Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

# Бакалаврська кваліфікаційна робота

на тему: «Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React»

Виконала: студентка 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Луп'як М.Д.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Черноволик Г.О.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Тарновський М.Г.
(прізвище та ініціали)

**Допущено до захисту**
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

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

Луп'як Марії Дмитрівні

1. Тема роботи – «Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React»

Керівник роботи: Черноволик Г.О., к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; проектування програмного засобу; розробка вебзастосунку; тестування програмного забезпечення; висновки; список використаних джерел; додатки.

3. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

4. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада Консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1-4 | Черноволик Г.О., к.т.н., доцент кафедри ПЗ | 24.03.25 | 01.06.25 |

7. Дата видачі завдання 24 березня 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз та постановка задачі | 25.03.25 - 05.04.25 | вик. |
| 2 | Проектування архітектури та розробка алгоритмів програми | 06.04.25 - 18.04.25 | вик. |
| 3 | Обґрунтування вибору засобів для реалізації системи | 19.04.25 - 21.04.25 | вик. |
| 4 | Розробка вебзастосунку | 22.04.25 - 17.05.25 | вик. |
| 5 | Тестування програмного забезпечення | 18.05.25 - 25.05.25 | вик. |
| 6 | Оформлення матеріалів до захисту БКР | 26.05.25 - 30.05.25 | вик. |

Студентка _____ Луп'як М.Д.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Черноволик Г.О.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

УДК 004:005.9

Луп’як М. Д. Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React : бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 95 с.

У бакалаврській кваліфікаційній роботі було розроблено вебзастосунок «CheckIt» для створення, організації та ведення особистих чеклістів. Система дозволяє користувачам створювати завдання вручну або автоматично за допомогою інтегрованого модуля штучного інтелекту, керувати списками, архівувати завершені записи, відстежувати прогрес та адаптувати інтерфейс під особисті вподобання. Реалізовано зручну багатосторінкову навігацію, автентифікацію через електронну пошту або Google, можливість зміни тем оформлення, а також перегляд завдань у вигляді календаря.

Було проведено аналіз інформаційного забезпечення системи, побудовано архітектуру вебзастосунку з поділом на клієнтську та серверну частини. Розроблено клієнтську та серверну частини програми. Також було проведено тестування, що підтвердило коректну роботу вебзастосунку.

Програмне забезпечення створено із використанням мови програмування C# для серверної частини (ASP.NET Core) та JavaScript з бібліотекою React для клієнтської. Також використовувались інструменти TypeScript, Tailwind CSS, Entity Framework Core, PostgreSQL, Google OAuth та система контролю версій Git.

Ключові слова: чекліст, React, ASP.NET Core, вебзастосунок, штучний інтелект, REST API.

# ABSTRACT

UDC 004:005.9

Lupiak M. D. Development of a web application for personal task management and time planning using the ASP.NET Core and React technology stack: bachelor's thesis in specialty 121 Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 95 p.

This bachelor's qualification work presents the development of the web application “CheckIt” for creating, organizing, and managing personal checklists. The system allows users to manually add tasks or generate them automatically using an integrated artificial intelligence module, manage lists, archive completed items, track progress, and customize the interface according to personal preferences. The application features convenient multi-page navigation, authentication via email or Google, theme switching, and calendar-based task visualization.

The study includes an analysis of the system’s information architecture and describes the application’s structure, divided into client-side and server-side components. Both parts of the application were developed, and comprehensive testing was performed to confirm its correct functionality.

The software was developed using the C# programming language for the server side (ASP.NET Core) and JavaScript with the React library for the client side. Additional tools and technologies used include TypeScript, Tailwind CSS, Entity Framework Core, PostgreSQL, Google OAuth, and the Git version control system.

Keywords: checklist, React, ASP.NET Core, web application, artificial intelligence, REST API.

ЗМІСТ

# ВСТУП

**Обґрунтування вибору теми дослідження.** Упродовж останнього десятиліття темп життя й обсяг інформаційних потоків зросли до рівня, коли традиційні паперові планери або списки у нотатках мобільного телефону перестали бути дієвими. Вміння правильно розпоряджатися часом набуває критичного значення як у професійній діяльності, так і в особистому житті.

Люди щодня стикаються з необхідністю виконання великої кількості завдань, що вимагає системного підходу до планування, організації та контролю виконання. У зв’язку з цим виникає потреба в інструментах, які б допомагали структурувати щоденні справи та підвищувати рівень самоорганізації.

Незважаючи на велику кількість програм для тайм-менеджменту, користувачі часто стикаються з цифровим хаосом. Сервіси або обмежуються списками справ, або працюють з Kanban-дошками, не поєднуючи всі інструменти та не підтримуючи ШІ. У результаті дані розпорошуються, втрачається контекст, а користувач змушений вручну структурувати цілі.

Запропонована тема дипломного проєкту покликана усунути ці недоліки шляхом створення вебплатформи, де список справ й календарне подання працюють як взаємодоповнювані режими, а модуль штучного інтелекту автоматично розбиває загальну мету на чіткі підзадачі. Використання відкритого та потужного стеку ASP.NET Core для серверної частини і React для клієнтської забезпечує високу продуктивність, масштабованість і просту інтеграцію з іншими сервісами.

Тому розробка вебплатформи для керування особистими задачами та планування, що об’єднує автоматичне створення планів на основі ШІ, спільний доступ до шаблонів і календарну інтеграцію є актуальною, оскільки дає змогу

скоротити час на рутину, мінімізувати фрагментацію даних та підвищити особисту продуктивність.

**Зв’язок роботи з науковими програмами, планами, темами.** Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

**Мета та завдання дослідження.** Метою є підвищення ефективності особистого тайм-менеджменту користувачів шляхом створення інтегрованого вебзастосунку, який забезпечує цілісне зберігання даних, автоматичну декомпозицію цілей за допомогою штучного інтелекту та гнучке календарне планування.

**Відповідно до поставленої мети потрібно вирішити такі завдання:**

– проаналізувати предметну область;
– провести порівняльний аналіз аналогів;
– провести аналіз підходів до вирішення завдання;
– обґрунтувати вибір архітектури, технологій та засобів розробки;
– спроєктувати алгоритми роботи системи;
– реалізувати клієнтську та серверну частини веб-астосунку;
– підготувати інструкцію користувача;
– провести тестування вебзастосунку.

**Об’єкт дослідження** – процес розробки вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React.

**Предмет дослідження** – методи та засоби розробки програмного забезпечення для планування часу й керування задачами, що передбачають використання сучасних технологій розробки інтерфейсів, серверної логіки та інструментів штучного інтелекту.

**Методи дослідження.** У процесі досліджень використовувались: теорія алгоритмів та структур даних для побудови логіки роботи з чеклістами та задачами,

теорія баз даних при проєктуванні структури збереження інформації, принципи UI/UX-дизайну для розробки зручного та адаптивного інтерфейсу, принципи розподілених систем і компонентного проєктування для створення багаторівневої архітектури, методи модульного та інтеграційного тестування для перевірки працездатності окремих компонентів і їх взаємодії в рамках усієї системи.

**Новизна отриманих результатів.**

1. Подальшого розвитку набув метод формування списку задач, що на відміну від аналогів, дозволяє користувачу автоматизувати розбиття загальної мети на окремі підзадачі за допомогою використання штучного інтелекту, що забезпечує користувачу економію часу та підвищення ефективності планування.

2. Подальшого розвитку набув метод інтегрованого планування та відстеження дедлайнів, що об'єднує функціонал створення чеклістів із календарним представленням та переліком майбітніх завдань, дозволяючи користувачам ефективно управляти часом та пріоритизувати дії.

**Практичне значення отриманих результатів** полягає в тому, що розроблена система надає користувачам зручну та інтуїтивно зрозумілу платформу для організації завдань, що дозволяє не лише ефективно створювати та управляти чеклістами, але й автоматизувати певні етапи планування для пришвидшення роботи.

**Особистий внесок здобувача.** Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто.

**Апробація матеріалів бакалаврської дипломної роботи.** Матеріали бакалаврської дипломної роботи доповідалися на LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025).

**Публікації.** Основні результати дослідження були опубліковані в матеріалах конференції [1].

# 1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Аналіз предметної області

Динамічний ритм сучасного життя змушує людину тримати під контролем десятки робочих, побутових і соціальних зобов’язань, тож уміння швидко фіксувати справи, вибудовувати пріоритети й розподіляти час стає критично важливим. Класичні паперові планувальники давно поступилися місцем цифровим сервісам, але більшість із них використовує вузький підхід до організації задач. Одні додатки зберігають лише лінійний список завдань, інші фокусуються на календарі, ще інші пропонують канбан-дошку, проте майже не існує рішень, які б гармонійно об’єднували всі ці функції в одному середовищі [2]. Через відсутність єдиного інтегрованого інструменту для планування, користувач змушений дублювати інформацію, синхронізувати дати, вручну декомпонувати великі цілі на підзадачі й витрачати час на пошук контексту, що зрештою знижує продуктивність та збільшує когнітивне навантаження.

Паралельно ринок масово інтегрує технології штучного інтелекту, здатні розуміти контекст і пропонувати релевантні підказки. Проте у сфері персонального тайм-менеджминту ці можливості майже не реалізовано й користувачеві досі доводиться власноруч розбивати загальну мету на дрібні кроки та розкладати їх у календарі. Водночас спостерігається виразний запит на інструменти, які б не лише зберігали записи, а й автоматизували рутинні завдання, наприклад, пропонували шаблони типових процесів, розбивали задачі на кроки та давали змогу ділитися шаблонами з іншими користувачами.

Таким чином предметна область дослідження охоплює цифрові механізми організації особистої діяльності, де основними викликами є фрагментація даних та відсутність інтелектуальної підтримки. Створення вебзастосунку, що об’єднає ці аспекти у межах реактивного інтерфейсу, доповнить їх ШІ-генерацією кроків

виконання для задач та бібліотекою публічних шаблонів, здатне усунути ці проблеми.

### 1.2 Порівняльний аналіз аналогів

На даному етапі розвитку цифрових технологій ринок програмного забезпечення для керування задачами й контрольними списками представлений великою кількістю рішень. Серед найпоширеніших – Todoist, Microsoft To Do, Trello та Notion. Усі ці сервіси мають спільну мету – допомогти користувачам ефективно організовувати завдання, структурувати справи та підвищувати особисту продуктивність. Проте, незважаючи на подібність у призначенні, вони істотно відрізняються за функціональністю, підходами до організації роботи з даними, можливостями інтеграції та рівнем автоматизації. Для глибшого розуміння переваг і недоліків кожного з рішень було проведено їх порівняльний аналіз.

Todoist (рисунок 1.1) – один із найпопулярніших інструментів для роботи зі списками справ. Він має простий інтерфейс, підтримує інтеграцію з календарями.. Водночас у Todoist відсутня підтримка архівації списків, немає динамічних шаблонів для багаторазового використання та не реалізовано функцій автоматичного розбиття цілей на підзадачі [3].

Microsoft To Do (рисунок 1.2) вирізняється глибокою інтеграцією в екосистему Microsoft та базовими можливостями для відстеження справ, але також позбавлений розширених функцій, таких як архівація списків, збереження у бібліотеці шаблнів та ШІ-підказки [4].

Trello (рисунок 1.3) орієнтований переважно на візуальне управління завданнями у форматі дошок, підтримує шаблони та архівацію, але не має календарної синхронізації за замовчуванням, аналітики та інтелектуальних функцій. Його можливості можна розширити за допомогою сторонніх Power-Up розширень, проте більшість із них є платними або потребують додаткового налаштування [5].

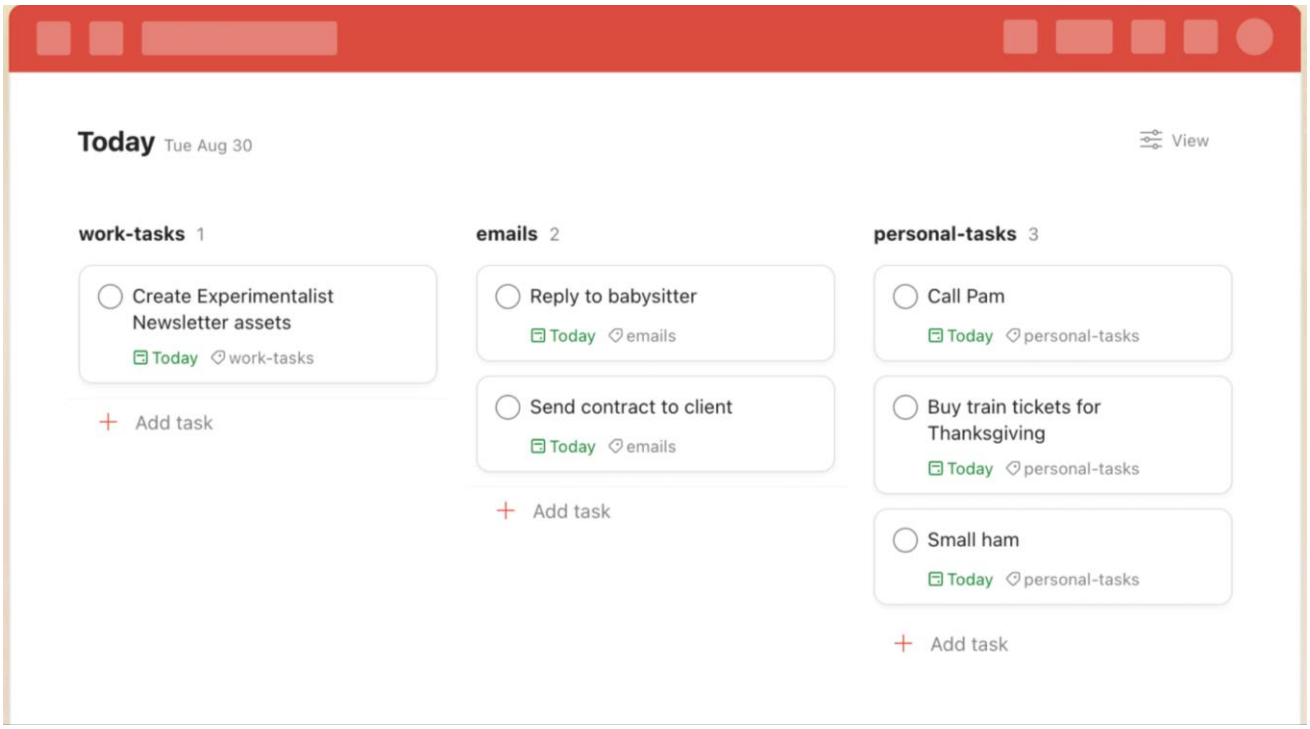



Рисунок 1.1 – Приклад роботи застосунку Todoist

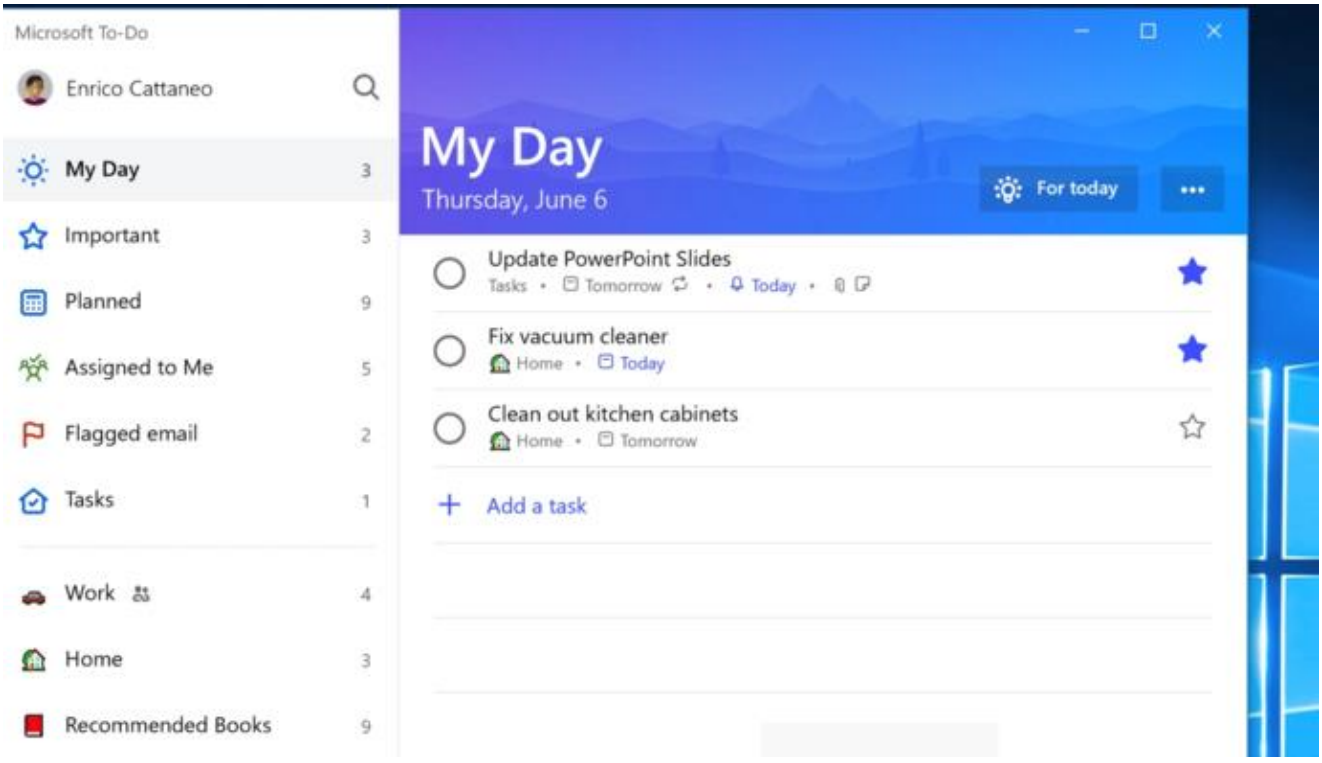



Рисунок 1.2 – Приклад роботи програми Microsoft To Do

Notion (рисунок 1.4) є універсальним інструментом для організації інформації, який поєднує можливості текстового редактора, таблиць, дошок та баз даних в одному середовищі. Його гнучкість дозволяє створювати власні системи продуктивності з нуля, адаптовані до будь-якого стилю роботи. Водночас такий підхід вимагає значних зусиль на етапі налаштування, оскільки більшість компонентів користувач формує самостійно. Хоча Notion підтримує базову генерацію тексту за допомогою вбудованого ШІ, він не забезпечує автоматичного структурування задач і повноцінної взаємодії з календарем, що обмежує його зручність у щоденному тайм-менеджменті. [6].

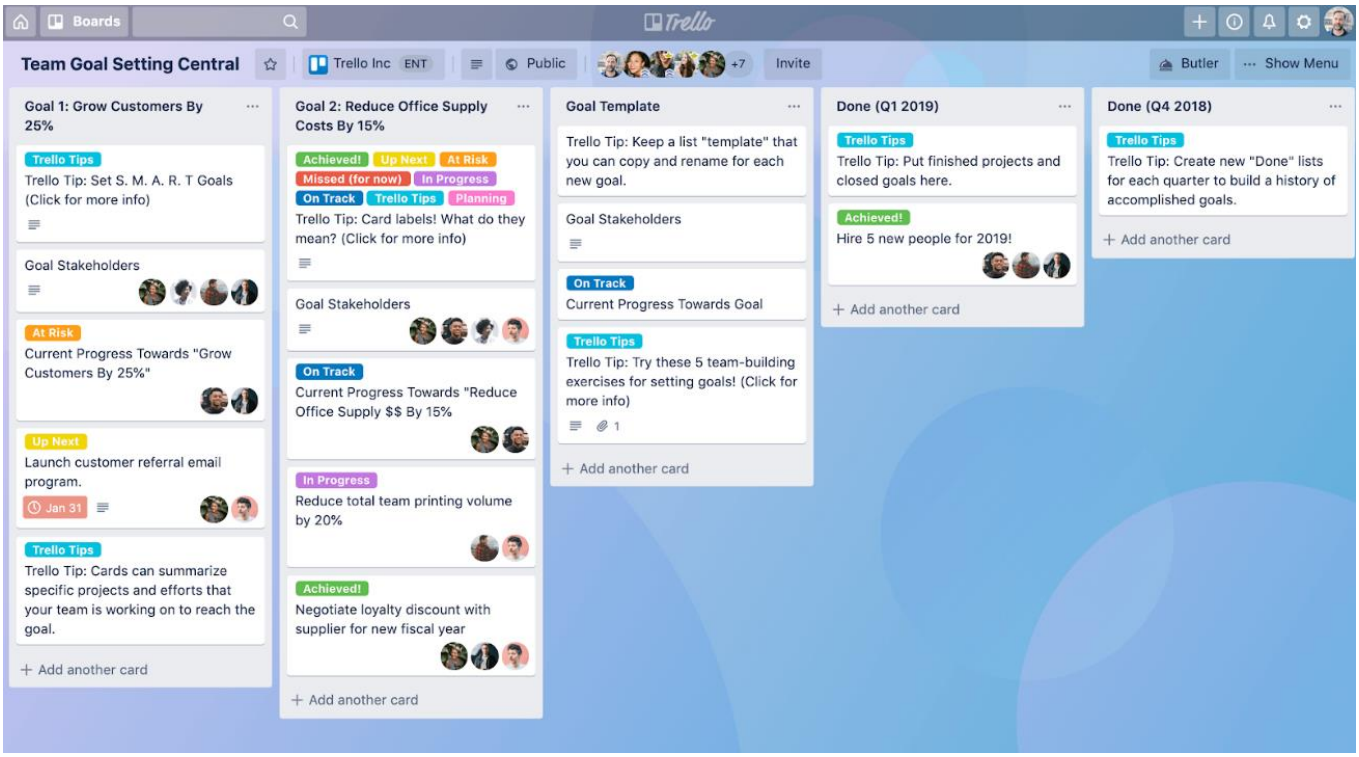



Рисунок 1.3 – Приклад роботи вебзастоснку Trello

Розроблений застосунок «CheckIt» поєднує переваги згаданих систем, усуваючи при цьому їхні обмеження. Він забезпечує автоматичну синхронізацію з календарем, підтримує архівацію та відновлення списків, реалізує механізм

динамічних шаблонів, які можна клонувати й повторно використовувати, має вбудовану аналітику виконання завдань та, що найголовніше, включає модуль штучного інтелекту, який автоматично декомпонує сформульовану мету на набір підзадач. Це значно спрощує процес планування, усуває необхідність вручну структурувати складні задачі та економить час користувачів.

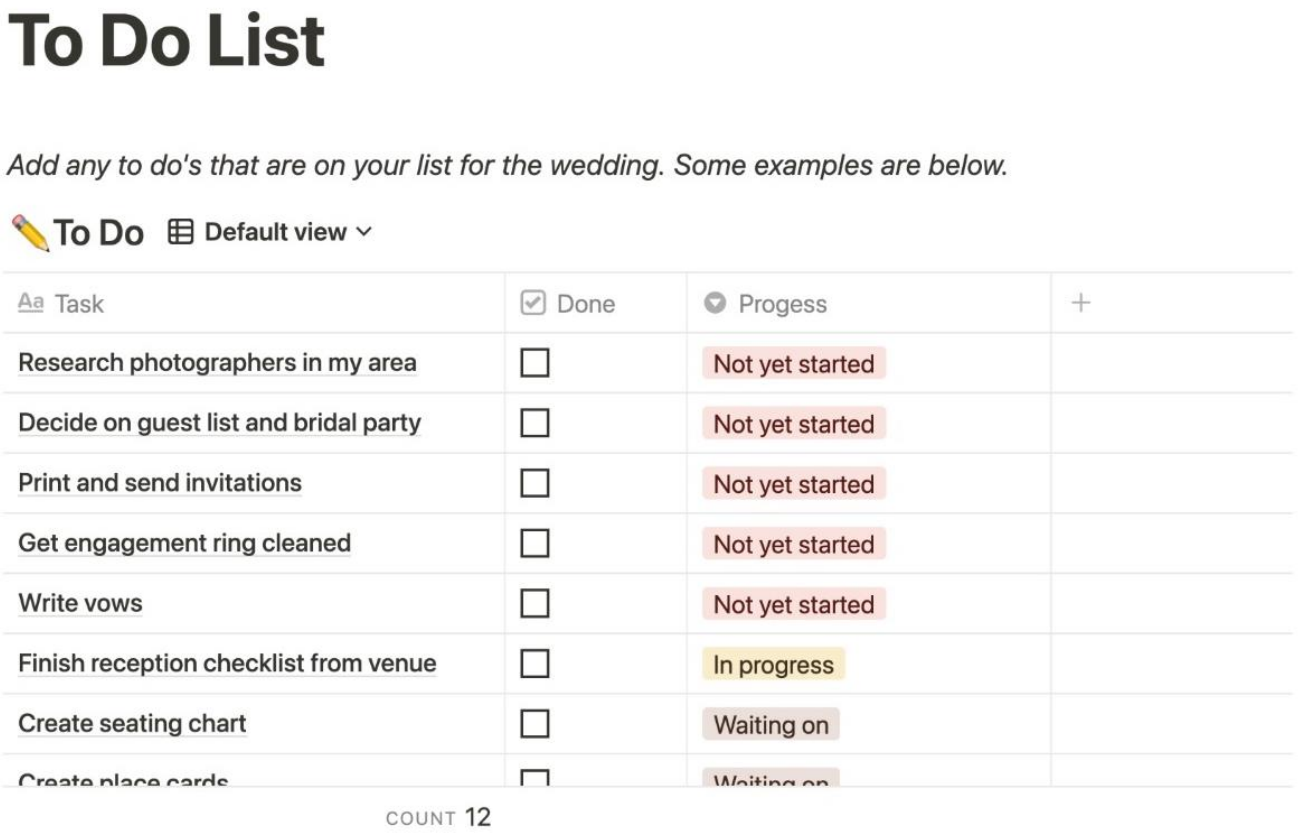



Рисунок 1.4 – Приклад роботи додатку Notion

Порівняльний аналіз дозволив наочно оцінити функціональні переваги кожного із сервісів, проаналізувавши наявність таких функцій як, як підтримка штучного інтелекту, шаблони, архівування, аналітика та календарна інтеграція. У таблиці 1.1 узагальнено результати порівняльного аналізу.

Таблиця 1.1 – Порівняльний аналіз аналогів

| Критерій | Todoist | Microsoft To Do | Trello | Notion | CheckIt |
|---|---|---|---|---|---|
| Синхронізація з календарем | 1 | 1 | 0 | 0 | 1 |
| Архівуваня списків | 0 | 0 | 1 | 1 | 1 |
| Динамічні шаблони | 0 | 0 | 1 | 1 | 1 |
| AI-декомпозиція цілей на підзадачі | 0 | 0 | 0 | 1 | 1 |
| Вбудована аналітика виконання завдань | 1 | 1 | 0 | 0 | 1 |
| Загальна оцінка | 40% | 40% | 40% | 60% | 100% |

Порівняльний аналіз вебсистем для управління особистими задачами та планування часу виявив, що кожна з цих платформ має свої переваги та недоліки. Результати аналізу підтверджують доцільність розробки додатку «CheckIt», оскільки він має більшу функціональність, ніж його аналоги.

### 1.3 Аналіз підходів до вирішення завдання

Для створення вебзастосунку, який забезпечує не лише зручне керування контрольними списками, але й розширений функціонал на основі штучного інтелекту, необхідно обрати сучасний технологічний стек, що дозволить поєднати масштабованість, безпеку та зручність використання. Архітектурні рішення мають забезпечити стабільну роботу системи, легкість подальшого розвитку функціоналу та технічну адаптивність.

Серверна частина застосунку реалізується з використанням ASP.NET Core [7], що дозволяє створювати потужні та продуктивні вебсервіси з чітким розмежуванням бізнес-логіки, даних і рівня доступу. Ця платформа підтримує REST API, що спрощує інтеграцію з клієнтською частиною, сторонніми сервісами та

мікросервісною архітектурою. Для безпеки застосовуються сучасні засоби автентифікації – JWT-токени [8] та підтримка авторизації через Google OAuth.

Клієнська частина додатку реалізується з використанням бібліотеки React [9], яка забезпечує побудову динамічного, інтуїтивно зрозумілого інтерфейсу. Завдяки компонентній структурі та можливості використовувати TypeScript, React дозволяє створювати гнучкий інтерфейс із підтримкою таких елементів, як перетягування елементів списків, контекстна навігація, візуальні індикатори прогресу та адаптивна верстка під мобільні пристрої.

Як основну систему керування базами даних обрано PostgreSQL [10], яка ідеально підходить для структурованих даних і підтримує складні зв’язки між об’єктами, такими як чеклісти, елементи, шаблони, користувачі тощо. Також планується використання Redis як швидкого кешу для зберігання сесій і тимчасових даних, а також як механізму реалізації черг повідомлень і сповіщень.

Особливою частиною архітектури є інтеграція модуля штучного інтелекту, який дозволяє автоматично розбивати загальні формулювання задач на логічно структуровані підзадачі. Для реалізації цієї функції використовується модель Google Gemini, яка доступна через API та забезпечує генерацію змістовних сценаріїв дій на основі введеного користувачем тексту. Застосунок надсилає запит до AI-сервісу, отримує список рекомендованих кроків і додає їх у вигляді підзадач до створеного чекліста. Такий підхід дозволяє скоротити час на планування та автоматизувати деталізацію загальних цілей без необхідності в додатковій ручній роботі.

Інтерфейс системи розробляється з урахуванням принципів сучасного UX-дизайну – мінімалізму, адаптивності, підтримки нічного темного режиму та швидкого доступу до основних функцій. Також передбачено створення мобільної версії застосунку з можливістю офлайн-доступу та синхронізацією після підключення до мережі.

Таким чином, обраний підхід поєднує перевірені технології розробки, такі як ASP.NET Core, React, PostgreSQL із сучасними AI-рішеннями на базі Google Gemini,

що дозволяє створити ефективний, безпечний і гнучкий інструмент для управління особистими завданнями з підтримкою інтелектуальної автоматизації.

### 1.4 Постановка задач для розробки програмного продукту

Проаналізувавши сучасні виклики у сфері особистого тайм-менеджменту та переваги й недоліки існуючих програмних рішень, а також враховуючи можливості застосування технологій штучного інтелекту, було визначено наступні завдання, які необхідно виконати для розробки вебзастосунку:

1. Розробити та обґрунтувати архітектуру вебзастосунку, що забезпечує цілісне зберігання даних та ефективну взаємодію між його компонентами.

2. Виконати порівняльний аналіз існуючих підходів до автоматичної декомпозиції цілей та обрати оптимальний для інтеграції зі штучним інтелектом.

3. Спроектувати та реалізувати загальний алгоритм роботи системи, включаючи логіку взаємодії користувача з функціями автоматичної декомпозиції та календарного планування.

4. Обґрунтувати вибір стеку технологій та інструментальних засобів розробки для реалізації клієнтської та серверної частин вебзастосунку.

5. Реалізувати функціонал створення та управління обліковими записами користувачів, забезпечивши безпеку даних.

6. Розробити інтуїтивно зрозумілий та адаптивний користувацький інтерфейс, що забезпечить зручну взаємодію користувача з усіма функціями системи, наприклад, створення чеклістів, календарне планування, відстеження прогресу тощо.

7. Забезпечити програмну реалізацію функцій створення, редагування та гнучкого календарного планування чеклістів та задач.

8. Інтегрувати та реалізувати модуль автоматичної декомпозиції цілей на підзадачі з використанням алгоритмів штучного інтелекту.

9. Підготувати детальну інструкцію користувача для ефективного використання всіх можливостей вебзастосунку.

10. Провести комплексне тестування розробленого програмного продукту для перевірки його функціональності, надійності та відповідності поставленим вимогам.

Технічне завдання на розробку наведено в додатку А.

### 1.5 Висновки

У результаті проведеного аналізу предметної області визначено основні проблеми сучасних інструментів для управління особистими завданнями, зокрема відсутність інтегрованих рішень, нестачу автоматизації та інтелектуальної підтримки.

Проведено порівняльний аналіз популярних аналогів, що підтвердив актуальність створення нового вебзастосунку з розширеним функціоналом. Обґрунтовано вибір архітектурних підходів і технологій, таких як ASP.NET Core, React, PostgreSQL та модель Google Gemini. Сформульовано чіткий перелік задач для реалізації програмного продукту, що охоплює розробку інтерфейсу, реалізацію функціоналу, інтеграцію ШІ та проведення тестування.

# 2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА РОЗРОБКА АЛГОРИТМІВ ПРОГРАМИ

## 2.1 Аналіз предметної області

Для реалізації вебзастосунку, що дозволяє ефективно керувати особистими завданнями, основну роль відіграє правильна організація даних, з якими працює система. У межах платформи «CheckIt» ці дані формуються безпосередньо користувачем під час взаємодії з додатком і включають інформацію про чеклісти, завдання, шаблони та параметри облікового запису.

Основним джерелом даних є діяльність користувача. Після реєстрації у системі створюється обліковий запис, де зберігається електронна адреса, пароль у зашифрованому вигляді, а також базові налаштування профілю. Ці дані використовуються для автентифікації та персоналізації інтерфейсу.

Головною структурною одиницею є чекліст – список завдань, об’єднаних за певною темою чи метою. Кожен чекліст містить назву, опис, дату створення, а також статус: активний чи архівований. Завдання всередині списку описуються параметрами, такими як текстовий заголовок, позначка виконання, та, за необхідності, короткий опис або дата завершення.

Під час роботи з додатком користувач має можливість створювати, редагувати та видаляти як самі списки, так і окремі задачі в них. Списки можна сортувати за датою створення або закріплювати як важливі, щоб вони відображались на початку панелі. Також реалізована можливість архівування завершених чеклістів без повного їх видалення, що дозволяє зберигати історію дій користувача.

Щоб прискорити процес створення нових завдань, у додатку передбачено систему шаблонів. Користувач може зберегти будь-який список як шаблон та згодом повторно його використовувати. Шаблони зберігаються у централізованому вигляді з можливістю надання доступу іншим користувачам для перегляду чи редагування.

Особливу роль відіграє модуль штучного інтелекту, який інтегровано у процес створення чекліста. Замість ручного заповнення кожного пункту, користувач може ввести загальну мету, наприклад «організувати переїзд», після чого система використовуючи модель Google Gemini, автоматично генерує перелік логічно пов’язаних підзадач. Ці підзадачі одразу додаються у форму нового списку, де їх можна редагувати або доповнювати.

Таким чином, обробка та структура даних у системі спрямовані на спрощення взаємодії, зменшення ручної роботи, підвищення зручності повторного використання раніше введеної інформації та швидке досягнення цілей користувачів.

## 2.2 Розробка загальної архітектури програмного продукту

Загальна архітектура вебзастосунку «CheckIt» побудована відповідно до принципів сучасної багаторівневої системи з чітким розмежуванням відповідальностей між компонентами. Такий підхід дозволяє досягти гнучкості, масштабованості, високої продуктивності та легкості супроводу. Архітектура базується на розділенні на клієнтську частину (Frontend), серверну частину (Backend) і базу даних, між якими здійснюється взаємодія за допомогою HTTP/HTTPS-протоколу [11] та RESTful API [12]. Схематичне зображення загальної структури представлено на рисунку 2.1.

Клієнтська частина застосунку реалізована з використанням JavaScript-бібліотеки React, яка забезпечує побудову гнучкого, динамічного інтерфейсу користувача. React дозволяє створювати модульну структуру у вигляді компонентів, кожен з яких відповідає за окремий елемент інтерфейсу. Уся логіка відображення, навігації та керування станом у клієнтській частині додатку реалізована в межах архітектурних шаблонів управління станом, таких як Context API. Передбачена також інтеграція з API-клієнтом для здійснення запитів до серверної частини та обробки відповідей.

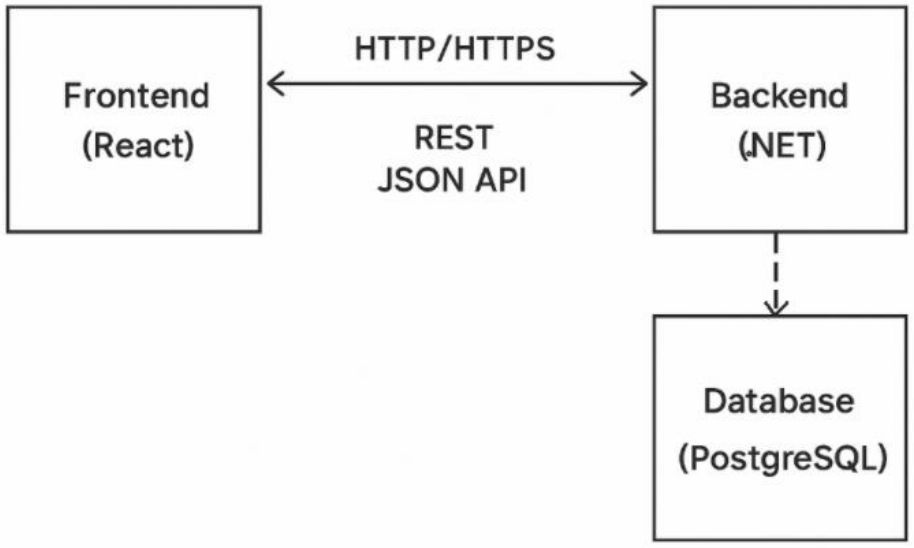



Рисунок 2.1 – Загальна архітектура системи «CheckIt»

Серверна частина побудована за допомогою використання ASP.NET Core і відповідає за обробку бізнес-логіки, авторизацію, валідацію даних, взаємодію з базою даних і зовнішніми сервісами.

Серверна частина структурована згідно з принципами чистої архітектури [13], де реалізовано чітке розділення на рівні (рисунок 2.2):

– API Layer – визначення кінцевих точок, обробка запитів і проміжне забезпечення безпеки;

– Application Layer – реалізація варіантів використання, сервісів, інтерфейсів і логіки валідації;

– Domain Layer – опис сутностей та бізнес-правил;

– Infrastructure Layer – доступ до бази даних, інтеграції з зовнішніми сервісами та робота з базою даних.

Уся взаємодія між клієнтською та серверною частинами відбувається через REST API, який обробляє дані у форматі JSON. Це дозволяє забезпечити просту та зрозумілу передачу даних, простоту налагодження й розширення функціоналу.

Для зберігання даних використовується PostgreSQL – потужна реляційна система керування базами даних, яка дозволяє реалізувати складну модель із таблицями користувачів, задач, чеклістів, шаблонів, подій календаря тощо.

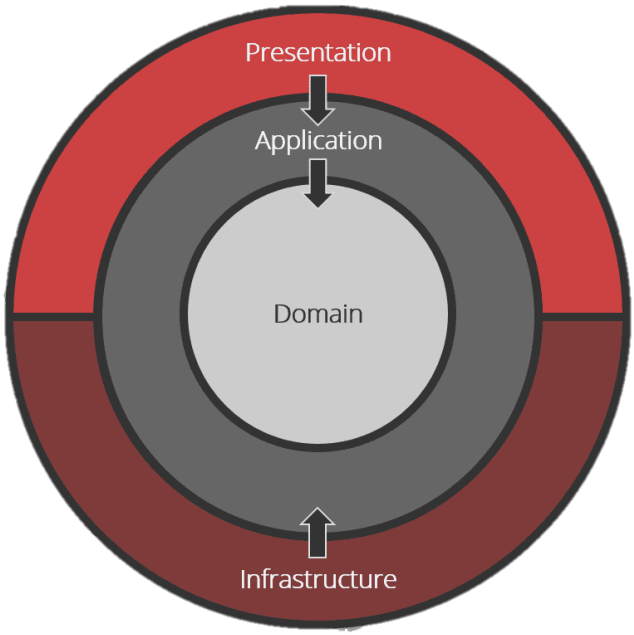



Рисунок 2.2 – Чиста архітектура

Серверна частина взаємодіє з БД за допомогою ORM [14], забезпечуючи цілісність даних, спрощення CRUD-операцій та зменшення кількості ручного SQL-коду. Доступ до бази даних організовано через шар інфраструктури, а взаємодія між рівнями логічно ізольована, що підвищує гнучкість і надійність системи.

Для забезпечення стабільної роботи та підвищення продуктивності вебзастосунку було закладено архітектурні принципи, що дозволяють легко масштабувати та підтримувати систему в майбутньому. Завдяки розподіленій структурі з чітким розмежуванням відповідальностей між клієнтською, серверною та інфраструктурною частинами, система збирігає високу гнучкість та підтримує впровадження нових можливостей без порушення наявної логіки.

Клієнтська частина реалізована з використанням бібліотеки React, що дозволяє будувати реактивний інтерфейс із сучасною адаптивною верстkою. Серверна частина побудована на основі ASP.NET Core, що забезпечує швидку обробку запитів, чітке розділення бізнес-логіки, підтримку REST API та безпечну автентифікацію користувачів.

Для роботи з реляційною структурою даних використовується PostgreSQL, яка дозволяє ефективно зберігати інформацію про задачі, чеклісти, шаблони та користувачів. Обмін даними між клієнтською та серверною частинами відбувається через HTTP/HTTPS із використанням формату JSON.

Окрему роль у системі відіграє інтеграція зі штучним інтелектом на базі Google Gemini, яка реалізована як зовнішній сервіс. Під час створення нового списку користувач може ввести загальну мету, а система автоматично генерує набір логічно структурованих підзадач. Це значно знижує навантаження на користувача, скорочує час планування та підвищує зручність роботи із задачами.

Уся архітектура проєкту орієнтована на розширюваність, тому у перспективі можливе додавання нових модулів, таких як спільне редагування, командна робота або інтеграція з календарними сервісами.

Таким чином, архітектура вебзастосунку «CheckIt» демонструє сучасний, модульний та гнучкий підхід до розробки, поєднуючи надійні технології з можливостями штучного інтелекту.

### 2.3. Розробка методу формування задач

Для забезпечення цілісної та логічної взаємодії користувача з системою було розроблено загальний алгоритм роботи вебзастосунку, а також окремі функціональні методи, що реалізують ключові сценарії. Особливістю реалізації є наявність двох нових підходів, які вигідно вирізняють цей застосунок серед аналогів

– це метод автоматизованого формування списку задач із використанням ШІ та метод інтегрованого календарного планування з прив'язкою до дедлайнів.

Метод формування задач реалізовано так, щоб надати користувачам максимальну гнучкість: вони можуть самостійно додавати завдання або скористатися генерацією підзадач на основі вказаної теми. Алгоритм цього процесу представлено на рисунку 2.3.

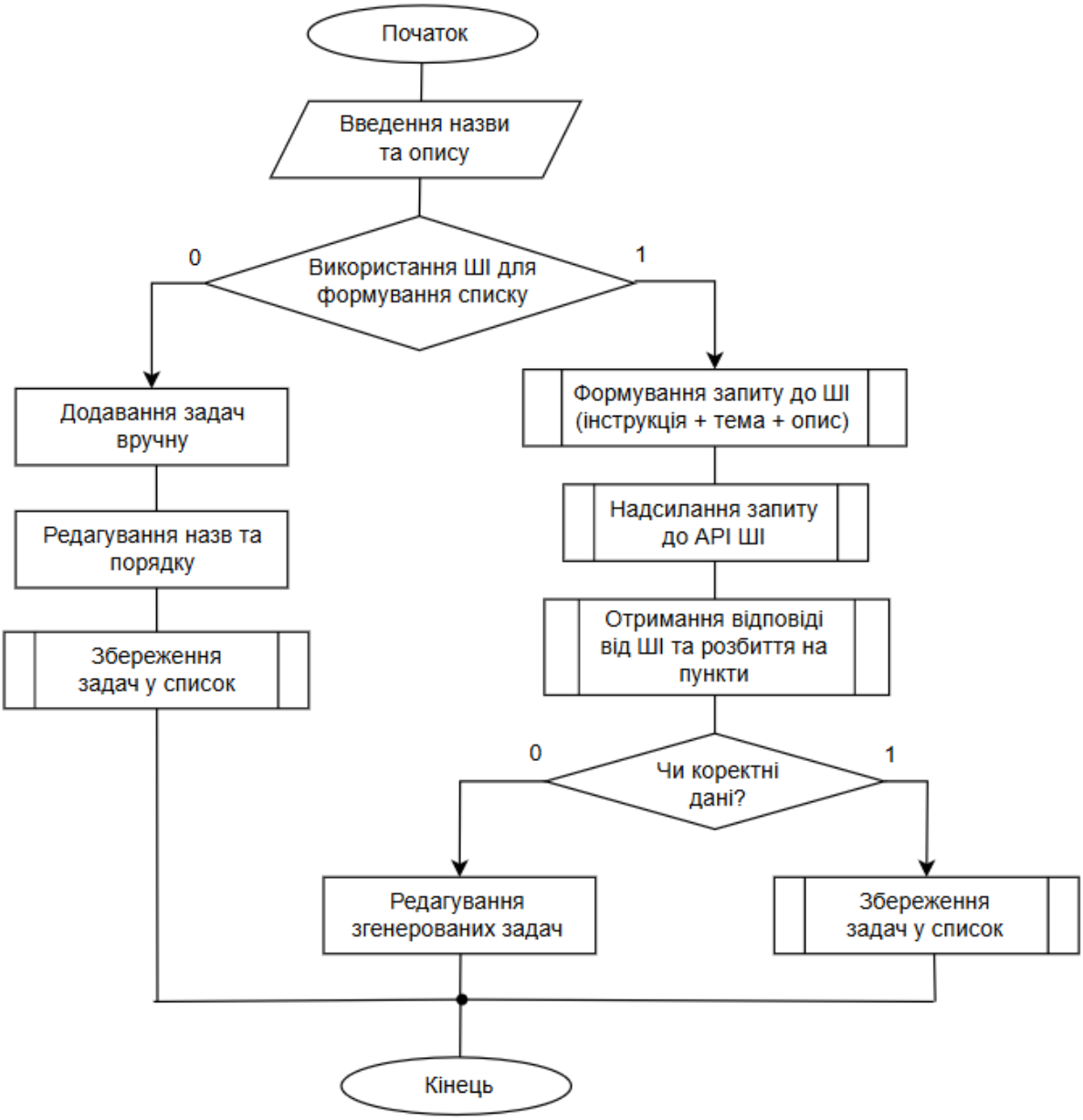



Рисунок 2.3 – Блок-схема методу формування задач

Після створення нового чекліста користувач вводить заголовок і, за бажанням, опис мети або контексту завдань. Далі система запитує, чи слід використовувати функцію генерації підзадач за допомогою ШІ. У разі підтвердження формується спеціалізований запит, який включає інструкцію для моделі, тему задачі та необов’язкові допоміжні уточнення. Запит надсилається до API зовнішньої мовної моделі Google Gemini, після чого відповідь обробляється і розбивається на окремі пункти.

Отримані дані автоматично перетворюються на структурований список завдань. Кожен елемент проходить валідацію, зокрема перевірку на відповідність допустимим форматам, обмеженням за довжиною та унікальність у рамках одного списку. У разі, якщо користувач не задоволений отриманим результатом, він має змогу відредагувати будь-який пункт або повністю змінити запропонований перелік.

У випадку, коли генерація через ШІ не використовується, користувач переходить до ручного додавання підзадач. Тут реалізовано інтерфейс швидкого введення, що підтримує функції редагування назв, зміну порядку задач за допомогою drag-and-drop, а також можливість дублювання елементів.

Завершальним етапом обох сценаріїв є збереження підготовленого списку до бази даних. У цьому процесі система автоматично додає часові мітки, ідентифікатори користувача та відповідну прив'язку до створеного чекліста.

Алгоритм забезпечує логічну послідовність дій, зручну адаптацію до обраного користувачем способу формування задач та передбачає можливість гнучкої взаємодії з інтерфейсом.

Таким чином, метод формування задач у вебзастосунку «CheckIt» об’єднує переваги ручного та автоматизованого підходів, дозволяє адаптувати результат під індивідуальні потреби та значно скорочує час, необхідний на початкове планування. Він відіграє центральну роль у загальній логіці роботи системи, оскільки саме з

якісно сформованих чеклістів починається подальше планування, відстеження дедлайнів та аналіз виконання.

### 2.4 Розробка методу інтегрованого планування та відстеження дедлайнів

Метод інтегрованого планування та відстеження дедлайнів реалізовано як взаємодію двох ключових підсистем: управління задачами всередині чекліста та візуалізації запланованих подій у календарному представленні. Такий підхід дозволяє користувачу не лише створювати завдання, але й бачити їх у часовому контексті, що є критично важливим для ефективного тайм-менеджменту.

Основна мета методу полягає у тому, щоб забезпечити узгоджене представлення завдань як у вигляді структурованих чеклістів, так і у форматі календаря. Це дозволяє поєднати переваги спискового підходу до планування з календарною візуалізацією, яка забезпечує розуміння загального навантаження та пріоритетів у часі. Блок-схема цього методу представлена на рисунку 2.4.

Процес починається з вибору або створення нового чекліста. Після цього користувач може вказати дедлайни для кожної задачі, задавши конкретну дату і за потреби час завершення. У разі, якщо дата не вказана, система пропонує шаблонні значення. Це дозволяє прискорити процес планування без втрати гнучкості.

Далі відбувається синхронізація даних чекліста із внутрішнім календарем. Завдання, що мають встановлені дедлайни, автоматично додаються до відповідних часових слотів у календарному інтерфейсі. Візуалізація відбувається у вигляді подій, які можна редагувати шляхом drag-and-drop або через спеціальне діалогове вікно. Це дозволяє користувачу швидко змінювати дати, переносити задачі чи переглядати їхній зміст без повернення до спискового представлення.

На додаток до календаря, метод включає формування окремого списку «Найближчі задачі», що автоматично оновлюється при кожній зміні чекліста або дедлайнів. У цей список потрапляють лише задачі, дедлайни яких знаходяться у

межах певного період, наприклад, наступні 7 днів. Таке представлення є надзвичайно зручним для щоденного планування, оскільки дозволяє сфокусувати увагу на актуальних діях без необхідності переглядати всі чеклісти.

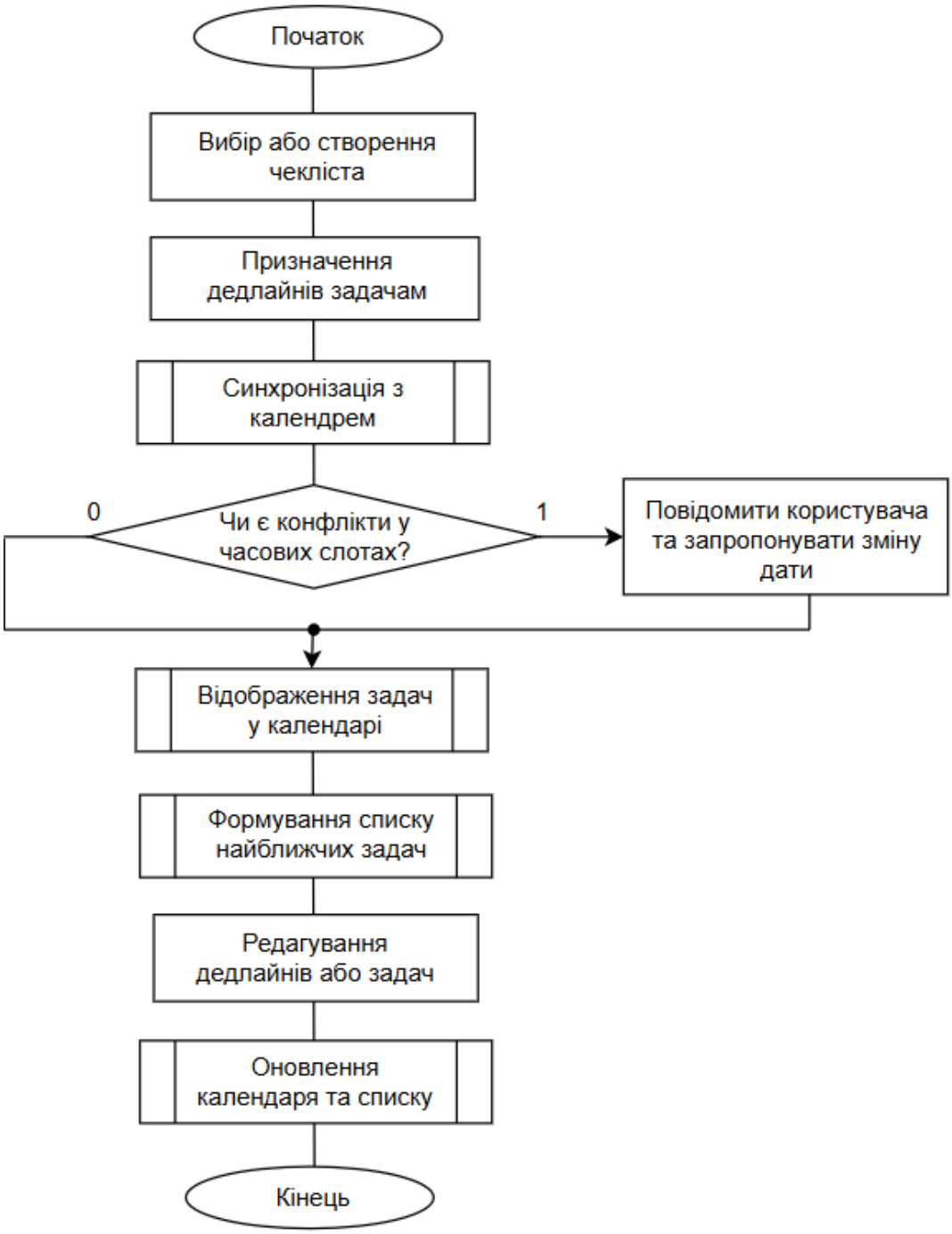



Рисунок 2.4 – Блок-схема методу інтегрованого планування та відстеження дедлайнів

У підсумку, запропонований метод дозволяє перетворити звичайний чекліст на потужний інструмент для керування часом, об'єднуючи інтерфейс задач зі зрозумілим календарним представленням і механізмами контролю термінів. Такий підхід суттєво покращує досвід користувача та розширює можливості використання вебзастосунку в контексті особистого планування, навчання або професійної діяльності.

## 2.5 Розробка алгоритмів роботи програми

Алгоритм взаємодії користувача з вебзастосунком побудовано з урахуванням сучасних підходів до UX-дизайну [15], що передбачають мінімізацію зайвих кроків, забезпечення швидкого доступу до основних функцій, гнучкість сценаріїв використання та підтримку ручного і автоматизованого введення інформації. Для цього процес було поділено на окремі етапи з чіткими переходами, що зручно візуалізуються у вигляді блок-схеми. Алгоритм взаємодії користувача із системою зображено на рисунку 2.5.

Після відкриття вебзастосунку користувач потрапляє на сторінку входу, де система виконує перевірку наявності активної сесії авторизації. Якщо користувач уже входив у систему, він автоматично перенаправляється на головну сторінку.

У випадку відсутності сесії, користувач має пройти авторизацію за допомогою логіну та пароля або через зовнішній провайдер Google. Цей етап забезпечує захист персональних даних і дозволяє пов’язувати збережені чеклісти з конкретним профілем. Якщо система не знаходить відповідного облікового запису в базі даних, користувачу пропонується пройти реєстрацію. Алгоритм цього процесу зображено на рисунку 2.6.

Після перевірки правильності введених даних, система створює обліковий запис у базі даних і ініціює сесію для подальшої роботи. Якщо в процесі виявлено

помилку, користувач отримує повідомлення та можливість її виправити. Такий підхід забезпечує безпечну реєстрацію та коректну ідентифікацію користувача.

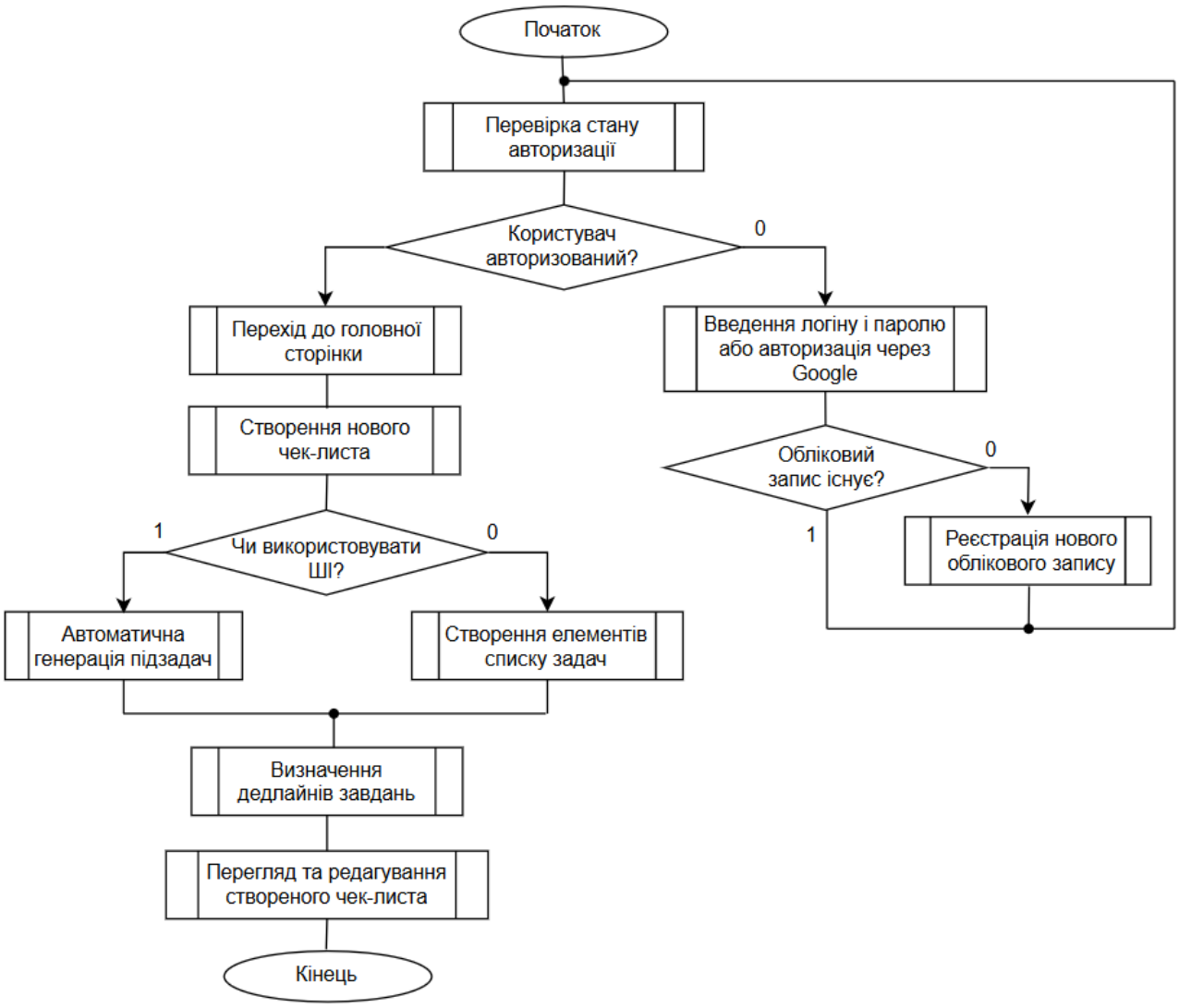



Рисунок 2.5 – Блок-схема основного алгоритму взаємодії з додатком

Після успішної автентифікації або створення акаунту, відкривається головна сторінка застосунку, де розміщені всі доступні функції: панель навігації, список активних чеклістів, архів, а також кнопка створення нового чекліста.

Процес створення нового списку задач починається з натискання відповідної кнопки. Користувач заповнює основні поля та має змогу активувати додаткову

опцію: використання штучного інтелекту. У випадку, якщо ця функція активована, система автоматично надсилає сформульовану мету до інтегрованого модуля штучного інтелекту на базі Google Gemini. Отримана у відповідь структурована послідовність підзадач додається у вигляді окремих елементів до нового чекліста.

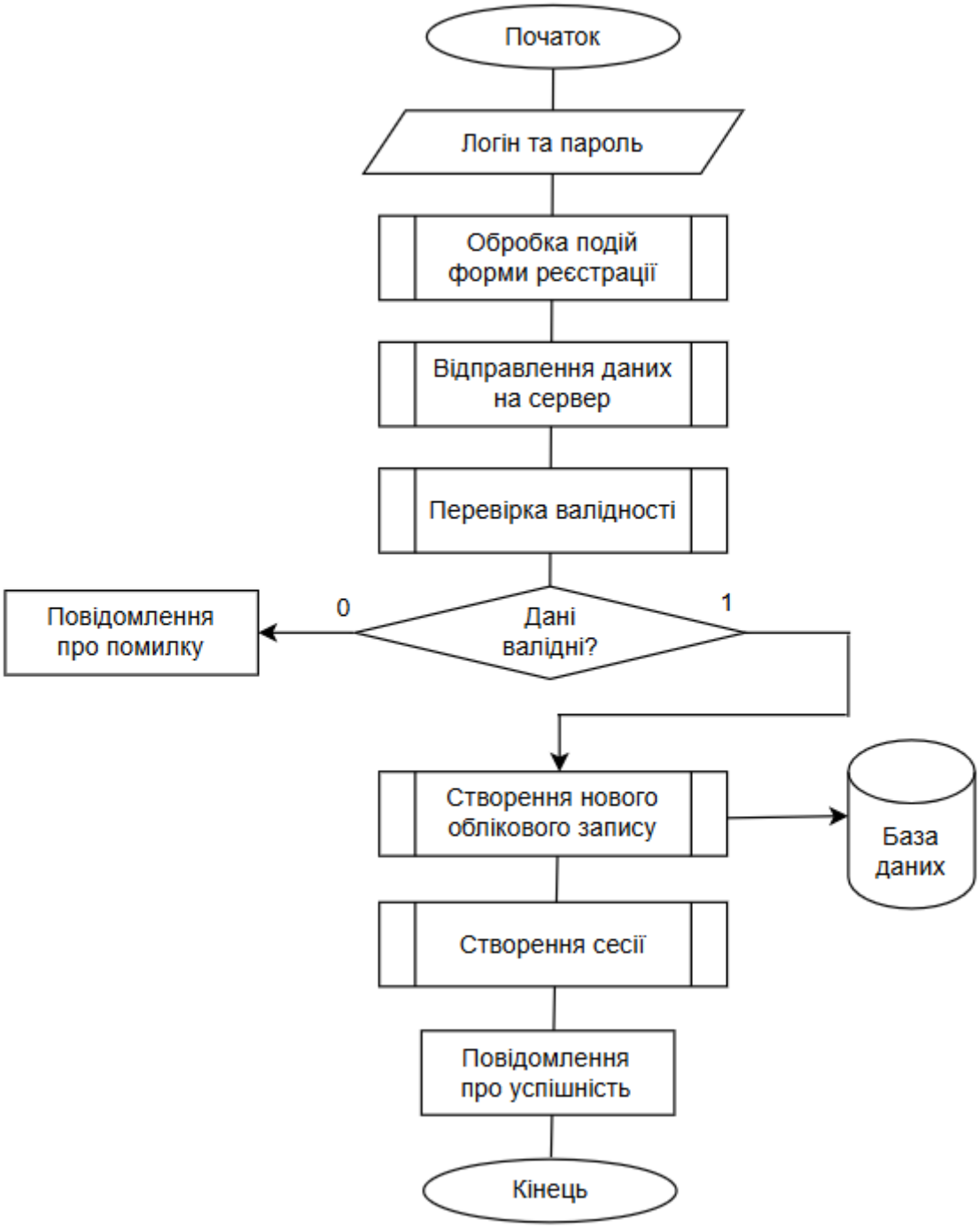



Рисунок 2.6 – Алгоритм створення облікового запису користувача

Якщо ж користувач вирішує не використовувати ШІ-функціонал, він може вручну створювати список задач. Цей сценарій є важливим для користувачів, які бажають повністю контролювати процес формування переліку завдань або мають специфічні вимоги до структури списку.

У наступному етапі, незалежно від способу формування задач, користувач має змогу встановити дедлайни для кожного елементу. Це дозволяє системі відстежувати терміни виконання задач, а в майбутньому – реалізовувати нагадування або підсвітку прострочених пунктів.

Останнім етапом є перегляд створеного чекліста в основному інтерфейсі. Користувач може вносити правки, редагувати, архівувати або видаляти завдання, позначати їх як виконані, додавати нові пункти або змінювати вже наявні. Таким чином, система підтримує повний цикл роботи із завданнями – від створення до архівації, та адаптується до будь-якого сценарію використання.

Щоб забезпечити гнучкість користування та адаптивність до різних сценаріїв, алгоритм роботи платформи передбачає розширення функціоналу у відповідь на дії користувача. Наприклад, під час редагування чек-листа можна активувати додаткові функції, такі як встановлення пріоритетів для окремих завдань, дублювання існуючих елементів або переміщення задач між списками. Також доступна можливість архівації завершених чек-листів без їх остаточного видалення, що дозволяє зберігати історію виконаної роботи.

Крім того, в архітектурі передбачено інтерактивну систему підказок, яка з’являється на ключових етапах взаємодії – створенні чек-листа, додаванні завдань або використанні модуля ШІ. Таке контекстне супроводження допомагає новим користувачам швидше освоїти функціонал системи без потреби в додаткових інструкціях.

## 2.6 Проєктування графічного інтерфейсу

Інтерфейс користувача відіграє важливу роль у забезпеченні інтуїтивної та швидкої взаємодії з вебзастосунком. У вебзастосунку «CheckIt» інтерфейс спроєктовано відповідно до принципів логічного групування функціональних блоків, адаптивного дизайну та мінімізації дій для досягнення результату.

При розробці використовувалась бібліотека React, яка дозволяє створювати гнучку компонентну структуру, що легко масштабується, розширюється та підтримує повторне використання елементів.

Основним середовищем роботи користувача є Dashboard – головний екран. Він відображає актуальну інформацію про задачі та надає доступ до швидких дій, таких як створення нового чекліста або перегляд запланованих завдань. На цьому екрані відображається блок з майбутніми задачами, календар, який відображає дедлайни, та список нещодавно створених списків. Структура даного екрана представлена на рисунку 2.3.

Складові основної сторінки:

1. Навігаційна панель
2. Зображення логотипу
3. Кнопка переходу на головну сторінку
4. Кнопка для переходу на стторінку зі списками
5. Кнопка для налаштування акаунту
6. Кнопка виходу з облікового запису
7. Кнопка створення нового чекліста
8. Список основних завдань
9. Вікно календаря
10. Робоча область

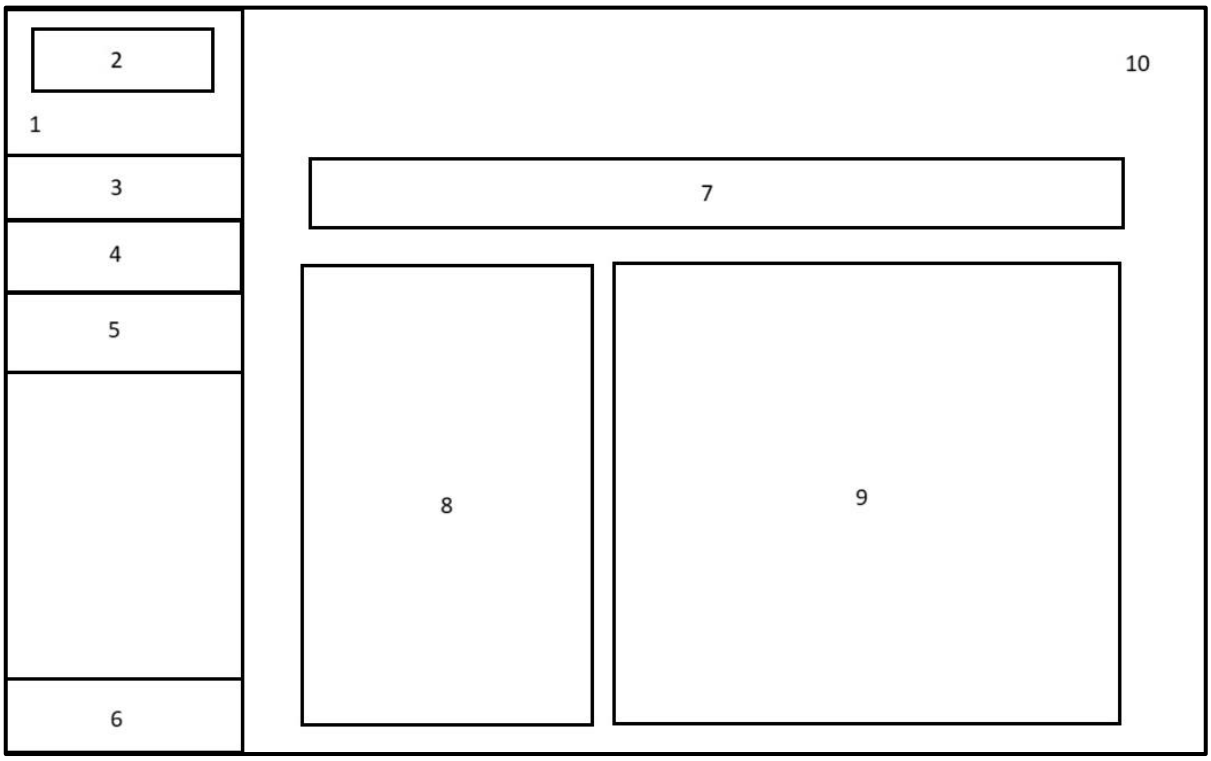



Рисунок 2.3 – Структура головної сторінки вебзастосунку «CheckIt»

Сторінка керування чеклістами дозволяє взаємодіяти з усіма створеними списками, шукати їх за назвою, закріплювати перемикати між активними та архівованими. Для кожного списку реалізовано окрему панель керування, що включає функції редагування, дублювання, архівації, видалення та визначення пріоритетів. Завдання всередині чекліста можна створювати вручну або генерувати автоматично за допомогою модуля штучного інтелекту. Структуру інтерфейсу цього розділу зображено на рисунку 2.4.

Складові сторінки керування чеклістами:

1. Робоча область
2. Кнопка створення нового чекліста
3. Кнопка переходу до активних чеклістів
4. Кнопка переходу до архівованих чеклістів
5. Кнопка для перегляду усіх чеклістів

6. Поле для пошуку
7. Область редагуваня чекліста
8. Назва списку задач
9. Кнопка для закріплення списку
10. Кнопка для дублікації чекліста
11. Кнопка для перейменування списку
12. Кнопка для архівування чекліста
13. Кнопка для видалення списку
14. Індикатор прогресу виконання
15. Дата створення чекліста
16. Список усіх завдань
17. Поле для введення нового завдання
18. Кнопка для створення нового елемента списку

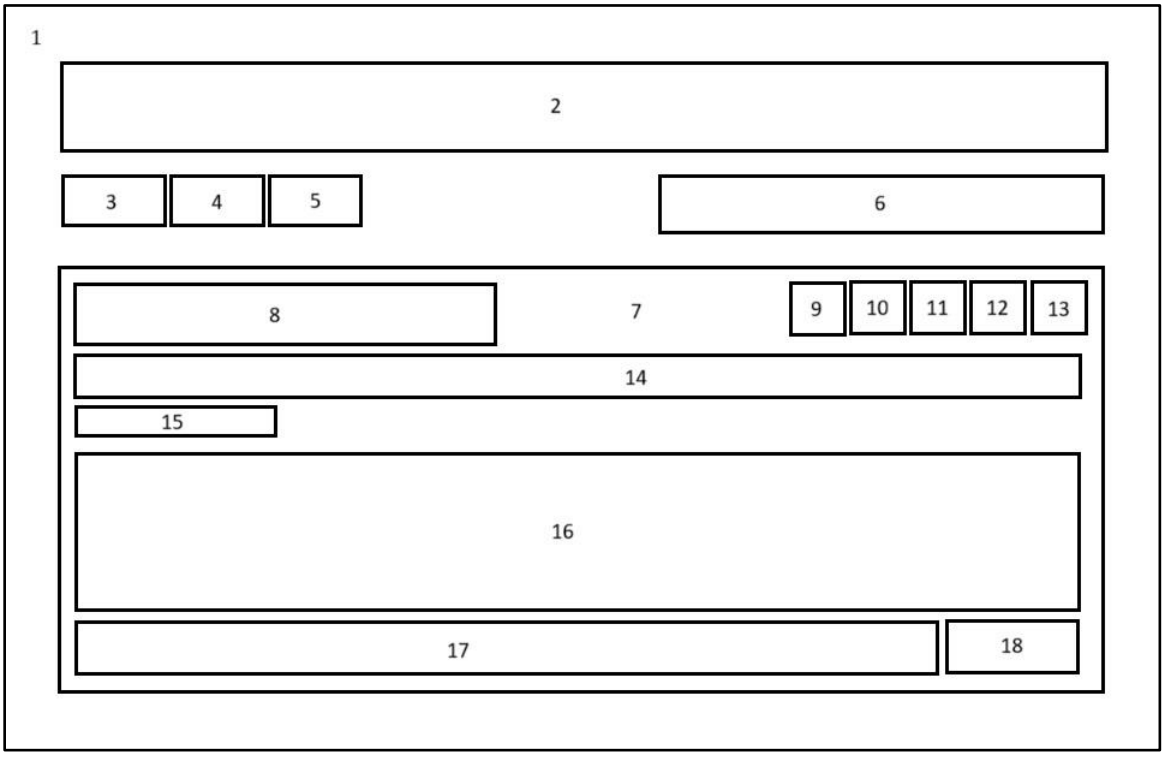



Рисунок 2.4 – Структурна схема сторінки керування чеклістами

Розділ інтерфейсу «Налаштування облікового запису» відповідає за персоналізацію середовища – зміну теми оформлення на світлу, темну або системну, оновлення пароля та керування інформацією профілю.

Просте компонування блоків дозволяє швидко знайти потрібну опцію без додаткових переходів.

Структура цієї сторінки представлена на рисунку 2.5.

Складові сторінки налаштування профілю:

1. Робоча область
2. Інформація про профіль користувача
3. Блок персоналізації інтерфейсу
4. Кнопка для перемикання на світлу тему
5. Кнопка для перемикання на темну тему
6. Кнопка для перемикання на системну тему
7. Форма зміни паролю користувача
8. Кнопка для скидання паролю
9. Поле введення поточного паролю
10. Поле введення нового паролю
11. Кнопка для підтвердження зміни паролю

Загальна структура інтерфейсу втілює принципи простоти, доступності та фокусованості на задачах користувача.

Інтерактивні елементи, плавна навігація та логічне групування функцій забезпечують швидке освоєння системи та знижують поріг входу для нових користувачів.

Завдяки адаптивному дизайну застосунок коректно працює на різних пристроях, включаючи планшети та смартфони.

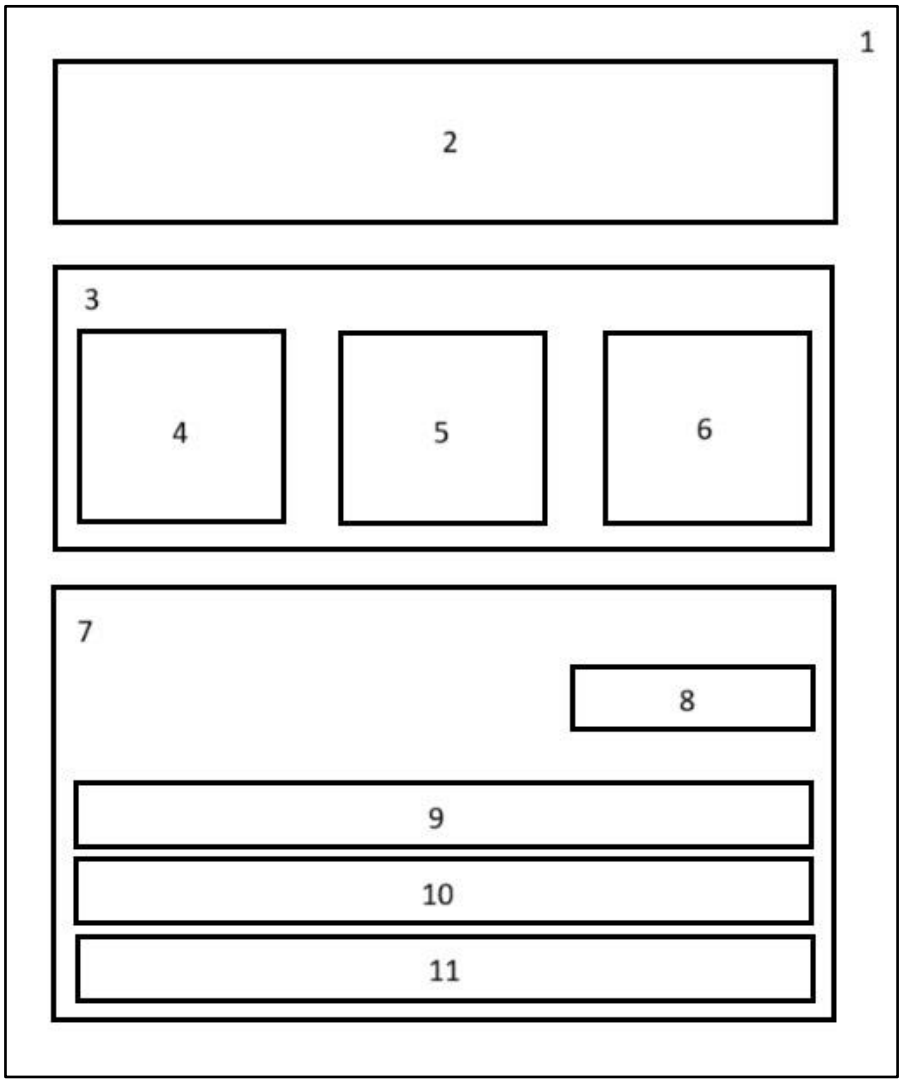



Рисунок 2.5 – Схема сторінки налаштувань профілю

Завдяки адаптивному дизайну застосунок коректно працює на різних пристроях, включаючи планшети та смартфони.

Таким чином, структура інтерфейсу «CheckIt» реалізована як зручне інструментальне середовище, що дозволяє ефективно організовувати особисті справи, не відволікаючись на технічні деталі.

## 2.7 Висновки

У межах розділу було спроєктовано ключові аспекти програмного засобу «CheckIt», що охоплюють організацію даних, архітектуру, інтерфейс та алгоритм взаємодії користувача з системою. Визначено структуру даних, яка забезпечує ефективне зберігання інформації про чеклісти, завдання, шаблони та облікові записи.

Запропоновано багаторівневу архітектуру з чітким розподілом відповідальностей між компонентами, що сприяє масштабованості й підтримці. Розроблено логіку інтерфейсу з урахуванням принципів зручності та адаптивності. Побудовано загальний алгоритм роботи вебзастосунку з інтеграцією модуля штучного інтелекту для автоматичного формування підзадач. Сукупність цих рішень формує надійну основу для реалізації повнофункціонального інструменту управління особистими завданнями.

# 3 РОЗРОБКА ВЕБЗАСТОСУНКУ

## 3.1 Обґрунтування вибору засобів для реалізації програми

Для реалізації вебзастосунку «CheckIt» було обрано сучасні інструменти, які поєднують у собі високу продуктивність, зручність у використанні та відповідність вимогам до масштабованої багаторівневої системи. Основним критерієм вибору засобів розробки була їхня здатність забезпечити ефективну взаємодію між клієнтською та серверною частинами, підтримку інтеграції зі штучним інтелектом, безпечну роботу з даними та можливість швидкого оновлення інтерфейсу.

Серверна частина вебзастосунку реалізована на базі фреймворку ASP.NET Core з використанням мови програмування C#. Цей інструментарій дозволяє створювати гнучкі, продуктивні вебсервіси з чіткою логічною структурою, високим рівнем безпеки, а також підтримкою REST API для взаємодії з клієнтською частиною. Реалізовано багаторівневу архітектуру з поділом на логічні шари, такі як API, Application, Domain, Infrastructure, що підвищує підтримуваність і дає змогу масштабувати систему без втрати цілісності.

Клієнтська частина реалізована за допомогою JavaScript-бібліотеки React із застосуванням TypeScript для забезпечення типізації та зниження ймовірності помилок на етапі компіляції. React дозволяє створити реактивний, модульний інтерфейс користувача з підтримкою швидкої взаємодії з системою без повного перезавантаження сторінки.

Для побудови стилів обрано Tailwind CSS – утилітарний CSS-фреймворк, який дозволяє швидко компонувати інтерфейс і забезпечує адаптивність дизайну. Стан додатку керується за допомогою Context API, що спрощує передачу даних між компонентами. Для навігації між сторінками використовується бібліотека React Router.

У якості системи управління базами даних обрано PostgreSQL – сучасну реляційну СУБД, яка забезпечує збереження чек-листів, завдань, шаблонів та

інформації про користувачів у структурованому вигляді. Для взаємодії з базою даних у серверній частині використано ORM-бібліотеку Entity Framework Core, яка дозволяє працювати з даними на рівні об'єктів, забезпечуючи зручне виконання CRUD-операцій і автоматичне створення SQL-запитів. Це значно спрощує розробку і підвищує цілісність моделі даних. Комунікація між клієнтом і сервером відбувається через HTTP/HTTPS з обміном даних у форматі JSON.

Для реалізації функціоналу на основі штучного інтелекту було використано модель Google Gemini, яка через зовнішній API дозволяє аналізувати сформульовану мету користувача і генерувати список релевантних підзадач. Отримані результати інтегруються в новий чек-лист автоматично, спрощуючи процес планування.

Такий підхід дозволяє комбінувати традиційне ручне введення задач із інтелектуальним доповненням, що суттєво підвищує зручність використання додатку. У разі необхідності розширення функціональності система допускає інтеграцію кешування з Redis, зокрема для тимчасового зберігання сесій або прискорення запитів.

Для розробки було використано середовище Visual Studio для серверної частини та Visual Studio Code для клієнтської. Контроль версій реалізовано через систему Git із розміщенням репозиторію на GitHub, що дозволяє командну співпрацю, відстеження змін і збереження резервних копій.

Збирання та тестування клієнтської частини здійснюється за допомогою Vite [16] – сучасного інструменту, який забезпечує швидкий старт проєкту, оптимізацію продуктивності та підтримку гарячого перезавантаження під час розробки. Обраний набір технологій дозволяє забезпечити надійну основу для розгортання повнофункціонального вебзастосунку з перспективою подальшого розширення та оптимізації.

### 3.2 Розробка клієнтської частини програми

Клієнтська частина вебзастосунку «CheckIt» реалізована з використанням JavaScript-бібліотеки React, що дозволяє будувати гнучкий компонентний інтерфейс з високою продуктивністю. Кожен елемент сторінки оформлений у вигляді окремого компонента, який відповідає за певну функцію або блок, що спрощує структурування коду, підвищує зручність супроводу та дозволяє багаторазове використання окремих елементів у різних частинах застосунку.

Для забезпечення типобезпеки у розробці інтерфейсу використано мову TypeScript, що дає змогу зменшити кількість помилок на етапі компіляції, а також покращити читаність і підтримуваність коду. Дизайн інтерфейсу побудовано з використанням утилітарного CSS-фреймворку Tailwind CSS [17], який забезпечує адаптивну верстку, гнучке компонування блоків і сучасний вигляд додатку.

Було реалізовано багаторівневу архітектуру клієнтської частини. Архітектура розділена на чотири основні частини:

1. Pages – окремі сторінки, такі як авторизація, реєстрація, панель користувача, сторінка управління списками. Для кожної сторінки визначено специфічну логіку та структуру компонування.

2. Components – багаторазові інтерфейсні елементи, серед яких реалізовано календар, список завдань, навігаційні панелі, форми, перемикачі теми тощо.

3. Context/State Management – реалізовано керування глобальним станом програми за допомогою Context API. Зокрема, забезпечено зберігання статусу авторизації, теми оформлення та поточних користувацьких налаштувань.

4. API Client – створено окремий модуль для надсилання запитів до серверної частини, обробки відповідей та повідомлень про помилки.

Усі сторінки додатку згруповані за логічним призначенням. На рисунку 3.1 представлено стартову сторінку, на якій користувач може перейти до авторизації та переглянути можливості вебсистеми.

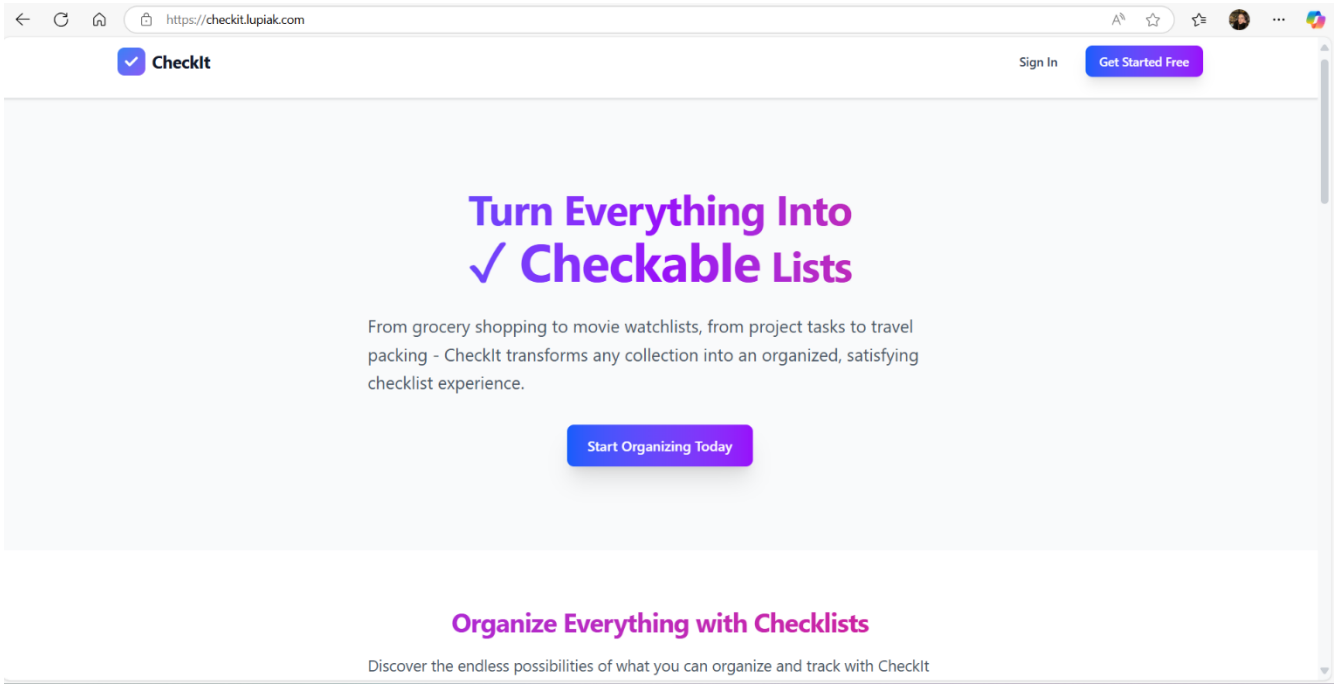



Рисунок 3.1 – Стартова сторінка додатку «CkeckIt»

Сторінка входу до системи реалізована з можливістю автентифікації як через email і пароль, так і через Google OAuth. Було використано компоненти форм з валідацією та динамічним відображенням повідомлень про помилки. Форму авторизації зображено на рисунку 3.2.

Після входу користувач потрапляє до сторінки «Dashboard» (рисунок 3.3), де реалізовано основну робочу панель із переліком майбутніх завдань та інтерактивним календарем. Інтерфейс поділено на два основні блоки – список актуальних задач з дедлайнами та календар з візуалізацією запланованих подій. Кожен пункт списку містить назву завдання та дату виконання.

Було впроваджено динамічне оновлення списків, підсвічування задач за термінами, а також індикатори прогресу, які допомагають оцінити рівень завершеності кожного чекліста (рисунок 3.4). З лівого боку розміщена панель навігації з доступом до активних, архівованих чеклістів, налаштувань облікового запису та кнопкою виходу з системи. Такий підхід дозволяє користувачеві швидко

перемикатися між розділами без додаткового завантаження сторінки, зберігаючи високу швидкодію додатку.

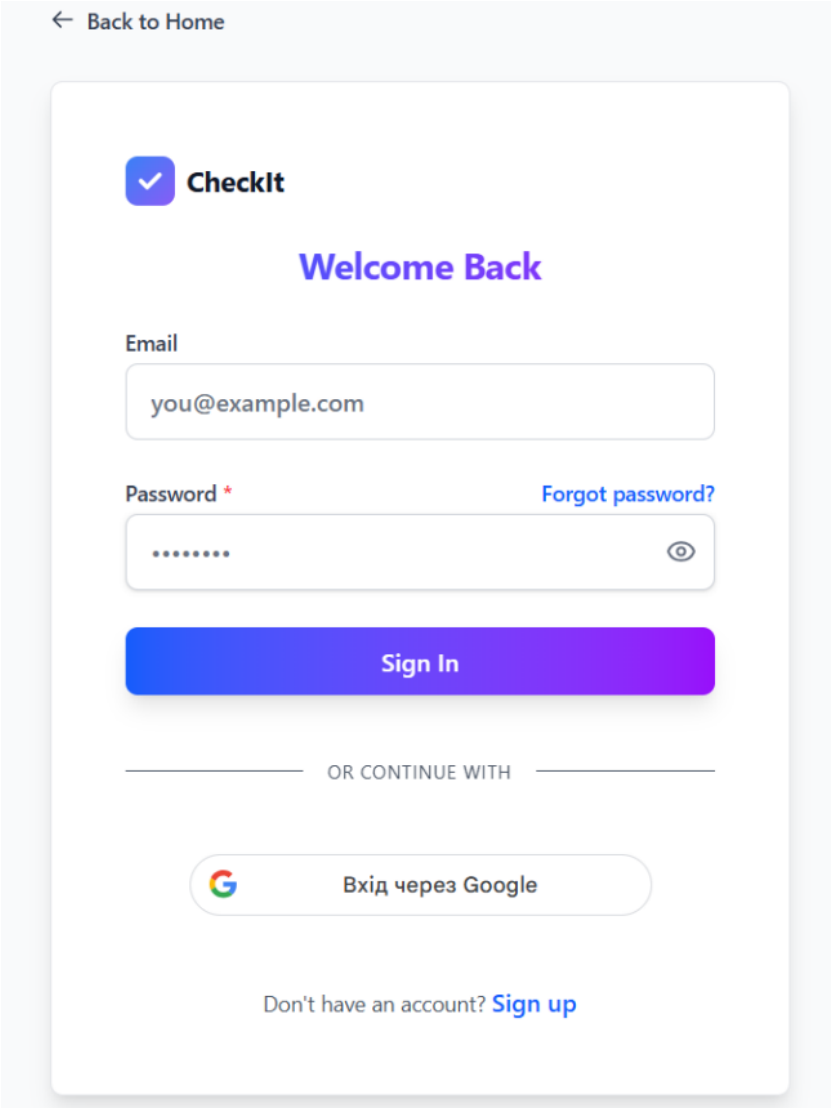



Рисунок 3.2 – Сторінка авторизації

На рисунку 3.5 представлено модальне вікно створення нового чекліста. Було реалізовано інтерфейс для введення заголовка та необов’язкового опису списку завдань.

Окремою функціональністю є можливість автоматичної генерації підзадач на основі введеного заголовка та опису – ця опція активується за допомогою прапорця

«Auto-generate tasks using AI». Після її увімкнення, система надсилає введені дані до модуля штучного інтелекту, який формує набір логічно пов’язаних кроків.

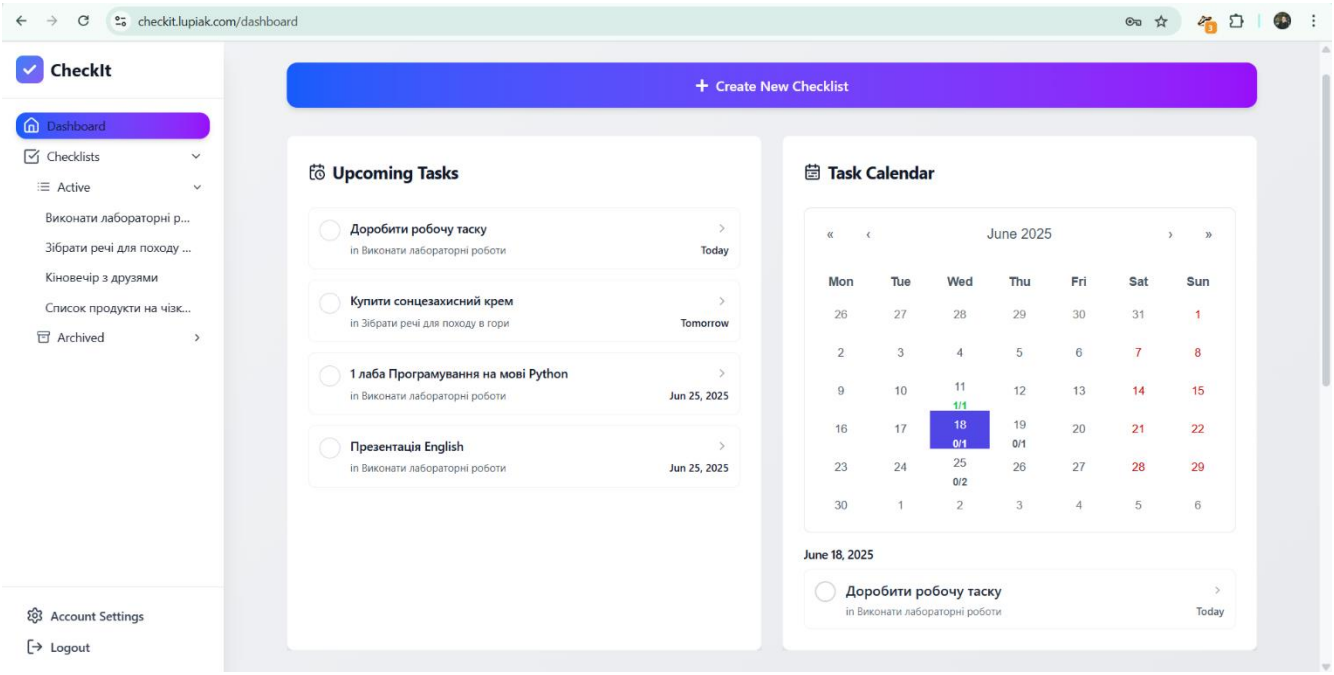



Рисунок 3.3 – Робоча панель з календарем і переліком задач

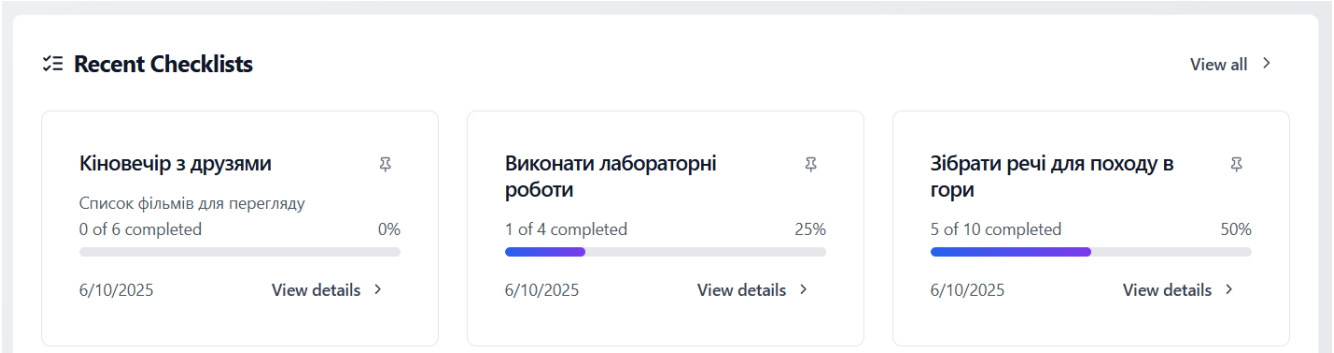



Рисунок 3.4 – Список останніх списків з відображенням прогресу

Користувач має можливість створювати власні чеклісти, наповнюючи їх довільними задачами вручну або на основі автоматичної генерації. На рисунку 3.6 зображено результат використання інтелектуального модуля, який після введення

заголовка «Кіновечір з друзями» сформував відповідний список фільмів для перегляду.

Додатково реалізовано можливість випадкового сортування пунктів (функція Shuffle) для автоматичного визначення порядку перегляду, що може бути корисним у неформальних сценаріях. Також реалізовано підтримку редагування назв пунктів у реальному часі, зміну їхнього порядку, збереження черговості, а також виведення дати створення списку. Кожен пункт можна позначити як виконаний, що дозволяє ефективно відстежувати прогрес у досягненні поставленої мети. Такий рівень функціональності забезпечує високу гнучкість у роботі з будь-яким типом списків.

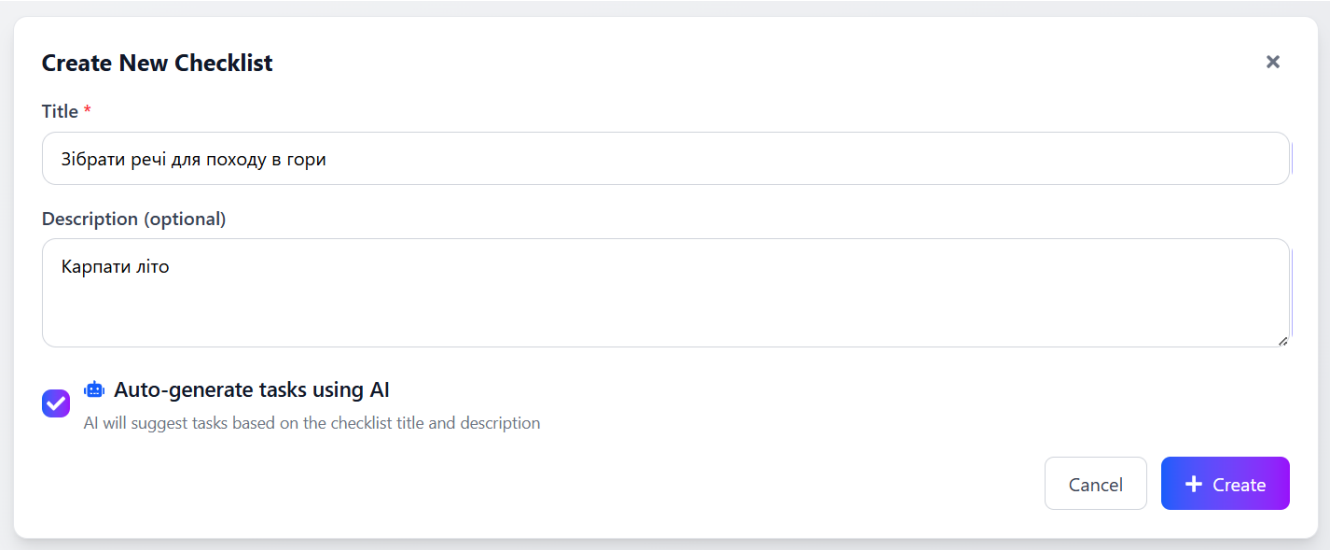



Рисунок 3.5 – Форма створення чекліста

На сторінці перегляду чеклістів було реалізовано функціонал пошуку та фільтрації, що дозволяє користувачеві швидко знаходити потрібні списки за назвою або станом. У правій частині інтерфейсу доступне перемикання між активними, архівованими та всіма списками. Це значно спрощує навігацію та підвищує зручність роботи зі збереженими даними. Окрім цього, впроваджено окрему панель для архівованих чеклістів, яка дозволяє зберігати завершені або тимчасово

неактуальні списки без їх остаточного видалення, забезпечуючи доступ до історії виконаних завдань. Функції пошуку та архівування списку представлено на риссунку 3.7.

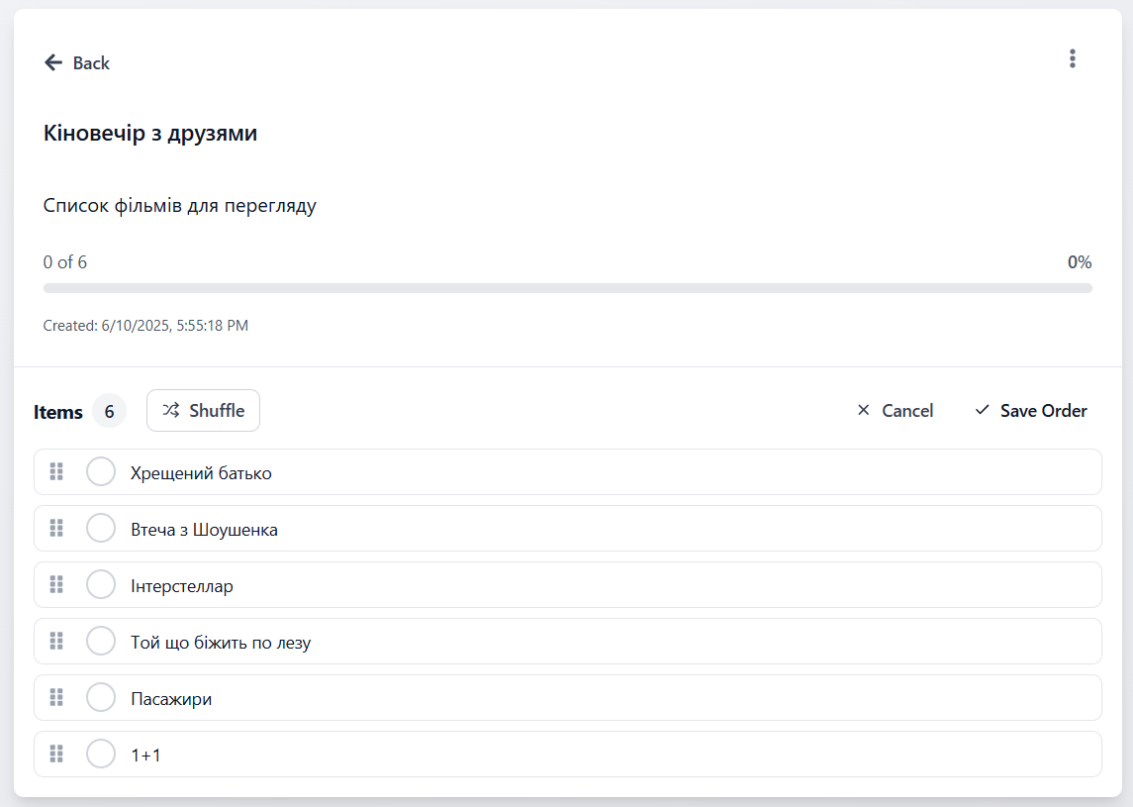



Рисунок 3.6 – Результат автоматичного створення списку

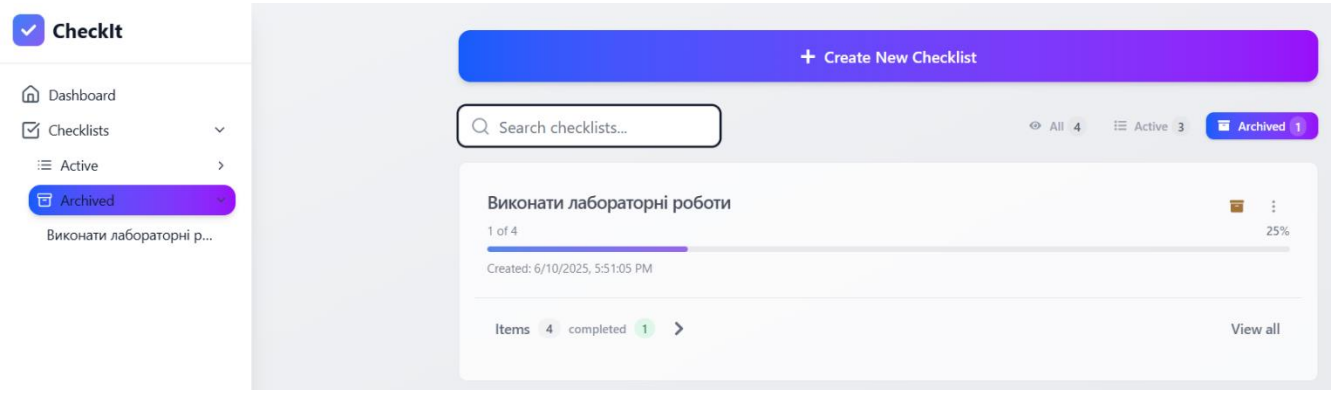



Рисунок 3.7 – Функції пошуку та перегляду архівованих списків

На сторінці налаштувань облікового запису було реалізовано інтерфейс персоналізації, що дає змогу користувачеві змінювати зовнішній вигляд системи. Доступні три режими відображення – світла тема, темна та системна, які одразу застосовуються до інтерфейсу без перезавантаження сторінки. Також реалізовано зміну пароля, що дозволяє встановити або оновити пароль доступу до облікового запису. На рисунку 3.8 зображено сторінку налаштувань облікового запису з демонстрацією зміни кольорової теми.

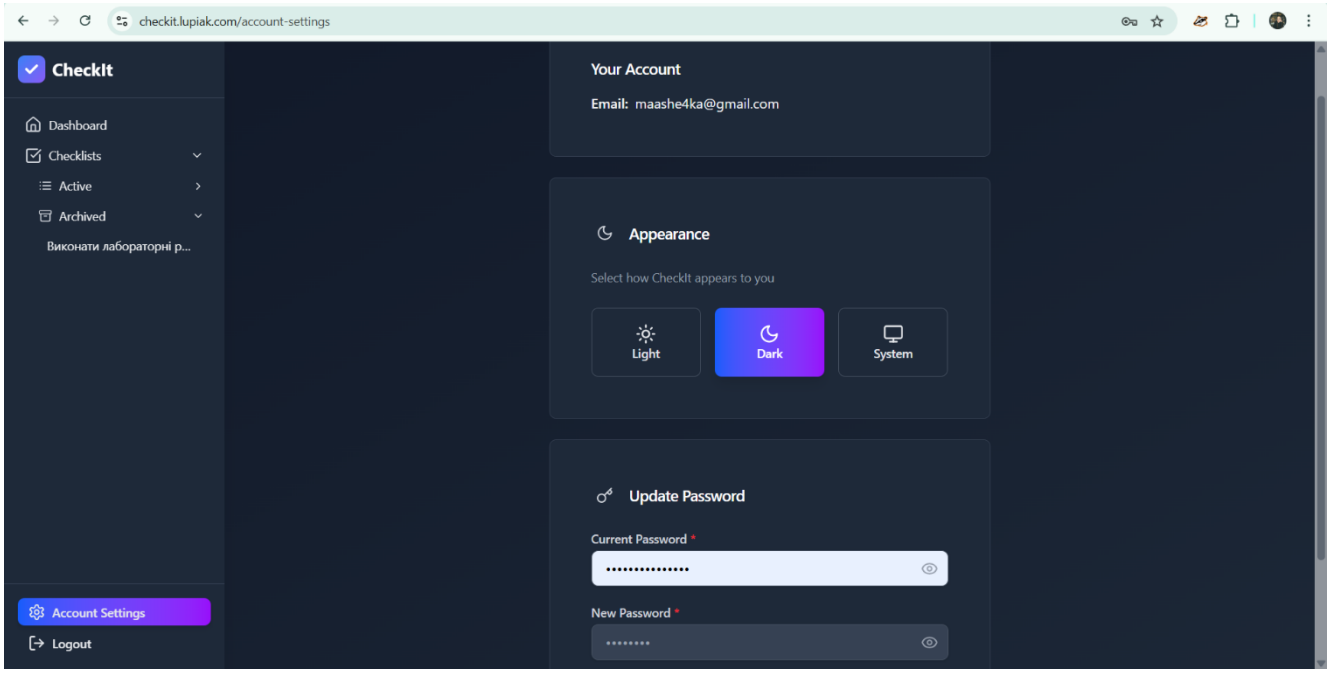



Рисунок 3.8 – Сторінка налаштувань облікового запису

На рисунку 3.9 показано форму встановлення пароля, яка є частиною налаштувань облікового запису. Було реалізовано перевірку складності введеного пароля відповідно до визначених вимог безпеки. Зокрема, система очікує, що пароль буде не коротшим за 8 символів і міститиме принаймні одну велику літеру, одну малу літеру та одну цифру. Кожна з цих умов автоматично перевіряється у реальному часі, що допомагає користувачу одразу бачити, які вимоги виконані.

Завдяки використанню Tailwind CSS було реалізовано адаптивність до мобільних пристроїв. Компоненти автоматично перебудовуються залежно від ширини екрана, що забезпечує комфортну роботу з мобільного телефону чи планшета. На рисунку 3.10 представлено мобільну версію інтерфейсу головної сторінки.

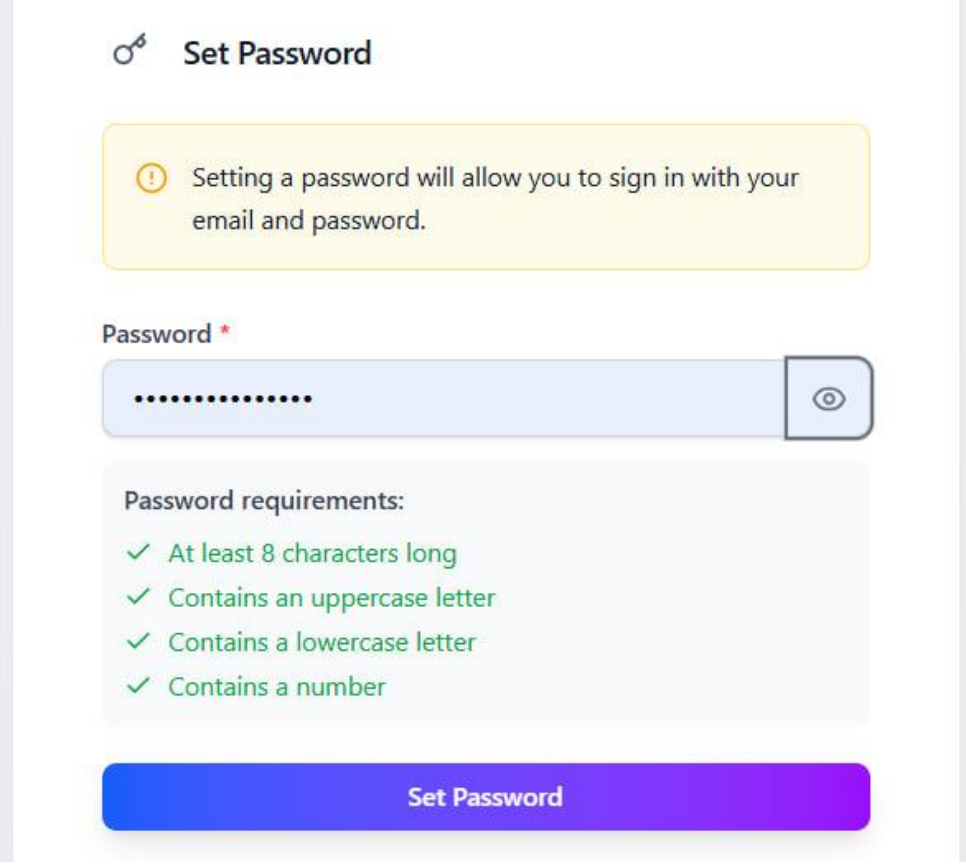



Рисунок 3.9 – Форма встановлення та зміни паролю

Таким чином, користувач отримує повноцінний доступ до всього функціоналу вебзастосунку незалежно від пристрою, яким він користується. Реалізована адаптивність дозволяє зберігати зручність навігації, читабельність вмісту та стабільність роботи інтерфейсу, що особливо важливо для користувачів, які планують завдання «на ходу» або користуються сервісом у мобільному середовищі.

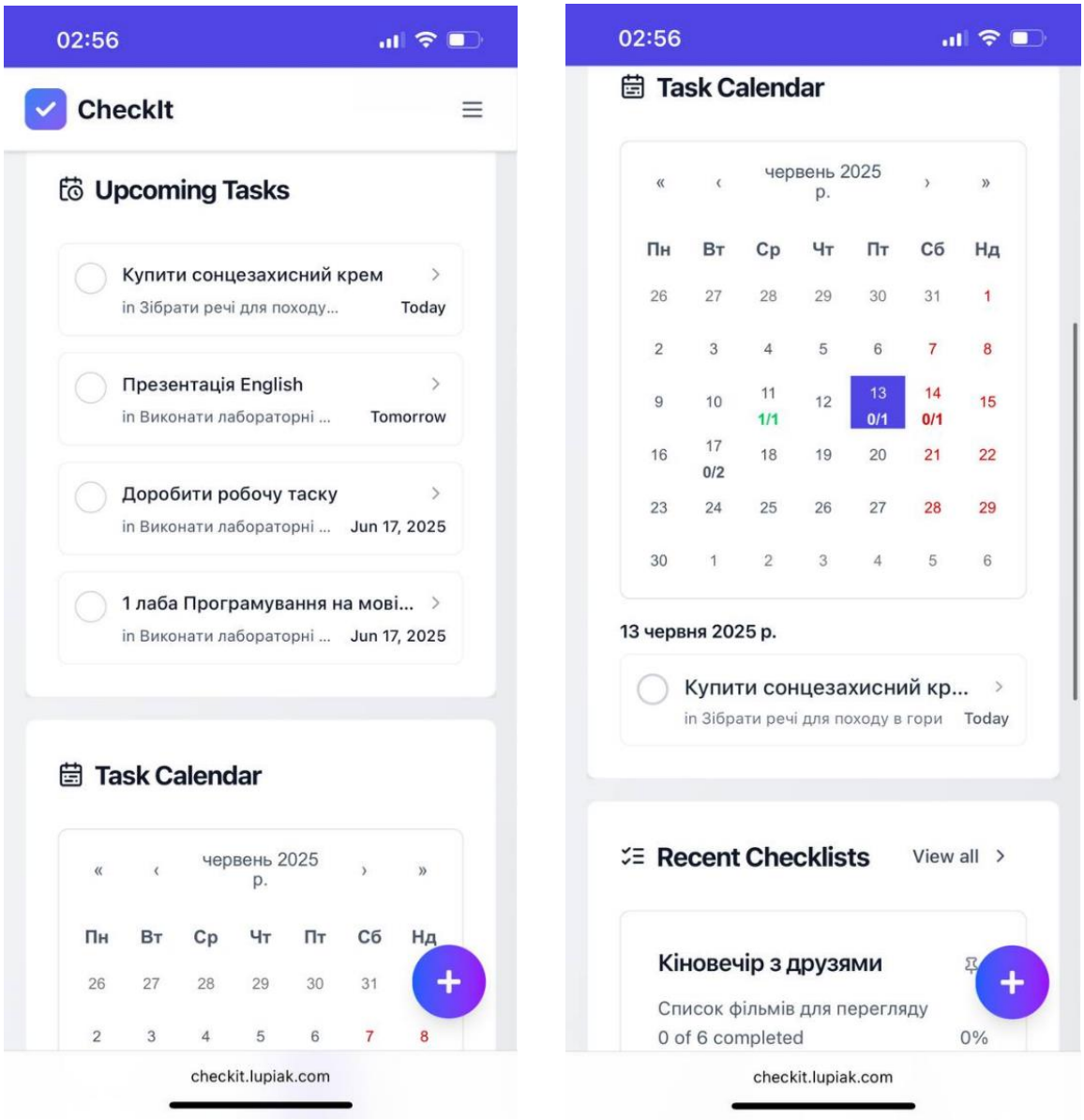



Рисунок 3.10 – Адаптована версія інтерфейсу

Клієнтська частина додатку «CheckIt» є повноцінною SPA (Single Page Application) [18], яка поєднує гнучку архітектуру, зручний інтерфейс та інтеграцію з інтелектуальним модулем, забезпечуючи швидку й ефективну взаємодію користувача з системою.

### 3.3 Розробка серверної частини застосунку

Серверна частина вебзастосунку CheckIt реалізована на основі сучасного фреймворку ASP.NET Core із використанням архітектурного підходу Domain-Driven Design (DDD), що забезпечує чіткий розподіл обов'язків між окремими рівнями системи. У структурі проєкту виділено кілька ключових шарів: API-рівень, Application-рівень, Domain-рівень та Infrastructure-рівень, кожен з яких виконує окрему функцію в загальній логіці програми.

API-рівень відповідає за опис маршрутів і обробку HTTP-запитів. У рамках цього рівня було реалізовано набір ендпоінтів для автентифікації та управління обліковим записом користувача. Було реалізовано функціонал реєстрації, входу за допомогою логіна й пароля або через обліковий запис Google, зміни пароля, перегляду профілю та прив'язки акаунту до Google. Усі запити проходять через систему авторизації, яка базується на JWT-токенах і забезпечує захист персональних даних. На рисунку 3.1 наведено відповідні ендпоінти, які згруповані в Swagger-інтерфейсі [19] для зручності тестування і візуалізації API.

Також було розроблено модуль ChecklistEndpoints, який містить всі ендпоінти для взаємодії з чеклістами. Для кожної операції, наприклад отримання списку, створення, редагування, видалення тощо, визначено окремий маршрут і відповідний обробник. На рисунку 3.2 зображено інтерфейс Swagger із ендпоінтами для роботи з чеклістами.

Application-рівень реалізує бізнес-логіку та інтерфейси сервісів. Тут знаходяться класи сервісів, які обробляють запити від API, здійснюють валідацію вхідних моделей, формують відповіді, обробляють помилки та викликають методи доменної логіки. У коді реалізовано сервіс IChecklistService, який надає основний функціонал для роботи з чеклістами – створення, редагування, архівування, додавання пунктів, зміна дедлайнів, перестановка порядку та інше.

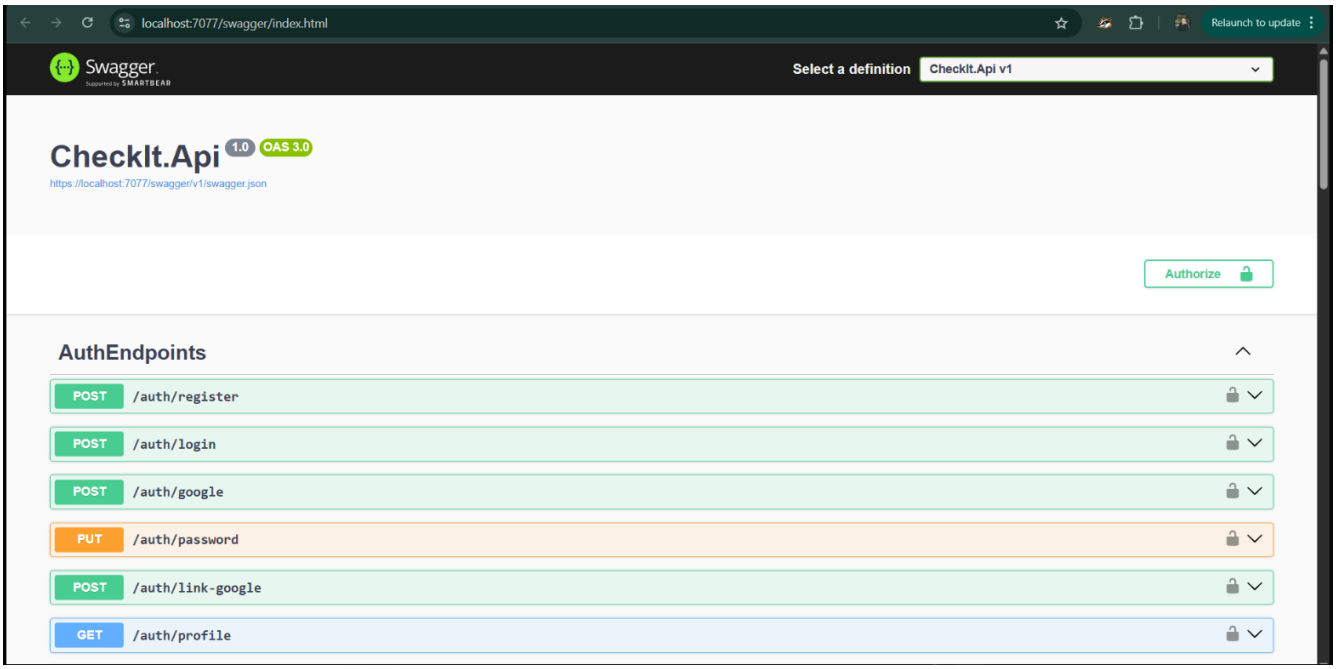



Рисунок 3.1 – Ендпоінти автентифікації

Domain-рівень включає опис сутностей та доменних подій, що забезпечують узгодженість бізнес-логіки. У цьому шарі описано правила, за якими система повинна працювати незалежно від зовнішніх джерел зокрема, перевірку прав доступу, допустимих форматів даних та логіку виконання операцій.

Infrastructure-рівень реалізує взаємодію з базою даних PostgreSQL через ORM-бібліотеку Entity Framework Core, а також містить реалізації репозиторіїв та інтеграції з зовнішніми сервісами, зокрема модулем штучного інтелекту для автоматичної генерації задач. Тут розташовані класи, що відповідають за збереження, оновлення та отримання даних із таблиць бази, а також обробку транзакцій. Також у цьому рівні реалізовано шаблони доступу до даних, які забезпечують гнучкість і зручність розширення функціоналу без порушення загальної структури. Завдяки чіткому розділенню обов’язків обидва рівні сприяють підтримуваності коду та полегшують впровадження нових можливостей у систему.

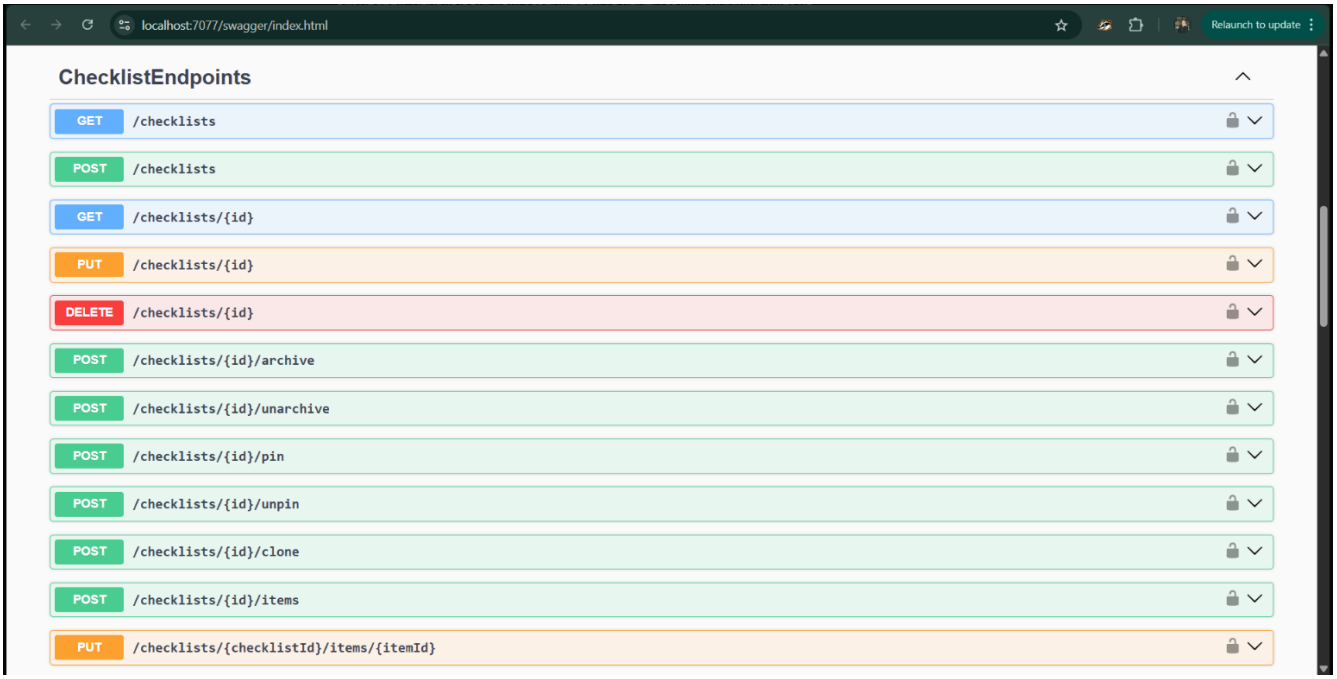



Рисунок 3.2 – Ендпоінти роботи з чеклістами

На прикладі модуля ChecklistEndpoints можна побачити реалізацію REST API, яке підтримує основні CRUD-операції. Користувач має змогу створити чекліст (POST /checklists), отримати список усіх своїх чеклістів (GET /checklists), змінити назву або опис (PUT /checklists/{id}), видалити список (DELETE /checklists/{id}) або надіслати його в архів. Крім того, реалізовано функціонал дублювання списків (/clone), закріплення у верхній частині інтерфейсу (/pin) та розархівування. Для роботи з окремими пунктами чекліста існують окремі маршрути, які дозволяють додавати елементи, змінювати їх статус виконання, оновлювати дату завершення та переставляти порядок пунктів.

Вся логіка супроводжується чітким обробленням результатів. При помилках користувач отримує відповідні статуси, наприклад, 401 Unauthorized, 404 Not Found або 400 Bad Request, а у випадку успіху повертаються дані у форматі JSON. Такий підхід забезпечує надійність API та спрощує його використання з боку клієнтської частини.

Таким чином, серверна частина «CheckIt» побудована на принципах чистої архітектури з чіткою відповідальністю кожного рівня. Завдяки цьому досягається висока підтримуваність коду, можливість легкого масштабування, безпечна робота з базами даних, а також можливість інтеграції з інтелектуальними модулями та сторонніми сервісами.

### 3.4 Висновки

У межах розділу було описано повний цикл розробки вебзастосунку «CheckIt», що включає обґрунтування вибору інструментів, розробку клієнтської та серверної частин, а також побудову логіки взаємодії з інтелектуальним модулем. Обрано сучасні технології, такі як ASP.NET Core, React, TypeScript, PostgreSQL та Tailwind CSS, що забезпечили високу продуктивність, масштабованість та адаптивність системи.

Розроблено багаторівневу архітектуру, що розділяє функціональність на логічні шари та сприяє підтримуваності проєкту. Інтерфейс користувача побудований на основі SPA-підходу, що дозволяє досягти високої швидкодії та зручності використання. Реалізовано механізм інтеграції зі штучним інтелектом для автоматичного формування підзадач, що підвищує ефективність планування.

Сукупність реалізованих функцій формує надійну платформу для подальшого розширення системи та впровадження нових можливостей управління персональними завданнями.

# 4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Тестування вебсистеми

Тестування є необхідною складовою професійної розробки програмного забезпечення, що дозволяє гарантувати його коректну роботу, відповідність вимогам та надійність у реальному використанні. У проєкті «CheckIt» було застосовано комплексний підхід до перевірки якості, який включає юніт-тестування, інтеграційне тестування, а також ручне тестування інтерфейсу користувача.

Юніт-тестування спрямоване на перевірку окремих компонентів у відриві від загальної системи. Це дозволяє виявити помилки у внутрішній логіці окремих класів, не залежачи від зовнішніх сервісів чи інтерфейсів. У межах «CheckIt» юніт-тести охоплюють доменні сутності (користувачі, чеклісти, елементи списків, токени), а також репозиторії, які відповідають за збереження даних у базі. Такі тести забезпечують правильність створення об’єктів, зміну станів, валідацію вхідних параметрів і винесення помилок.

Інтеграційне тестування перевіряє, як окремі частини системи працюють разом – наприклад, як HTTP-запити обробляються контролерами, як сервіси взаємодіють з базою даних, як реалізована бізнес-логіка. Цей тип тестування критично важливий для багаторівневої архітектури, яка реалізована в «CheckIt», оскільки дозволяє переконатися, що вся вертикаль – від користувацького запиту до збереження в базі – працює стабільно [20].

Найважливішою частиною застосунку є функціональність управління чеклістами, реалізована у вигляді сервісу. Тести модулю ChecklistServiceTests охоплюють 27 кейсів і перевіряють створення, оновлення, архівацію, зміну статусу елементів, сортування, а також валідацію запитів. Серед прикладів – тест, який перевіряє, чи інтеграція зі штучним інтелектом правильно додає згенеровані підзадачі до нового списку. Інші тести перевіряють, наприклад, що не можна

оновити чекліст із невалідним ID або з доступом неавторизованого користувача. На рисунку 4.1 представлено успішне проходження усіх перевірок бізнес-логіки чеклістів.

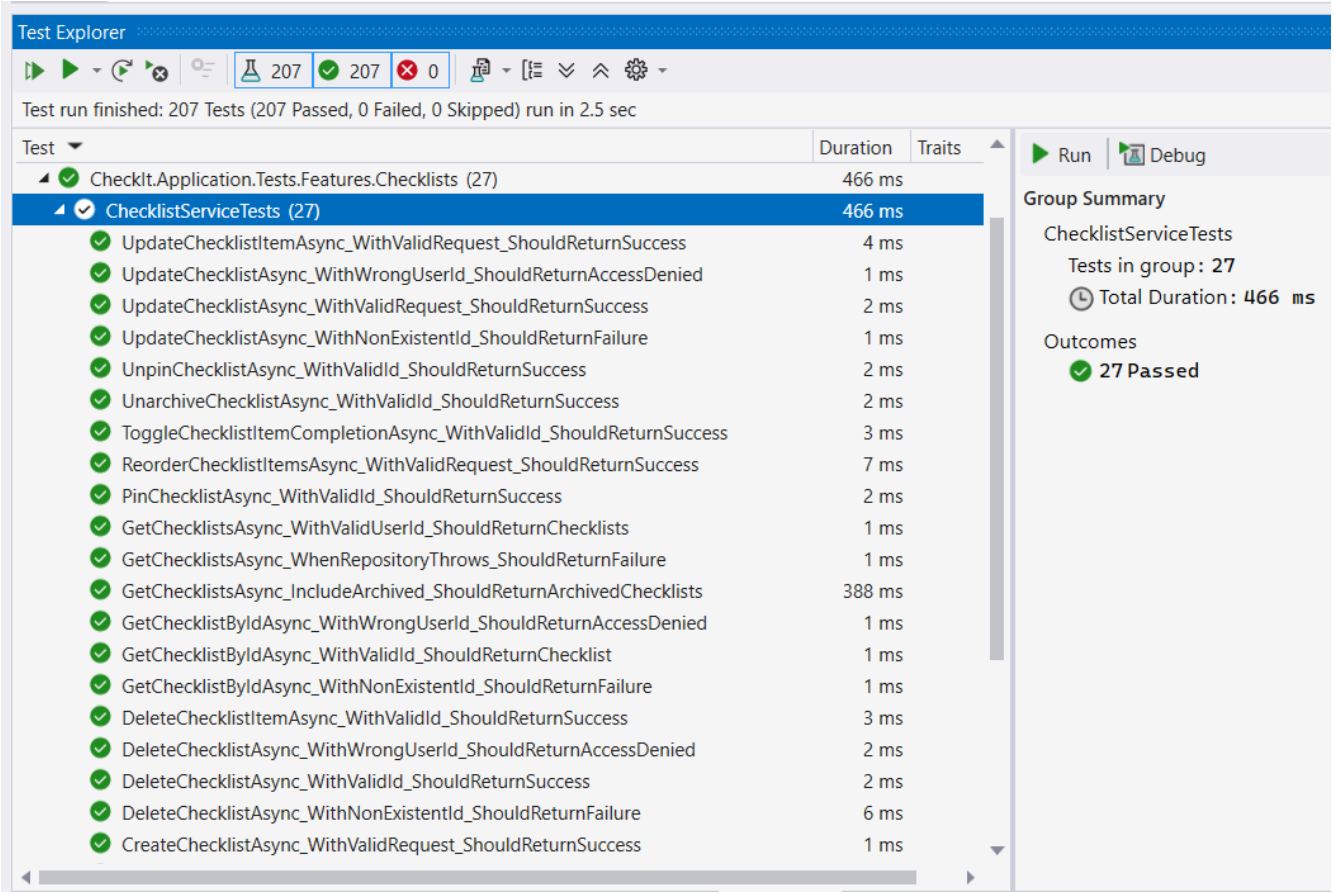



Рисунок 4.1 – Тести для перевірки бізнес-логіки для роботи з чеклістами

Для тестування REST API було розроблено 31 тест (рисунок 4.2) , які охоплюють додавання, редагування, видалення, оновлення термінів, архівацію та зміну статусу пунктів чекліста.

Наприклад, UpdateChecklistItem_WithInvalidRequest_ShouldReturnBadRequest перевіряє, чи система правильно реагує на некоректні параметри. Інші тести гарантують, що користувач отримає 404 у разі спроби змінити неіснуючий чекліст або пункт. Ці тести є ключовими для перевірки коректної роботи фронтенду з сервером через REST API, а також для захисту від некоректного вводу.

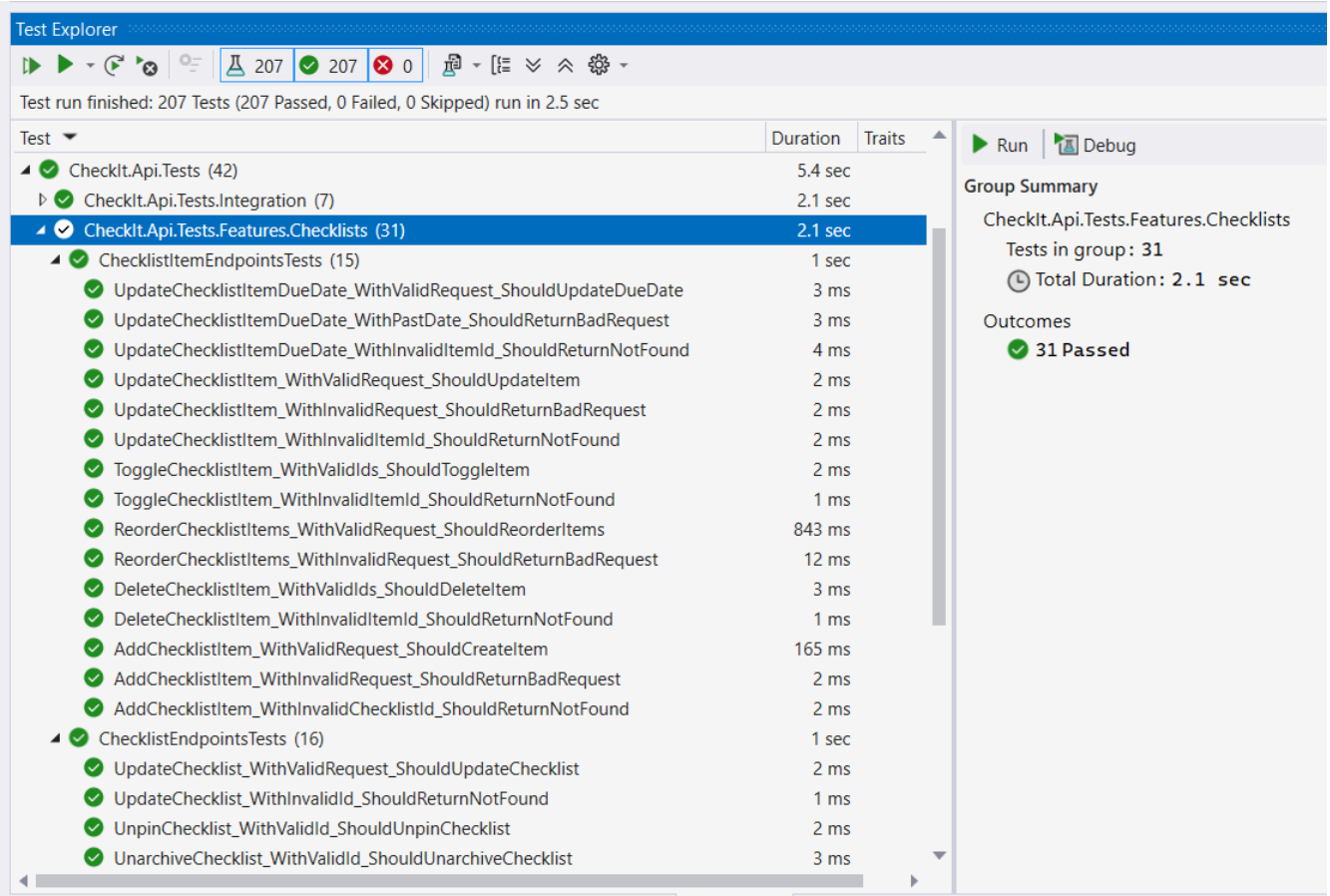



Рисунок 4.2 – Тести для перевірки роботи API

Безпечна та стабільна автентифікація є основою захисту користувацьких даних. У межах модуля AuthServiceTests було протестовано 15 сценаріїв, включаючи реєстрацію з новим або існуючим email, логін із дійсними й хибними даними, зміну пароля, генерацію токенів відновлення доступу та Google-автентифікацію. усі тести автентифікації завершилися успішно, що підтверджує надійність механізму доступу.

Також було розроблено тестові сценарії повного циклу створення, оновлення, дублювання та видалення чеклістів через HTTP (рисунок 4.3). Вони імітують реальні дії користувача та перевіряють, чи збережені дані відповідають очікуваним результатам. Наприклад, CloneChecklist_ShouldCreateExactCopy перевіряє, чи

створено точну копію на основі оригіналу. Ці тести перевіряють повну взаємодію всіх шарів – від API до збереження в PostgreSQL

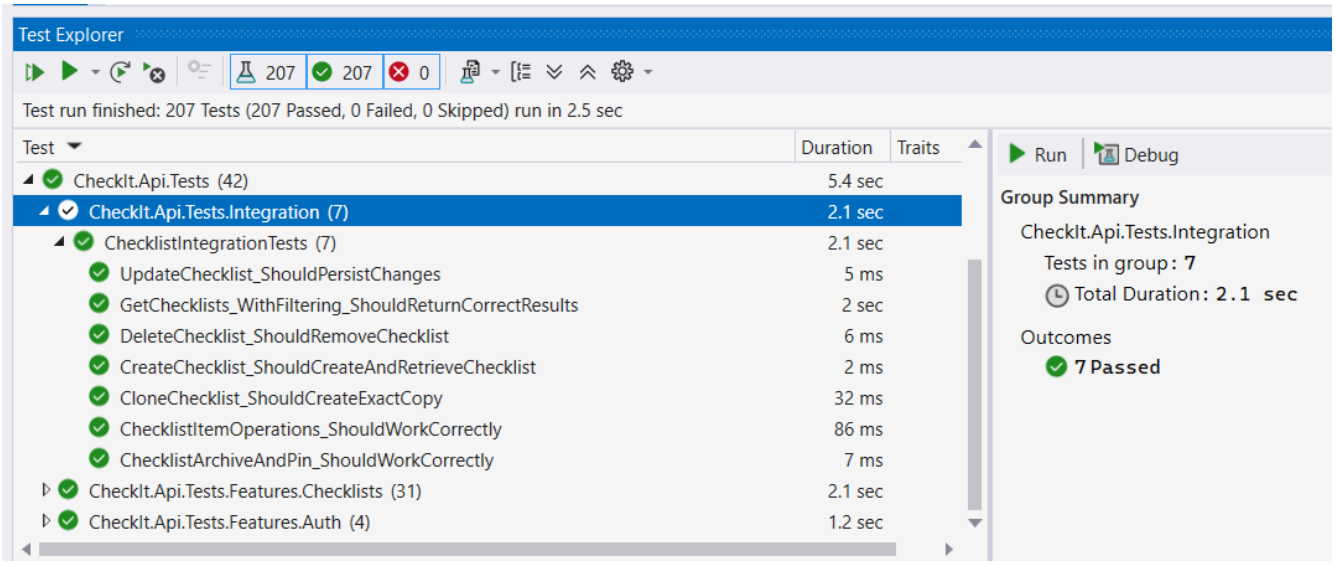



Рисунок 4.3 – Інтеграційні тести для перевірки повного циклу роботи

Окрім автоматизованих тестів, проводилося також ручне тестування клієнтської частини. Усі ключові сторінки – авторизація, панель задач, створення чеклістів, інтерфейс налаштувань – були перевірені на коректність відображення, стабільність взаємодії, адаптивність на мобільних пристроях та відповідність очікуваній поведінці. Особливу увагу приділено роботі модулю ШІ – автогенерація підзадач, їх збереження та редагування перевірялися вручну.

В результаті у межах тестування було реалізовано 207 автоматизованих тестів, усі з яких успішно виконано. Високий рівень покриття функціональності, перевірка кожного шару системи – від логіки домену до зовнішнього API – у поєднанні з ручним тестуванням інтерфейсу забезпечують високу якість і стабільність роботи вебзастосунку «CheckIt». Такий підхід дозволяє з упевненістю перейти до етапу розгортання, масштабування та використання застосунку в реальному середовищі.

## 4.2 Розробка інструкції користувача

Для забезпечення зручного використання вебзастосунку «CheckIt» була створена інструкція користувача, яка дозволяє швидко ознайомитися з можливостями системи та ефективно почати з нею працювати. Основна мета інструкції – зробити взаємодію з програмою інтуїтивно зрозумілою навіть для користувачів, які вперше мають справу з подібними інструментами.

Робота з «CheckIt» починається зі входу до системи. На головній сторінці користувачу доступні дві форми авторизації: стандартна через електронну пошту та пароль, а також зручна авторизація через Google-акаунт. У випадку, якщо обліковий запис ще не створено, можна зареєструватися, заповнивши просту форму. Після успішного входу користувач автоматично потрапляє на робочу панель.

Інтерфейс побудований таким чином, щоб усі ключові елементи були зібрані в одному місці. Центральне місце займає список створених чеклістів, у якому кожен пункт містить назву, дату створення, індикатор прогресу та позначки про виконані завдання. Праворуч розташовано календар, що допомагає візуально оцінити розподіл завдань у часі.

Створити новий список дуже просто: достатньо натиснути кнопку «Створити чекліст» і заповнити поля з назвою та, за бажанням, описом. Особливістю системи є функція автоматичної генерації підзадач на основі введеної теми – для цього достатньо активувати відповідний прапорець. У результаті користувач одразу отримує структурований список логічно пов’язаних кроків, згенерованих модулем штучного інтелекту. Це значно прискорює планування та зменшує навантаження на користувача.

Редагування чеклістів відбувається напряму в інтерфейсі. Можна змінювати назви пунктів, позначати їх як виконані, змінювати порядок або видаляти. Також доступна функція випадкового сортування задач, яка може бути корисною для неформальних чи творчих завдань. Усі зміни зберігаються автоматично.

Щоб спростити роботу з великою кількістю записів, реалізовано пошук та фільтрацію. Користувач може швидко знайти потрібний список за назвою або переглянути лише активні чи архівовані чеклісти. Завершені або неактуальні списки можна переміщати в архів, де вони зберігатимуться без остаточного видалення.

У системі також передбачено налаштування облікового запису. Користувач має можливість змінити тему інтерфейсу (світла, темна або системна), змінити пароль та переглянути основну інформацію профілю. Усі ці зміни застосовуються миттєво без потреби перезавантаження сторінки.

Інтерфейс розроблено з урахуванням адаптивності, тому він коректно працює як на комп’ютерах, так і на мобільних пристроях. Зміна розміру екрана не впливає на функціональність – усі елементи зберігають доступність і зручність використання.

Після завершення роботи із системою користувач може вийти зі свого акаунта через кнопку «Вийти», розміщену в боковому меню. Це дозволяє завершити сесію та захистити персональні дані.

Розроблена інструкція дозволяє користувачу швидко розібратися в основних можливостях системи та використовувати її повний функціонал без додаткових пояснень. Такий підхід підвищує загальну зручність користування та сприяє позитивному досвіду взаємодії з програмою.

### 4.3 Висновки

У межах четвертого розділу було проведено всебічне тестування вебзастосунку «CheckIt» з метою перевірки його стабільності, надійності та відповідності функціональним вимогам. Завдяки реалізованим юніт- та інтеграційним тестам вдалося забезпечити високе покриття коду і перевірити критичні ділянки логіки, такі як управління чеклістами, автентифікація користувачів та обробка REST API-запитів. Загалом виконано 207 автоматизованих тестів, жоден з яких не завершився з помилкою, що свідчить про стабільну роботу

системи на всіх рівнях. Додатково проведене ручне тестування інтерфейсу дозволило переконатися в коректності візуального відображення, зручності навігації та працездатності функцій у браузерах і на мобільних пристроях. Окрім цього, була створена інструкція користувача, яка описує основні етапи взаємодії з вебзастосунком – від авторизації до створення й редагування чеклістів.

# ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено веб-систему «CheckIt» для створення та керування чеклістами з підтримкою інтелектуальної генерації завдань. Розроблений застосунок дозволяє користувачам реєструватися, створювати списки задач, автоматично генерувати підзадачі за допомогою штучного інтелекту, відстежувати прогрес виконання завдань, працювати з архівами, а також персоналізувати інтерфейс. Усі дані користувачів зберігаються у реляційній базі PostgreSQL, а взаємодія між клієнтською та серверною частинами реалізована через REST API. Бакалаврську кваліфікаційну роботу виконано відповідно до методичних вказівок [21].

У процесі розробки було використано мову програмування C# для серверної частини (на базі ASP.NET Core), JavaScript із бібліотекою React і TypeScript – для клієнтської частини, а також бібліотеки Tailwind CSS, Entity Framework Core, Context API, Google OAuth 2.0, Git та інші сучасні інструменти веброзробки.

У першому розділі було проведено аналіз актуальних рішень для створення списків завдань, розглянуто особливості існуючих інструментів та обґрунтовано потребу в новому застосунку з інтеграцією штучного інтелекту. Визначено мету та задачі дослідження, сформульовано вимоги до системи.

У другому розділі описано архітектуру системи, побудовано інформаційну модель, визначено структуру даних, сценарії використання та ключові модулі, які відповідають за логіку керування задачами, користувачами, шаблонами та генерацією чеклістів.

У третьому розділі обґрунтовано вибір інструментів розробки. Було створено клієнтську частину на базі React, а серверну – з використанням багаторівневої архітектури ASP.NET Core. Реалізовано інтеграцію з моделлю штучного інтелекту для автоматичного формування підзадач. Розроблено базу даних та інтерфейс користувача з адаптивною верстkою.

У четвертому розділі проведено тестування системи, включно з юніт- та інтеграційними тестами, які охоплюють критично важливі частини – автентифікацію, роботу з чеклістами, REST API та бізнес-логіку. Також виконано ручне тестування інтерфейсу. Загалом успішно пройдено 207 автоматизованих тестів. Розроблено інструкцію користувача, що дозволяє легко опанувати функціонал системи.

Таким чином, усі поставлені завдання було реалізовано, а розроблений вебзастосунок «CheckIt» повністю відповідає вимогам, поставленим на початку роботи. Система є готовим інструментом для подальшого впровадження, використання або масштабування.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Луп’як М.Д. Порівняльний аналіз методів аутентифікації у вебзастосунках / М.Д. Луп’як, Г.О. Черноволик // Матеріали LIV Всеукраїнської науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії, Вінниця, 2025 [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24415/20166

2. Додатки для тайм-менеджиенту [Електронний ресурс] – Режим доступу до ресурсу: https://wonder-web.com.ua/en/blog/marketing-articles/top-10-program-dlya-efektivnogo-tajm-menedzhmenta-i-planuvannya/

3. Todoist [Електронний ресурс] – Режим доступу до ресурсу: https://www.todoist.com/

4. Microsoft To Do [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/de-de/microsoft-365/microsoft-to-do-list-app

5. Trello [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlasian.com/de/soft-ware/trello

6. Notion [Електронний ресурс] – Режим доступу до ресурсу: https://www.notion.com/help/guides/what-is-notion

7. ASP .NET Core [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core

8. JWT-токени [Електронний ресурс] – Режим доступу: https://supertokens.com/blog/what-is-jwt

9. Adam Boduch, Roy Derks. React and React Native: A complete hands-on guide to modern web and mobile development with React.js – 2020. – 526 с.

10. PostgreSQL [Електронний ресурс] – Режим доступу: https://www.postgresql.org/about/

11. HTTP-протокол [Електронний ресурс] – Режим доступу: https://hostiq.ua/wiki/ukr/http-https/

12. RESTful API [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/restful-api/

13. Clean architecture [Електронний ресурс] – Режим доступу: https://codilime.com/blog/clean-architecture/

14. ORM [Електронний ресурс] – Режим доступу: https://www.youstable.com/uk/блозі/що-таке-orm-у-програмуванні/

15. User Experience Design [Електронний ресурс] – Режим доступу: https://www.interaction-design.org/literature/topics/ux-design?srsltid=AfmBOopBU29-G13adgoT6zVV7wGrTGW6WfqjVF52XKqk6esjPMSQSzpjJ

16. Vite [Електронний ресурс] – Режим доступу: https://www.corbado.com/blog/vite-react

17. Tailwind CSS [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/css/introduction-to-tailwind-css/

18. Single Page Application [Електронний ресурс] – Режим доступу: https://www.sana-commerce.com/e-commerce-terms/what-is-a-single-page-application/

19. Swagger [Електронний ресурс] – Режим доступу: https://blog.hubspot.com/website/what-is-swagger

20. Аттіла Ковач. Modern Software Testing Techniques: A Practical Guide for Developers and Testers. Нью-Йорк: Apress, 2024. 266с

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О.Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

# ДОДАТКИ

## Додаток А – Технічне завдання

61

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«_25_» __березня__ 2025 р.

**Технічне завдання**
**на бакалаврську кваліфікаційну роботу «Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React»**
**за спеціальністю**
**121 – Інженерія програмного забезпечення**

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Черноволик Г.О.
«_24_» __березня__ 2025 р.

Виконала:
студентка гр. 4ПІ-21б Луп'як М.Д.
«_24_» __березня__ 2025 р.

м. Вінниця – 2025 рік

**1. Найменування та галузь застосування**

Бакалаврська кваліфікаційна робота: «Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React». Галузь застосування – вебтехнології.

**2. Підстава для розробки.**

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

**3. Мета та призначення розробки.**

Метою роботи є підвищення ефективності особистого тайм-менеджменту користувачів шляхом створення інтегрованого вебзастосунку, який забезпечує цілісне зберігання даних та автоматичну декомпозицію цілей за допомогою ШІ.

**4. Вихідні дані для проведення НДР**

1. Task Management Software Trends Analysis Report [Електронний ресурс] – Режим доступу до ресурсу: https://www.grandviewresearch.com/industry-analysis/task-management-software-market

2. Building a To-Do App with React and TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://blog.logrocket.com/building-a-todo-app-with-react-typescript/

3. ASP.NET Core Fundamentals [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/

4. Gemini API documentation by Google [Електронний ресурс] – Режим доступу до ресурсу: https://ai.google.dev/docs

**5. Технічні вимоги**

Вхідні дані – назва чекліста, опис, спосіб формування задач, дедлайни.

Вихідні дані – згенеровані або введені задачі, інформація про виконання.

**6. Конструктивні вимоги.**

Інтерфейс програми поовинен відповідати естетичним та ергономічним

вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

**7. Перелік технічної документації, що пред'являється по закінченню робіт:**

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

**8. Вимоги до рівня уніфікації та стандартизації**

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

**9. Стадії та етапи розробки**

| № з/п | Назва етапів бакалаврської кваліфікаційної роботи | Строк виконання етапів роботи |
|---|---|---|
| 1 | Аналіз та постановка задачі | 25.03.25 - 05.04.25 |
| 2 | Проектування архітектури та розробка алгоритмів програми | 06.04.25 - 18.04.25 |
| 3 | Обґрунтування вибору засобів для реалізації системи | 19.04.25 - 21.04.25 |
| 4 | Розробка вебзастосунку | 22.04.25 - 17.05.25 |
| 5 | Тестування програмного забезпечення | 18.05.25 - 25.05.25 |
| 6 | Оформлення матеріалів до захисту БКР | 26.05.25 - 30.05.25 |

**10. Порядок контролю та прийняття.**

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

**Додаток Б – Протокол перевірки на наявність текстових запозичень**



ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core і React»

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7 %

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:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений

Керівник _____ Черноволик Г.О. к.т.н., доцент каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Луп'як М.Д.
(підпис) (прізвище, ініціали)

# Додаток В – Лістинг програми

```
using CheckIt.Application.Features.Checklists.Interfaces;
using CheckIt.Application.Features.Checklists.Models;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace CheckIt.Api.Endpoints;

public static class ChecklistEndpoints
{
    public static void MapChecklistEndpoints(this IEndpointRouteBuilder app)
    {
        var group = app.MapGroup("/checklists")
            .RequireAuthorization();

        // Checklist operations
        group.MapGet("/", GetAllChecklists)
            .WithName("GetChecklists")
            .WithDescription("Get all checklists for the authenticated user")
            .Produces<List<ChecklistResponse>>(StatusCodes.Status200OK)
            .Produces(StatusCodes.Status401Unauthorized)
            .Produces(StatusCodes.Status500InternalServerError)
            .WithOpenApi();

        group.MapGet("/{id}", GetChecklistById)
            .WithName("GetChecklist")
            .WithDescription("Get a specific checklist by ID")
            .Produces<ChecklistResponse>(StatusCodes.Status200OK)
            .Produces(StatusCodes.Status401Unauthorized)
            .Produces(StatusCodes.Status404NotFound)
```

```
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPost("/", CreateChecklist)
    .WithName("CreateChecklist")
    .WithDescription("Create a new checklist")
    .Produces<ChecklistResponse>(StatusCodes.Status201Created)
    .Produces(StatusCodes.Status400BadRequest)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPut("/{id}", UpdateChecklist)
    .WithName("UpdateChecklist")
    .WithDescription("Update a checklist's metadata")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status400BadRequest)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapDelete("/{id}", DeleteChecklist)
    .WithName("DeleteChecklist")
    .WithDescription("Delete a checklist")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();
```

```
group.MapPost("/{id}/archive", ArchiveChecklist)
    .WithName("ArchiveChecklist")
    .WithDescription("Archive a checklist")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPost("/{id}/unarchive", UnarchiveChecklist)
    .WithName("UnarchiveChecklist")
    .WithDescription("Unarchive a checklist")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPost("/{id}/pin", PinChecklist)
    .WithName("PinChecklist")
    .WithDescription("Pin a checklist")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPost("/{id}/unpin", UnpinChecklist)
    .WithName("UnpinChecklist")
    .WithDescription("Unpin a checklist")
    .Produces(StatusCodes.Status204NoContent)
```

```
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPost("/{id}/clone", CloneChecklist)
    .WithName("CloneChecklist")
    .WithDescription("Clone a checklist with all its items")
    .Produces<ChecklistResponse>(StatusCodes.Status201Created)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

// Checklist item operations
group.MapPost("/{id}/items", AddChecklistItem)
    .WithName("AddChecklistItem")
    .WithDescription("Add an item to a checklist")
    .Produces<ChecklistItemResponse>(StatusCodes.Status201Created)
    .Produces(StatusCodes.Status400BadRequest)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPut("/{checklistId}/items/{itemId}", UpdateChecklistItem)
    .WithName("UpdateChecklistItem")
    .WithDescription("Update a checklist item")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status400BadRequest)
    .Produces(StatusCodes.Status401Unauthorized)
```

```
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapDelete("/{checklistId}/items/{itemId}", DeleteChecklistItem)
    .WithName("DeleteChecklistItem")
    .WithDescription("Delete a checklist item")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPost("/{checklistId}/items/{itemId}/toggle", ToggleChecklistItem)
    .WithName("ToggleChecklistItem")
    .WithDescription("Toggle completion status of a checklist item")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();

group.MapPut("/{checklistId}/items/reorder", ReorderChecklistItems)
    .WithName("ReorderChecklistItems")
    .WithDescription("Reorder items in a checklist")
    .Produces(StatusCodes.Status204NoContent)
    .Produces(StatusCodes.Status400BadRequest)
    .Produces(StatusCodes.Status401Unauthorized)
    .Produces(StatusCodes.Status404NotFound)
    .Produces(StatusCodes.Status500InternalServerError)
    .WithOpenApi();
```

```
    group.MapPut("/{checklistId}/items/{itemId}/due-date", UpdateChecklistItemDueDate)
        .WithName("UpdateChecklistItemDueDate")
        .WithDescription("Update the due date of a checklist item")
        .Produces(StatusCodes.Status204NoContent)
        .Produces(StatusCodes.Status400BadRequest)
        .Produces(StatusCodes.Status401Unauthorized)
        .Produces(StatusCodes.Status404NotFound)
        .Produces(StatusCodes.Status500InternalServerError)
        .WithOpenApi();
}

private static async Task<IResult> GetAllChecklists(
    [FromQuery] bool includeArchived,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.GetChecklistsAsync(userId, includeArchived, cancellationToken);
    if (!result.IsSuccess)
    {
        return Results.Problem(result.Errors.FirstOrDefault() ?? "Failed to retrieve checklists");
    }

    return Results.Ok(result.Value);
```

```
}

private static async Task<IResult> GetChecklistById(
    Guid id,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.GetChecklistByIdAsync(userId, id, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.Problem(result.Errors.FirstOrDefault() ?? "Failed to retrieve checklist");
    }

    return Results.Ok(result.Value);
}

private static async Task<IResult> CreateChecklist(
    [FromBody] CreateChecklistRequest request,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
```

```
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.CreateChecklistAsync(userId, request, cancellationToken);
    if (!result.IsSuccess)
    {
        return Results.BadRequest(result.Errors);
    }

    return Results.Created($"/checklists/{result.Value!.Id}", result.Value);
}

private static async Task<IResult> UpdateChecklist(
    Guid id,
    [FromBody] UpdateChecklistRequest request,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.UpdateChecklistAsync(userId, id, request, cancellationToken);
    if (!result.IsSuccess)
    {
```

```
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.BadRequest(result.Errors);
    }

    return Results.NoContent();
}

private static async Task<IResult> DeleteChecklist(
    Guid id,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.DeleteChecklistAsync(userId, id, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.BadRequest(result.Errors);
    }

    return Results.NoContent();
}
```

```
private static async Task<IResult> ArchiveChecklist(
    Guid id,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.ArchiveChecklistAsync(userId, id, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.BadRequest(result.Errors);
    }

    return Results.NoContent();
}

private static async Task<IResult> UnarchiveChecklist(
    Guid id,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
```

```
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.UnarchiveChecklistAsync(userId, id, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.BadRequest(result.Errors);
    }

    return Results.NoContent();
}

private static async Task<IResult> PinChecklist(
    Guid id,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.PinChecklistAsync(userId, id, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
```

```
            ? Results.NotFound()
            : Results.Problem(result.Errors.FirstOrDefault() ?? "Failed to pin checklist");
    }

    return Results.NoContent();
}

private static async Task<IResult> UnpinChecklist(
    Guid id,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.UnpinChecklistAsync(userId, id, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.Problem(result.Errors.FirstOrDefault() ?? "Failed to unpin checklist");
    }

    return Results.NoContent();
}

private static async Task<IResult> CloneChecklist(
```

```
    Guid id,
    [FromBody] CloneChecklistRequest request,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.CloneChecklistAsync(userId, id, request, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.Problem(result.Errors.FirstOrDefault() ?? "Failed to clone checklist");
    }

    return Results.Created($"/checklists/{result.Value!.Id}", result.Value);
}

private static async Task<IResult> AddChecklistItem(
    Guid id,
    [FromBody] CreateChecklistItemRequest request,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
```

```
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.AddChecklistItemAsync(userId, id, request, cancellationToken);
    if (!result.IsSuccess)
    {
        return result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied")
            ? Results.NotFound()
            : Results.BadRequest(result.Errors);
    }

    return Results.Created($"/checklists/{id}/items/{result.Value!.Id}", result.Value);
}

private static async Task<IResult> UpdateChecklistItem(
    Guid checklistId,
    Guid itemId,
    [FromBody] UpdateChecklistItemRequest request,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }
```

```
        var result = await checklistService.UpdateChecklistItemAsync(userId, checklistId, itemId, request,
cancellationToken);
        if (!result.IsSuccess)
        {
            if (result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied") ||
                result.Errors.Contains("Checklist item not found") || result.Errors.Contains("Item does not belong to
the specified checklist"))
            {
                return Results.NotFound();
            }
            return Results.BadRequest(result.Errors);
        }

        return Results.NoContent();
    }

    private static async Task<IResult> DeleteChecklistItem(
        Guid checklistId,
        Guid itemId,
        IChecklistService checklistService,
        HttpContext httpContext,
        CancellationToken cancellationToken)
    {
        var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
        {
            return Results.Unauthorized();
        }

        var result = await checklistService.DeleteChecklistItemAsync(userId, checklistId, itemId,
cancellationToken);
        if (!result.IsSuccess)
```

```
        {
            if (result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied") ||
                result.Errors.Contains("Checklist item not found") || result.Errors.Contains("Item does not belong to the specified checklist"))
            {
                return Results.NotFound();
            }
            return Results.BadRequest(result.Errors);
        }

        return Results.NoContent();
    }
    private static async Task<IResult> ToggleChecklistItem(
        Guid checklistId,
        Guid itemId,
        IChecklistService checklistService,
        HttpContext httpContext,
        CancellationToken cancellationToken)
    {
        var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
        {
            return Results.Unauthorized();
        }
        var result = await checklistService.ToggleChecklistItemCompletionAsync(userId, checklistId, itemId, cancellationToken);
        if (!result.IsSuccess)
        {
            if (result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied") ||
                result.Errors.Contains("Checklist item not found") || result.Errors.Contains("Item does not belong to the specified checklist"))
            {
```

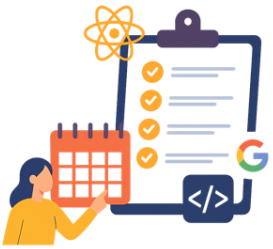
```
            return Results.NotFound();
        }
        return Results.BadRequest(result.Errors);
    }

    return Results.NoContent();
}
private static async Task<IResult> ReorderChecklistItems(
    Guid checklistId,
    [FromBody] ReorderChecklistItemsRequest request,
    IChecklistService checklistService,
    HttpContext httpContext,
    CancellationToken cancellationToken)
{
    var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
    {
        return Results.Unauthorized();
    }

    var result = await checklistService.ReorderChecklistItemsAsync(userId, checklistId, request,
cancellationToken);
    if (!result.IsSuccess)
    {
        if (result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied"))
        {
            return Results.NotFound();
        }
        return Results.BadRequest(result.Errors);
    }
```

```
        return Results.NoContent();
    }

    private static async Task<IResult> UpdateChecklistItemDueDate(
        Guid checklistId,
        Guid itemId,
        [FromBody] UpdateItemDueDateRequest request,
        IChecklistService checklistService,
        HttpContext httpContext,
        CancellationToken cancellationToken)
    {
        var userIdString = httpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (string.IsNullOrEmpty(userIdString) || !Guid.TryParse(userIdString, out var userId))
        {
            return Results.Unauthorized();
        }
        var result = await checklistService.UpdateChecklistItemDueDateAsync(userId, checklistId, itemId, request,
cancellationToken);
        if (!result.IsSuccess)
        {
            if (result.Errors.Contains("Checklist not found") || result.Errors.Contains("Access denied") ||
                result.Errors.Contains("Checklist item not found") || result.Errors.Contains("Item does not belong to
the specified checklist"))
            {
                return Results.NotFound();
            }
            return Results.BadRequest(result.Errors);
        }

        return Results.NoContent();
    }
```

# Додаток Г – Ілюстративна частина

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

«Розробка вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core та React»



Студентка: гр. 4ПІ-21б
**Луп'як М.Д.**
Керівник: к.т.н., доц. каф. ПЗ
**Черноволик Г.О.**

Вінниця 2025

Рисунок Г.1 – слайд 1

- **Метою роботи** є підвищення ефективності особистого тайм-менеджменту користувачів шляхом створення інтегрованого вебзастосунку, який забезпечує цілісне зберігання даних, автоматичну декомпозицію цілей за допомогою штучного інтелекту та гнучке календарне планування.
- **Об'єкт дослідження** – процес розробки вебзастосунку для керування особистими задачами та планування часу з використанням стеку технологій ASP.NET Core та React.
- **Предмет дослідження** – методи та засоби розробки програмного забезпечення для планування часу й керування задачами, що передбачають використання сучасних технологій розробки інтерфейсів, серверної логіки та інструментів штучного інтелекту.

Рисунок Г.2 – слайд 2

## АКТУАЛЬНІСТЬ

У сучасному суспільстві, де велике значення має ефективне планування особистих задач та оптимальне розподілення часу, зростає попит на інструменти управління завданнями, що забезпечують зручну організацію повсякденних справ.

Створення вебсистеми, яка поєднуватиме простоту використання з можливостями автоматизації процесів, є актуальною.

Рисунок Г.3 – слайд 3

## ЗАДАЧІ

1. Проаналізувати предметну область;
2. провести порівняльний аналіз аналогів;
3. провести аналіз підходів до вирішення завдання;
4. обґрунтувати вибір архітектури, технологій та засобів розробки;
5. спроєктувати алгоритми роботи системи;
6. реалізувати клієнтську та серверну частини веб-астосунку;
7. підготувати інструкцію користувача;
8. провести тестування вебзастосунку.

Рисунок Г.4 – слайд 4

## НОВИЗНА

1. Подальшого розвитку набув **метод формування списку задач**, що на відміну від аналогів, дозволяє користувачу автоматизувати розбиття загальної мети на окремі підзадачі за допомогою використання штучного інтелекту, що забезпечує користувачу економію часу та підвищення ефективності планування.
2. Подальшого розвитку набув **метод інтегрованого планування та відстеження дедлайнів**, що об'єднує функціонал створення чеклістів із календарним представленням та переліком майбітніх завдань, дозволяючи користувачам ефективно управляти часом та пріоритизувати дії.

Рисунок Г.5 – слайд 5

## ПРАКТИЧНА ЦІННІСТЬ РЕЗУЛЬТАТІВ

Практична цінність результатів полягає у моливості використання розробленого вебзастосунку для ефективногокерування особистими задачами, щоденного планування та організації часу в повсякденному житті, навчанні чи професійній діяльності.

Рисунок Г.6 – слайд 6

## АНАЛОГИ

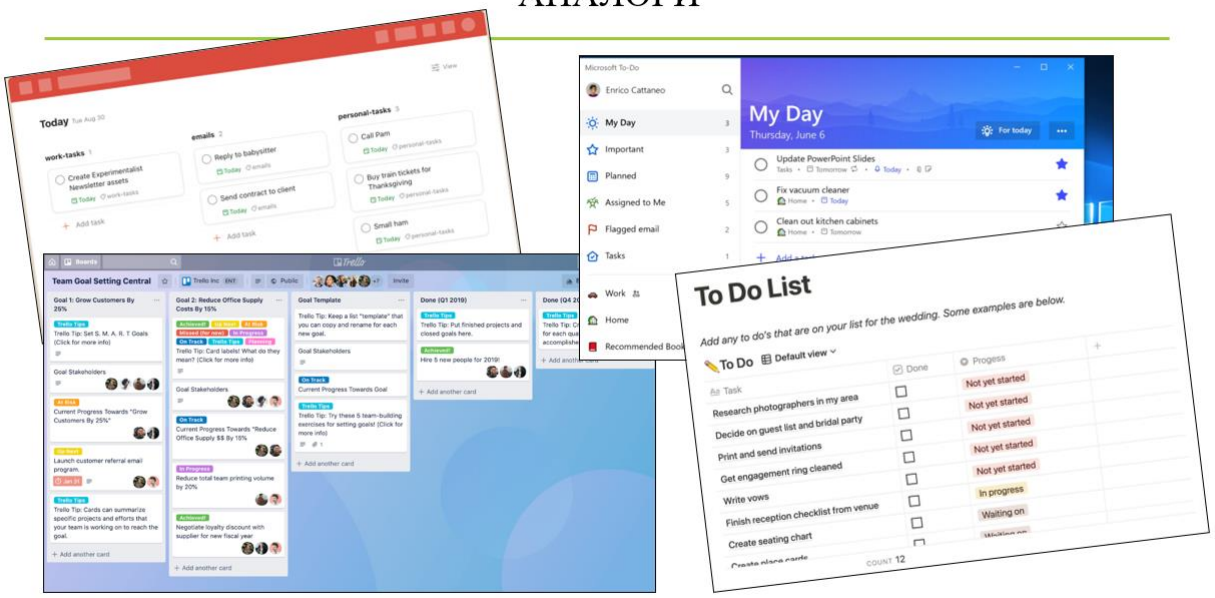



Рисунок Г.7 – слайд 7

## ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ

| Критерій | Todoist | Microsoft To Do | Trello | Notion | CheckIt |
|---|---|---|---|---|---|
| Синхронізація з календарем | 1 | 1 | 0 | 0 | 1 |
| Архівування списків | 0 | 0 | 1 | 1 | 1 |
| Динамічні шаблони | 0 | 0 | 1 | 1 | 1 |
| AI-декомпозиція цілей на підзадачі | 0 | 0 | 0 | 1 | 1 |
| Вбудована аналітика виконання завдань | 1 | 1 | 0 | 0 | 1 |
| Загальна оцінка | 40% | 40% | 40% | 60% | 100% |

Рисунок Г.8 – слайд 8

## МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ

У процесі досліджень використовувались: теорія алгоритмів та структур даних для побудови логіки роботи з чеклістами та задачами, теорія баз даних при проєктуванні структури збереження інформації, принципи UI/UX-дизайну для розробки зручного та адаптивного інтерфейсу, принципи розподілених систем і компонентного проєктування для створення багаторівневої архітектури, методи модульного та інтеграційного тестування для перевірки працездатності окремих компонентів і їх взаємодії в рамках усієї системи.

Рисунок Г.9 – слайд 9

## ТЕХНОЛОГІЧНИЙ СТЕК

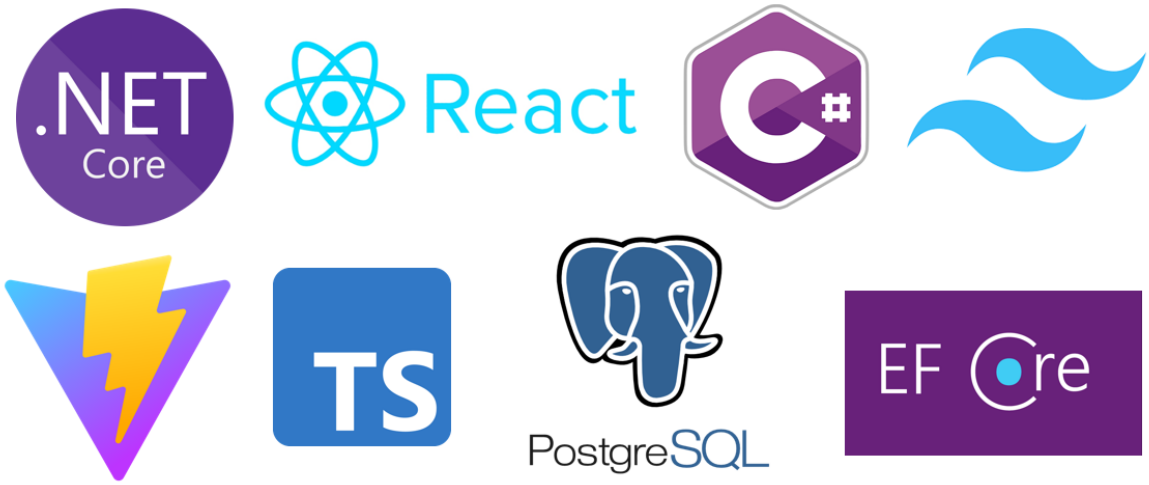



Рисунок Г.10 – слайд 10

## ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ

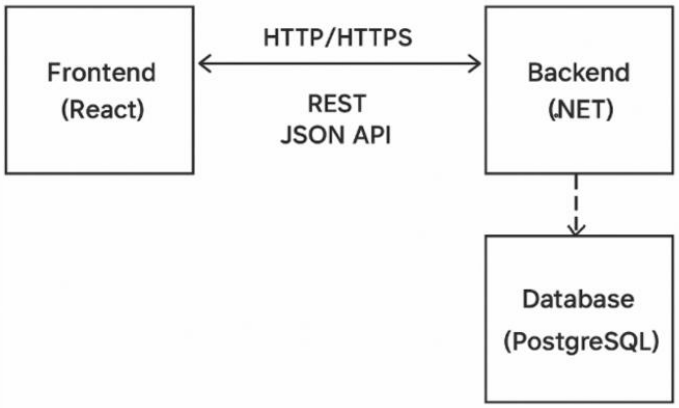



Рисунок Г.11 – слайд 11

## АЛГОРИТМ МЕТОДУ ФОРМУВАННЯ ЗАДАЧ

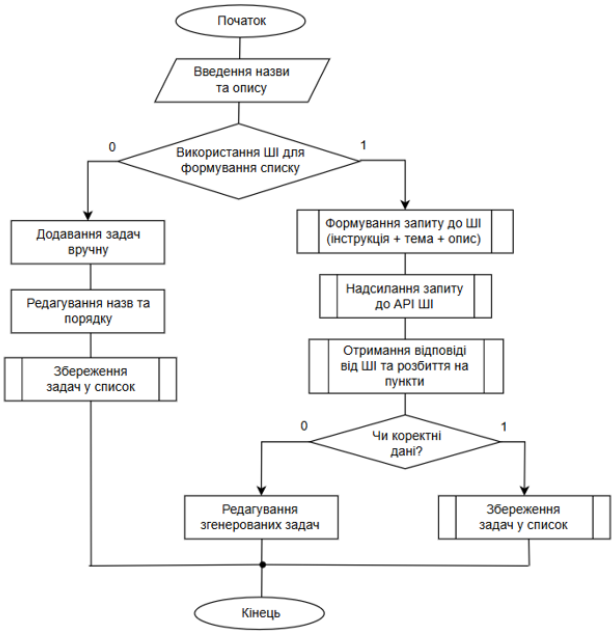



Рисунок Г.12 – слайд 12

# АЛГОРИТМ МЕТОДУ ІНТЕГРОВАНОГО ПЛАНУВАННЯ

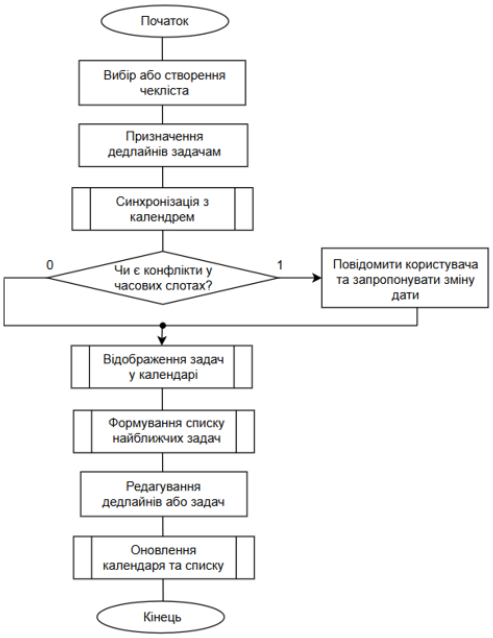



Рисунок Г.13 – слайд 13

# ЗАГАЛЬНИЙ АЛГОРИТМ РОБОТИ СИСТЕМИ

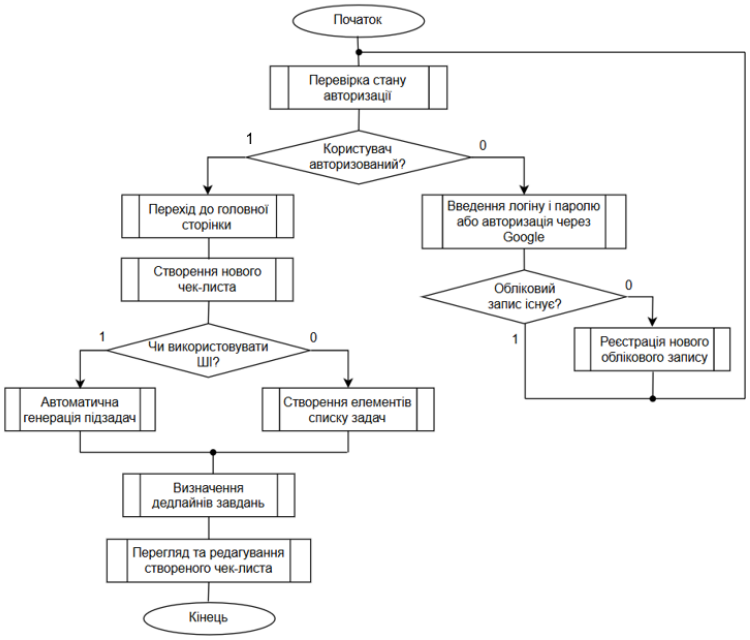



Рисунок Г.14 – слайд 14

# АЛГОРИТМ СТВОРЕННЯ ОБЛІКОВОГО ЗАПИСУ

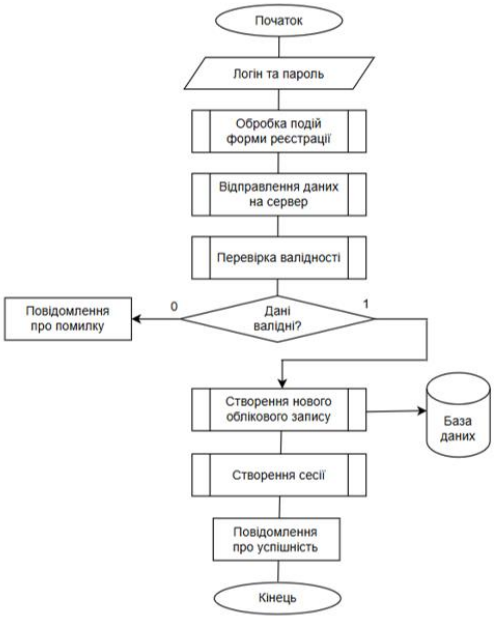



Рисунок Г.15 – слайд 15

# ДЕМОНСТРАЦІЯ РОБОТИ ВЕБЗАСТОСУНКУ

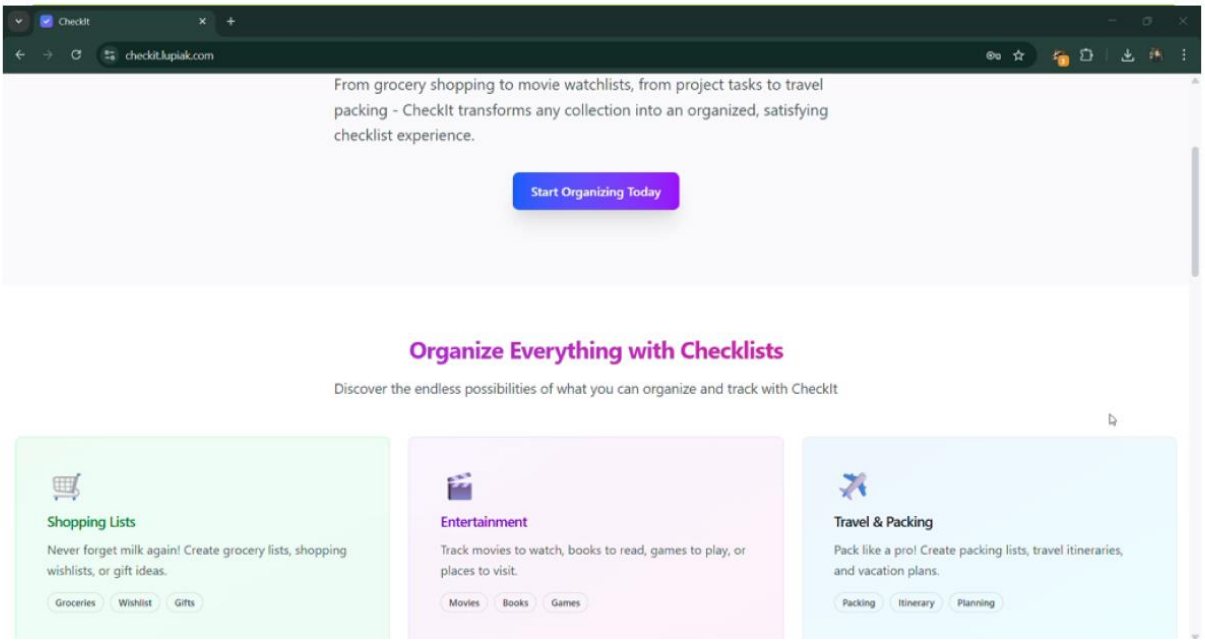



Рисунок Г.16 – слайд 16

# ПУБЛІКАЦІЇ Й АПРОБАЦІЇ

Матеріали бакалаврської дипломної роботи доповідалися на LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025).
За тематикою дослідження опубліковано одну наукову працю в матеріалах конференції.

Рисунок Г.17 – слайд 17

# ВИСНОВКИ

У результаті бакалаврської кваліфікаційної роботи було розроблено вебзастосунок для керування особистими задачами та планування часу. Для розробки клієнтської частини використано React з TypeScript, для серверної – ASP.NET Core.
Було виконано такі завдання:

1. проаналізовано предметну область;
2. проведено порівняльний аналіз аналогів;
3. проведено аналіз підходів до вирішення завдання;
4. обґрунтувано вибір архітектури, технологій та засобів розробки;
5. спроєктовано алгоритми роботи системи;
6. реалізовано клієнтську та серверну частини веб-астосунку;
7. розроблено інструкцію користувача;
8. проведено тестування вебзастосунку.

Рисунок Г.18 – слайд 18

ДЯКУЮ ЗА УВАГУ!

Рисунок Г.19 – слайд 19