захисту
Зав. кафедри Грабовий А.А.
«06» грудня 20 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Андрію ФЕДОРОВУ

1. Тема роботи: Розробки WEB-додатку для організації особистого часу
керівник роботи: В'ячеслав ПЕРЕПЕЛИЦЯ, асистент
затверджені наказом вищого навчального закладу "20" серпня 2025 року №97

2. Термін подання студентом роботи 30 травня 2025 р.

3. Вихідні дані до роботи: мова програмування – з використанням об'єктно-орієнтованого підходу; цільова платформа – веб-середовище; веб-додаток має забезпечувати можливість створення, перегляду, редагування та видалення задач, підтримувати авторизацію та реєстрацію користувачів, надсилання нагадувань електронною поштою, адаптивний та інтуїтивно зрозумілий інтерфейс, а також базову інтеграцію з зовнішніми календарними сервісами.

4. Зміст текстової частини (перелік питань, які потрібно розробити): вступ; обґрунтування доцільності створення веб-додатку для організації особистого часу; аналіз і порівняння існуючих рішень; проектування архітектури системи; програмна реалізація веб-додатку; тестування функціональності та аналіз результатів; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
схема алгоритму реєстрації та створення задачі у веб-додатку; загальна структурна схема функціонування веб-системи; діаграма взаємодії компонентів системи; UML-діаграма класів основних моделей; приклад роботи додатку в інтерфейсі користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Перепелиця В. І., асистент каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обґрунтування доцільності розробки WEB-додатку для організації особистого часу	05.05.25	07.05.25	
2	Розробка WEB-додатку для організації особистого часу	08.05.25	11.05.25	
3	Програмна реалізація WEB-додатку для організації особистого часу	12.05.25	15.05.25	
4	Висновки та аналіз отриманих результатів	16.05.25	26.05.25	
5	Оформлення додатків	27.05.25	29.05.25	
6	Розробка інструкції користувача	30.05.25	01.06.25	
7	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Андрій ФЕДОРОВ**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **В'ячеслав ПЕРЕПЕЛИЦЯ**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 92 сторінок формату А4, на яких наведено 16 рисунків, 2 таблиці, список використаних джерел містить 24 найменувань.

Метою роботи є розширення веб-застосунку для організації особистого часу шляхом впровадження функціоналу планування задач, нагадувань та керування ними в особистому кабінеті.

У роботі обґрунтовано актуальність використання веб-застосунків як інструментів для планування та самоменеджменту. Проведено огляд та аналіз існуючих рішень, визначено їхні переваги та недоліки, на основі чого сформульовано вимоги до системи. Спроєктовано структуру та архітектуру застосунку, зокрема: моделі користувача й задач, алгоритми реєстрації та керування подіями, систему email-нагадувань.

Додаток реалізовано мовою програмування Python із використанням мікрофреймворку Flask. Для візуального оформлення використано Bootstrap 5, а для зберігання даних — SQLite у поєднанні з ORM-бібліотекою SQLAlchemy. Авторизацію реалізовано за допомогою Flask-Login, а систему сповіщень — через Flask-Mail. Проведено ручне тестування основного функціоналу, яке підтвердило стабільну роботу системи та відповідність заданим вимогам.

Ключові слова: WEB-додаток, організація часу, Flask, Python, нагадування, планування задач, Bootstrap.

ABSTRACT

The bachelor’s qualification work consists of 92 A4-sized pages, containing 16 figures, 2 tables, and a list of 24 references.

The purpose of this work is to develop a web application for organizing personal time by implementing functionality for task planning, reminders, and personal task management.

The work substantiates the relevance of using web applications as tools for planning and self-management. An overview and analysis of existing solutions were conducted, their strengths and weaknesses were identified, and requirements for the system were formulated. The structure and architecture of the application were designed, including: user and task models, algorithms for registration and event handling, and an email reminder system.

The application was implemented using the Python programming language and the Flask microframework. Bootstrap 5 was used for the frontend design, while SQLite in combination with the SQLAlchemy ORM library was used for data storage. Authorization was implemented using Flask-Login, and notifications were handled via Flask-Mail. Manual testing of the main functionality confirmed the stability of the system and its compliance with the specified requirements.

Keywords: web application, time management, Flask, Python, reminders, task planning, Bootstrap.

ЗМІСТ

ВСТУП

У сучасному світі цифрові технології активно впливають на стиль життя людини, трансформуючи підходи до організації часу, навчання, роботи та побуту. Стрімкий розвиток інтернету, мобільних пристроїв та веб-сервісів відкриває нові можливості для створення інструментів, які дозволяють ефективно планувати справи, відстежувати особисті завдання та досягати цілей. Особливої актуальності такі рішення набувають у повседневному середовищі, де правильне розподілення часу є ключовим для успішного навчання, саморозвитку та відпочинку. Сучасні веб-застосунки, завдяки своїй доступності, адаптивності та інтерактивності, стають незамінним інструментом для підтримки особистого тайм-менеджменту.

Актуальність створення веб-додатку для організації особистого часу зумовлена зростаючою потребою у цифрових сервісах, які надають прості та зручні інтерфейси для планування, контролю завдань і отримання нагадувань. Існуючі сервіси, хоч і мають потужний функціонал, часто перевантажені зайвими можливостями, не завжди адаптовані до потреб або потребують платного доступу до ключових функцій. Розробка спеціалізованого веб-застосунку, орієнтованого на простоту, швидкість взаємодії та базову функціональність (створення задач, редагування, нагадування), дозволяє покрити реальні потреби аудиторії, водночас не створюючи надмірного навантаження.

Метою даної бакалаврської кваліфікаційної роботи розширення функціональних можливостей веб-додатку для організації особистого часу, за рахунок забезпечення простого та зручного інтерфейсу для створення й керування задачами, базової функціональності для повсякденного планування, системи нагадувань через електронну пошту.

Об’єктом дослідження є процеси управління особистими задачами у веб-середовищі.

Предметом дослідження є методи та засоби створення веб-додатку на основі фреймворку Flask для підтримки функціональності особистого тайм-менеджменту.

Для досягнення поставленої мети визначено наступні задачі дослідження:

1. Обґрунтувати доцільність розробки веб-застосунку для організації особистого часу.
2. Проаналізувати існуючі сервіси та визначити їх переваги й недоліки.
3. Спроєктувати архітектуру, структуру та інтерфейс веб-застосунку.
4. Реалізувати основні функції: реєстрацію, створення, редагування, перегляд і видалення задач.
5. Розробити механізм нагадувань про задачі через електронну пошту.
6. Провести тестування функціональності застосунку та підготувати інструкцію користувача.

Практична цінність роботи полягає у створенні простого та доступного інструменту для планування особистих справ. Розроблений WEB-додаток може бути використаний широким колом користувачів для ефективного управління повсякденними завданнями, покращення самодисципліни та формування звички до планування. Запропоноване рішення сприяє цифровій грамотності та поширенню практик особистої організації.

1. ОГЛЯД І АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Огляд існуючих застосунків для організації особистого часу

У сучасному освітньому середовищі дедалі частіше використовують цифрові інструменти для планування навчального процесу, розкладу занять та дедлайнів. Найпопулярніші з них – Google Календар, Todoist, Notion, Trello, Any.do, ClickUp, iStudiez Pro, MyStudyLife, TickTick, RescueTime, Focus To-Do та інші. Далі наведено огляд цих застосунків за такими критеріями: короткий опис, аудиторія, основні функції, особливості для студентів, кількість користувачів, платформи, ціни та рейтинги.

Google Calendar – безкоштовний веб- і мобільний календар від Google, призначений для планування подій, зустрічей, завдань і особистого розкладу[1]. Цільова аудиторія – широке коло користувачів екосистеми Google: від приватних осіб і студентів до бізнесу, додаток зображено на рисунку 1.1.

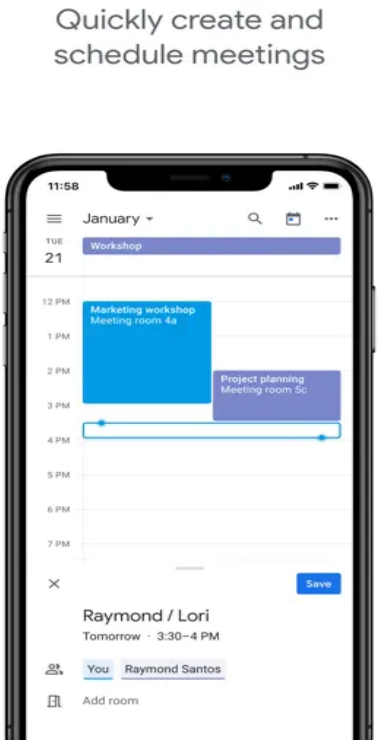

Рисунок 1.1 – Веб-додаток Google Calendar

WEB-додаток інтегрований з іншими сервісами Google (Gmail, Meet тощо), тому зручний для тих, хто уже користується Gmail або Google Workspace. Його **о**сновні функції та переваги**:**

– Створення й редагування подій з датою, часом, місцем і учасниками; можливість ділитися подіями та календарями.

– Автоматичне додавання подій з пошти (квитки на подорож, бронювання) [1].

– Нагадування про майбутні події (за часом чи місцем).

– Підтримка кількох календарів (наприклад, особистий, навчальний, робочий) з можливістю кольорового маркування.

– Різні види відображення: щоденний, тижневий, місячний та режим «Агенда».

Інтеграція з Google Meet та іншими відеоплатформами – автоматичне створення посилань на онлайн-зустрічі. Переваги – простота та зрозумілість інтерфейсу, глибока інтеграція з продуктами Google, безкоштовність, синхронізація через акаунт Google. Це зменшує потребу користувачів тримати кілька різних інструментів.

Користувачі можуть використовувати Google Календар для створення розкладу занять (з підтримкою класичного або блочного режиму), планування дедлайнів домашніх робіт та екзаменів. Кольорове розмежування предметів і подій робить розклад наочнішим. Також є можливість додати сповіщення про терміни виконання завдань і подивитись перелік всіх майбутніх справ. Інтеграція з додатком «Завдання» дозволяє відображати список справ поряд із календарем. Кількість користувачів понад 500 млн активних користувачів на місяць [1]. Платформи доступу це веб (браузер), мобільні додатки для Android і iOS (iPhone/iPad), а також можливість доступу через десктопні браузери. Безкоштовний, є розширені функції для Google Workspace, але сам календар не вимагає оплати.

Наступний додаток – Todoist, один з найпопулярніших кросплатформених планувальників завдань [2]. Підходить як окремим

користувачам (особистий менеджмент справ), так і командам, вигляд застосунку зображено на рисунку 1.2.

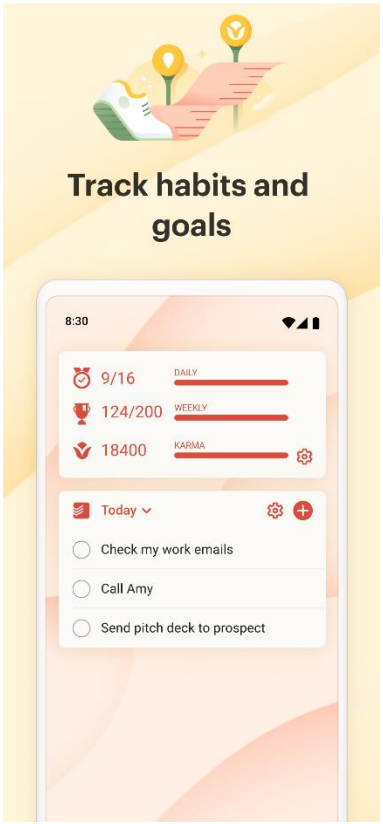

Рисунок 1.2 – Веб-додаток Todoist

Його аудиторія – широке коло: від студентів, котрі планують навчальні задачі, до професіоналів і менеджерів проєктів. WEB-додаток відзначається простим дизайном та гнучким функціоналом для організації як щоденних справ, так і комплексних проєктів. Основними функціями програми являється:

- Створення завдань і списків, проєктів та підзадач.
- Встановлення термінів (дати і час), повторюваних завдань, пріоритетів. Автоматичний розпізнавач дат у тексті («завтра», «через 3 дні»).
- Теги, фільтри та сортування для швидкого пошуку завдань.
- Спільна робота над проєктами: призначення завдань іншим, коментування, прикріплення файлів.
- Нагадування з push-повідомленнями (для преміум-версії).

– Веб-кліпінг (додавання задачі з email, браузера, голосом через Alexa/Google Assistant).

– Статистика продуктивності («Карма»), що мотивує закривати завдання.

Переваги Todoist – інтуїтивність, наявність готових шаблонів проєктів, інтеграція з календарями й іншими сервісами (Gmail, Outlook, Slack, та ін.), а також кросплатформеність (доступ у браузері, мобілці, десктопі) [3]. Todoist дозволяє ефективно «звільнити розум від хаосу» і систематизувати завдання. Студенти можуть створювати в Todoist окремі «проєкти» під кожен курс чи дисципліну, розбиваючи їх на секції («Джерела», «Екзамени», «Завдання», «Лекції»). Наприклад, у статті Todoist рекомендують організувати «Завдання та проекти» за предметами, додавати дедлайни домашніх завдань і записувати дати іспитів [4]. Додаток підтримує синхронізацію зі стороннім календарем, тож студент може бачити свої справи поряд із розкладом занять. Налаштування повторюваних рутинних задач допомагає виробити корисні звички і не забути щотижневі справи. Платформи доступу: веб-версія (todoist.com), додатки для Android, iOS (iPhone/iPad), Windows, macOS, Linux, розширення для браузера, плагіни до пошти та голосові помічники. Базова версія – безкоштовна, до 5 проєктів, 3 фільтри, 5 спільних учасників. Преміум (Pro) – ≈$4 на місяць з розширеними можливостями, необмежена кількість проєктів, нагадування, більше файлів [2]. Для команд є бізнес-плани.

Notion – універсальний «все-в-одному» робочий простір для нотаток, баз даних, завдань і документів [5]. Це гнучкий конструктор сторінок і баз, у якому можна створювати власну систему організації, зображено на рисунку 1.3.

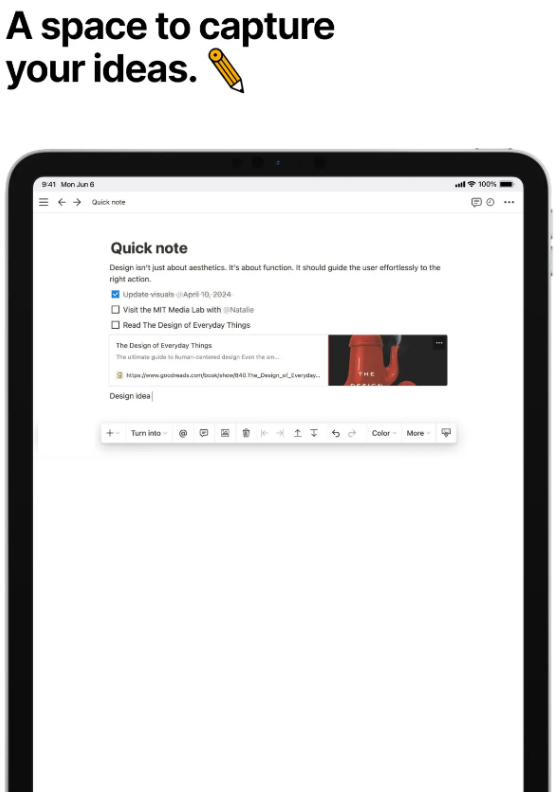

Рисунок 1.3 – Веб-додаток Notion

Його використовують як індивідуальні користувачі для планування життя та навчання, так і команди в бізнесі. Окрема величезна аудиторія – студенти і освітяни. Notion пропонує безкоштовні «Студентські» акаунти з такими функціями та перевагами:

– Блокова архітектура: будь-яку сторінку можна заповнювати блоками (текст, списки, таблиці, зображення, код, математичні формули, календар, дошка Kanban тощо). Блоки легко переставляються.

– Бази даних: створення розширених баз з різними видами подання (таблиця, канбан, календар, галерея), фільтрами й зв’язками між таблицями.

– Велика галерея готових шаблонів (для лекційних нотаток, планерів семестрів, бібліотек ресурсів тощо).

– Спільне редагування документів у реальному часі.

– Інтеграції з Google Calendar, Slack, Figma, GitHub та іншими сервісами; також Notion API.

– Notion AI: вбудований штучний інтелект для генерації тексту, підсумовування, перекладів тощо.

Переваги Notion – необмежені можливості кастомізації, що дозволяють налаштувати робоче середовище під будь-які потреби; підтримка спільної роботи та велика спільнота користувачів зі спільними шаблонами. Notion активно рекламує себе як інструмент для студентів. Він безкоштовний для особистого використання, а для студентів діє безкоштовне оновлення «Plus Plan» [6]. Студенти використовують Notion для ведення конспектів (структурують їх базами даних за темами), створення планерів курсу, відстеження дедлайнів і завдань, ведення портфоліо проєктів тощо [5]. Наприклад, можна зробити БД курсових робіт із зв’язками до лекційних нотаток або розкладом семестру. Платформи доступу: веб-версія (notion.so), настільні додатки для Windows і macOS, мобільні – Android і iOS (iPhone/iPad), а також версії для Linux через неофіційні клієнти або веб. Є безкоштовний план (для особистого користування, з обмеженнями на обсяг елементів і завантажень). Платні плани: Personal Pro ≈$10/міс за рік, Team ≈$20/міс за рік тощо. Для студентів і викладачів доступний безкоштовний «Plus Plan» [6].

ClickUp – «єдиний додаток для всього» («all-in-one productivity platform»), який об’єднує управління завданнями, документами, чатом, цілями та інші інструменти в одному середовищі [7]. Первинно орієнтований на бізнес і команди, але також просувається як рішення для студентів – компанія має окремі підрозділи «для студентів» і «освіти». Користувачі – як малі команди, так і великі корпорації (відомо, що 80% компаній зі списку Fortune 500 використовують ClickUp [7]), а також окремі користувачі, котрі переходять з інших таск-менеджерів.

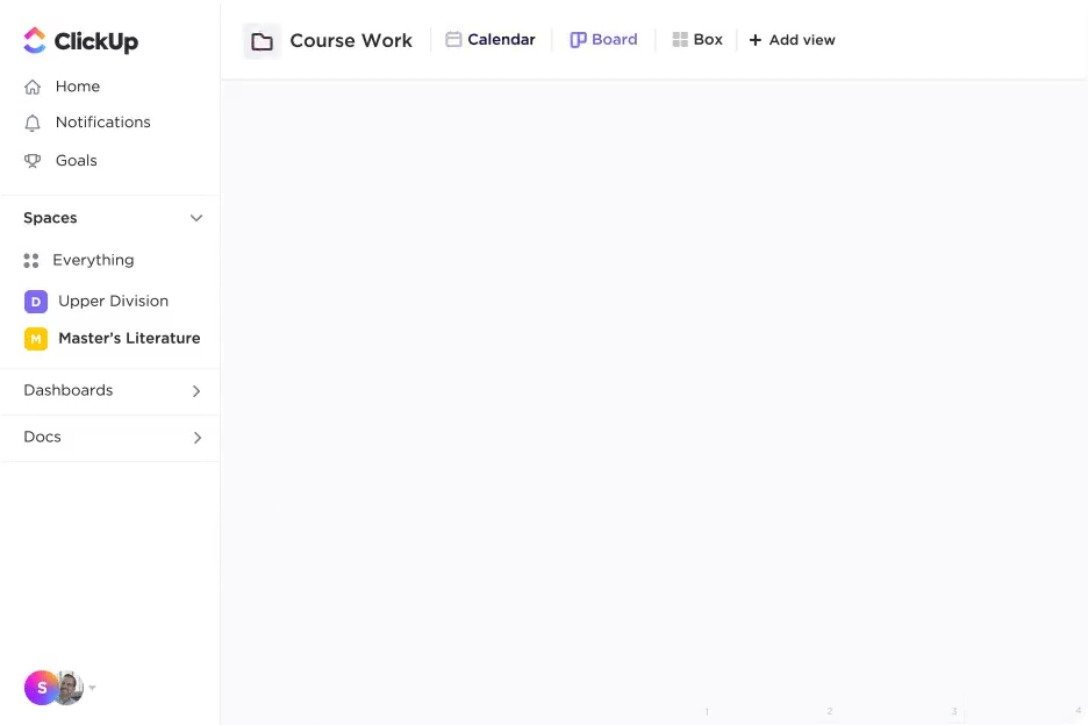

Рисунок 1.4 – Веб-додаток ClickUp

Інтерфейс ClickUp: універсальний робочий простір з інтегрованими списками завдань, календарем та візуальними поданнями. На рисунку 1.4 видно приклад організації навчальних завдань за курсом. Основні функції та переваги:

– Завдання і підзадачі: гнучкі статуси, дедлайни, пріоритети, залежності між завданнями.

– Безліч різних подань роботи: список, канбан (дошка), календар, діаграми Ганта, таблиці, часова лінія (Timeline), майнд-мепи тощо.

– Документи і Білі дошки (Whiteboards), можливість створювати довгострокові тексти, заметки, бази знань прямо в ClickUp; та спільні білі дошки для брейнштормінгу.

– Вбудований чат для команди, а також коментарі в завданнях і документах.

– Вбудовані автоматизації (схожі на Trello Butler) та понад 1000 інтеграцій.

– ClickUp AI: модуль штучного інтелекту для створення тексту, підсумків.

Переваги – надзвичайна функціональність та висока кастомізованість. Зворотній зв’язок від користувачів: ClickUp називають «платформою, що дозволяє робити на 62% більше завдань за той же час». ClickUp спеціально позиціонує себе як інструмент для освіти [7]. Студентська версія пропонує організувати всі курсові справи в одному місці: можна створити завдання під кожне заняття, додавати дедлайни, прикріплювати конспекти в Docs, планувати та перетягувати їх в календар. Існують готові шаблони для Cornell Notes та загальний студентський шаблон. В ClickUp також можна вести групові чати для командних проєктів та розподіляти завдання між членами групи. Таким чином, усі академічні задачі (лекції, домашні завдання, проєкти, іспити) відстежуються в єдиному середовищі. Платформи: веб, десктопні додатки для Windows і macOS, мобільні – Android і iOS (також є клієнт для Linux від спільноти). Є розширення для Chrome, Outlook, Slack та Alexa. Є безкоштовний план «Free Forever» з великою кількістю можливостей, безлімітна кількість завдань, 24 хмара-сховища, 100 МБ кожен файл, 1 робочий простір тощо [7]. Платні плани: Unlimited ≈$7/міс за рік, Business ≈$12/міс, Business Plus, Enterprise (≈$20–25) з додатковими фічами. Для студентів і навчальних закладів діє спеціальна знижка або безкоштовний доступ за наявності освітнього email.

Усі наведені застосунки мають детальну документацію і спільноти користувачів. На офіційних сайтах кожного сервісу можна переглянути актуальні інструкції та приклади використання. Окрім текстових описів, для ілюстрації інтерфейсу часто публікуються скріншоти – наприклад, на початку розділу ClickUp наведено типовий екран організації навчальних завдань у цьому застосунку.

1.2. Платформи для розробки веб-застосунків

Flask – це легкий і гнучкий мікрофреймворк на Python, призначений для створення невеликих і середніх веб-застосунків. Як вказано в офіційній документації, Flask є «легким WSGI веб-фреймворком», створеним для

швидкого старту з можливістю масштабуватися до складних застосунків. Він надає мінімальний набір компонентів (напр., маршрутизацію URL і шаблонізацію через Jinja2), не нав'язуючи жорсткої структури проєкту. Таке «мінімалістичне» ядро («micro-framework»), яке передбачає підключення лише потрібних розширень, гарантує легкість і гнучкість розробки [8].

– Мінімалістичний дизайн: Flask надає лише базову функціональність, тому розробник може сам обирати компоненти та бібліотеки для проєкту.

– Легкість і масштабованість: Завдяки невеликій вазі і малої кількості залежностей Flask добре масштабується при зростанні проєкту. Він підтримує модульну архітектуру, де функціональність можна розділити на окремі «синьки» (Blueprints) і розширення, що спрощує подальший розвиток і тестування.

– Простота освоєння: Flask легко вивчити новачкам, особливо тим, хто вже знайомий з Python. Документація Flask чітка і повна, що спрощує роботу: відладжувач та вбудований сервер полегшують розробку і тестування.

– Розширюваність: Велика спільнота надає численні офіційні і сторонні розширення (ORM, авторизація, форми тощо), які інтегруються з Flask «як ніби вбудовані».

Таким чином, Flask підходить для створення веб-сайтів і REST API будь-якої складності (від простих MVP до бізнес-систем) завдяки своїй гнучкості та модульності.

Для клієнтської частини веб-застосунків використовують HTML і CSS – стандартні технології розмітки та стилізації. Щоб пришвидшити розробку інтерфейсу, застосовують готові CSS-фреймворки: наприклад, Bootstrap або Tailwind CSS. Вони надають набір готових стилів і компонентів для адаптивного дизайну. Bootstrap пропонує «все необхідне з коробки» (кнопки, форми, меню, модальні вікна тощо) разом із JavaScript-плагінами для інтерактивності. Використання Bootstrap дає змогу швидко отримати однаково виглядаючі та адаптивні елементи інтерфейсу на різних пристроях [9]. Tailwind CSS, натомість, застосовує utility-first підхід: замість готових компонентів він

надає безліч утилітарних CSS-класів (для розмірів, відступів, кольорів тощо). Це забезпечує високу гнучкість дизайну – розробник сам компонуватиме класи, формуючи унікальний інтерфейс. В обох випадках фреймворки значно прискорюють створення адаптивного фронтенду. Для динамічної поведінки на клієнті використовують JavaScript. Простий JS-код може керувати взаємодією користувача (валидація форм, відображення меню тощо). Для складніших односторінкових застосунків (SPA) часто застосовують React або Vue.js. Flask легко поєднується з будь-якою JS-бібліотекою: статичні файли збираються окремо і подаються через Flask, або ж фронтенд і бекенд працюють окремо через REST/GraphQL API. Зокрема, при розгортанні фронтенд-фреймворку на Vercel чи інший сервіс часто спілкування відбувається через HTTP API до Flask на бекенді.

Для зберігання даних веб-застосунків використовують реляційні бази даних. SQLite – це «вбудована» файлова СУБД, яка не потребує окремого серверного процесу [10]. Вона зберігає всю базу в одному файлі та підтримує ACID-транзакції, що робить її зручною для малих проєктів, прототипів або локального розроблення. Однак SQLite має обмежену паралельну обробку запитів (одночасно лише один запис). PostgreSQL (Postgres) позиціонує себе як «найбільш просунута» відкрита реляційна СУБД світу. Вона розроблена бути високо розширюваною і стандартно сумісною SQL-базою. Postgres підтримує «об’єктно-реляційні» можливості (успадкування таблиць, перевантаження функцій тощо) і ефективно працює з паралельними транзакціями завдяки MVCC-механізму (Multiversion Concurrency Control), що забезпечує ізольованість і стійкість ACID без блокувань читання. Postgres хоча і трохи менш розповсюджений за MySQL, але має багату екосистему інструментів (pgAdmin, тощо) і активно використовується в продукційних рішеннях. MySQL – дуже популярна СУБД з відкритим кодом, що довгий час лідирувала за рейтингами популярності. Багато великих сайтів (Twitter, Facebook, Netflix тощо) використовують MySQL завдяки його швидкодії та надійності. MySQL створювався з акцентом на швидкість роботи, хоча історично трохи поступався

суворій SQL-стандартній сумісності. Він працює як серверна служба з багатокористувацьким доступом: процес-сервер контролює підключення клієнтів і права доступу. Завдяки широкій спільноті для MySQL існує безліч додаткових інструментів (phpMyAdmin, DBeaver тощо). Для роботи з будь-якою з цих баз даних у Flask зручно застосовувати ORM – об'єктно-реляційні мапери. Найпопулярніший у Python – SQLAlchemy. Це «пітонічний» SQL-інструмент з ORM, що забезпечує ефективний доступ до БД. Розробник працює з Python-об'єктами (класи відповідають таблицям) і викликає методи для запитів, а SQLAlchemy сам генерує потрібні SQL-команди [11]. Крім зручності кодування, SQLAlchemy гарантує незалежність від СУБД: код практично не змінюється, якщо замінити, скажімо, SQLite на PostgreSQL або MySQL. ORM також надає механізми кешування, відкладеного завантаження («lazy loading») та інші оптимізації виконання запитів.

Після розробки WEB-додатоку потрібно розгорнути на хостингу. Поширені PaaS-платформи для Python/Flask: Heroku, Render, Vercel.

Heroku – класична хмарна платформа (PaaS), яка «спрощує розгортання й масштабування Python-додатків» [12]. Для деплою достатньо залити код у Git-репозиторій, налаштувати Procfile і Heroku автоматично підхоплює Flask-сервер (зазвичай із gunicorn). Heroku пропонує множину адд-онів (БД, кешування), простий масштаб (додати динамо) і відладжений робочий процес.

Render – сучасна альтернатива Heroku з безкоштовним тарифом. Render дозволяє хостити веб-сервіси на будь-якій мові та фреймворку (Node.js, Django, Flask тощо) [12]. Коли ви пушите в гілку GitHub/GitLab/Bitbucket, Render сам виконує збірку та деплой застосунку. Кожен сервіс отримує піддомен на render.com, можна додати власний домен. Render підтримує також внутрішню мережу сервісів для приватних сполучень.

Vercel – хостинг, орієнтований переважно на фронтенд (React, Next.js), проте останнім часом підтримує й Python із Flask. У Vercel Python-функції виконуються як Serverless Functions. Наприклад, можна налаштувати проект Flask, який обробляє HTTP-запити через api/–маршрути, а Vercel автоматично

роутить їх у serverless-оточення. Це дає змогу легко запускати API на Vercel разом із фронтендом у єдиному проекті.

GitHub Pages – безкоштовний хостинг статичних сайтів (HTML/CSS/JS) із GitHub-репозиторію. Він підходить для фронтенду таких застосунків (напр., якщо UI зібраний в статичний bundle). Веб-сторінки розгортаються просто: достатньо створити репозиторій username.github.io і запушити туди файли – зміни будуть відразу опубліковані. GitHub Pages не може запускати бекенд-код (тільки статику).

Крім перерахованого, можливі інші хостинги (AWS, Google Cloud, DigitalOcean тощо), але Heroku/Vercel/Render використовуються через простоту та інтеграцію з Git-системою.

Сучасна розробка вимагає використання систем контролю версій. Git – найпопулярніша децентралізована VCS. Розробники зберігають увесь код у Git-репозиторії (локально і на віддаленому сервері). Публічні чи приватні репозиторії розміщуються на сервісах як GitHub, GitLab чи Bitbucket. Git відстежує кожну зміну у файлах, фіксує метадані (хто/коли з чим працював) і дозволяє відкотитися до будь-якої попередньої версії. Використання Git (зокрема GitHub) дає такі переваги:

– Єдине джерело правди: репозиторій на GitHub стає «центральною» версією проєкту, де зберігається актуальний код, історія комітів і документація. Навіть при спільній роботі чи перемиканні між машинами завжди очевидно, де остання стабільна версія і в які зміни було внесено.

– Відстеження змін: Git дозволяє «відстежувати все, що змінилося». Усі додані, видалені або змінені рядки коду фіксуються разом із коментарями до комітів. Це дає повний журнал розвитку проєкту і полегшує розуміння причини кожного виправлення чи нової фічі.

– Співпраця: за допомогою гілок (branches) і Pull Request (запитів на злиття) різні розробники можуть паралельно працювати над різним функціоналом, а потім інтегрувати зміни у головну гілку. Системи CI/CD

(безперервної інтеграції) часто налаштовуються на GitHub для автоматичного тестування при пуші.

Отже, Git + платформи як GitHub є невід'ємним компонентом процесу: вони структурують роботу команди і гарантують безпечне зберігання та відтворення коду.

Для ефективної розробки потрібні зручні інструменти:

Редактори/IDE. Серед найпопулярніших – Visual Studio Code і PyCharm. VSCode – легкий, безкоштовний редактор з розширеннями Python, що забезпечує інтелектуальне підсвічування коду, автодоповнення (IntelliSense), linting, відлагодження та інтеграцію з Git [13]. PyCharm (від JetBrains) – повнофункціональне IDE, яке «з коробки» підтримує рефакторинг, тестування, інтегровану роботу з БД та інші інструменти [14]. Обидва середовища полегшують навігацію по коду, дозволяють керувати віртуальними середовищами і пакунками без виходу з редактора.

Пакетні менеджери. Для керування залежностями Python-проєкту традиційно використовують pip (менеджер пакетів) і вбудовані середовища venv (для ізоляції проєктів). Останнім часом поширюється Poetry – сучасний інструмент для визначення залежностей і створення пакунків Python. Poetry автоматизує встановлення і блокування версій бібліотек, надає прості команди (add, remove, install) для керування залежностями, а також підтримує збирання та публікацію пакунків. Він гарантує відтворювані збірки (детерміністичний підхід) і автоматично створює віртуальне середовище за потреби.

Правильно налаштоване середовище (IDE + віртуальне оточення + залежності) значно полегшує розробку, тести і деплой проекту:

– API-first. У сучасній практиці часто застосовують API-перший підхід: спочатку проектується API-документація (напр., OpenAPI), а потім розробляють бекенд і фронтенд. Це дозволяє розділити роботу: фронтенд-розробники можуть починати за моках API одночасно з розробкою серверної логіки. API-first стратегія сприяє модульності і масштабованості сервісів.

– REST vs GraphQL. Найпопулярнішими архітектурами для API є REST та GraphQL. REST передбачає набір кінцевих точок і стандартні HTTP-методи (GET, POST, PUT, DELETE) для маніпуляцій з ресурсами. GraphQL – це мова запитів до API, де клієнт сам описує, які поля даних йому потрібні, без прив'язки до конкретних endpoint'ів. Обидва підходи дозволяють виконувати операції з даними (отримання, створення, оновлення, видалення), але GraphQL забезпечує більшу гнучкість у виборі даних в одному запиті. REST і GraphQL використовуються залежно від потреб: REST традиційно простий і зрозумілий, GraphQL вигідний при багатьох різних клієнтах із змінними запитами.

– Безпека. З розвитком кіберзагроз безпека веб-застосунку є критично важливою. Сучасний підхід – security by design: WEB-додаток повинен захищати дані за умов «за замовчуванням». Практики включають використання HTTPS, валідацію та санітизацію всіх вхідних даних, захист від SQL-ін'єкцій, XSS, CSRF тощо. Як зазначено, веб-застосунки постійно піддаються атакам з метою викрадення даних, тому «високо безпечні веб-рішення» стають важливішими, ніж будь-коли. Розробник повинен регулярно оновлювати залежності, слідкувати за рекомендаціями OWASP і тестувати додаток на вразливості.

– Адаптивний (responsive) дизайн. Сайт чи WEB-додаток має однаково добре виглядати й працювати на різних пристроях – від смартфонів до десктопів. З огляду на різноманітність екранів, «організації повинні будувати додатки, які добре виглядають і працюють на всіх пристроях». Відповідно, при розробці інтерфейсу використовують mobile-first та адаптивні макети (flexbox, grid, медіазапити), а CSS-фреймворки забезпечують готові засоби для створення адаптивних шаблонів.

Таким чином, стек з Flask на бекенді поєднується із сучасними фронтенд-технологіями (HTML/CSS/JS), популярними базами даних та сервісами хостингу, що відповідає актуальним практикам веб-розробки. Вище наведено ключові платформи і інструменти, які використовуються при створенні веб-застосунку для організації особистого часу.

1.3. Аналіз популярних рішень

Google Календар – це онлайн-сервіс для планування подій і зустрічей. Його називають «потужним інструментом планування» і використовують мільйони людей. У програмі можна створювати події, встановлювати сповіщення, об’єднувати кілька календарів і інтегруватися з Gmail та іншими сервісами Google. Інтерфейс Google Календаря спроектовано у вигляді звичних місячного/тижневого/щоденного переглядів, що є доволі інтуїтивним для користувачів (навчальна крива невелика). Спеціальних студентських функцій немає – проте студенти використовують його для розкладів занять і дедлайнів. Додаток має багатомовний інтерфейс (підтримує українську, англійську та багато інших мов) і доступний безкоштовно всім користувачам. Він працює у веб-браузері та в мобільних додатках (Android, iOS), а також підтримує офлайн-режим (можна переглядати останні 4 тижні подій без підключення).

Todoist – кросплатформовий таск-менеджер, оптимізований для створення списків завдань і проєктів. Незважаючи на лаконічний дизайн, він «має всі інструменти, необхідні для будь-якого робочого процесу»: проєкти та підпроєкти, дедлайни, повтори, пріоритети, ярлики (теги), фільтри та інші можливості. Функціонал включає смарт-додатки для різних пристроїв (десктоп, мобільні, розширення браузера). Інтерфейс Todoist чистий і мінімалістичний; багато користувачів відзначають, що він інтуїтивний, але може виявитися неочікувано насиченим для новачків. Спеціальних функцій для студентів немає. Локалізація: програма підтримує багато інших мов (англійська, іспанська, української наразі немає). Модель ціноутворення – freemium: базовий доступ безкоштовний (до 5 проєктів), а розширені можливості (календарний режим, індивідуальні нагадування, історія без обмеження тощо) – у платних тарифах (“Pro” ~4 $ на місяць при річній передплаті). Платформа: веб-версія, мобільні програми (Android, iOS) і десктопні (Windows, macOS, Linux), офлайн-функціонал обмежений.

Notion – універсальне «AI-середовище» для нотаток, баз даних, задач і співпраці. Це єдине «рабоче простір», де можна зберігати вікі-сторінки, документи, таблиці, дошки (Kanban), завдання тощо. Сервіс пропонує шаблони для будь-яких цілей і гнучкий інтерфейс на основі блоків (текстові, табличні, список, код, тощо). Через велику кількість можливостей інтерфейс досить складний і має помітну криву навчання: для ефективної роботи потрібен час на освоєння. Notion особливо популярний серед студентів завдяки безкоштовному освітньому плану (унліміт-сервіс для студентів і викладачів). Програма локалізована обмежено: є англійська, корейська, японська, французька, німецька, іспанська, португальська та кілька бета-версій (фінська, скандинавські мови тощо), але нема української. Ціна: базовий доступ безкоштовний, є платні тарифи Plus, Business тощо. Доступність: повноцінна веб-версія і клієнти для iOS, Android, Windows та macOS; підтримує роботу офлайн (локальні дані) та синхронізацію з сервером.

Trello – візуальний канбан-інструмент для управління проєктами. Основні елементи – дошки, списки та картки; на картки можна додавати чеклісти, ярлики, дедлайни, вкладення тощо. Є автоматизація (Butler), шаблони, інтеграції (Power-Ups) і календарний перегляд. Створити дошку просто – «зареєструйся, створи дошку – і ти готовий до роботи». Інтерфейс дуже наочний і простий у використанні (канбан-дошки з drag&drop); студентам підходить для командних проєктів, але спеціальних навчальних опцій немає. Щодо локалізації: Trello перекладено на десятки мов, у тому числі на українську. Ціна: базова версія безкоштовна (limit 10 Power-Ups на дошку), а платні плани Standard (~5 $/міс при річній передплаті) і Premium (~10 $/міс) додають не лімітовані дошки, розширені представлення тощо. Доступність: веб-версія, мобільні додатки (iOS, Android), десктоп (Windows, macOS); офлайн-режим можливий лише у кешованому браузері.

ClickUp – «все-в-одному» платформа для організації задач і проектів. Має широкий набір функцій: списки задач, підзадачі, чеклісти, документи, цілі, дашборди, відстеження часу, повідомлення, чат і навіть

Kanban/календар/гарнт/картки тощо (15+ видів переглядів). За рахунок багатофункціональності інтерфейс досить перевантажений. Оглядова оцінка: «UI дуже складний і неінтуїтивний». Проте для студентів ClickUp пропонує готові шаблони: планування розкладу, завдань, нотатки (Cornell Notes) та інші, що дозволяє використовувати його і у навчальному процесі. Локалізація: інтерфейс в основному англомовний, української немає (є звернення користувачів з проханням додати їх). Ціна: базова версія безкоштовна (Free Forever), платні Unlimited ~7 $/міс і Business ~12 $/міс при оплаті за рік. Доступність: веб-додаток, клієнти для iOS, Android, Windows, macOS (також є Linux-версія), офлайн-режим – лише через мобільне кешування.

Any.do – простий таск-менеджер з календарем. Основні можливості: створення списків задач, планування «сучасного дня» (щоденний планувальник), календарне переглядання справ, нагадування (таймінг/місцезнаходження), нотатки і вкладення. Програма має чистий інтерфейс із мінімалістичним дизайном, що робить її зручною у використанні. Специфічних студентських функцій немає, проте, як і інші таск-менеджери, підходить для планування домашніх завдань. Any.do підтримує українську, і багато інших мов на мобільних платформах. Є безкоштовна версія (необмежені завдання й підзадачі) та платні плани Premium (від ~$4.99/міс при річній передплаті) і сімейний тариф. Доступність: веб-версія, мобільні додатки (iOS, Android), десктопні (Windows, Mac), розширення для браузерів (Chrome, Firefox) та інтеграція з Siri, офлайн-підтримка базових функцій.

TickTick – універсальний планувальник із поєднанням списку справ, календаря та Pomodoro-таймера. Він дозволяє створювати завдання з дедлайнами й повторенням, встановлювати постійні/розумні нагадування, вести кілька календарних представлень (щоденний, тижневий, місячний), а також вбудований трекер звичок і Pomodoro-таймер для збереження фокусу. Інтерфейс TickTick вважається більш чистим і інтуїтивним, ніж у багатьох конкурентів. З точки зору студентів – немає спеціалізованих функцій, але для них пропонується 25%-ва знижка на Premium-підписку. Локалізація: додаток

підтримує українську (переклади здійснювалися волонтерами). Модель ціни: базовий функціонал доступний безкоштовно, є платний Premium (річна підписка ~$35.99). Платформи: синхронізується між усіма пристроями (веб, iOS, Android, Windows, Mac, Apple Watch); підтримує офлайн-режим (оновлення за підключення).

Focus To-Do – мобільний/десктопний додаток, що поєднує планувальник завдань із Pomodoro-таймером. Функціонал включає списки справ з підзадачами, встановлення дедлайнів і повторів, докладний журнал виконання та статистику, а також власне Pomodoro-таймер з короткими та довгими перервами. Інтерфейс простий, орієнтований на фокус: великий таймер Pomodoro та мінімум відволікаючих елементів. Для студентів корисно, що додаток прямо позиціонується для «роботи та навчання», з підтримкою фокус-сесій і трекінгу навчальних завдань. Локалізація: підтримує українську, англійську та кілька інших мов. Доступні безкоштовна версія та покупки всередині програми (Наприклад, $1.99/міс Premium або $11.99 за lifetime). Платформи: є мобільні додатки (iOS, Android), десктопні програми (Windows, Mac), розширення для Chrome, а також синхронізація даних між ними, офлайн-підтримка – повна (дані зберігаються локально та синхронізуються при підключенні).

iStudiez Pro – планувальник занять і оцінок для студентів. Дозволяє вводити розклад занять із деталями (дні, час, викладач, контакти), а також домашні завдання та екзамени. Вбудований трекер оцінок показує прогрес за кожним предметом. Інтерфейс спроектовано під потреби навчання – він інтуїтивний для створення розкладу (кольори, іконки для предметів). Оцінюванню приділено окремий модуль, що перетворює додаток на повноцінного «навчального помічника». Локалізація: є українська, англійська та інші мови інтерфейсу. Ціна: платний додаток (безкоштовної версії немає); коштує ~$2.99 на iOS/Android або $9.99 для Mac-версії. Доступність: програма є на iPhone/iPad, Android, а також на Mac (також синхронізується через iCloud);

офлайн-функціонал – так, адже дані зберігаються локально із можливістю резервної копії.

MyStudyLife – мобільний і веб-додаток, орієнтований виключно на студентів. Він інтегрує розклад занять, планування домашніх робіт, відстеження екзаменів і нагадування в єдиному календарі. Є добовий, тижневий і місячний перегляди розкладу, «Xtra»-меню для задач поза навчанням (спортивні секції, гуртки тощо). Окрім стандартних функцій планування, реалізовано вбудований Помодоро-таймер для фокус-стадій та трекер оцінок і цілей. Інтерфейс простий і оптимізований під потреби студентів (відповідає назві «особистий асистент навчання»). MyStudyLife локалізовано на українську (серед інших мов: в'єтнамська, китайська спрощена та традиційна). Доступно безкоштовно (є можливість придбання Pro-функцій для відключення реклами чи додаткових опцій). Працює у веб-браузері, а також має мобільні додатки (Android/iOS) з онлайн-синхронізацією; офлайн-режим дозволяє переглядати вже завантажений розклад.

Порівняльна характеристика популярних веб-ресурсів для планування подій занесено в таблицю 1.1.

Таблиця 1.1 – Порівняння популярних веб-ресурсів

WEB-додатки	**Функціональність**	**Зручність інтерфейсу**	**Адаптація студентам**	**Локалізація**	**Ціна**	**Доступність**
Google Календар	Планування подій, нагадування, сповіщення, інтеграції	Звичний календарний інтерфейс, простий у освоєнні	Без спеціальних «студентських» функцій	UA, RU, EN та інші (багатомовний)	Безкоштовно	Web, Android, iOS; офлайн-перегляд
Todoist	Завдання, проєкти, пріоритети, ярлики, фільтри	Мінімалістичний інтерфейс, інтуїтивний для користувача	Без студентських опцій	Підтримує інші (без УКР)	Безкоштовно + Premium (~$4/міс)	Web, iOS, Android, Windows, macOS, Linux, розширення

Продовження таблиці 1.1

WEB-додатки	Функціональність	Зручність інтерфейсу	Адаптація студентам	Локалізація	Ціна	Доступність
Notion	Документи, бази даних, вікі-сторінки, шаблони, завдання	Гнучкий, багато налаштувань; висока крива навчання	Є безкоштовний освітній план для студентів	EN, KO, JP, FR, DE, ES (UA – немає)	Безкоштовно + платні (Plus, Business)	Web, iOS, Android, Windows, macOS
Trello	Канбан-дошки, списки, картки, дедлайни, Butler-автоматизація	Дуже візуальний і простий	Без специфічних функцій (але підходить для груп)	UA, та багато інших мов	Безкоштовно + Standard	Web, iOS, Android, Windows, macOS
ClickUp	Повний набір: завдання, документи, цілі, діаграми, чат	Дуже багатофункціональний, але перевантажений	Є студентські шаблони (розклад, завдання тощо)	EN (немає UA)	Безкоштовно + Unlimited $7/міс, Business $12/міс	Web, iOS, Android, Windows, macOS, Linux
TickTick	Списки, календари, нагадування, Pomodoro, трекер звичок	Інтерфейс чистий і зручний	Немає спеціальних функцій; 25% знижка для студентів	Підтримує UA та інші	Безкоштовно + Premium (~$35.99/рік)	Web, iOS, Android, Windows, Mac (синхронізація)
Focus To-Do	Планер завдань + Pomodoro-таймер, звіти та статистика	Інтерфейс простий, спеціально для фокус-нагодованих сесій	Так (орієнтована на продуктивне навчання/роботу)	UA, EN, CN та інші	Безкоштовно + in-app покупки (Premium)	Web, iOS, Android, Windows, Mac, Chrome ext.

1.4. Переваги та недоліки існуючих застосунків

Організація особистого часу є важливою складовою ефективного навчання та саморозвитку, особливо для студентів, які часто поєднують навчання, роботу та особисті справи. У зв'язку з цим на ринку існує велика кількість застосунків, які пропонують різні функціональні можливості для планування часу, створення нагадувань, ведення списків справ тощо. Однак кожен із цих інструментів має свої сильні та слабкі сторони, які варто враховувати при виборі відповідного рішення.

У таблиці нижче наведено порівняльний аналіз переваг та недоліків найбільш поширених застосунків для організації особистого часу, з урахуванням таких факторів, як функціональність, зручність інтерфейсу, адаптація під потреби студентів, доступність, ціна та локалізація [15].

Аналіз переваг і недоліків розглянутих веб-ресурсів для планування подій показано в таблиці 1.2.

Таблиця 1.2 – Аналіз переваг і недоліків розглянутих веб-ресурсів

Назва застосунку	**Переваги**	**Недоліки**
Notion	- «Все-в-одному» простір з високою гнучкістю: можна створювати необмежену кількість сторінок, блоків, баз даних та шаблонів. - Підходить для командної роботи: є коментарі, @-згадки, версії сторінок та гнучкі права доступу. - Доступний безкоштовний тариф з базовим функціоналом; преміум-версія порівняно недорога.	- Висока складність освоєння: масив можливостей призводить до стрімкого зростання кривої навчання. - Неможливість відстежувати цілі чи генерувати звіти «з коробки» – відсутні вбудовані інструменти звітності. - Мобільний додаток поступається за функціональністю десктопному; кількість інтеграцій менша, ніж у деяких конкурентів.

Продовження таблиці 1.2

Назва застосунку	**Переваги**	**Недоліки**
Todoist	- Інтуїтивний і охайний інтерфейс; містить детальні інструменти організації задач. - Зручне планування завдань і термінів, підтримує повторювані задачі та шаблони проєктів. - Синхронізується між пристроями, підтримує нагадування.	- Обмежена кількість сторонніх інтеграцій. - Багато можливостей одночасно можуть «спантеличити» новачків – є крута крива навчання для нових користувачів. - Інколи повідомляють про затримки синхронізації та сповіщень, через що оновлення не завжди відбувається миттєво.
Google Calendar	- Безкоштовний, має простий «чистий» інтерфейс; тісно інтегрований з екосистемою Google. - Підтримує кілька календарів з різними правами доступу, дозволяє встановлювати нагадування та розсилати запрошення на події.	- Обмежені можливості кастомізації вигляду. -Слабка інтеграція з багатьма зовнішніми сервісами. - Неможливість виконання складного планування і обмеження у безпеці даних.
ClickUp	- Дуже гнучкий і настроюваний інтерфейс: можна створювати власні робочі процеси і налаштовувати простір під потреби команди. - Інтуїтивно зрозумілий інтерфейс – всі завдання та проєкти відображаються в єдиному дашборді. - Багато інструментів для співпраці: коментарі, чеклісти, @-згадки; є призначення відповідальних, дедлайни, спільний доступ до документів.	- Багатофункціональність ускладнює освоєння: багато інструментів потребують часу на навчання. - Обмежена кількість доступних інтеграцій; немає телефонної підтримки. - Складна тарифна політика: важко обрати оптимальний план, можна переплатити за непотрібні опції.

Основними перевагами більшості сучасних застосунків для організації часу є наявність базових інструментів планування (календарі, списки справ, нагадування) та синхронізація даних між пристроями. Більшість з них пропонують безкоштовні версії з основним функціоналом і роблять інтерфейс інтуїтивним (спрощений дизайн, drag&drop тощо) [16]. Натомість розвинутий набір можливостей призводить до того, що новим користувачам потрібен час на освоєння програми. Типовими недоліками є обмеженість налаштувань інтерфейсу і кастомізації, а також блокування багатьох корисних опцій у безкоштовних пакетах (наприклад, розширені нагадування чи аналітика). Крім того, для повної роботи таких сервісів зазвичай потрібен стабільний інтернет. У сукупності розглянуті застосунки значно полегшують щоденне планування й організацію особистого часу, але для максимальної ефективності часто потрібен час на вивчення їхніх можливостей та, за необхідності, перехід на платні функції.

1.5. Постановка задачі на розробку застосунку

У результаті аналізу існуючих рішень для організації особистого часу було виявлено, що жоден з розглянутих застосунків повністю не відповідає всім потребам сучасного студента. Хоча такі сервіси, як Google Calendar, Todoist, Trello, Notion, Microsoft To Do та Any.do, пропонують широкий спектр функцій, вони мають певні обмеження, які можуть ускладнювати або обмежувати їх ефективне використання в освітньому середовищі.

Серед основних недоліків, на які варто звернути увагу, є відсутність локалізації українською мовою в деяких сервісах, обмеження функціональності у безкоштовних версіях, відсутність спеціалізованих функцій для студентів (наприклад, управління навчальними модулями, інтеграції з розкладом занять, моніторингу дедлайнів курсових робіт тощо), а також складність інтерфейсу, що може відштовхувати менш досвідчених користувачів.

Ураховуючи викладене, доцільним є створення власного веб-застосунку для організації особистого часу, який буде орієнтований насамперед на потреби студентської аудиторії. Основна мета такого застосунку — забезпечити інтуїтивно зрозумілий, безкоштовний, функціональний і локалізований інструмент, який дозволятиме ефективно планувати особистий час, відстежувати завдання, формувати графік занять, отримувати сповіщення про важливі події та інтегруватися з іншими сервісами за потреби.

З урахуванням вищезазначеного, можна сформулювати наступні завдання, які має вирішувати WEB-додаток:

– Надання зручного інтерфейсу для створення, редагування та видалення завдань;

– Побудова календаря або таймлайну для візуального відображення графіка користувача;

– Можливість формування тижневого/місячного розкладу з можливістю відображення подій, дедлайнів, занять;

– Підтримка сповіщень (через браузер або мобільні повідомлення) для нагадування про важливі події;

– Підтримка української мови та сучасних технологій веб-розробки;

– Можливість масштабування та подальшого розвитку застосунку (наприклад, створення мобільної версії).

Таким чином, розробка власного веб-застосунку для організації особистого часу дозволить усунути наявні недоліки в існуючих рішеннях і створити ефективний інструмент для студентської аудиторії, який сприятиме підвищенню особистої продуктивності, дисципліни та ефективності управління часом.

1.6. Висновки до розділу 1

У першому розділі було здійснено огляд і аналіз сучасних рішень, які використовуються для організації особистого часу. Розглянуто функціональні

можливості таких популярних сервісів, як Google Calendar, Todoist, Notion, Trello, а також вітчизняних аналогів. Було проаналізовано їх переваги й недоліки, зокрема з точки зору зручності використання, адаптивності, підтримки української локалізації, вартості та доступності.

Також було розглянуто платформи й технології, що використовуються для розробки веб-застосунків, включаючи мови програмування, фреймворки та бібліотеки. Проведено порівняння рішень з точки зору їх застосування у створенні інтерфейсів, зберіганні даних, реалізації авторизації та інтеграцій із зовнішніми сервісами.

На основі проведеного аналізу сформульовано постановку задачі, яка охоплює створення веб-застосунку з функціоналом реєстрації, створення та керування завданнями, нагадуванням про дедлайни, адаптивним інтерфейсом і потенційною інтеграцією з іншими сервісами.

Результати першого розділу слугуватимуть теоретичним підґрунтям для подальшого етапу — проектування архітектури, інтерфейсу та реалізації застосунку, що буде розглянуто в наступному розділі.

2 ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ОСОБИСТОГО ЧАСУ

2.1. Огляд вимог до функціональності застосунку

Проєктування програмного забезпечення починається з чіткого формування вимог до його функціональності та загальної концепції використання. У контексті даної дипломної роботи розглядається розробка веб-застосунку, що має забезпечити користувачеві можливість ефективного планування особистого часу. Основна цільова аудиторія — студенти вищих навчальних закладів, які зазвичай поєднують навчання, самостійну підготовку, роботу, дозвілля та інші життєві активності, потребуючи при цьому зручного інструменту для організації повсякденного навантаження.

Додаток має функціонувати як автономна веб-платформа, доступна з будь-якого сучасного браузера без необхідності встановлення додаткового програмного забезпечення [17]. Це дозволяє охопити ширше коло користувачів і гарантувати доступність сервісу незалежно від операційної системи чи типу пристрою. Основні функціональні можливості повинні включати систему обліку користувачів з можливістю реєстрації, авторизації та подальшого доступу до персонального середовища, в межах якого користувач може створювати та редагувати власні завдання, формувати розклад, отримувати нагадування та візуально планувати свою діяльність.

Особливу увагу слід приділити інтуїтивній побудові інтерфейсу, що дозволить швидко зорієнтуватися навіть користувачам без спеціальної технічної підготовки. Важливо забезпечити гнучкість у налаштуванні завдань — наприклад, можливість вказання точного часу, пріоритету, повторюваності або дедлайнів. Також необхідною є реалізація календарного представлення інформації, що дозволить візуалізувати розподіл навантаження у часі [18].

Ще одним важливим функціональним аспектом є можливість надсилання нагадувань про заплановані події. У межах веб-застосунку це може бути реалізовано у формі локальних браузерних сповіщень або ж через інтеграцію з

електронною поштою користувача. Така функція дозволяє уникнути пропущених дедлайнів або важливих подій, що особливо актуально в умовах високого ритму студентського життя.

Крім того, є можлива реалізація інтеграції із зовнішніми сервісами — зокрема, з Google Calendar або іншими аналогами, які вже використовуються в обширному крузі користувачів. Це дозволить забезпечити синхронізацію між подіями, уникати дублювання інформації та створить можливість використання застосунку в якості єдиного центру управління часом.

Окремо слід згадати питання безпеки. Зважаючи на необхідність обробки персональних даних (електронна пошта, облікові записи, нотатки), система повинна передбачати відповідні механізми захисту, включно з безпечним зберіганням паролів, захистом від несанкціонованого доступу та відповідністю загальним принципам конфіденційності [19].

Застосування зазначеного функціонального набору є цілком виправданим з огляду на сучасні тенденції в сфері особистої продуктивності та цифрового планування. Середовище характеризується значною кількістю змінних — розклад занять може змінюватися щотижня, дедлайни завдань часто оновлюються, виникає потреба у гнучкому керуванні позанавчальною діяльністю, зустрічами, додатковими курсами або проєктною роботою. Саме тому інструмент, який дозволяє оперативно створювати, переглядати, редагувати або видаляти записи, отримувати нагадування та візуалізувати особисте навантаження, є вкрай актуальним.

Доцільність інтеграції із зовнішніми календарями, такими як Google Calendar, пояснюється тим, що чимало користувачів вже мають сформовані звички взаємодії з цими системами. Наявність такої інтеграції сприяє плавному переходу на новий додаток без втрати важливої інформації. Крім того, вона дозволяє уніфікувати інформаційні потоки, зменшити ризик дублювання подій та підвищити загальну ефективність роботи користувача з часом.

Таким чином, розглянутий функціональний спектр забезпечує не лише задоволення базових потреб користувача, а й створює передумови для

глибшого залучення до процесу особистого планування, формування звичок тайм-менеджменту та підвищення загального рівня самоорганізації.

2.2. Структура системи застосунку

Структура розроблюваного веб-застосунку для організації особистого часу базується на класичній клієнт-серверній архітектурі, що передбачає чіткий поділ між інтерфейсом користувача, логікою обробки даних та збереженням інформації. Такий підхід дозволяє забезпечити масштабованість, гнучкість у розширенні функціоналу, а також незалежність між окремими частинами системи.

Серверна частина реалізована за допомогою фреймворку Flask, який надає необхідну інфраструктуру для обробки HTTP-запитів, маршрутизації, керування сесіями та взаємодії з базою даних. В якості бази даних використовується SQLite — легка файлова СУБД, що добре підходить для невеликих веб-застосунків із середнім навантаженням [8]. Робота з базою даних здійснюється за допомогою ORM-бібліотеки SQLAlchemy, яка дозволяє уникнути прямого написання SQL-запитів і спрощує логіку маніпуляції даними.

Фронтенд застосунку побудований на HTML-шаблонах із використанням системи шаблонізації Jinja2, яка є інтегрованою складовою Flask [12]. Для стилізації інтерфейсу використовується фреймворк Bootstrap, що дозволяє забезпечити адаптивний дизайн та візуальну зручність користувача без необхідності глибокого занурення у CSS. JavaScript наразі не застосовується, проте в подальшому може бути інтегрований для підвищення інтерактивності інтерфейсу.

Аутентифікація та управління сесіями реалізовано з використанням розширення Flask-Login, яке забезпечує безпечне зберігання інформації про авторизованого користувача, обмеження доступу до приватного контенту та підтримку механізмів реєстрації та входу в систему [20].

Для реалізації функції нагадувань у вигляді електронних листів застосовується бібліотека Flask-Mail, яка дозволяє відправляти повідомлення користувачам з попередженнями про заплановані події або дедлайни [21]. Це розширення надає інтерфейс для підключення SMTP-серверу та конфігурації шаблонів листів.

Незважаючи на відсутність адміністративної панелі, у системі передбачено подальше розширення функціональності, зокрема через інтеграцію із зовнішніми сервісами, такими як Google Calendar або подібними API. Така інтеграція дозволить у майбутньому значно підвищити зручність користування і збагатити можливості взаємодії із застосунком.

Типова взаємодія користувача із системою включає кілька послідовних кроків, що охоплюють увесь життєвий цикл користування веб-застосунком. Спершу користувач реєструється або входить у систему через відповідну форму. Після успішної авторизації він потрапляє на головну сторінку, де може створювати нові задачі, переглядати вже створені, редагувати або видаляти їх. Крім того, система періодично або за розкладом надсилає електронні повідомлення-нагадування про майбутні або прострочені події. Такий цикл дозволяє ефективно організовувати особистий час користувача та підтримувати високу залученість у плануванні власної діяльності як показано на рисунку 2.1.

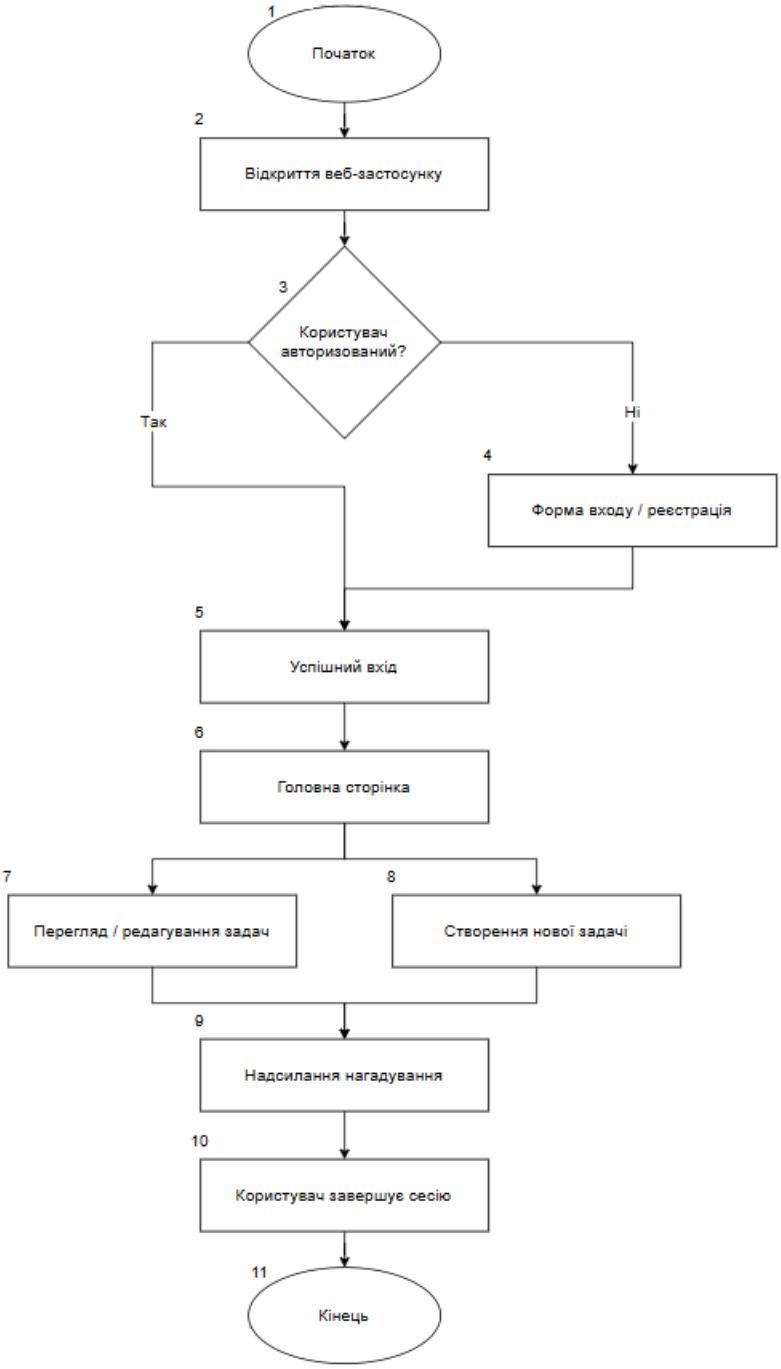

Рисунок 2.1 – Типова схема взаємодії користувача із системою

Зображена схема взаємодії демонструє типовий сценарій користування, починаючи від авторизації й закінчуючи керуванням задачами та завершенням сесії. Така послідовність дій дозволяє забезпечити логічну та інтуїтивну навігацію користувача, що є ключовим чинником зручності та ефективності у щоденному використанні системи.

Як показано на рисунку 2.2, запити від користувача обробляються сервером, який у разі потреби звертається до бази даних або сторонніх API. Реалізовано класичну схему взаємодії між користувачем, інтерфейсом, серверною частиною та допоміжними сервісами. Основним каналом комунікації виступає веб-браузер, через який користувач ініціює HTTP-запити до серверу, побудованого на Flask. У відповідь сервер виконує обробку запиту, взаємодіє з базою даних SQLite через ORM SQLAlchemy, формує сторінку з використанням шаблонів Jinja2 та надсилає її у вигляді HTML-відповіді у браузер [23].

Рисунок 2.2 – Схема взаємодії основних компонентів веб-застосунку

У випадку необхідності система може ініціювати надсилання електронного листа через Flask-Mail, а також передбачено можливість інтеграції із зовнішніми API, наприклад для синхронізації з календарями чи іншими сервісами управління задачами.

2.3. Розробка архітектури застосунку

Архітектура веб-додатку побудована за принципами клієнт-серверної моделі з розділенням відповідальностей між фронтендом, бекендом та базою

даних. Веб-браузер користувача виступає як клієнтська частина, що надсилає HTTP-запити до Flask-сервера. Серверна частина реалізована за допомогою Python-фреймворку Flask та використовує SQLAlchemy як ORM для взаємодії з базою даних SQLite. Формування HTML-вмісту здійснюється через шаблонізатор Jinja2, а для стилізації інтерфейсу застосовується фреймворк Bootstrap. Надсилання нагадувань реалізовано через модуль Flask-Mail. Вся ця архітектура показана на рисунку 2.3.

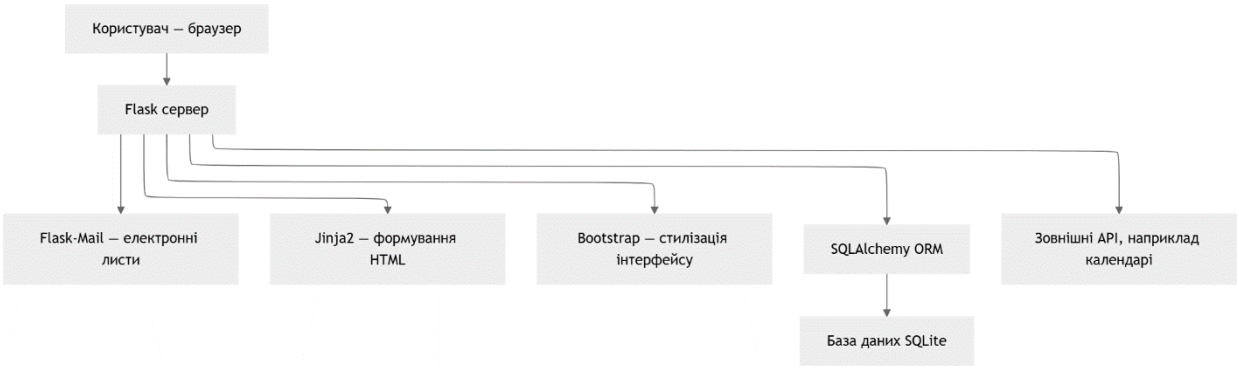

Рисунок 2.3 – Архітектура веб-застосунку

Запропонована архітектура забезпечує модульність, простоту масштабування та легкість у підтримці системи. Кожен компонент виконує чітко визначену роль, що дозволяє в майбутньому доповнювати функціональність, не порушуючи цілісності застосунку. Такий підхід сприяє стабільній роботі системи та забезпечує зручність у її подальшому розширенні.

У межах архітектури застосунку реалізовано дві основні моделі даних: Користувач (User) та Задача (Task). Клас Користувач відповідає за ідентифікацію зареєстрованих осіб, зберігання їх логіну, email-адреси та хешованого пароля. Він також має зв’язок один-до-багатьох із класом Задача, що дозволяє кожному користувачеві створювати власний набір завдань. Логіка обробки запитів у веб-застосунку реалізована через набір маршрутів Flask, кожен з яких відповідає за певну дію користувача. Основні дії включають перегляд головної сторінки із завданнями, реєстрацію, авторизацію, створення, редагування та видалення задач, а також зміну їх статусу. На рисунку 2.4

представлено умовну UML-діаграму, яка ілюструє ці дії як логічно окремі компоненти контролера.

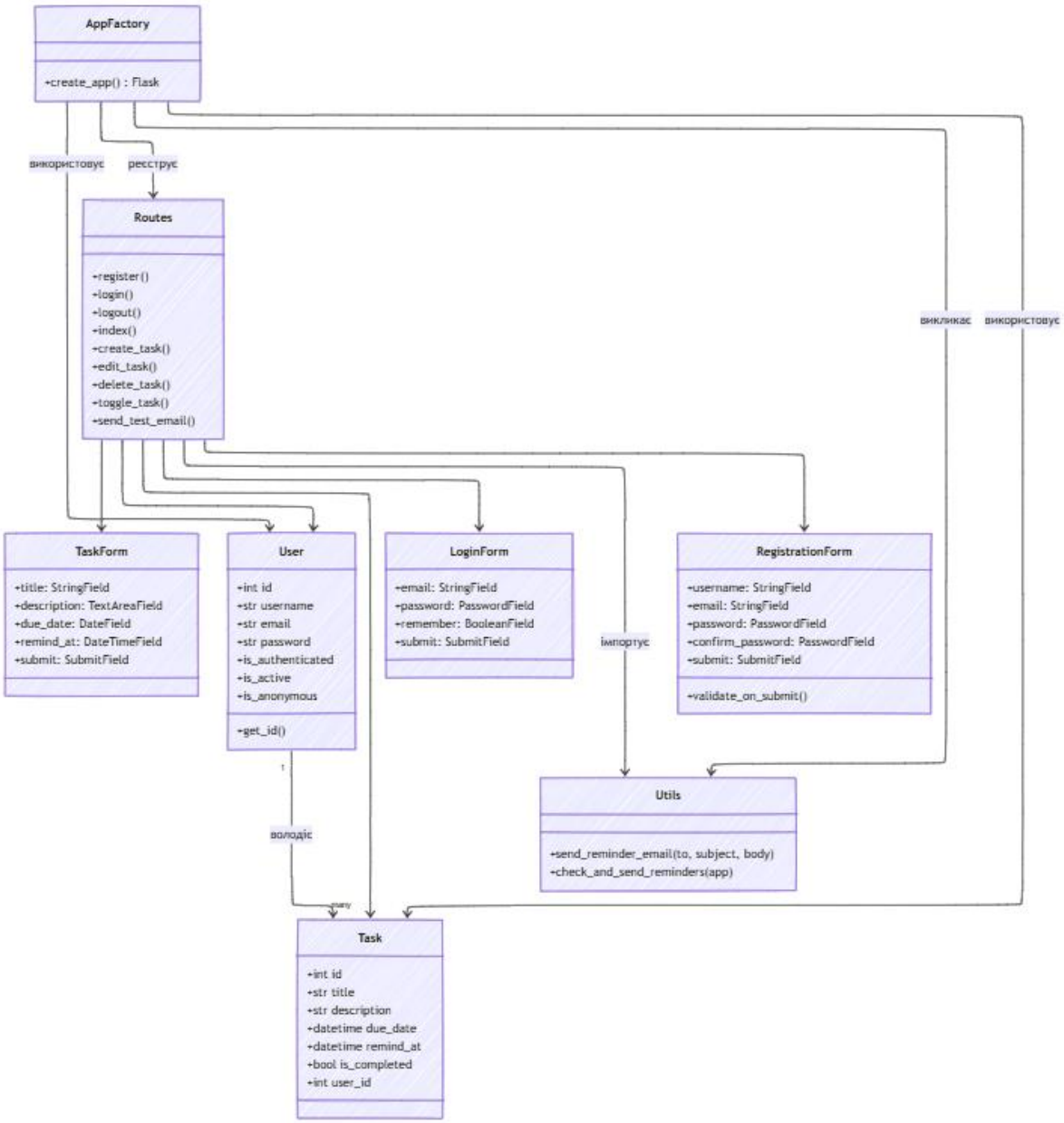

Рисунок 2.4 – UML-діаграма класів застосунку

У рамках розробки програмного забезпечення важливим етапом є формалізація логіки виконання основних процесів системи. Це дозволяє не лише краще зрозуміти внутрішню структуру роботи застосунку, а й візуалізувати послідовність дій користувача та реакції системи. З цією метою доцільно використовувати схемно-алгоритмічне подання основних сценаріїв взаємодії, що забезпечує наочність та структурованість опису функціонування.

Побудова блок-схем алгоритмів дозволяє представити ключові механізми, які відбуваються під час виконання дій користувачем: реєстрації, авторизації, створення задачі, редагування інформації тощо. Такі схеми відображають послідовність дій, умови переходу між етапами, перевірки даних та відповіді сервера. Їхнє використання у процесі проєктування значно спрощує реалізацію логіки на етапі програмування та допомагає уникнути помилок, пов’язаних з нечітко визначеною поведінкою системи.

У цьому розділі буде наведено низку алгоритмічних схем, що описують основні сценарії використання веб-застосунку. Кожен з алгоритмів супроводжується покроковим текстовим описом і відповідною блок-схемою на рисунку 6.3.

І сценарій

Алгоритм реєстрації користувача складається з таких кроків:

Крок 1. Якщо користувач натиснув кнопку «Зареєструватися», то перейти до кроку 2, інакше – до кроку 14.

Крок 2. Користувач вводить своє ім’я у відповідне поле форми реєстрації.

Крок 3. Користувач вводить свою електронну пошту у відповідне поле форми реєстрації.

Крок 4. Користувач вводить пароль у відповідне поле форми реєстрації.

Крок 5. Користувач підтверджує пароль у відповідному полі форми реєстрації.

Крок 6. Якщо ім’я або електронна пошта вже існують у базі даних, то перейти до кроку 7, інакше – до кроку 8.

Крок 7. Сервер повертає повідомлення про помилку, і користувач повертається до кроку 2.

Крок 8. Сервер хешує пароль користувача.

Крок 9. Сервер створює новий запис користувача в базі даних.

Крок 10. Сервер надсилає повідомлення «Акаунт створено! Тепер ви можете увійти».

Крок 11. Користувач перенаправляється на сторінку входу.

Крок 12. Користувач натискає кнопку «Увійти», і перейти до кроку 14.

Крок 13. Якщо користувач не натискає «Увійти», то повернутися до кроку 1.

II сценарій

Алгоритм автентифікації користувача складається з таких кроків:

Крок 14. Якщо користувач натиснув кнопку «Увійти», то перейти до кроку 15, інакше – до кроку 26.

Крок 15. Користувач вводить свою електронну пошту у відповідне поле форми входу.

Крок 16. Користувач вводить свій пароль у відповідне поле форми входу.

Крок 17. Користувач обирає, чи запам’ятати його (опція «Запам’ятати мене»).

Крок 18. Сервер перевіряє, чи існує користувач із такою електронною поштою в базі даних.

Крок 19. Якщо користувач існує і пароль збігається, то перейти до кроку 20, інакше – до кроку 21.

Крок 20. Сервер автентифікує користувача, і перейти до кроку 22.

Крок 21. Сервер повертає повідомлення «Неуспішна спроба входу. Перевірте email і пароль», і повернутися до кроку 15.

Крок 22. Користувач перенаправляється на головну сторінку.

Крок 23. Сервер відображає привітання з ім’ям користувача та список задач.

Крок 24. Сервер додає опції для створення задачі та виходу.

Крок 25. Користувач обирає дію, і перейти до кроку 26.

III сценарій

Алгоритм створення та управління задачею складається з таких кроків:

Крок 26. Якщо користувач авторизований, то перейти до кроку 27, інакше – до кроку 43.

Крок 27. Якщо користувач обрав створення задачі, то перейти до кроку 28, інакше – до кроку 34.

Крок 28. Користувач вводить назву задачі у відповідне поле форми.

Крок 29. Користувач вводить опис задачі, дату нагадування та дедлайн у відповідні поля.

Крок 30. Якщо введені дані валідні (наприклад, формат дати), то перейти до кроку 31, інакше – до кроку 32.

Крок 31. Сервер повертає повідомлення про помилку введення, і повернутися до кроку 29.

Крок 32. Сервер створює нову задачу в базі даних із введеними даними.

Крок 33. Сервер показує повідомлення «Завдання створено!» і перенаправляє на головну сторінку.

Крок 34. Якщо користувач обрав редагування задачі, то перейти до кроку 35, інакше – до кроку 40.

Крок 35. Користувач вводить нову назву задачі у відповідне поле.

Крок 36. Користувач вводить новий опис задачі у відповідне поле.

Крок 37. Якщо введені дані валідні, то перейти до кроку 38, інакше – до кроку 39.

Крок 38. Сервер повертає повідомлення про помилку оновлення, і повернутися до кроку 35.

Крок 39. Сервер оновлює запис задачі в базі даних.

Крок 40. Сервер показує повідомлення «Задачу оновлено успішно!» і перенаправляє на головну сторінку.

Крок 41. Якщо користувач обрав видалення задачі, то перейти до кроку 42, інакше – до кроку 43.

Крок 42. Сервер видаляє задачу з бази даних, показує повідомлення «Задачу видалено успішно!» і перенаправляє на головну сторінку.

IV сценарій

Алгоритм позначення задачі як виконаної складається з таких кроків:

Крок 43. Якщо користувач авторизований, то перейти до кроку 44, інакше – до кроку 59.

Крок 44. Якщо користувач хоче позначити задачу як виконану, то перейти до кроку 45, інакше – до кроку 50.

Крок 45. Сервер перевіряє, чи користувач є власником задачі.

Крок 46. Якщо користувач має права, то перейти до кроку 47, інакше – до кроку 48.

Крок 47. Сервер змінює статус задачі (is_completed) на протилежний (виконана/невиконана).

Крок 48. Сервер повертає повідомлення «У вас немає прав змінювати цю задачу» і перенаправляє на головну сторінку.

Крок 49. Сервер показує повідомлення «Статус задачі оновлено» і перенаправляє на головну сторінку.

Крок 50. Якщо користувач хоче вийти з системи, то перейти до кроку 51, інакше – до кроку 59.

Крок 51. Сервер завершує сесію користувача.

Крок 52. Користувач перенаправляється на сторінку привітання.

Крок 53. Користувач бачить опції входу або реєстрації.

Крок 54. Якщо користувач обирає вхід, то перейти до кроку 14.

Крок 55. Якщо користувач обирає реєстрацію, то перейти до кроку 1.

Крок 56. Якщо користувач хоче продовжити роботу, то перейти до кроку 26.

Крок 57. Якщо користувач не обирає дію, то перейти до кроку 59.

Крок 58. Користувач завершує роботу з програмою.

Крок 59. Кінець роботи програми.

Алгоритм роботи цих випадків зображенно на рисунку 6.3.

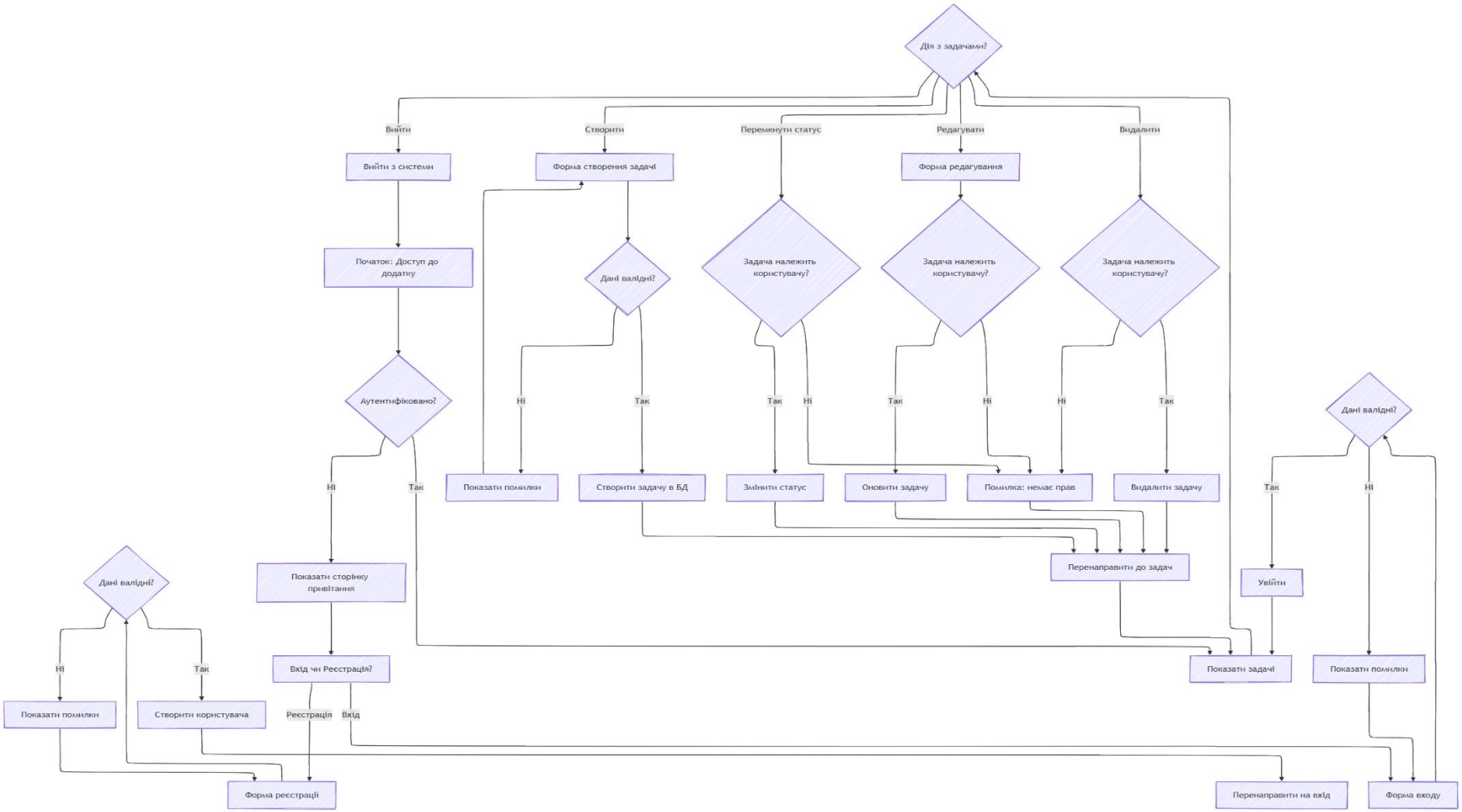

Рисунок 6.3 – Блок-схема алгоритму

2.4. Висновоки до розділу 2

У даному розділі було здійснено системне проєктування веб-застосунку для організації особистого часу студентів. На основі сформульованих функціональних потреб та результатів аналізу існуючих рішень було визначено основні вимоги до майбутнього програмного забезпечення. Серед ключових можливостей, які повинен підтримувати додаток, визначено: створення й редагування задач, формування календарного представлення подій, надсилання нагадувань електронною поштою, підтримку користувацької реєстрації та авторизації, а також адаптацію інтерфейсу до потреб цільової аудиторії.

Було побудовано логічну структуру системи з виділенням основних компонентів: клієнтської частини, серверної частини на Flask, бази даних SQLite з ORM SQLAlchemy та допоміжних модулів для обробки електронної пошти та авторизації. Представлено діаграму архітектури, яка ілюструє взаємозв’язки між модулями системи, а також UML-діаграми класів, що описують моделі «Користувач» та «Задача», їх атрибути та зв’язки.

Також було побудовано схеми, які описують типову взаємодію користувача із системою, включно з блок-схемою алгоритму реєстрації, що деталізує ключові етапи перевірки, створення облікового запису та верифікації електронної пошти. Таке поетапне проєктування дозволило сформувати цілісну картину логіки роботи майбутнього застосунку й створити передумови для його ефективної розробки.

За результатами виконаної роботи можна зробити висновок, що сформована архітектурна модель відповідає сучасним вимогам до зручності, масштабованості та функціональної гнучкості веб-сервісів. Розроблений підхід дозволяє ефективно перейти до наступного етапу — безпосередньої реалізації програмного продукту.

3. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ ОСОБИСТОГО ЧАСУ

3.1. Вибір технологій та інструментів для реалізації застосунку

У процесі реалізації веб-застосунку для організації особистого часу було прийнято рішення використовувати стек технологій, який поєднує простоту, ефективність та можливість швидкої розробки MVP-продукту. Основними критеріями для вибору технологій були: підтримка розробки веб-інтерфейсу, зручність у реалізації backend-логіки, можливість масштабування та інтеграції з зовнішніми сервісами, доступність документації та активність спільноти.

Мовою програмування було обрано Python, зокрема мікрофреймворк Flask, що забезпечує легкий та зрозумілий спосіб створення серверної частини. Flask відзначається простою структурою проєкту, широким вибором розширень (Flask-Login, Flask-Mail, Flask-WTF тощо), а також добре підходить для невеликих проєктів, які можуть масштабуватись у майбутньому [8]. Також у Flask реалізовано підтримку шаблонів через Jinja2, що дозволяє ефективно формувати HTML-код на стороні сервера.

```
db = SQLAlchemy()
login_manager = LoginManager()
mail = Mail()
def create_app():
    app = Flask(__name__)
    app.config.from_object('config.Config')  # Підключення конфігурацій
    db.init_app(app)        # Ініціалізація бази даних
    login_manager.init_app(app)  # Ініціалізація менеджера входу
    mail.init_app(app)      # Ініціалізація пошти
    from app.routes import routes
    app.register_blueprint(routes)  # Реєстрація blueprint'у з маршрутами
    return app
```

У цьому фрагменті створюється об’єкт Flask-додатку, завантажується конфігурація з окремого класу Config, а також виконується підключення ключових компонентів — SQLAlchemy (ORM для бази даних), Flask-Login для авторизації користувачів і Flask-Mail для відправлення email-повідомлень. Це дозволяє відокремити основну логіку від конфігурацій і забезпечує масштабованість додатку. Такий підхід забезпечує розширюваність проєкту та дозволяє структурувати логіку в окремих модулях.

Візуальна частина застосунку реалізована з використанням CSS-фреймворку Bootstrap [9]. Завдяки цьому забезпечено адаптивність інтерфейсу, тобто можливість коректного відображення як на десктопних, так і на мобільних пристроях. Крім того, Bootstrap дозволив швидко створити форми, панелі навігації та кнопки, дотримуючись єдиного стилю оформлення.

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card shadow rounded">
        <div class="card-body">
          <h2 class="mb-4 text-center">Вхід</h2>
          <form method="POST">
            <div class="mb-3">
              <label for="email" class="form-label">Email</label>
              <input type="email" class="form-control" id="email" name="email" required>
            </div>
            <div class="mb-3">
              <label for="password" class="form-label">Пароль</label>
              <input type="password" class="form-control" id="password" name="password" required>
            </div>
            <div class="d-grid">
              <button type="submit" class="btn btn-primary">Увійти</button>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

Продемонстрований HTML-шаблон показує використання класів Bootstrap для структурування контенту, відображення форм введення та кнопок.

Використані стандартні класи Bootstrap (container, form-control, btn, card) для структурування інтерфейсу форми входу. Такий підхід забезпечує адаптивність, візуальну послідовність і зменшує потребу в написанні власного CSS-коду.

У якості системи управління базами даних було обрано SQLite у поєднанні з ORM-бібліотекою SQLAlchemy [11]. Такий вибір пояснюється простотою налаштування та зручністю взаємодії з базою даних у вигляді об'єктів Python. Це значно спрощує роботу зі збереженням, оновленням та видаленням даних користувачів та їхніх задач.

Для реалізації системи авторизації та автентифікації обрано Flask-Login. Це розширення дозволяє реалізувати реєстрацію, вхід, вихід користувача, зберігання його сесії, а також обмеження доступу до функціоналу, що вимагає автентифікації.

Функціональність email-нагадувань реалізується за допомогою Flask-Mail [21]. В даному випадку використовується SMTP-сервер (наприклад, Mailtrap) для надсилання листів на електронну пошту користувача згідно з визначеним графіком задач.

```
MAIL_SERVER = 'sandbox.smtp.mailtrap.io'
MAIL_PORT = 587
MAIL_USE_TLS = True
MAIL_USERNAME = '*****'
MAIL_PASSWORD = '*****'
MAIL_DEFAULT_SENDER = '*****'
```

У наведеному прикладі показано підключення до тестового SMTP-сервера Mailtrap [22]. Параметри MAIL_USERNAME та MAIL_PASSWORD надаються самим сервісом Mailtrap і використовуються лише в середовищі розробки. Такий варіант зручний на етапі розробки, оскільки дозволяє безпечно перевіряти роботу email-функціоналу без надсилання реальних листів користувачам.

Інтерфейс не містить адміністративної панелі, оскільки додаток орієнтований на кінцевого користувача (студента), і всі дії відбуваються в межах особистого кабінету.

Вибрані технології:

– Python 3.11 (мова програмування загального призначення з підтримкою ООП та багатим набором бібліотек)

– Flask 2.x (легкий мікрофреймворк для розробки веб-додатків на Python)

– Flask-Mail (розширення для надсилання електронних листів через SMTP у Flask-застосунках)

– Flask-Login (розширення для реалізації авторизації, автентифікації та управління сесією користувача)

– Jinja2 (шаблонізатор для формування HTML-коду на стороні сервера у Flask)

– SQLAlchemy (ORM-бібліотека для роботи з базами даних у вигляді об’єктів Python)

– SQLite (вбудована реляційна база даних, що не потребує окремого серверу)

– Bootstrap 5 (фреймворк для створення адаптивного та привабливого інтерфейсу користувача)

Для розробки використовувалось інтегроване середовище розробки PyCharm Community Edition. Код додатку структурується у вигляді модулів, що полегшує супровід проєкту.

3.2. Реалізація основних функцій застосунку

Основною метою даного застосунку є надання користувачеві можливості ефективного керування особистими завданнями. У межах цього підпункту розглянуто реалізацію ключового функціоналу: реєстрації та авторизації користувача, створення, редагування, перегляду та видалення задач. Для

побудови логіки застосовано фреймворк Flask, а маршрути реалізовано в окремому файлі views.py, що відповідає за обробку HTTP-запитів.

Реєстрація користувача реалізована через маршрут /register, який підтримує методи GET і POST. У разі запиту GET відображається HTML-форма, де користувач може ввести логін, електронну пошту та пароль. Після заповнення та надсилання форми (POST-запит) система перевіряє, чи вже існує користувач із таким email. Якщо так — повертається повідомлення про помилку; якщо ні — створюється новий запис у базі даних.

```
if form.validate_on_submit():
    hashed_password = generate_password_hash(form.password.data)
    user = User(username=form.username.data, email=form.email.data, password=hashed_password)
    db.session.add(user)
    db.session.commit()
    login_user(user)
    flash('Акаунт створено! Ви увійшли до системи.', 'success')
    return redirect(url_for('routes.index'))
```

Цей код демонструє логіку обробки запиту: перевірку email, хешування пароля та збереження нового користувача. Після успішної реєстрації користувач автоматично входить у систему та перенаправляється на головну сторінку.

Форма реєстрації створена у відповідному HTML-шаблоні, що містить поля для введення даних та кнопку підтвердження. У шаблоні використовуються стандартні класи Bootstrap для побудови форми, що забезпечує адаптивність і зручність введення на будь-якому пристрої.

Процес входу в систему реалізований через маршрут /login. Аналогічно до реєстрації, спочатку відображається форма, після чого дані перевіряються. Якщо введені облікові дані правильні, користувач аутентифікується і перенаправляється до особистого кабінету.

Авторизація здійснюється за допомогою функції login_user з бібліотеки Flask-Login, яка встановлює сесію користувача. Також передбачено механізм виходу з системи за допомогою маршруту /logout.

Ще однією ключовою функцією є створення задачі. Реалізація виконується в маршруті /create, де через форму користувач має змогу вказати заголовок задачі, опис, дедлайн, дату нагадування тощо. Після натискання кнопки “Зберегти” ці дані передаються на сервер і записуються у відповідну таблицю бази даних.

```
if request.method == 'POST':
    title = request.form['title']
    description = request.form['description']
    due_date_str = request.form.get('due_date')
    due_date = datetime.strptime(due_date_str, "%Y-%m-%d") if due_date_str else None
    task = Task(
        title=title,
        description=description,
        due_date=due_date,
        user_id=current_user.id
    )
    db.session.add(task)
    db.session.commit()
    flash('Завдання створено!', 'success')
    return redirect(url_for('routes.index'))
```

Тут відображено логіку обробки POST-запиту — створення нового екземпляра класу Task та його збереження в базі даних. Також здійснюється перевірка, чи всі необхідні поля заповнені, та обробка помилок.

Форма включає стандартні елементи введення (input, textarea, date), а також додаткові атрибути для визначення формату введення дати. Для зручності використовується розмітка Bootstrap.

На головній сторінці користувач бачить список своїх задач. Завдання автоматично фільтруються за автором, а також сортуються за датою створення. Передбачено інтерфейс для редагування та видалення задач:

```
@routes.route('/')
def index():
    if current_user.is_authenticated:
        tasks = Task.query.filter_by(user_id=current_user.id).order_by(Task.created_at.desc()).all()
        return render_template('index.html', tasks=tasks)
    else:
        return render_template('index.html')
```

Далі у шаблоні застосовано цикл for для виведення списку задач з бази даних. Для кожної задачі відображаються її атрибути: назва, опис, статус виконання, дати нагадування та завершення, а також кнопки редагування й видалення:

```
{% for task in tasks %}
  <div class="card mb-3">
    <div class="card-body">
      <h5 class="card-title">{{ task.title }}</h5>
      <p class="card-text">{{ task.description }}</p>
      <p class="card-text">
        <small class="text-muted">
          Завершити до: {{ task.due_date.strftime('%Y-%m-%d') if task.due_date else 'Не вказано' }}
        </small>
      </p>
      <form action="{{ url_for('routes.toggle_task', task_id=task.id) }}" method="POST" class="d-
inline">
        <button type="submit" class="btn btn-sm {{ 'btn-success' if task.is_completed else 'btn-outline-
secondary' }}">
          {{ '✓ Завершено' if task.is_completed else 'Позначити як завершене' }}
        </button>
      </form>
```

```
    <a href="{{ url_for('routes.edit_task', task_id=task.id) }}" class="btn btn-sm btn-
warning">Редагувати</a>
    <form action="{{ url_for('routes.delete_task', task_id=task.id) }}" method="POST" class="d-
inline">
      <button type="submit" class="btn btn-sm btn-danger">Видалити</button>
    </form>
   </div>
  </div>
{% endfor %}
```

Крім цього, реалізовано функцію позначення задачі як виконаної або невиконаної. Це здійснюється через окремий маршрут, який оновлює відповідне поле is_completed у базі даних.

Таким чином, реалізовано повний базовий цикл взаємодії користувача із системою: від реєстрації до повного керування особистими задачами. Реалізація кожної функції відповідає сучасним вимогам до зручності, безпеки та доступності.

3.3. Тестування функціональності застосунку

Завершальним етапом розробки веб-застосунку є тестування його функціональних можливостей. Метою тестування є перевірка коректності роботи основних модулів системи, виявлення можливих помилок та підтвердження відповідності розробленого продукту поставленим вимогам.

Оскільки додаток створено у середовищі Flask, для ручного тестування використовувався вбудований локальний сервер розробника. Усі основні маршрути було перевірено через веб-браузер на предмет:

– відображення сторінок (реєстрація, вхід, створення задачі, головна сторінка);

– роботи форм (додавання задачі, редагування, авторизація);

– коректності перенаправлень після дій користувача;

– обробки помилок при неправильному введенні даних.

```
D:\personal_time_organizer\.venv\Scripts\python.exe D:\personal_time_organizer\run.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 158-191-160
127.0.0.1 - - [30/May/2025 07:30:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2025 07:30:25] "GET /logout HTTP/1.1" 302 -
127.0.0.1 - - [30/May/2025 07:30:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2025 07:30:28] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2025 07:32:14] "POST /login HTTP/1.1" 200 -
```

Рисунок 3.1 – Локальний запуск програми

На рисунку 3.1 зображено запуск застосунку на локальному хості, що свідчить про успішну ініціалізацію всіх компонентів системи без помилок.

У процесі тестування було перевірено функціональність реєстрації та авторизації користувачів. Зокрема, система успішно створює нові облікові записи, захищає паролі шляхом хешування, та обмежує доступ до функціоналу для неавторизованих користувачів. Приклад реєстрації зображенно на рисунку 3.2.

Рисунок 3.2 – Вкладка реєстрації

Після заповнення реєстраційної форми і натискання кнопки «Зареєструватись», система створює нового користувача у базі даних і автоматично виконує вхід.

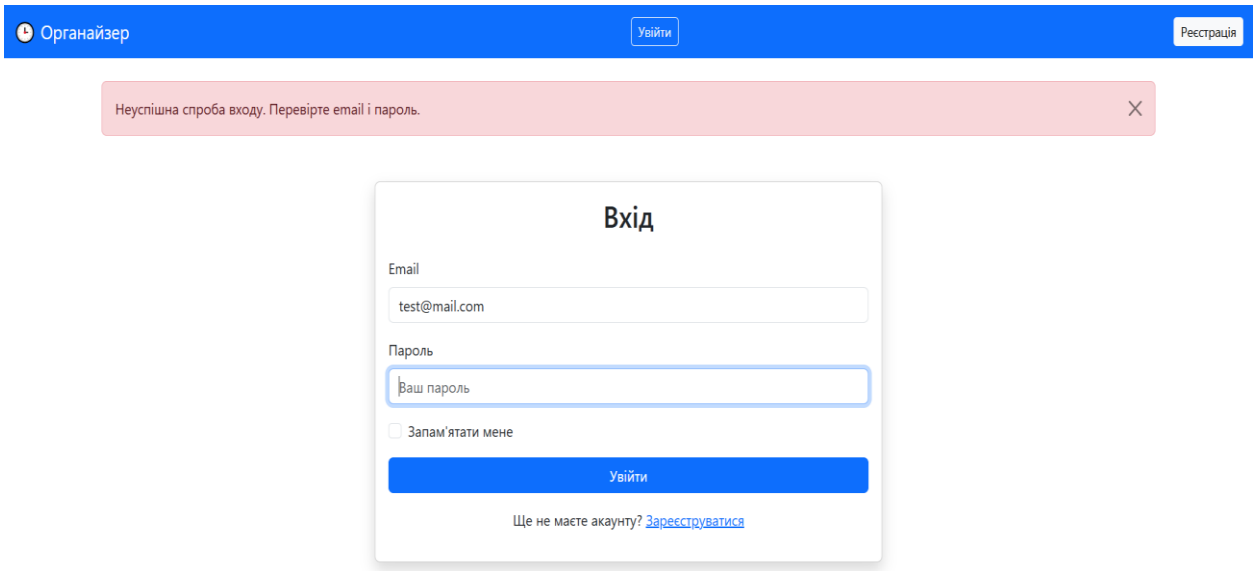

Рисунок 3.3 – Вхід з невірними даними

Цей приклад на рисунку 3.3 демонструє обробку помилок: при введенні невірного логіну чи пароля система не пропускає користувача до кабінету, а відображає повідомлення про невдалу авторизацію.

При введені данних для входу на сторінці реєстрації користувача перекидує на головну сторінку. На ній можливо переглядати усі діючі завдання, а якщо потрібно то видаляти чи редагувати їх як зображено на рисунку 3.4.

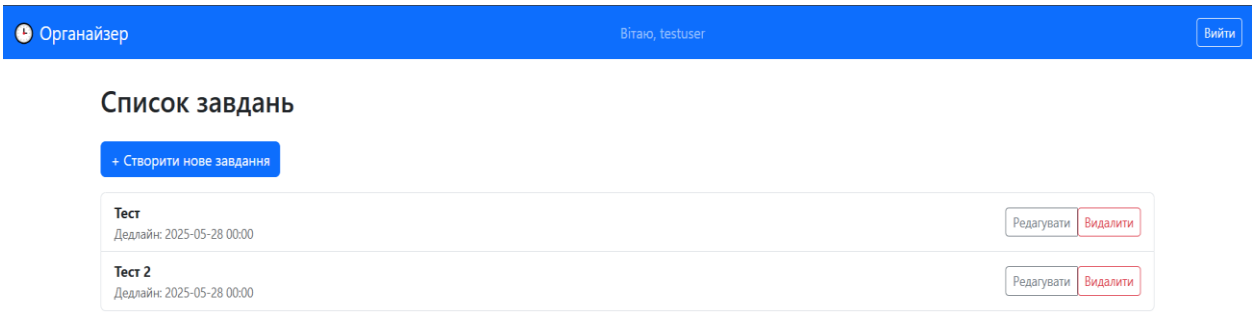

Рисунок 3.4 – Головна сторінка застосунку

Окрему увагу приділено перевірці роботи з задачами. Створення, перегляд, редагування, позначення як виконаних та видалення задач працює стабільно. Дані передаються до бази даних, оновлюються або видаляються відповідно до обраної дії, як зображенно на рисунку 3.5.

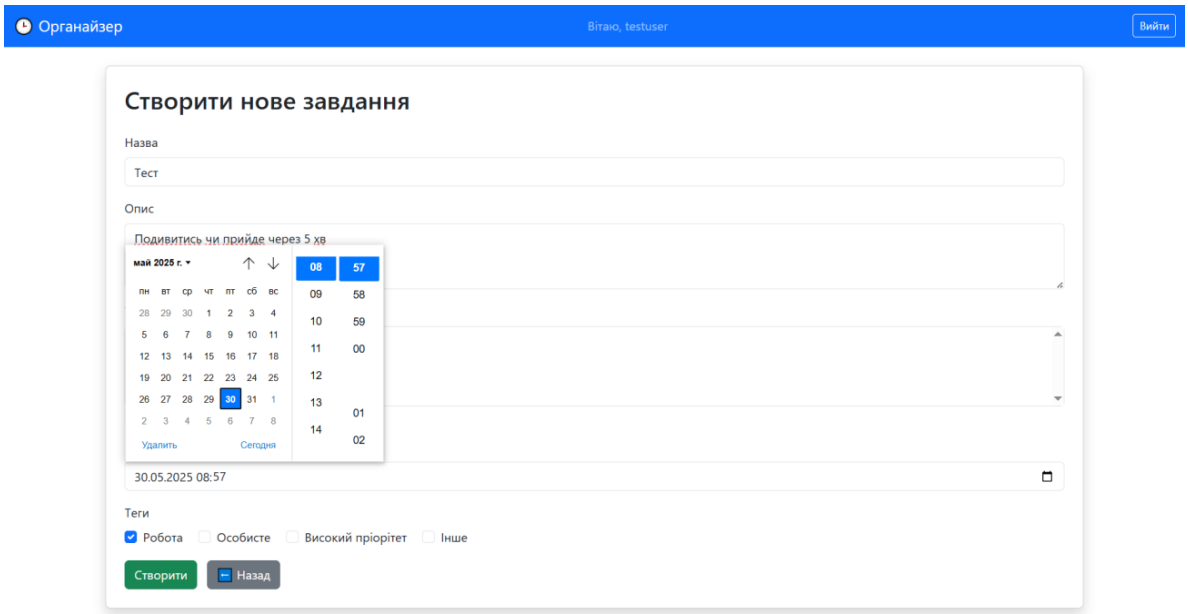

Рисунок 3.5 – Створення задач користувачем

Під час додавання задачі у формі можна задати не лише текстове наповнення, а й дату нагадування, дедлайн та статус виконання. Після збереження всі дані відображаються у загальному списку задач користувача.

Користувач на головній сторінці може використовувати пошук або фільтр по тегам щоб швидше найти необхідну задачу як зображенно на рисунку 3.6.

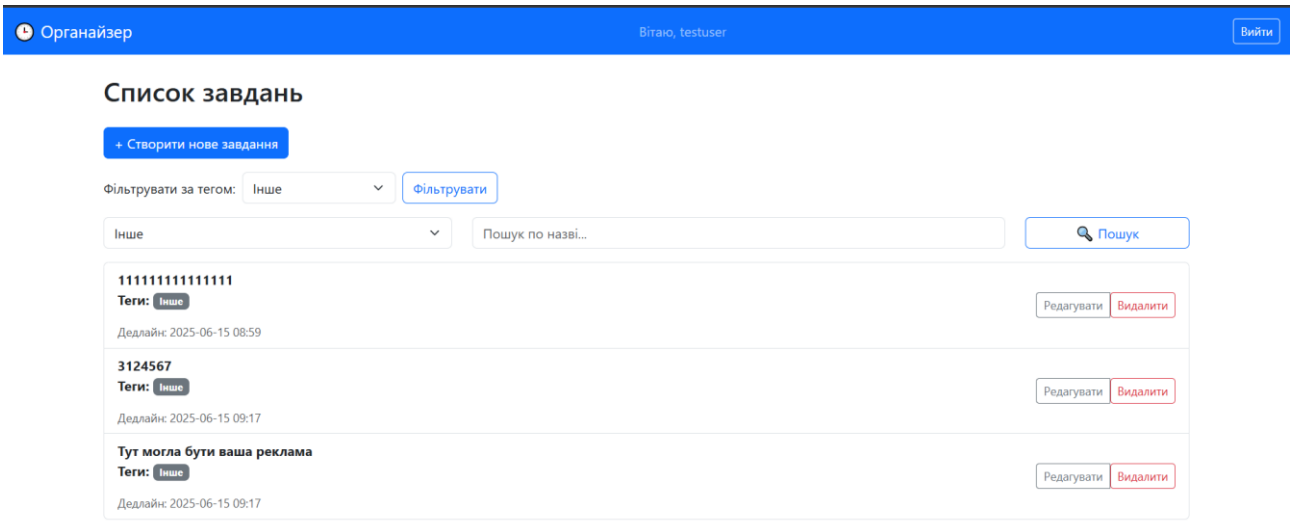

Рисунок 3.6 – Фільтрування задавдань за тегом

Було протестовано функцію нагадувань — система формує листа і надсилає його на email одразу після створення задачі з вказаним полем remind_at, як зображено на рисунку 3.7.

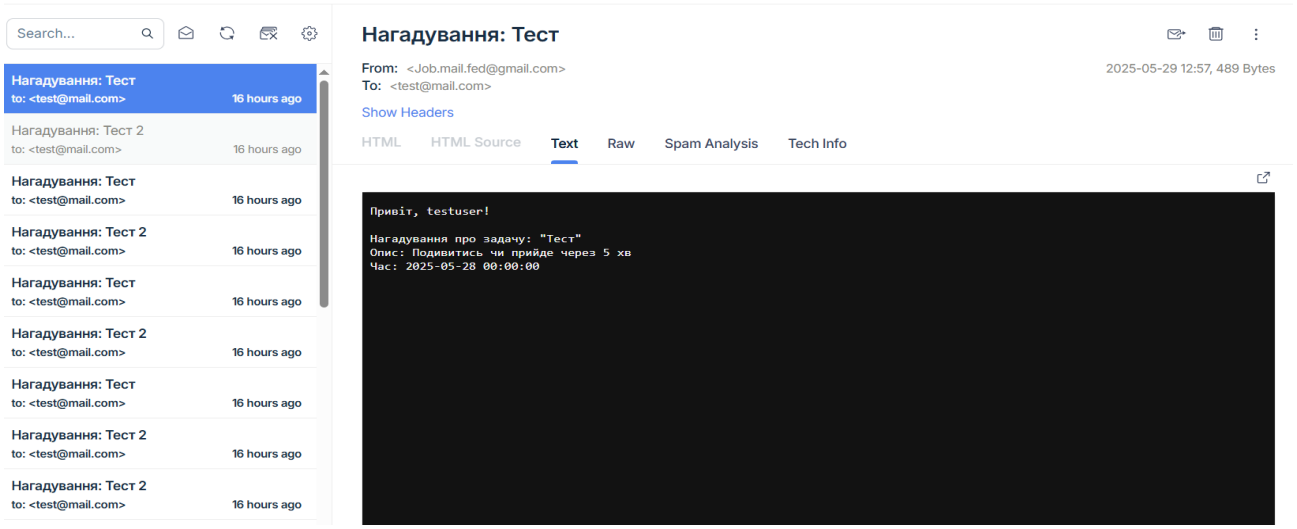

Рисунок 3.7 – Лист нагадування на тестовій почті

Це підтверджує, що механізм надсилання сповіщень через Flask-Mail працює коректно та може бути використаний для реального інформування користувачів.

Загалом, під час тестування WEB-додаток продемонстрував стабільну роботу на рівні базового функціоналу. Критичних помилок не виявлено, усі заплановані дії виконуються згідно з вимогами. У подальшому можлива реалізація автоматизованого тестування через бібліотеки unittest або pytest.

Таким чином, результати тестування підтверджують працездатність основних компонентів системи: автентифікації, роботи з задачами, відправки сповіщень та загальної взаємодії з інтерфейсом. Це дозволяє перейти до фінального оформлення та підготовки застосунку до презентації.

3.4. Висновки до розділу 3

У третьому розділі було детально розглянуто процес безпосередньої реалізації веб-застосунку для організації особистого часу. Було обґрунтовано вибір технологій, зокрема використання Python-фреймворку Flask, бібліотек Flask-Mail і Flask-Login, шаблонізатора Jinja2, ORM-інструмента SQLAlchemy та

фреймворку верстки Bootstrap. Такий стек забезпечив простоту реалізації, гнучкість і можливість подальшого розширення функціоналу.

Розглянуто реалізацію ключових функцій застосунку: реєстрація, авторизація, створення, редагування та видалення задач, відображення списку завдань користувача. У межах функціоналу також було реалізовано систему електронних сповіщень з використанням email-нагадувань через SMTP-сервер.

Проведене тестування підтвердило стабільність роботи основного функціоналу. WEB-додаток коректно обробляє запити, взаємодіє з базою даних, відправляє нагадування та відображає інформацію в інтерфейсі. Скріншоти інтерфейсу та фрагменти коду підтверджують практичну реалізацію задекларованих можливостей.

Таким чином, на основі проектних рішень другого розділу, у третьому розділі було повністю реалізовано базову версію програмного продукту, що виконує функції індивідуального планувальника задач з можливістю розширення у подальших версіях.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було досліджено сучасні цифрові інструменти для організації особистого часу, проаналізовано існуючі веб-застосунки та сервіси з подібною функціональністю, визначено їх переваги й недоліки. На основі отриманих даних було сформульовано вимоги до функціоналу, структури й архітектури власного веб-застосунку, орієнтованого передусім на потреби студентів.

У результаті дослідження, проєктування та реалізації було створено WEB-додаток для організації особистого часу, який поєднує ключові можливості:

- реєстрація та авторизація користувачів;
- створення, редагування, видалення та перегляд задач;
- система email-нагадувань про задачі з вказаним дедлайном;
- виведення задач у зручному інтерфейсі за допомогою Bootstrap;
- побудова структури бази даних із використанням SQLAlchemy.

Для реалізації проєкту було обрано мову програмування Python і фреймворк Flask, що забезпечують гнучкість, простоту розгортання та розширюваність. Для розробки інтерфейсу застосовано Bootstrap 5, а збереження даних реалізовано за допомогою SQLite. У застосунку також інтегровано Flask-Login для управління сесією користувача та Flask-Mail для надсилання сповіщень. Архітектура побудована на основі шаблонів MVC, що забезпечує логічну структуру, модульність та масштабованість.

Мануальне тестування показало коректну роботу усіх основних функцій: система стабільно реагує на дії користувача, коректно обробляє дані форм, виконує перенаправлення, та надсилає нагадування за вказаним часом. Інтерфейс виявився зручним і зрозумілим навіть для користувачів без технічного досвіду.

Усі поставлені на початку дослідження завдання були успішно виконані:

1. Обґрунтовано доцільність створення веб-застосунку для організації особистого часу.

2. Проведено аналіз існуючих рішень і визначено вимоги до системи.

3. Спроєктовано архітектуру та інтерфейс майбутнього застосунку.
4. Реалізовано функціонал задач, авторизації та нагадувань.
5. Проведено тестування та верифікацію функціональності.

Крім того, звіт про виконану роботу було складено відповідно до вимог і методичних вказівок, що гарантує його структурованість, логічність викладення матеріалу та відповідність академічним стандартам [24].

Таким чином, поставлену задачу – розробити WEB-додаток для організації особистого часу – було успішно реалізовано. Створене рішення демонструє практичну цінність і може бути основою для подальшого розвитку: зокрема, впровадження фільтрації задач за категоріями, інтеграції з календарями (Google Calendar API), мобільної адаптації або реалізації push-сповіщень для браузерів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Workspace Statistics. URL: https://surli.cc/gazvcr (дата звернення: 16.05.2025).
2. Amazing Stats and Facts about Todoist. URL: https://surl.lu/spffkm (дата звернення: 16.05.2025).
3. Todoist: Planner & Calendar. URL: https://surl.li/febcjz (дата звернення: 16.05.2025).
4. A Student's Guide to Todoist. URL: https://surli.cc/yhrijo (дата звернення: 16.05.2025).
5. Notion: Notes, Tasks, AI. URL: https://surl.li/emjdxr (дата звернення: 16.05.2025).
6. Notion is that single place. URL: https://surl.li/vvggag (дата звернення: 16.05.2025).
7. About ClickUp. URL: https://surl.li/hybcze (дата звернення: 16.05.2025).
8. Flask Documentation. URL: https://flask.palletsprojects.com/ (дата звернення: 19.05.2025).
9. Bootstrap 5 Documentation. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 18.05.2025).
10. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 19.05.2025).
11. SQLAlchemy ORM Tutorial for Python Developers. URL: https://docs.sqlalchemy.org/en/20/orm/ (дата звернення: 19.05.2025).
12. Jinja2 Templating Engine Documentation. URL: https://jinja.palletsprojects.com/en/3.1.x/ (дата звернення: 18.05.2025).
13. Visual Studio Code – офіційна документація. URL: https://code.visualstudio.com/docs (дата звернення: 20.05.2025).
14. PyCharm Community Edition – огляд можливостей. URL: https://www.jetbrains.com/pycharm/features/ (дата звернення: 21.05.2025).

15. Організація особистого часу: методики, інструменти, практики. URL: https://osvita.ua/school/lessons-summary/90842/ (дата звернення: 15.05.2025).
16. Що таке ORM та як воно працює в Python. URL: https://dou.ua/forums/topic/25283/ (дата звернення: 16.05.2025).
17. Введення в ООП. Основні принципи. URL: https://w3schoolsua.github.io/hyperskill/oop-intro.html (дата звернення: 21.05.2025).
18. UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 22.05.2025).
19. Функціональне програмування в Python. URL: https://foxminded.ua/funktsionalne-prohramuvannia/ (дата звернення: 22.05.2025).
20. Flask-Login Documentation. URL: https://flask-login.readthedocs.io/en/latest/ (дата звернення: 19.05.2025).
21. Flask-Mail – Flask Extension. URL: https://pythonhosted.org/Flask-Mail/ (дата звернення: 19.05.2025).
22. Mailtrap: Safe Email Testing. URL: https://mailtrap.io/ (дата звернення: 20.05.2025).
23. Використання Jinja2 для побудови веб-інтерфейсів. URL: https://habr.com/ru/articles/554832/ (дата звернення: 16.05.2025).
24. Методичні вказівки до виконання курсової роботи з дисципліни «Організація баз даних та знань» / Уклад.: Т. О. Савчук, О. В. Ольшанська. Вінниця : ВНТУ, 2021. 38 с.

ДОДАТКИ

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробки WEB-додатку для організації особистого часу
Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)
Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,37 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Перепелиця В.І.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Федоров А.Ю.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)

Лістинг програми

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from flask_migrate import Migrate
from flask_mail import Mail
from apscheduler.schedulers.background import BackgroundScheduler
from datetime import datetime, timedelta
import atexit
from config import Config

db = SQLAlchemy()
migrate = Migrate()
login_manager = LoginManager()
login_manager.login_view = 'routes.login'
mail = Mail()

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)

    db.init_app(app)
    migrate.init_app(app, db)
    login_manager.init_app(app)
    mail.init_app(app)

    from app.routes import routes as routes_blueprint
    from app.utils import check_and_send_reminders

    # Планувальник
    scheduler = BackgroundScheduler()
    scheduler.add_job(
        func=lambda: check_and_send_reminders(app),
        trigger='interval',
        minutes=10,
        id='reminder_job'
    )
    scheduler.start()
    atexit.register(lambda: scheduler.shutdown())
```

```
    app.register_blueprint(routes_blueprint)

    with app.app_context():
        db.create_all()

    return app
```

```
from flask_wtf import FlaskForm
from wtforms import StringField, TextAreaField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from app.models import User

class RegistrationForm(FlaskForm):
    username = StringField('Ім\'я користувача', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Пароль', validators=[DataRequired()])
    confirm_password = PasswordField('Підтвердити пароль', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Зареєструватися')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('Цей користувач вже існує.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('Цей email вже зареєстровано.')

class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Пароль', validators=[DataRequired()])
    remember = BooleanField('Запам\'ятати мене')
    submit = SubmitField('Увійти')

class TaskForm(FlaskForm):
    title = StringField('Назва задачі', validators=[DataRequired()])
    description = TextAreaField('Опис')
    submit = SubmitField('Створити')
```

```
from app import db
from flask_login import UserMixin
from datetime import datetime

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), nullable=False, unique=True)
    email = db.Column(db.String(150), nullable=False, unique=True)
    password = db.Column(db.String(150), nullable=False)
    tasks = db.relationship('Task', backref='author', lazy=True)

task_tags = db.Table('task_tags',
    db.Column('task_id', db.Integer, db.ForeignKey('task.id')),
    db.Column('tag_id', db.Integer, db.ForeignKey('tag.id'))
)

class Tag(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True, nullable=False)

    def __repr__(self):
        return f'<Tag {self.name}>'


class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(140), nullable=False)
    description = db.Column(db.Text, nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    is_completed = db.Column(db.Boolean, default=False)
    remind_at = db.Column(db.DateTime, nullable=True)
    due_date = db.Column(db.DateTime, nullable=True)
    tags = db.relationship('Tag', secondary='task_tags', backref='tasks')

    def __repr__(self):
        return f'<Task {self.title}>'
```

```
from flask import render_template, redirect, url_for, flash, request, Blueprint
from flask_login import login_user, current_user, logout_user, login_required
from werkzeug.security import generate_password_hash, check_password_hash
```

```
from app import db
from app.forms import RegistrationForm, LoginForm, TaskForm
from app.models import User, Task
from datetime import datetime
from app import login_manager
from app.models import User
from app.models import Tag


routes = Blueprint('routes', __name__)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


@routes.app_context_processor
def inject_user():
    return dict(current_user=current_user)

@routes.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('routes.index'))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password = generate_password_hash(form.password.data)
        user = User(username=form.username.data, email=form.email.data, password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash('Акаунт створено! Тепер ви можете увійти.', 'success')
        return redirect(url_for('routes.login'))
    return render_template('register.html', form=form)

@routes.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('routes.index'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and check_password_hash(user.password, form.password.data):
```

```
            login_user(user, remember=form.remember.data)
            return redirect(url_for('routes.index'))
        else:
            flash('Неуспішна спроба входу. Перевірте email і пароль.', 'danger')
    return render_template('login.html', form=form)

@routes.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('routes.index'))

@routes.route('/')
def index():
    from app.models import Tag

    tags = Tag.query.all()
    selected_tag_id = request.args.get('tag')
    search_query = request.args.get('q', '').strip()
    selected_tag = Tag.query.get(selected_tag_id) if selected_tag_id else None

    if current_user.is_authenticated:
        query = Task.query.filter_by(user_id=current_user.id)

        if selected_tag:
            query = query.filter(Task.tags.contains(selected_tag))
        if search_query:
            query = query.filter(Task.title.ilike(f'%{search_query}%'))

        tasks = query.all()

        return render_template(
            'index.html',
            tasks=tasks,
            tags=tags,
            selected_tag=selected_tag,
            search_query=search_query
        )
    else:
        return render_template(
            'index.html',
            tasks=[],
```

```
            tags=tags,
            selected_tag=None,
            search_query=""
        )

@routes.route('/tasks/<int:task_id>/edit', methods=['GET', 'POST'])
@login_required
def edit_task(task_id):
    task = Task.query.get_or_404(task_id)
    if task.user_id != current_user.id:
        flash('У вас немає прав для редагування цієї задачі.', 'error')
        return redirect(url_for('routes.index'))
    if request.method == 'POST':
        task.title = request.form.get('title')
        task.description = request.form.get('description')
        db.session.commit()
        flash('Задачу оновлено успішно!', 'success')
        return redirect(url_for('routes.index'))
    return render_template('edit_task.html', task=task)


@routes.route('/create-task', methods=['GET', 'POST'])
@login_required
def create_task():
    tags = Tag.query.all()

    if request.method == 'POST':
        title = request.form['title']
        description = request.form['description']
        due_date_str = request.form.get('due_date')
        due_date = datetime.strptime(due_date_str, "%Y-%m-%dT%H:%M") if due_date_str else None
        remind_at_str = request.form.get('remind_at')
        remind_at = datetime.strptime(remind_at_str, '%Y-%m-%dT%H:%M') if remind_at_str else None

        selected_tag_ids = request.form.getlist('tags')
        selected_tags = Tag.query.filter(Tag.id.in_(selected_tag_ids)).all()


        task = Task(
            title=title,
            description=description,
            due_date=due_date,
```

```
            remind_at=remind_at,
            user_id=current_user.id,
            tags = selected_tags
        )
        tag_ids = request.form.getlist('tags')
        task.tags = Tag.query.filter(Tag.id.in_(tag_ids)).all()

        task.tags = [Tag.query.get(int(tag_id)) for tag_id in selected_tag_ids]

        db.session.add(task)
        db.session.commit()
        flash('Завдання створено!', 'success')
        return redirect(url_for('routes.index'))

    return render_template('create_task.html', tags=tags)


@routes.route('/tasks/<int:task_id>/delete', methods=['POST'])
@login_required
def delete_task(task_id):
    task = Task.query.get_or_404(task_id)
    if task.user_id != current_user.id:
        flash('У вас немає прав для видалення цієї задачі.', 'error')
        return redirect(url_for('routes.index'))
    db.session.delete(task)
    db.session.commit()
    flash('Задачу видалено успішно!', 'success')
    return redirect(url_for('routes.index'))

@routes.route('/tasks/<int:task_id>/toggle', methods=['POST'])
@login_required
def toggle_task(task_id):
    task = Task.query.get_or_404(task_id)
    if task.user_id != current_user.id:
        flash('У вас немає прав змінювати цю задачу.', 'error')
        return redirect(url_for('routes.index'))
    task.is_completed = not task.is_completed
    db.session.commit()
    flash('Статус задачі оновлено.', 'success')
    return redirect(url_for('routes.index'))
```

```
@routes.route('/send-test-email')
@login_required
def send_test_email():
    from app.utils import send_reminder_email
    send_reminder_email(
        to=current_user.email,
        subject="Тестове нагадування",
        body="Це тестове нагадування з Flask!"
    )
    return "Нагадування відправлено!"
```

```
from app import db, mail
from app.models import Task, User
from flask_mail import Message
from datetime import datetime, timedelta

def send_reminder_email(to, subject, body):
    msg = Message(subject, recipients=[to], body=body)
    mail.send(msg)

def check_and_send_reminders(app):
    with app.app_context():
        now = datetime.utcnow()
        upcoming = now + timedelta(minutes=10)

        tasks = Task.query.filter(
            Task.due_date != None,
            Task.due_date <= upcoming,
            Task.is_completed == False
        ).all()

        for task in tasks:
            user = User.query.get(task.user_id)
            if user:
                subject = f"Нагадування: {task.title}"
                body = f"Привіт, {user.username}!\n\nНагадування про задачу: \"{task.title}\"\nОпис: {task.description
or 'Без опису'}\nЧас: {task.due_date}"
                send_reminder_email(user.email, subject, body)
```

```
<!DOCTYPE html>
<html lang="uk">
<head>
```

```
    <meta charset="UTF-8">
    <title>{% block title %}Органайзер часу{% endblock %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary mb-4">
        <div class="container-fluid">
            <a class="navbar-brand" href="{{ url_for('routes.index') }}">🕒 Органайзер</a>
            {% if current_user.is_authenticated %}
                <span class="navbar-text me-3">Вітаю, {{ current_user.username }}</span>
                <a href="{{ url_for('routes.logout') }}" class="btn btn-outline-light btn-sm">Вийти</a>
            {% else %}
                <a href="{{ url_for('routes.login') }}" class="btn btn-outline-light btn-sm me-2">Увійти</a>
                <a href="{{ url_for('routes.register') }}" class="btn btn-light btn-sm">Реєстрація</a>
            {% endif %}
        </div>
    </nav>
    <div class="container">
        {% with messages = get_flashed_messages(with_categories=true) %}
          {% if messages %}
            {% for category, message in messages %}
              <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
                {{ message }}
                <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
              </div>
            {% endfor %}
          {% endif %}
        {% endwith %}
        {% block content %}{% endblock %}
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

```
{% extends "base.html" %}

{% block title %}Створити нове завдання{% endblock %}

{% block content %}
<div class="container mt-4">
  <div class="card shadow rounded p-4">
```

```
  <h2 class="mb-4">Створити нове завдання</h2>
  <form method="POST">
   <div class="mb-3">
    <label for="title" class="form-label">Назва</label>
    <input type="text" class="form-control" id="title" name="title" required>
   </div>
   <div class="mb-3">
    <label for="description" class="form-label">Опис</label>
    <textarea class="form-control" id="description" name="description" rows="3"></textarea>
   </div>
   <div class="mb-3">
    <label for="tags" class="form-label">Теги</label>
    <select class="form-select" id="tags" name="tags" multiple>
     {% for tag in tags %}
      <option value="{{ tag.id }}">{{ tag.name }}</option>
     {% endfor %}
    </select>
    <small class="form-text text-muted">Утримуйте Ctrl (Cmd) для вибору кількох тегів.</small>
   </div>
   <div class="mb-3">
    <label for="due_date" class="form-label">Крайній термін</label>
    <input type="datetime-local" class="form-control" id="due_date" name="due_date">
   </div>
   <div class="mb-3">
    <label class="form-label">Теги</label><br>
    {% for tag in tags %}
     <div class="form-check form-check-inline">
      <input class="form-check-input" type="checkbox" name="tags" value="{{ tag.id }}" id="tag_{{ tag.id }}">
      <label class="form-check-label" for="tag_{{ tag.id }}">{{ tag.name }}</label>
     </div>
    {% endfor %}
   </div>
   <button type="submit" class="btn btn-success">Створити</button>
   <a href="{{ url_for('routes.index') }}" class="btn btn-secondary ms-2">⬅ Назад</a>
  </form>
 </div>
</div>
{% endblock %}
```

```
{% extends "base.html" %}
```

```
{% block title %}Редагувати завдання{% endblock %}

{% block content %}
<div class="container mt-4">
  <div class="card shadow rounded p-4">
    <h2 class="mb-4">Редагувати завдання</h2>
    <form method="POST">
      <div class="mb-3">
        <label for="title" class="form-label">Назва</label>
        <input type="text" class="form-control" id="title" name="title" value="{{ task.title }}" required>
      </div>
      <div class="mb-3">
        <label for="description" class="form-label">Опис</label>
        <textarea class="form-control" id="description" name="description" rows="3">{{ task.description
}}</textarea>
      </div>
      <div class="mb-3">
        <label for="due_date" class="form-label">Крайній термін</label>
        <input type="datetime-local" class="form-control" id="due_date" name="due_date" value="{{
task.due_date.strftime('%Y-%m-%dT%H:%M') if task.due_date }}">
      </div>
      <button type="submit" class="btn btn-primary">Зберегти</button>
      <a href="{{ url_for('routes.index') }}" class="btn btn-secondary ms-2">⬅ Назад</a>
    </form>
  </div>
</div>
{% endblock %}
```

```
{% extends 'base.html' %}
{% block title %}Мої завдання{% endblock %}

{% block content %}
<h2 class="mb-4">Список завдань</h2>

<div class="mb-3">
    <a href="{{ url_for('routes.create_task') }}" class="btn btn-primary">+ Створити нове завдання</a>
</div>
<form method="get" class="mb-3">
  <div class="row g-2 align-items-center">
    <div class="col-auto">
      <label for="tag" class="form-label mb-0">Фільтрувати за тегом:</label>
```

```
    </div>
    <div class="col-auto">
      <select name="tag" id="tag" class="form-select">
        <option value="">Усі</option>
        {% for tag in tags %}
          <option value="{{ tag.id }}" {% if selected_tag and tag.id == selected_tag.id %}selected{% endif %}>
            {{ tag.name }}
          </option>
        {% endfor %}
      </select>
    </div>
    <div class="col-auto">
      <button type="submit" class="btn btn-outline-primary">Фільтрувати</button>
    </div>
  </div>
</form>
<form method="GET" class="row mb-3">
    <div class="col-md-4">
        <select name="tag" class="form-select" onchange="this.form.submit()">
            <option value="">-- Всі теги --</option>
            {% for tag in tags %}
                <option value="{{ tag.id }}" {% if selected_tag and selected_tag.id == tag.id %}selected{% endif %}>{{
tag.name }}</option>
            {% endfor %}
        </select>
    </div>
    <div class="col-md-6">
        <input type="text" name="q" class="form-control" placeholder="Пошук по назві..." value="{{ search_query
}}">
    </div>
    <div class="col-md-2">
        <button type="submit" class="btn btn-outline-primary w-100">🔍 Пошук</button>
    </div>
</form>
{% if tasks %}
    <ul class="list-group">
        {% for task in tasks %}
            <li class="list-group-item d-flex justify-content-between align-items-center">
                <div>
                    <strong>{{ task.title }}</strong>
                    <p>
                      <strong>Теги:</strong>
```

```
                {% for tag in task.tags %}
                  <span class="badge bg-secondary">{{ tag.name }}</span>
                {% endfor %}
              </p>
              {% if task.due_date %}
                <span class="text-muted small d-block">Дедлайн: {{ task.due_date.strftime('%Y-%m-%d
%H:%M') }}</span>
              {% endif %}
              {% if task.is_completed %}
                <span class="badge bg-success">Завершено</span>
              {% endif %}
            </div>
            <div class="btn-group">
              <a href="{{ url_for('routes.edit_task', task_id=task.id) }}" class="btn btn-sm btn-outline-
secondary">Редагувати</a>
              <form method="post" action="{{ url_for('routes.delete_task', task_id=task.id) }}"
style="display:inline;">
                <button type="submit" class="btn btn-sm btn-outline-danger" onclick="return confirm('Видалити
задачу?')">Видалити</button>
              </form>
            </div>
          </li>
        {% endfor %}
      </ul>
{% else %}
  <p class="text-muted">Завдань ще немає.</p>
{% endif %}
{% endblock %}
```

```
{% extends "base.html" %}

{% block title %}Вхід{% endblock %}

{% block content %}
<div class="container mt-5">
 <div class="row justify-content-center">
  <div class="col-md-6">
   <div class="card shadow rounded">
    <div class="card-body">
     <h2 class="mb-4 text-center">Вхід</h2>
     <form method="POST">
      {{ form.hidden_tag() }}
```

```
          <div class="mb-3">
            {{ form.email.label(class="form-label") }}
            {{ form.email(class="form-control", placeholder="Ваш email") }}
            {% if form.email.errors %}
              <div class="text-danger small">{{ form.email.errors[0] }}</div>
            {% endif %}
          </div>
          <div class="mb-3">
            {{ form.password.label(class="form-label") }}
            {{ form.password(class="form-control", placeholder="Ваш пароль") }}
            {% if form.password.errors %}
              <div class="text-danger small">{{ form.password.errors[0] }}</div>
            {% endif %}
          </div>
          <div class="form-check mb-3">
            {{ form.remember(class="form-check-input") }}
            {{ form.remember.label(class="form-check-label") }}
          </div>
          <div class="d-grid">
            <button type="submit" class="btn btn-primary">Увійти</button>
          </div>
        </form>
        <div class="mt-3 text-center">
          <p>Ще не маєте акаунту? <a href="{{ url_for('routes.register') }}">Зареєструватися</a></p>
        </div>
      </div>
    </div>
  </div>
</div>
{% endblock %}
```

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Реєстрація</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
<div class="container">
  <h2 class="mt-5">Реєстрація</h2>
```

```
    <form method="POST">
        {{ form.hidden_tag() }}
        <div class="form-group">
            {{ form.username.label }} {{ form.username(class="form-control") }}
        </div>
        <div class="form-group">
            {{ form.email.label }} {{ form.email(class="form-control") }}
        </div>
        <div class="form-group">
            {{ form.password.label }} {{ form.password(class="form-control") }}
        </div>
        <div class="form-group">
            {{ form.confirm_password.label }} {{ form.confirm_password(class="form-control") }}
        </div>
        {{ form.submit(class="btn btn-primary") }}
    </form>
</div>
</body>
</html>
```

```
<h1>Мої задачі</h1>
<a href="{{ url_for('routes.create_task') }}">➕ Створити нову задачу</a><br><br>
<ul>
<h2>Ваші задачі:</h2>
{% for task in tasks %}
    <div style="border: 1px solid #ccc; margin: 10px; padding: 10px;">
        <strong>{{ task.title }}</strong><br>
        <p>{{ task.description }}</p>
        <a href="{{ url_for('routes.edit_task', task_id=task.id) }}">Редагувати</a>
    </div>
    <form action="{{ url_for('routes.delete_task', task_id=task.id) }}" method="POST" style="display:inline;">
    <button type="submit" onclick="return confirm('Ви впевнені, що хочете видалити цю задачу?');">
        Видалити
    </button>
</form>
<form action="{{ url_for('routes.toggle_task', task_id=task.id) }}" method="POST" style="display:inline;">
    <button type="submit">
        {% if task.is_completed %}
            Позначити як невиконану
        {% else %}
            Позначити як виконану
```

```
        {% endif %}
    </button>
</form>
<li {% if task.is_completed %} style="text-decoration: line-through; color: gray;" {% endif %}>
    {{ task.title }} - {{ task.description }}
</li>
{% else %}
    <p>У вас ще немає задач.</p>
{% endfor %}
</ul>
```

```
class Config:
    SECRET_KEY = 'your_secret_key'
    SQLALCHEMY_DATABASE_URI = 'sqlite:///site.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    MAIL_SERVER = 'sandbox.smtp.mailtrap.io'
    MAIL_PORT = 2525
    MAIL_USE_TLS = True
    MAIL_USE_SSL = False
    MAIL_USERNAME = ''
    MAIL_PASSWORD = ''
    MAIL_DEFAULT_SENDER = ''
```

```
from app import app, db

with app.app_context():
    db.create_all()
    print("Створено")
```

```
from app import create_app, db
from app.models import Task, User
from app.utils import send_reminder_email
from datetime import datetime, timedelta

app = create_app()

with app.app_context():
    now = datetime.utcnow()
    upcoming = now + timedelta(minutes=10)
```

```
    tasks = Task.query.filter(
        Task.due_date != None,
        Task.due_date >= now,
        Task.due_date <= upcoming,
        Task.is_completed == False
    ).all()

    for task in tasks:
        user = User.query.get(task.user_id)
        if user:
            send_reminder_email(
                to=user.email,
                subject="Нагадування про задачу",
                body=f"Привіт, {user.username}! Не забудьте про задачу: {task.title}\n\n{task.description}"
            )
            print(f"Нагадування надіслано: {task.title} -> {user.email}")
```

```
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True)
```

ДОДАТОК В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«РОЗРОБКИ WEB-ДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ ОСОБИСТОГО ЧАСУ»

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Федоров А. Ю.
(прізвище та ініціали)
Керівник: доктор філософії, асистент кафедри КН
Перепелиця В. І.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

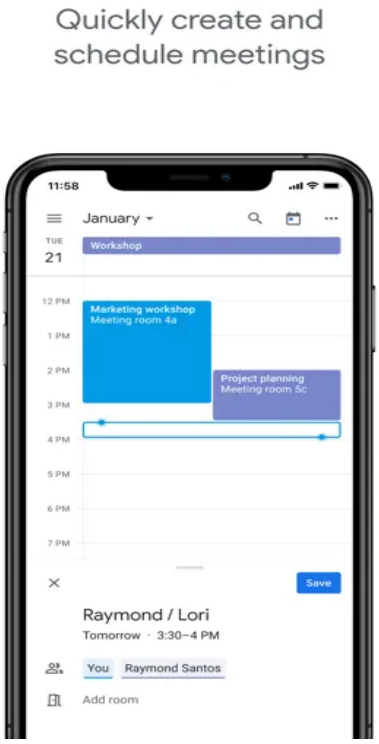

Рисунок В.1 – Веб-додаток Google Calendar

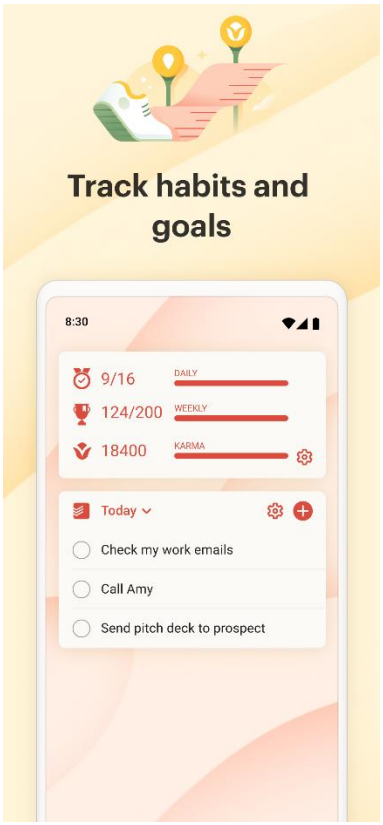

Рисунок В.2 – Веб-додаток Todoist

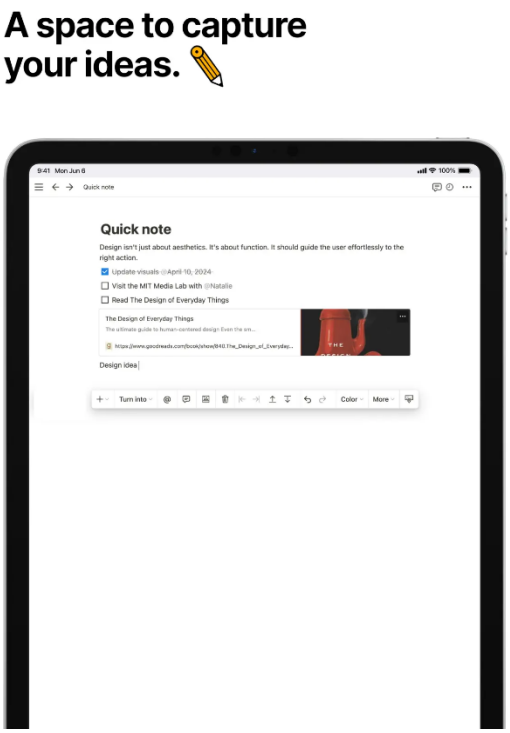

Рисунок В.3 – Веб-додаток Notion

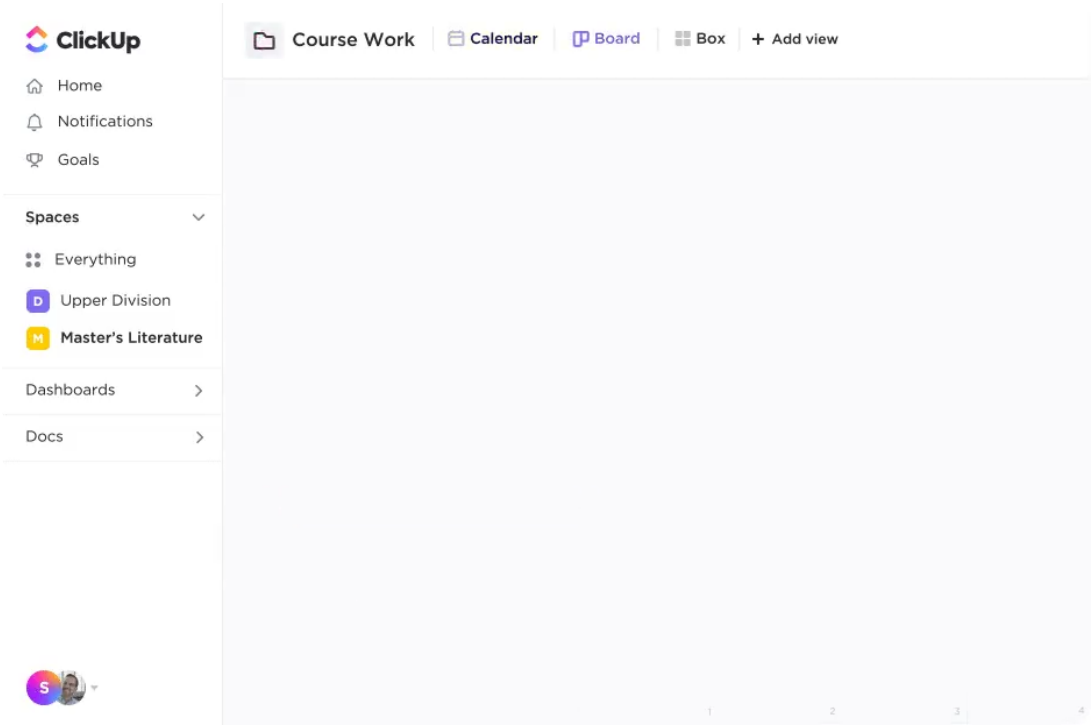

Рисунок В.4 – Веб-додаток ClickUp

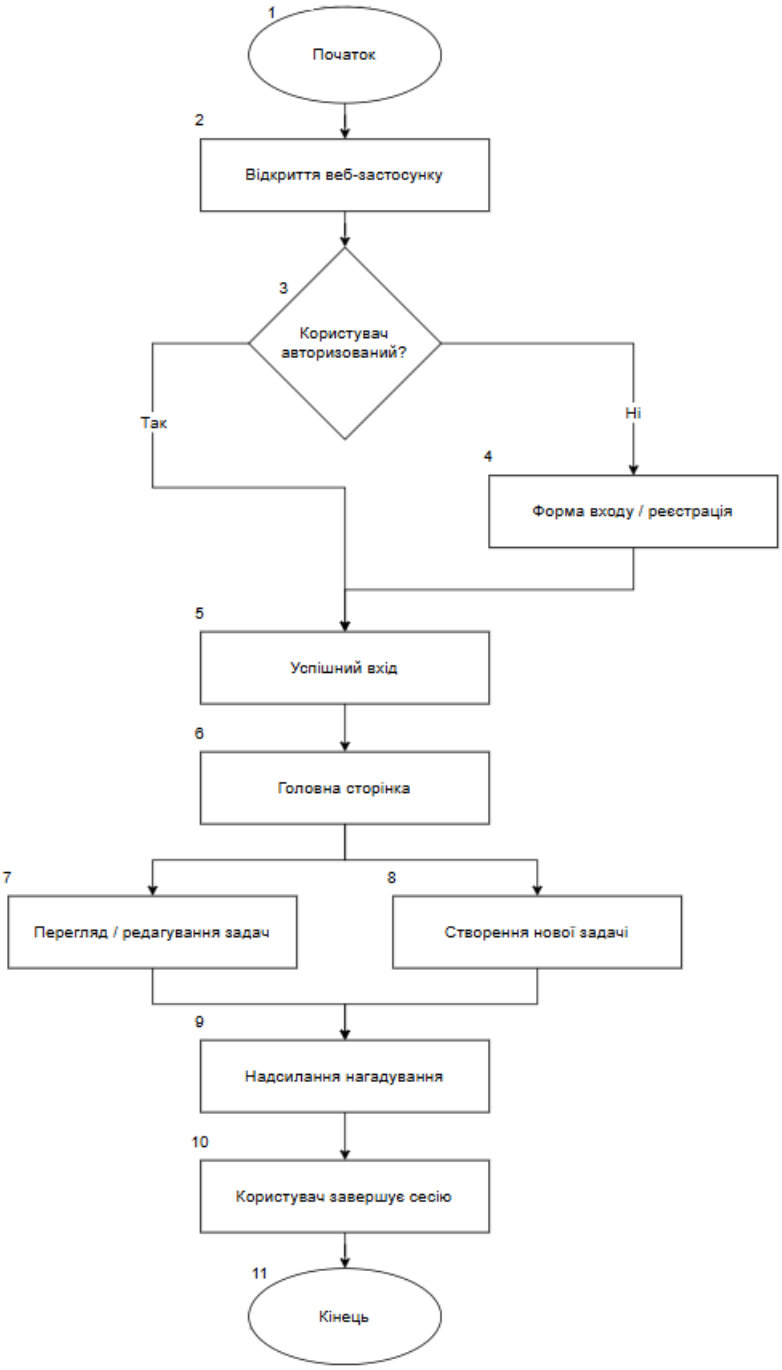

Рисунок В.5 – Типова схема взаємодії користувача із системою

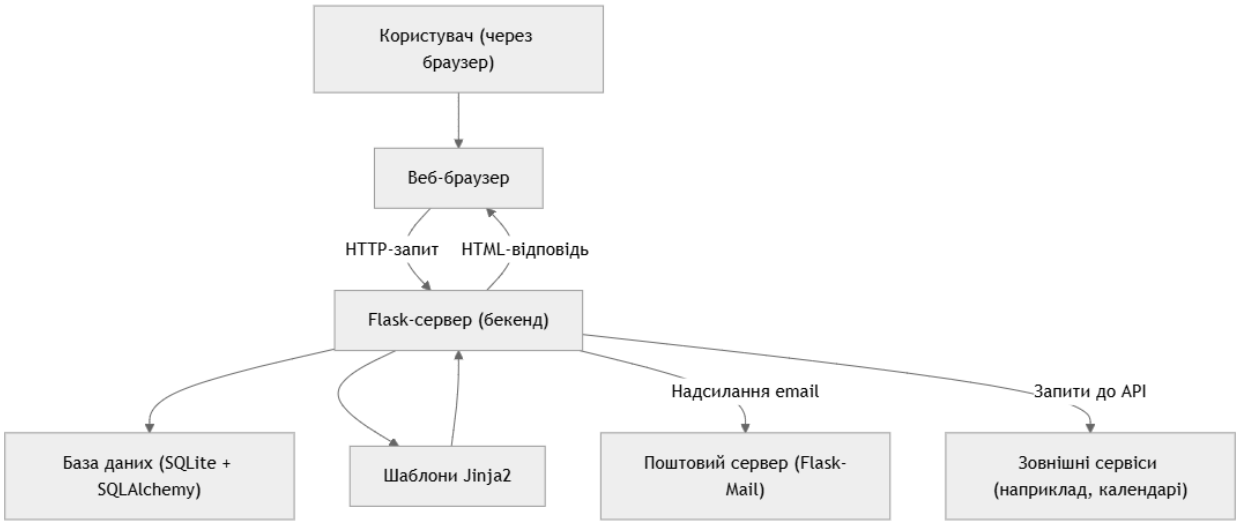

Рисунок В.6 – Схема взаємодії основних компонентів веб-застосунку

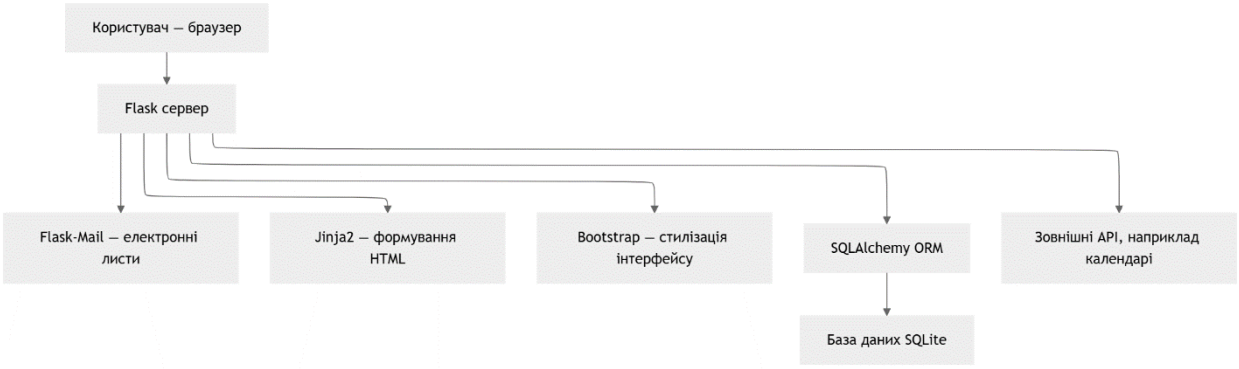

Рисунок В.7 – Архітектура веб-застосунку

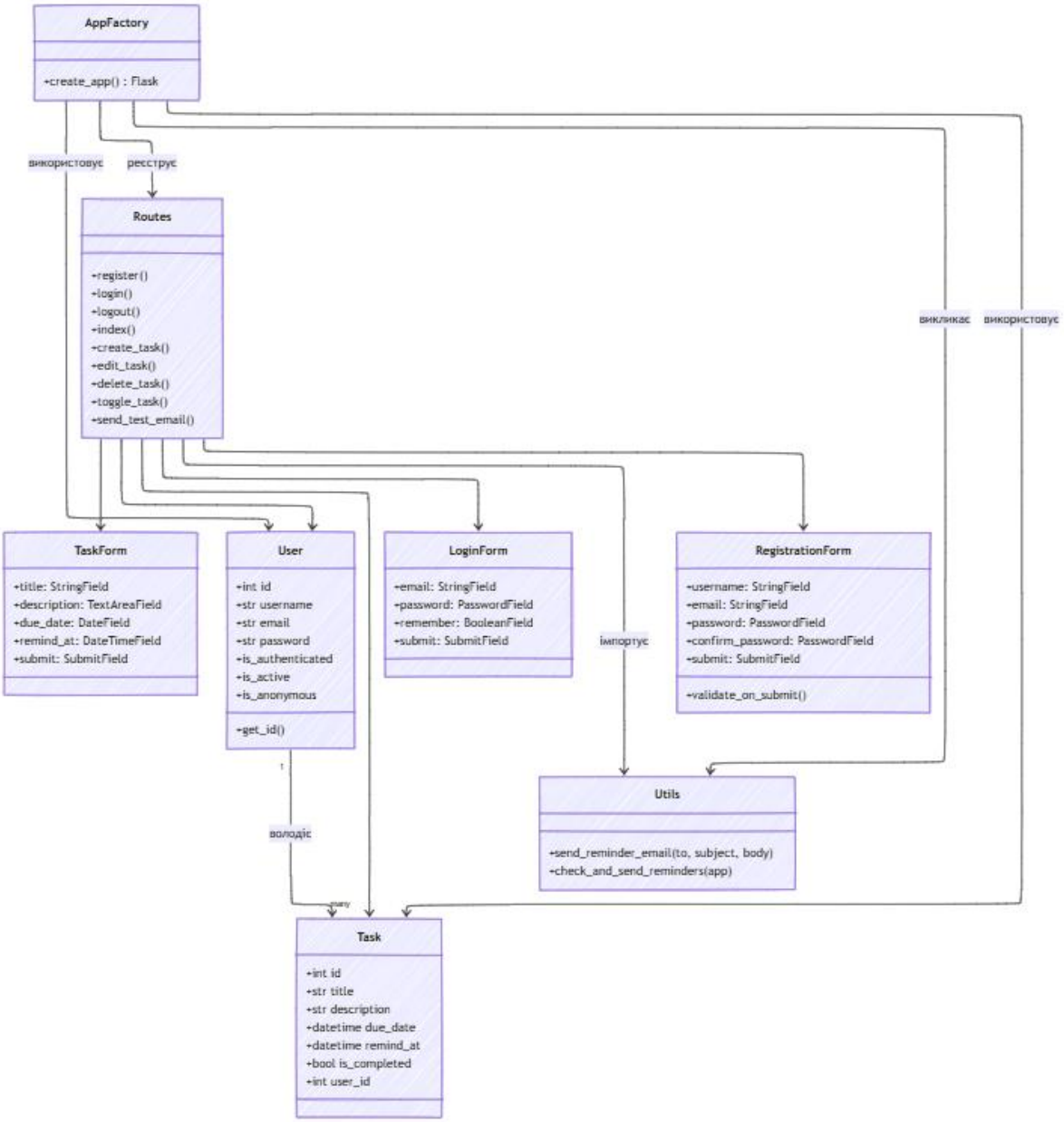

Рисунок В.8 – UML-діаграма класів застосунку

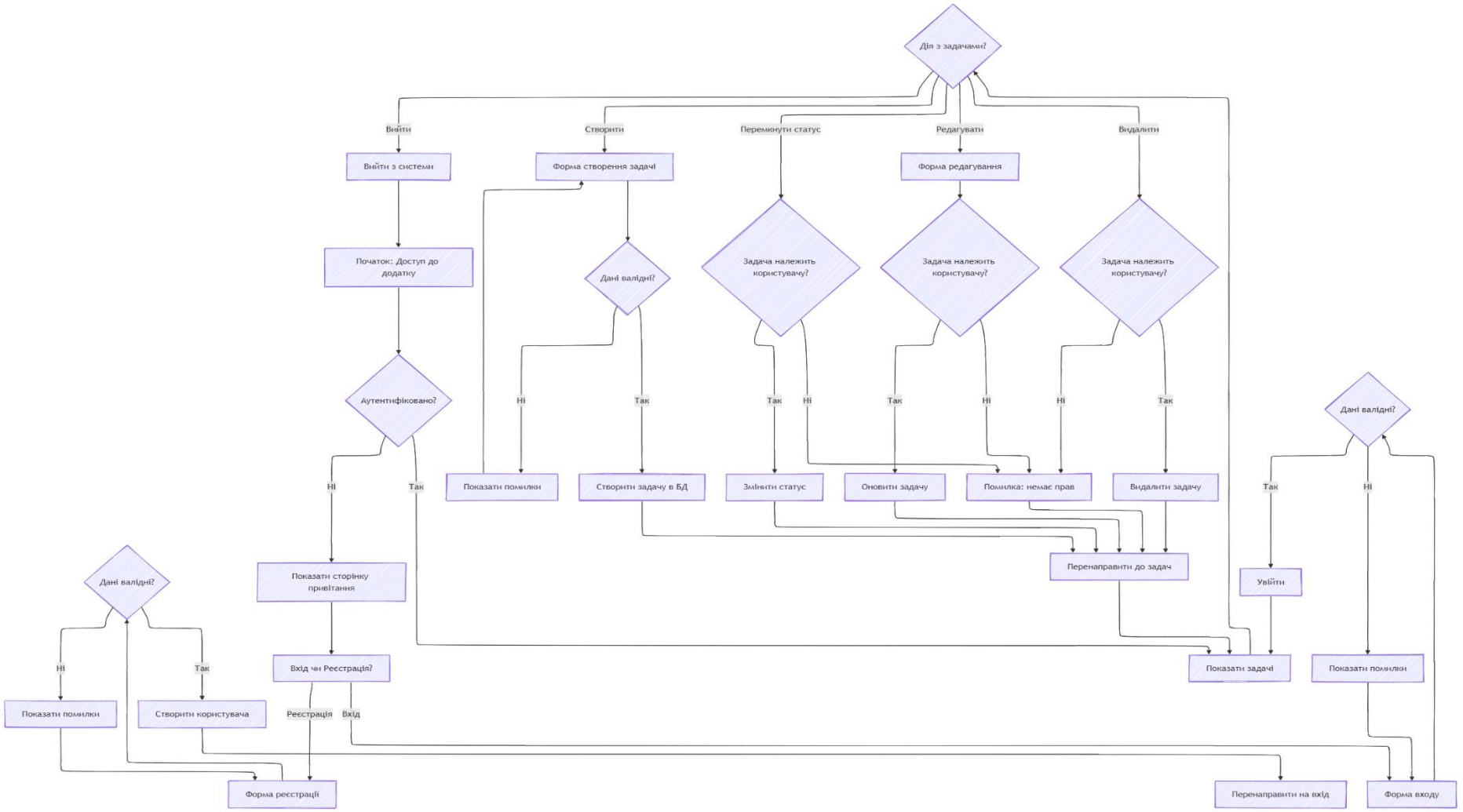

Рисунок В.9 – Блок-схема алгоритму

```
D:\personal_time_organizer\.venv\Scripts\python.exe D:\personal_time_organizer\run.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 158-191-160
127.0.0.1 - - [30/May/2025 07:30:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2025 07:30:25] "GET /logout HTTP/1.1" 302 -
127.0.0.1 - - [30/May/2025 07:30:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2025 07:30:28] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2025 07:32:14] "POST /login HTTP/1.1" 200 -
```

Рисунок В.10 – Локальний запуск програми

Рисунок В.11 – Вкладка реєстрації

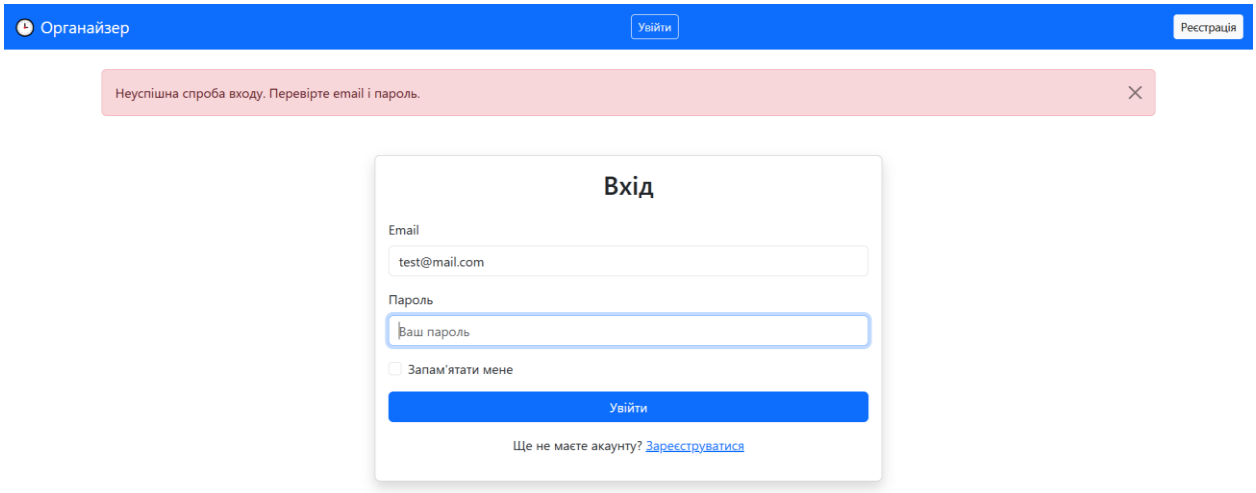

Рисунок В.12 – Вхід з невірними даними

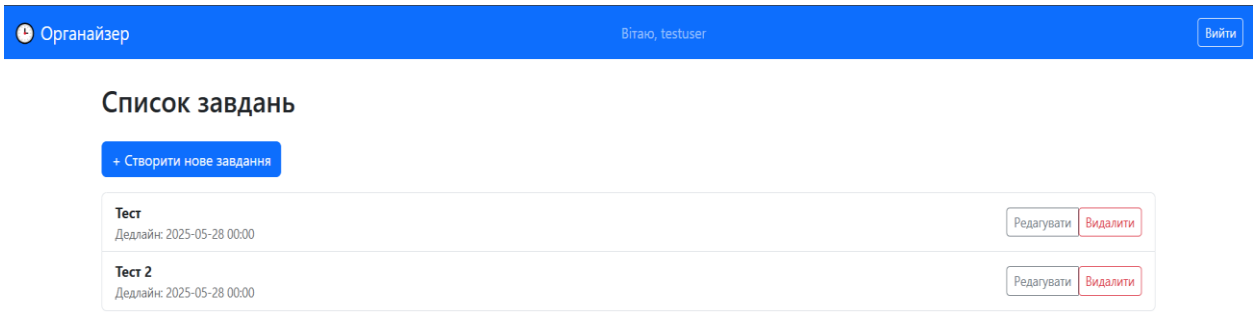

Рисунок В.13 – Головна сторінка застосунку

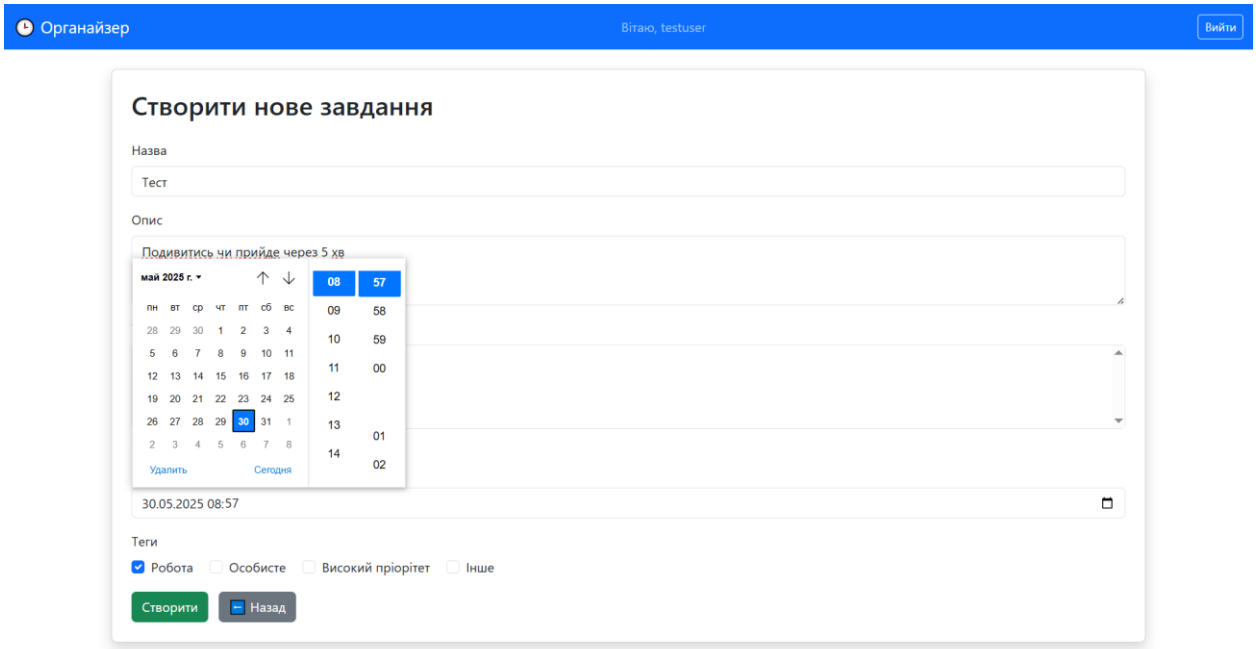

Рисунок В.14 – Створення задач користувачем

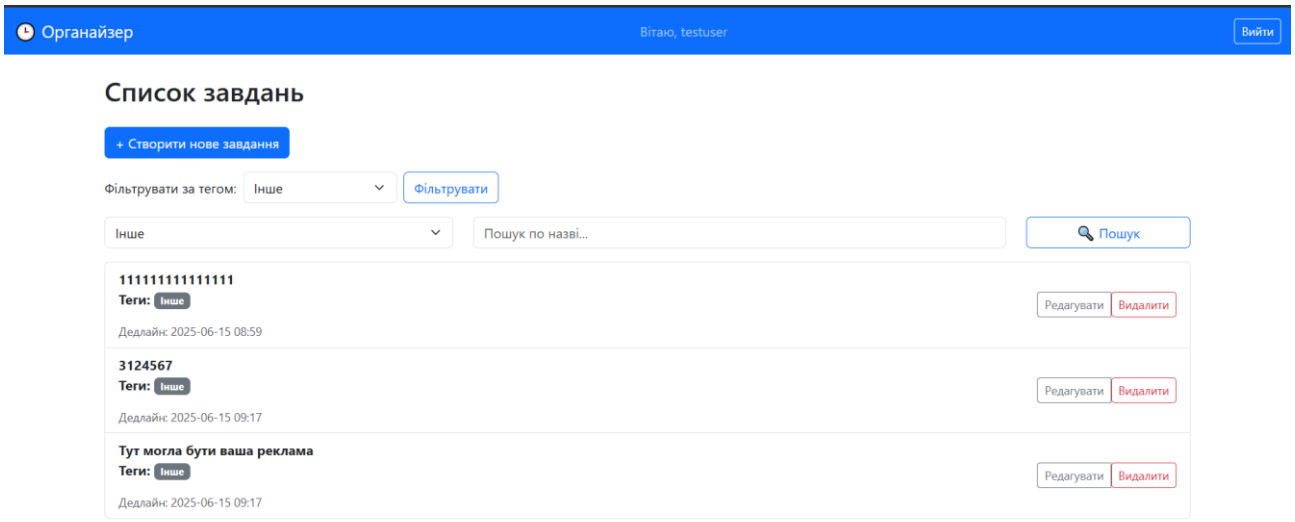

Рисунок В.15 – Фільтрування задавдань за тегом

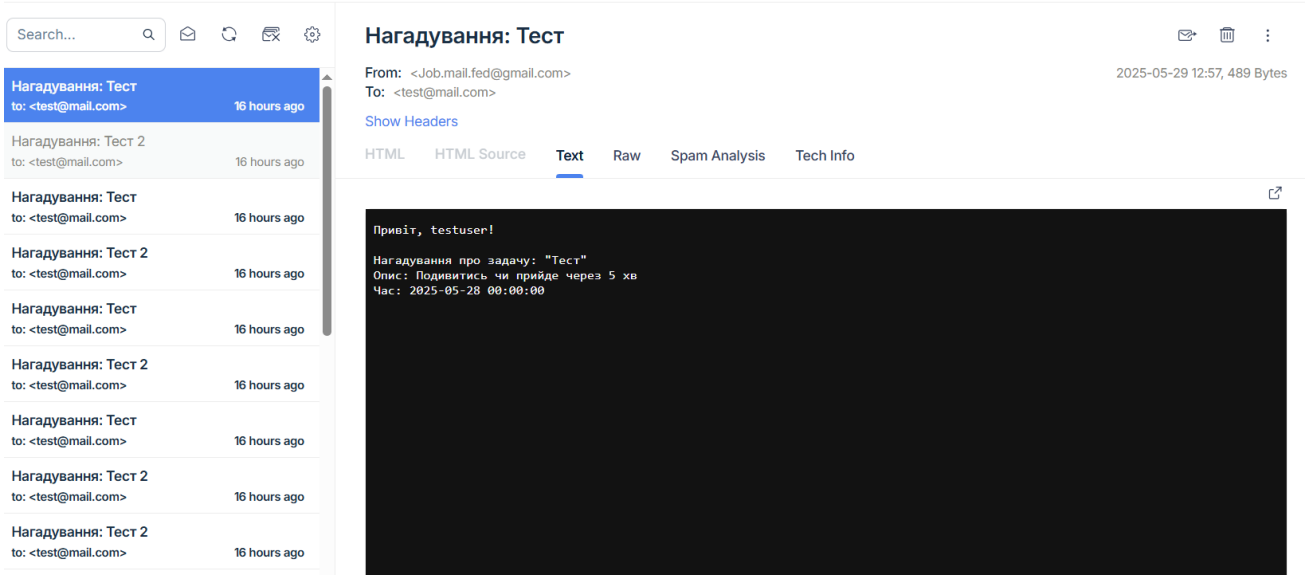

Рисунок В.16 – Лист нагадування на тестовій почті

ДОДАТОК Г (довідниковий)

Інструкція користувача

Загальна інформація

Після запуску веб-застосунку користувачу відкривається головна сторінка з навігаційним меню. Доступні основні розділи:

- Головна (список задач)
- Створити задачу
- Вхід / Реєстрація
- Вихід (після входу)

Доступ до функціональних можливостей

Щоб отримати доступ до функцій додатку, користувачу необхідно авторизуватися або створити новий акаунт. Після входу користувач отримує доступ до особистого списку задач.

Реєстрація нового користувача

1. Перейти на сторінку «Реєстрація».
2. Вказати ім’я користувача, електронну пошту та пароль.
3. Натиснути кнопку «Зареєструватися».
4. Після створення акаунту користувач може увійти в систему.

Вхід до акаунту

1. Перейти на сторінку «Вхід».
2. Ввести зареєстровану електронну пошту та пароль.
3. Натиснути кнопку «Увійти».
4. У випадку правильних даних, користувач потрапляє на головну сторінку з власними задачами.

Створення задачі

1. Перейти в розділ «Створити задачу».
2. Вказати назву, опис (необов’язково), дату дедлайну та час нагадування.
3. Вибрати один або кілька тегів (наприклад, Робота, Особисте тощо).
4. Натиснути кнопку «Створити». Задача з’явиться у списку на головній сторінці.

Перегляд і редагування задач

На головній сторінці користувач бачить список усіх створених задач, які можна:

- Переглянути
- Відредагувати (кнопка «Редагувати»)
- Відмітити як виконану / не виконану
- Видалити

Фільтрація задач за тегами

У верхній частині головної сторінки доступна панель фільтрації. Користувач може:

- Обрати тег (наприклад, Особисте) — і система покаже лише задачі з цим тегом.
- Очистити фільтр — щоб повернутися до повного списку.

Нагадування про задачі

Система автоматично перевіряє задачі, для яких встановлений час нагадування. Якщо задача не завершена і час нагадування близький — на вказану електронну пошту надсилається повідомлення.

Вихід з акаунту

Користувач може натиснути на кнопку «Вихід» у навігаційному меню, щоб завершити сесію.