Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Інтелектуальний аналіз продуктивності користувачів у системі управління завданнями за допомогою OpenAI API

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Колодій О. О. (прізвище та ініціали)

Керівник (підпис) Стоянов Ю. М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2024

АНОТАЦІЯ

Кваліфікаційна робота магістра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2024. Сторінок 123, рисунків 24, презентація.

Тема: Інтелектуальний аналіз продуктивності користувачів у системі управління завданнями за допомогою OpenAI API.

Метою кваліфікаційної роботи було створення програмного забезпечення для управління завданнями користувача з використанням ШІ для допомоги користувачеві з планом виконання і аналізом успішності виконання останніх завдань.

У результаті було розроблено та розгорнуто клієнт-серверний додаток, здатний до виконання поставлених завдань.

Для можливості використання багатьма користувачами також розроблено модель реєстрації та авторизації за допомогою електронної пошти та паролю з перевіркою чи електронна пошта вже використовується.

Ключові слова: продуктивність користувачів, система управління завданнями, аналіз даних, штучний інтелект, OpenAI API, клієнт-серверний додаток, реєстрація користувачів, авторизація, планування завдань, аналіз успішності.

ABSTRACT

Master's Thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 "Software Engineering." TNTU, 2024. Pages: 123, Figures: 24, Presentation.

Title: Intelligent Analysis of User Productivity in Task Management Systems Using OpenAI API.

The aim of the master's thesis was to create software for user task management utilizing AI to assist users with execution plans and analyze the success of recently completed tasks.

As a result, a client-server application was developed and deployed, capable of performing the assigned tasks.

To enable multi-user functionality, a registration and authentication model was implemented using email and password with email verification to prevent duplicate accounts.

Keywords: user productivity, task management system, data analysis, artificial intelligence, OpenAI API, client-server application, user registration, authentication, task planning, success analysis.

ЗМІСТ

ВСТУП

Продуктивність є одним із ключових показників ефективності роботи людей в умовах сучасного інформаційного суспільства. Постійне зростання обсягу інформації, ускладнення бізнес-процесів та збільшення кількості різноманітних завдань потребують нових підходів до їх координації й аналізу. Виконання задач у визначені строки стає критично важливим як для індивідуальних користувачів, так і для команд та організацій різного масштабу. У цих умовах ефективні системи управління завданнями виступають інструментами, що не лише допомагають організувати поточну діяльність, а й сприяють довгостроковому плануванню, структуризації робочого процесу та прогнозуванню результатів.

Звичайні підходи до управління завданнями, засновані переважно на формальних дедлайнах і стандартизованих критеріях оцінювання, часто не враховують індивідуальні особливості користувачів — їхній робочий ритм, часові зони, індивідуальні схильності та особливості навчання. Така недостатня персоналізація знижує ефективність застосовуваних методик, ускладнює адаптацію систем під конкретні потреби та призводить до того, що користувачі не завжди отримують максимальну користь від доступних інструментів.

Використання штучного інтелекту (ШІ) в управлінні завданнями дозволяє перейти на якісно новий рівень персоналізації та аналітики. Наприклад, алгоритми машинного навчання можуть бути застосовані для глибокого аналізу даних, як це вже зроблено у сфері нейроінтерфейсів для інтерпретації мозкових сигналів. Інтеграція OpenAI API відкриває можливості для впровадження інтелектуальних моделей, здатних генерувати рекомендації щодо оптимізації робочих процесів, підвищення ефективності розподілу ресурсів, а також покращення балансу між робочим навантаженням та відпочинком користувачів.

Актуальність теми обумовлена як зростаючою складністю організації робочого процесу, так і стрімким розвитком технологій штучного інтелекту. У

сучасних умовах постійних змін, віддаленої роботи, глобалізації бізнесу та зростання конкуренції потрібні рішення, здатні оперативно адаптуватися до нових викликів. Інтеграція ШІ в системи управління завданнями дозволяє не лише підвищити ефективність роботи окремих фахівців, а й сприяє формуванню високопродуктивних команд, які працюють більш злагоджено та досягають кращих стратегічних результатів. Такі рішення допомагають оптимізувати час, ресурси і зусилля, спрямовані на виконання проєктів, створюючи передумови для сталого розвитку та підвищення конкурентоспроможності організацій.

Метою кваліфікаційної роботи є створення клієнт-серверного програмного забезпечення для управління завданнями з використанням OpenAI API з метою аналізу продуктивності користувачів та надання їм персоналізованих рекомендацій. Основними завданнями є не лише розробка інтуїтивно зрозумілого інтерфейсу, але й комплексне опрацювання різних аспектів робочого процесу: збір та систематизація даних, формування історії виконання завдань, оцінка динаміки продуктивності користувачів, генерація індивідуальних порад. Окрім того, розробка механізмів реєстрації та авторизації спрямована на забезпечення безпечного та конфіденційного доступу до функціоналу, що дає змогу застосунку ефективно працювати з різними категоріями користувачів, включно з командами різних розмірів.

Предметом дослідження виступає методологія інтеграції ШІ-технологій у процес управління завданнями, а також аналіз способів використання OpenAI API для створення гнучких і адаптивних рекомендаційних систем. Зокрема, цікавим аспектом є дослідження ефективних методів підвищення точності прогнозів щодо витрат часу, оптимізації черговості виконання завдань та вдосконалення особистих робочих стратегій користувачів. Дослідження включає оцінку того, як поєднання технічних рішень (штучний інтелект, клієнт-серверна архітектура, реляційна база даних) та методологічних підходів (агільний розвиток, безперервний аналіз продуктивності) допомагає покращити загальну ефективність управління роботою.

Результати цієї роботи можуть бути використані для подальших досліджень у сфері інтелектуальних систем управління робочими процесами, заснованих на персоналізації та адаптації під потреби конкретних користувачів і команд. Вони можуть стати підґрунтям для розробки більш складних алгоритмічних моделей, що враховуватимуть психологічні та поведінкові особливості, а також контекстуальні фактори, як-от сезонні коливання навантаження чи специфіка окремих галузей. Запропонований підхід здатен впливати не лише на особистий рівень — від підвищення особистої мотивації та відповідальності користувача, — а й на професійний, де ефективне управління завданнями сприятиме зростанню продуктивності команд і організацій у цілому, створюючи нові можливості для їхнього сталого розвитку та конкурентних переваг.

1 АНАЛІЗ ВИМОГ ДО ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблематика та підходи до аналізу продуктивності користувачів у системах управління завданнями

Управління завданнями виступає фундаментальним механізмом планування та координації діяльності як окремих осіб, так і цілих колективів. Індивідуальні користувачі, фрілансери, команди проєктних офісів, малі підприємства та великі корпорації покладаються на спеціалізовані інструменти для організації своєї роботи. Такі інструменти повинні забезпечувати ефективне розподілення ресурсів, визначення пріоритетів, дотримання строків і аналіз прогресу. Однак, попри численний арсенал доступних систем, питання якості аналізу продуктивності користувачів та ефективності управління завданнями залишається недостатньо вирішеним [1, 2].

Сучасні платформи з управління завданнями, такі як Trello, Asana чи ClickUp, пропонують інструменти для організації списків справ, делегування робочих завдань, встановлення дедлайнів і моніторингу виконання. Проте вони часто орієнтуються лише на кількісні показники, такі як кількість виконаних завдань, і рідко враховують індивідуальні аспекти продуктивності [8]. Недостатня інтеграція методів обробки природної мови (NLP) унеможливлює виявлення прихованих проблем або емоційного контексту, як це показано у дослідженнях про аналіз текстів для медичних нейроінтерфейсів [9].

Основні проблеми полягають у:

- Фокусі на кількісних метриках без глибинного аналізу контексту виконання [8].
- Відсутності адаптивності у рекомендаціях [9].
- Обмеженій інтеграції текстового аналізу для глибшого розуміння завдань користувачів [8].

Таким чином, впровадження технологій ШІ, таких як OpenAI API, дозволить суттєво підвищити якість аналізу продуктивності користувачів та зробити рекомендації більш персоналізованими.

Однією з ключових проблем є відсутність індивідуалізованого підходу. Багато користувачів працюють із різними типами завдань, що потребують різної концентрації, знань або творчої діяльності. Деякі завдання можуть містити довгі текстові описи, складні інструкції чи дискусії у вигляді коментарів. Проте більшість платформ не включають глибинний аналіз текстових даних, який міг би дати змогу виявити приховані закономірності, критичні моменти або емоційне ставлення до виконання роботи [2, 3]. Наприклад, фрази на кшталт "дуже складно" чи "терміново" залишаються без уваги з боку систем, хоча вони прямо вказують на специфіку завдання чи його важливість.

Крім того, практично відсутні механізми адаптивності. Сучасні системи рідко пропонують рекомендації, адаптовані під стиль роботи конкретного користувача. Вони не враховують ні пік продуктивності протягом дня, ні індивідуальні часові ритми, ні схильність користувача до певних типів завдань. Тому рекомендації щодо планування, які надають більшість платформ, є надто загальними та не здатні впливати на поведінкові особливості користувача [2, 4]. Наприклад, якщо один користувач ефективніше працює вранці, а інший — пізно ввечері, то стандартні поради "розпочати важливе завдання зранку" будуть однаково застосовані до обох, не враховуючи їхньої індивідуальної продуктивності.

Недостатня інтелектуалізація цих систем також є значним недоліком. Деякі платформи пропонують прості аналітичні інструменти, однак рідко коли вдаються до використання машинного навчання чи штучного інтелекту. Через це вони не можуть автоматично виявляти складні тенденції у виконанні завдань або пропонувати глибоко персоналізовані рекомендації. Сучасні технології дозволяють аналізувати великий обсяг даних, виявляти патерни продуктивності та надавати індивідуальні рекомендації, але більшість платформ не реалізують цей потенціал на практиці [3, 4]. Відсутність штучного інтелекту робить

неможливим автоматичний аналіз емоційної складової тексту, визначення тональності, а також врахування контексту виконання завдань.

Таким чином, основними проблемами є:

Фокус на кількості завдань: Кількісні метрики часто не відображають реальної картини. Наприклад, одна людина може виконати 10 простих завдань за день, а інша — 2 складних завдання, які потребували глибокої аналітики. За формальними показниками перший користувач виглядає більш продуктивним, хоча другий виконав набагато більш комплексну роботу [1].

Обмеження адаптивності: Уніфіковані рекомендації не враховують унікальні робочі ритми, інтереси та навички користувачів, через що знижується практична користь таких порад [2].

Недостатня інтеграція текстового аналізу: Текстові дані залишаються невикористаним ресурсом, хоча вони можуть виявити приховані проблеми, точні причини затримок або рівень мотивації користувача [3].

Обмежена інтелектуалізація: Відсутність штучного інтелекту та машинного навчання унеможливлює автоматизований аналіз поведінки користувачів, виявлення нових підходів до планування та підвищення ефективності [4].

Для подолання цих недоліків необхідно впроваджувати інструменти, здатні аналізувати якісні аспекти робочого процесу та адаптуватися до потреб кожного користувача. Одним із можливих рішень є використання OpenAI API, який дозволяє застосовувати передові моделі штучного інтелекту. Цей підхід може суттєво підвищити точність і корисність аналізу продуктивності, оскільки система зможе аналізувати текст, виявляти специфічні патерни у поведінці користувача та надавати персоналізовані рекомендації в реальному часі [5]. Наприклад, якщо користувач часто скаржиться на нестачу часу у своїх нотатках до завдань, система зможе автоматично пропонувати стратегії тайм-менеджменту або планування пріоритетів.

Таким чином, інтеграція штучного інтелекту й інструментів, на кшталт OpenAI API, відкриває нові можливості для інтелектуалізації систем управління

завданнями та дозволяє глибше зрозуміти поведінку користувачів. Це створює умови для переходу від оцінки "кількість виконаних завдань" до більш осмисленого аналізу "якість та ефективність виконання", що в підсумку сприятиме підвищенню загальної продуктивності та задоволеності користувачів [1-5].

1.2 Вимоги до інтелектуального аналізу продуктивності

Ефективна система аналізу продуктивності має враховувати декілька ключових аспектів, серед яких слід виділити збір, обробку, аналіз та візуалізацію даних. Ці етапи дозволяють не лише оцінити кількісні показники ефективності (наприклад, кількість виконаних завдань чи дотримання дедлайнів), але й розкрити якісні характеристики продуктивності користувача. Завдяки цьому можливим стає формування персоналізованих рекомендацій, спрямованих на підвищення ефективності роботи, враховуючи специфіку конкретної предметної області, види завдань та індивідуальні особливості користувача.

Перший етап — збір даних. Система повинна фіксувати історію створення, зміни та завершення завдань, а також аналізувати, коли саме користувач досягає пікової продуктивності. Наприклад, це можуть бути часові інтервали, протягом яких користувач виконує найбільшу кількість завдань, найскладніші завдання або завдання з найкоротшим часом виконання. Врахування періодів низької активності чи затримок із завершенням завдань дозволить виявити приховані перешкоди. Причинами можуть бути зовнішні фактори (відволікання, неузгодженість із колегами) або внутрішні (втома, низька мотивація) [6]. Глибше розуміння цих аспектів дає змогу оптимізувати розподіл зусиль, визначати пріоритети, запобігати повторенню помилок та уникати непродуктивних сценаріїв роботи.

Другим важливим аспектом є обробка та аналіз текстових даних. Часто завдання супроводжуються описами, коментарями, обговореннями або нотатками, у яких користувач може зазначати труднощі, специфіку виконання чи причини затримок. Використання технологій обробки природної мови (NLP) дозволяє автоматично визначати ключові теми, що повторюються в завданнях. Наприклад, якщо користувач регулярно зазначає, що бракує часу або необхідних інструментів, система може це врахувати при формуванні порад. Аналіз емоційного тону описів дає можливість виявити рівень стресу, демотивацію або, навпаки, задоволення від виконаної роботи. Такий аналіз допомагає зрозуміти, як саме користувач сприймає власну діяльність, виявити психологічні бар’єри або чинники, які знижують продуктивність [3].

Важливою складовою є візуалізація даних, оскільки вона допомагає перетворити складні аналітичні висновки на наочні та легкодоступні для сприйняття матеріали. Візуалізація може бути представлена у вигляді різних типів графіків, діаграм, часових ліній та інтерактивних панелей керування. Зокрема, можливість фільтрувати дані за певними параметрами (наприклад, за статусом завдання, строком виконання чи категоріями складності) дозволяє користувачеві адаптувати відображення інформації під власні потреби. Таким чином, дані стають більш зрозумілими, а користувач отримує інструмент для швидкої оцінки ситуації та прийняття рішень щодо подальших дій [16].

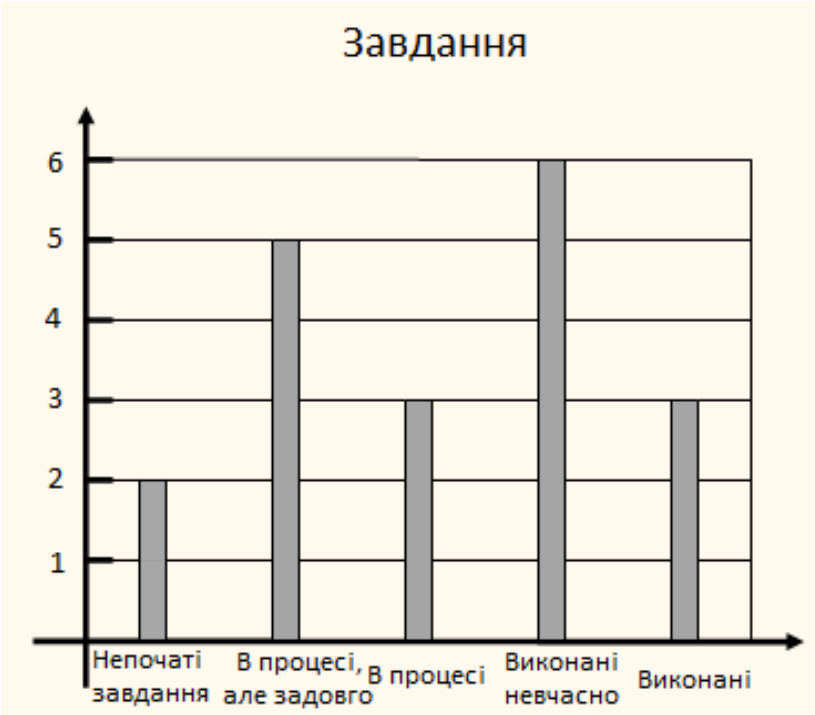


Рисунок 1.1 - Приклад графіку, яким можна відобразити результати виконання завдань користувача

На рисунку 1.1 зображено приклад стовпчикової діаграми, яка може відображати співвідношення різних статусів завдань користувача та їхню своєчасність. Наприклад, на такому графіку можна побачити скільки завдань взагалі не були розпочаті, скільки знаходяться у процесі виконання згідно з запланованим часом чи перевищують його, які завдання вдалося завершити вчасно, а які — із запізненням. Такий підхід допомагає не лише оперативно оцінити поточний стан справ, а й виявити закономірності, які повторюються з часом. Наприклад, якщо користувач систематично перевищує розрахунковий час на завдання однієї категорії, то можна дійти висновку про необхідність більш ретельного планування, додаткового навчання чи коригування пріоритетів.

Такий підхід сприяє більш глибокому аналізу ситуації, оскільки користувач може експериментувати з даними, шукаючи приховані патерни. Наприклад, аналізуючи графік продуктивності за кілька місяців, користувач помітить, що восени ефективність знижується, і спробує зрозуміти причини цього явища [17, 18]. Можливо, в цей період виникає більше складних завдань, або користувачеві бракує ресурсів чи підтримки.

Отже, поєднання збору, аналізу і візуалізації даних дає системі інструментарій для об’єктивної оцінки ефективності користувача та її динаміки. Текстовий аналіз дозволяє не обмежуватись кількісними показниками, а враховувати емоційні, мотиваційні та контекстні фактори. Завдяки цьому користувач отримує не просто статистику, а комплексну картину своєї діяльності, що сприяє більш раціональному плануванню, вчасному виявленню проблем і максимізації ефективності робочого процесу.

1.3 Використання сучасних технологій для аналізу продуктивності

Технології штучного інтелекту відіграють ключову роль у створенні інтелектуальних систем. Використання NLP дає можливість аналізувати текстові дані, визначати ключові слова, теми та тональність. Наприклад, опис завдання з ключовими словами "терміново" чи "високий пріоритет" може автоматично виділятися як критичне [3, 18].

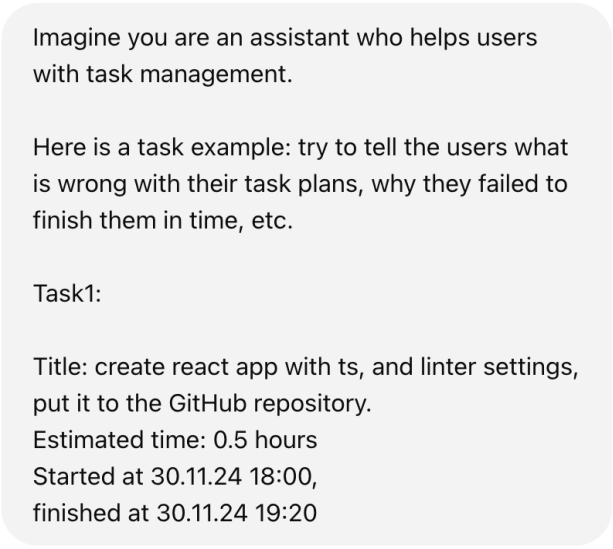
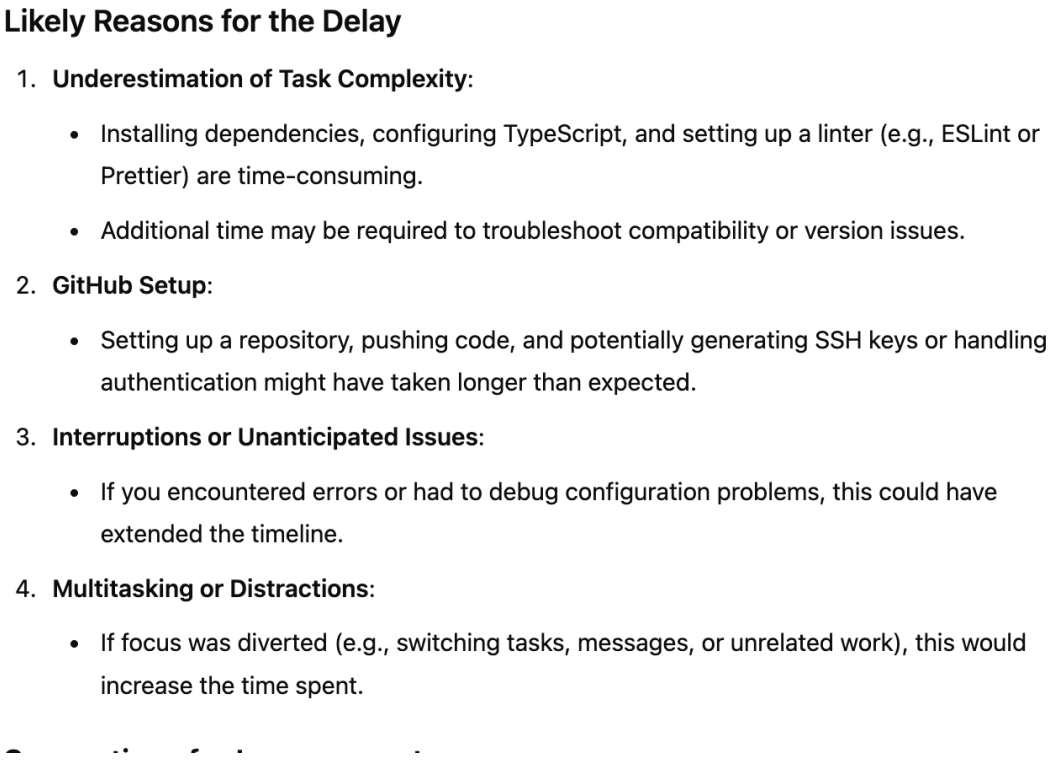


Рисунок 1.2 - Приклад запиту до ChatGPT, який повинен здійснити аналіз завдання користувача

На рисунку 1.2 зображено тестовий запит до ChatGPT з проханням оцінити виконання завдання користувача, яке він планував виконати за пів години, але це зайняло йому 1,33 години.

Моделі машинного навчання допомагають системі прогнозувати продуктивність користувача на основі історичних даних. Вони також можуть генерувати рекомендації для підвищення ефективності. Наприклад, якщо користувач має схильність до затримок у виконанні завдань після обіду, система може запропонувати змінити розклад або делегувати менш пріоритетні завдання [6].

Важливим елементом є інтеграція OpenAI API, який дозволяє створювати динамічні рішення для аналізу та взаємодії з користувачем. API забезпечує високу точність у генерації текстових рекомендацій та аналізі даних [5].



Likely Reasons for the Delay

1. **Underestimation of Task Complexity**:
   - Installing dependencies, configuring TypeScript, and setting up a linter (e.g., ESLint or Prettier) are time-consuming.
   - Additional time may be required to troubleshoot compatibility or version issues.
2. **GitHub Setup**:
   - Setting up a repository, pushing code, and potentially generating SSH keys or handling authentication might have taken longer than expected.
3. **Interruptions or Unanticipated Issues**:
   - If you encountered errors or had to debug configuration problems, this could have extended the timeline.
4. **Multitasking or Distractions**:
   - If focus was diverted (e.g., switching tasks, messages, or unrelated work), this would increase the time spent.

а) причини затримки у виконанні

Suggestions for Improvement

1. **Break Down the Task**:
   - Split the task into smaller, measurable subtasks like:
     - "Create React app with TypeScript" (15 minutes).
     - "Add and configure linter" (20 minutes).
     - "Push code to GitHub repository" (10 minutes).
   - This would give a more realistic estimate for each phase.
2. **Add Buffer Time**:
   - Include buffer time for potential issues. For example, a 1-hour estimate would have been more realistic for this task.
3. **Document Errors**:
   - Keep a record of what caused delays (e.g., npm install failed, linter configuration not compatible) to adjust future time estimates and improve workflows.
4. **Focus and Optimize Workflow**:
   - Minimize distractions during task execution.
   - Pre-plan repetitive setups by creating templates (e.g., a starter repo with TypeScript and linter pre-configured).

б) Поради для покращення оцінки часу виконання

Рисунок 1.3 - Приклад відповіді ШІ, яка вказує на можливі причини затримки в виконані завдання і можливі рішення для покращення оцінки виконання в майбутньому

Рисунок 1.3 відображає аналіз ChatGPT завдання користувача, яке він виконав, але за значно більший час ніж розраховував. На рисунку а ШІ намагається пояснити можливі причини затримки, чому саме так вийшло, що користувач вважав, що завдання він виконає швидше ніж це вийшло. На рисунку б ШІ надає поради користувачу, для того, щоб в подальшому він розраховував час правильно.

Крім того, важливо врахувати можливість інтеграції з іншими платформами управління завданнями. Наприклад, система може підключатися

до Trello або Asana для імпорту завдань, зберігаючи час користувачів. Це також спрощує впровадження нових функцій в існуючі робочі процеси [1].

Дані мають бути представлені у зручній формі. Графіки, діаграми та текстові пояснення повинні доповнювати одне одного, щоб користувач міг швидко зрозуміти, які аспекти його роботи потребують уваги. Інтерактивні елементи, такі як фільтри чи інструменти порівняння, дозволять налаштовувати візуалізації відповідно до потреб користувача [7].

2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Постановка завдання розробки

Кожна програмна система є складним перетворювачем, який трансформує вхідні дані в цінну інформацію або дії, що сприяють досягненню поставлених цілей. Поведінка такої системи формується на етапі розробки, а її властивості визначаються вимогами користувачів та особливостями предметної області. У даній магістерській роботі виконано розробку програмного забезпечення для аналізу продуктивності користувачів у системах управління завданнями.

Метою роботи є створення інтелектуальної інформаційної системи для управління завданнями (ToDo App), яка надає користувачам не лише інструменти для організації роботи, а й інструменти інтелектуального аналізу продуктивності за допомогою інтеграції OpenAI API. Система покликана автоматизувати аналіз індивідуальних показників ефективності, визначати слабкі місця у плануванні, а також генерувати персоналізовані рекомендації для підвищення продуктивності користувача [8].

У ході дослідження та розробки було визначено, що майбутня система повинна підтримувати функції збору, обробки, аналізу та візуалізації даних про виконання завдань. Зокрема, система забезпечує:

- збір даних про створення, виконання, перенесення або видалення завдань;
- аналіз текстових описів завдань за допомогою методів обробки природної мови (NLP);
- визначення емоційного тону та ключових тем описів завдань;
- створення звітів про динаміку продуктивності з використанням інтерактивних графіків;
- генерацію рекомендацій для оптимізації планування та підвищення ефективності [9].

Важливою частиною проєкту є розробка системи реєстрації та авторизації користувачів. Вона забезпечує персоналізацію досвіду роботи із застосунком, захищає дані користувачів та гарантує контроль доступу до функціоналу програми. Реєстрація дозволяє новим користувачам створити обліковий запис, вводячи необхідні дані, а авторизація дає змогу ідентифікувати користувачів і забезпечувати доступ до їхніх персональних даних і налаштувань [12].

Для забезпечення високого рівня безпеки в системі використовуються токени JSON Web Token (JWT) для реалізації безпечної та ефективної аутентифікації. Кожен обліковий запис прив’язаний до унікального ідентифікатора користувача, що дозволяє системі аналізувати дані на основі індивідуальних показників і створювати персоналізовані рекомендації.

Розроблена система має виглядати як модульний програмний продукт із компонентами для інтеграції в інші платформи, зокрема через API. Для аналізу даних використовуються методи математичного моделювання, включно з розрахунком статистичних показників продуктивності та прогнозуванням поведінки користувача на основі історичних даних [13].

На основі аналізу було визначено, що програмна система повинна включати:

1. Модуль інтеграції OpenAI API для обробки текстових даних завдань і генерації рекомендацій.
2. Алгоритми аналізу продуктивності з підтримкою індивідуального налаштування ключових метрик.
3. Інструменти візуалізації даних, які включають дашборд для користувачів із графіками динаміки продуктивності, прогнозами та рекомендаціями.
4. Систему ролей та доступів для багатокористувацького використання.
5. Систему реєстрації та авторизації, яка підтримує створення облікових записів і безпечний доступ до даних [5].

Програмний продукт передбачає роботу з такими основними функціями:

- введення даних про завдання;
- обробка текстових описів завдань для аналізу їхнього змісту;

- створення звітів на основі часових і статистичних метрик;
- візуалізація даних на інтерактивних графіках із можливістю фільтрації за обраними параметрами;
- надання персоналізованих рекомендацій, сформованих на основі моделей штучного інтелекту;
- захищений вхід та налаштування доступу до індивідуальних даних.

Результатом виконання поставленого завдання стане створення сучасної інформаційної системи, яка здатна не лише підвищити продуктивність користувачів, але й запропонувати їм нові підходи до планування та управління часом за допомогою інтелектуального аналізу.

2.2 Визначення актантів та варіантів використання

Розробка програмного забезпечення передбачає проектування сценаріїв його використання, які допомагають визначити основні функції системи та можливості для користувачів. У контексті ToDo застосунку з використанням штучного інтелекту було визначено ключові варіанти використання, що охоплюють усі аспекти взаємодії користувача з системою.

Додатково, розроблено діаграму варіантів використання (рисунок 2.1), яка ілюструє основні сценарії взаємодії користувача із застосунком.

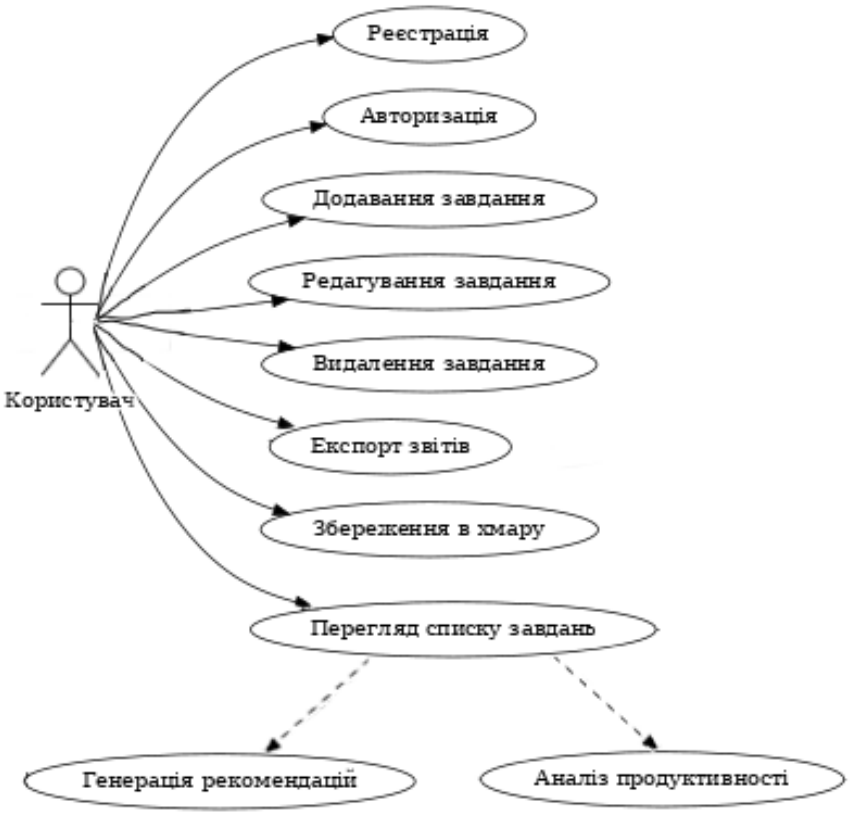


Рисунок 2.1 - Діаграма варіантів використання системи для актанту “Користувач”

На рисунку 2.1 представлена діаграма варіантів використання для основного актора — користувача. Вона ілюструє взаємозв’язки між основними функціями системи та їхнім використанням у реальних сценаріях.

Ключові варіанти використання:

1. Авторизація та реєстрація.

Реєстрація нового користувача та вхід у систему через email і пароль є базовими функціями. Поточна реалізація не підтримує двофакторну автентифікацію (2FA) та OTP, однак передбачає можливість їх додавання у майбутньому. Після авторизації користувач отримує доступ до свого списку завдань і статистики.

2. Створення нового завдання.

Користувач може додавати завдання, вказуючи їхню назву, опис, дедлайн, пріоритетність і категорію. Додатково система дозволяє встановлювати періодичність завдань для автоматичного повторення (наприклад, щоденно або щотижнево).

3. Редагування завдання.

Існує можливість змінювати параметри створених завдань, такі як опис, дедлайн, пріоритет чи категорія, у разі зміни обставин або вимог.

4. Видалення завдання.

Завдання, що більше не є актуальними, можуть бути видалені користувачем. Це допомагає підтримувати актуальність списку завдань.

5. Сортування та фільтрація завдань.

Користувач може сортувати завдання за різними критеріями: за датою створення, дедлайном або пріоритетністю. Фільтри дозволяють виділити, наприклад, лише термінові або завершені завдання.

6. Перегляд рекомендацій щодо продуктивності.

Інтеграція з OpenAI API дозволяє аналізувати виконання завдань та надавати рекомендації. Наприклад, система може запропонувати оптимальний час для виконання певних завдань на основі попередньої поведінки користувача.

7. Візуалізація результатів аналізу.

Система представляє аналітичні дані у вигляді графіків і діаграм, що дозволяє користувачу отримати чітке уявлення про свою продуктивність. Наприклад, графіки показують розподіл завершених завдань за тиждень, середню тривалість виконання завдань тощо.

8. Збереження даних у базі даних.

Дані зберігаються у PostgreSQL, розміщеній на платформі neon.tech, для подальшого аналізу та доступу в будь-який час. Це гарантує збереження та цілісність даних.

9. Експорт завдань.

Система дозволяє експортувати дані про завдання у форматах JSON чи CSV для зберігання, аналізу або інтеграції з іншими інструментами. Поточна

версія не підтримує інтеграції з календарями, такими як Google Calendar чи Outlook, але ця функція може бути реалізована в майбутньому.

10. Основний актор

Основним актором системи є користувач, який має доступ до всіх функцій, включно з можливостями створення, редагування, видалення завдань, аналізу продуктивності та використання рекомендацій. Уся функціональність спрямована на максимальну зручність і простоту взаємодії з системою.

2.3 Вибір архітектури та розробка програмного продукту

Розробка програмного забезпечення є складним і багатоступеневим процесом, який потребує ретельного вибору відповідної моделі розробки та архітектури. Вибір цих аспектів має вирішальне значення для успішної реалізації проекту, забезпечення його гнучкості, масштабованості та зручності використання. У рамках цього проекту було обрано еволюційну модель розробки, що дозволила поступово вдосконалювати функціональність і враховувати зміни у вимогах.

Еволюційна модель є ітеративним підходом, який дозволяє створювати програмне забезпечення шляхом поступового впровадження функцій та вдосконалення. Цей метод ідеально підходить для проектів із невизначеними або змінними вимогами. Основні особливості еволюційної моделі:

Ітеративність — розробка виконується серіями ітерацій. Кожна ітерація завершується функціонально працюючим прототипом, який можна оцінити та протестувати.

1. Поступові зміни — кожна ітерація включає розширення існуючого функціоналу або додавання нового, враховуючи зміни в умовах проекту чи відгуки користувачів.

2. Активна участь зацікавлених сторін — користувачі, розробники та інші учасники проекту працюють спільно, формуючи пріоритети та надаючи зворотний зв’язок.
3. Перегляд та оцінка — результати кожної ітерації ретельно аналізуються, що дозволяє вчасно виявляти та усувати помилки, а також адаптувати проект до змін.
4. Еволюційна модель сприяє досягненню високої відповідності вимогам користувачів та дозволяє швидко реагувати на їхні потреби. Цей підхід став основою для організації роботи над проектом, забезпечуючи послідовний розвиток системи. Для наочності модель представлена на рисунку 2.2.

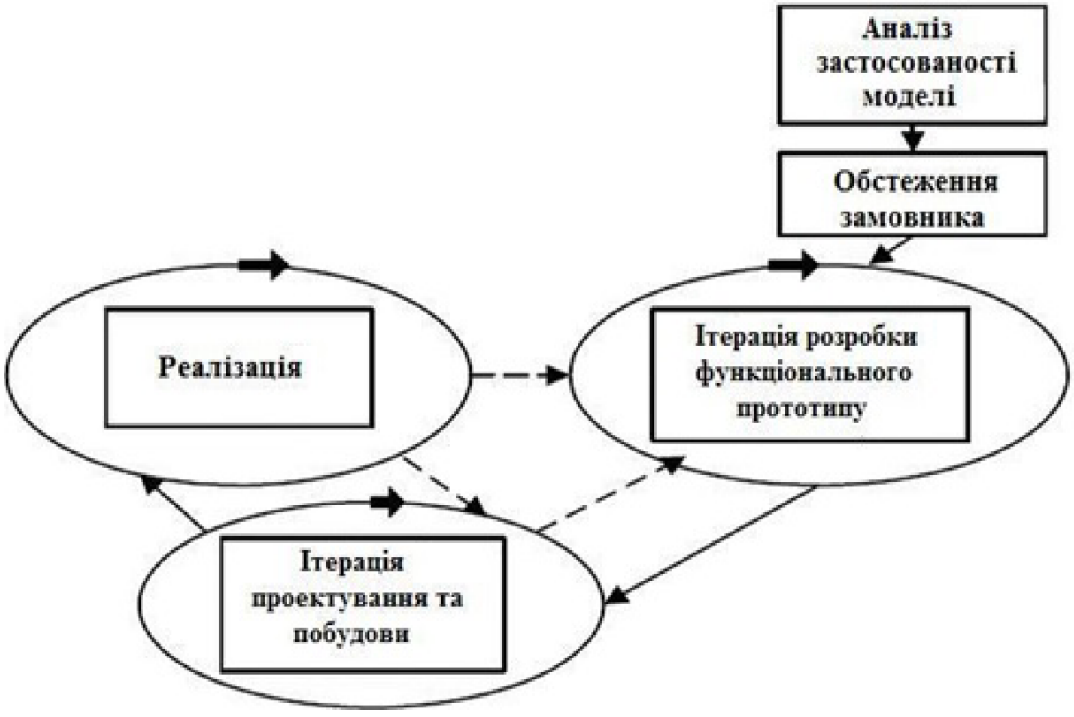


Рисунок 2.2 – Модель еволюційного прототипіювання

Архітектура проекту базується на клієнт-серверній моделі, яка добре підходить для веб-додатків і передбачає взаємодію між клієнтською частиною (frontend), серверною частиною (backend) та базою даних [10, 11]. Основні компоненти архітектури:

Клієнтська частина відповідає за взаємодію з користувачем. Вона обробляє введення даних, відображає результати роботи системи та надсилає запити до серверної частини через API.

Серверна частина забезпечує бізнес-логіку додатка та обробку запитів, а також виконує операції з базою даних. Ця частина відповідальна за обробку завдань, авторизацію користувачів і взаємодію з OpenAI API для аналітичних розрахунків [19].

Для зберігання завдань і користувацької інформації обрано реляційну базу даних PostgreSQL, яка надає надійний механізм роботи з даними, можливість масштабування та високий рівень безпеки.

Хмарна інфраструктура. Для розгортання проекту використано наступні платформи:

Frontend: GitHub Pages — безкоштовна платформа для публікації статичних веб-додатків.

Backend: Render.com — сервіс для розгортання серверних додатків, що підтримує Node.js і інші технології.

Database: Neon.tech — хмарна платформа для роботи з PostgreSQL, яка забезпечує простоту налаштування та масштабування.

Діаграма компонентів:

У проекті реалізовано декілька модулів, кожен із яких виконує певну функцію:

Авторизація користувачів — підтримка традиційної реєстрації, але поки що без OTP або 2FA, які можуть бути додані у майбутньому.

Управління завданнями — створення, редагування, сортування та видалення завдань.

Аналітичний модуль — аналіз продуктивності користувачів за допомогою OpenAI API.

API для інтеграції — обробка запитів між клієнтом і сервером.

На рисунку 2.3 представлено схему взаємодії між цими компонентами.

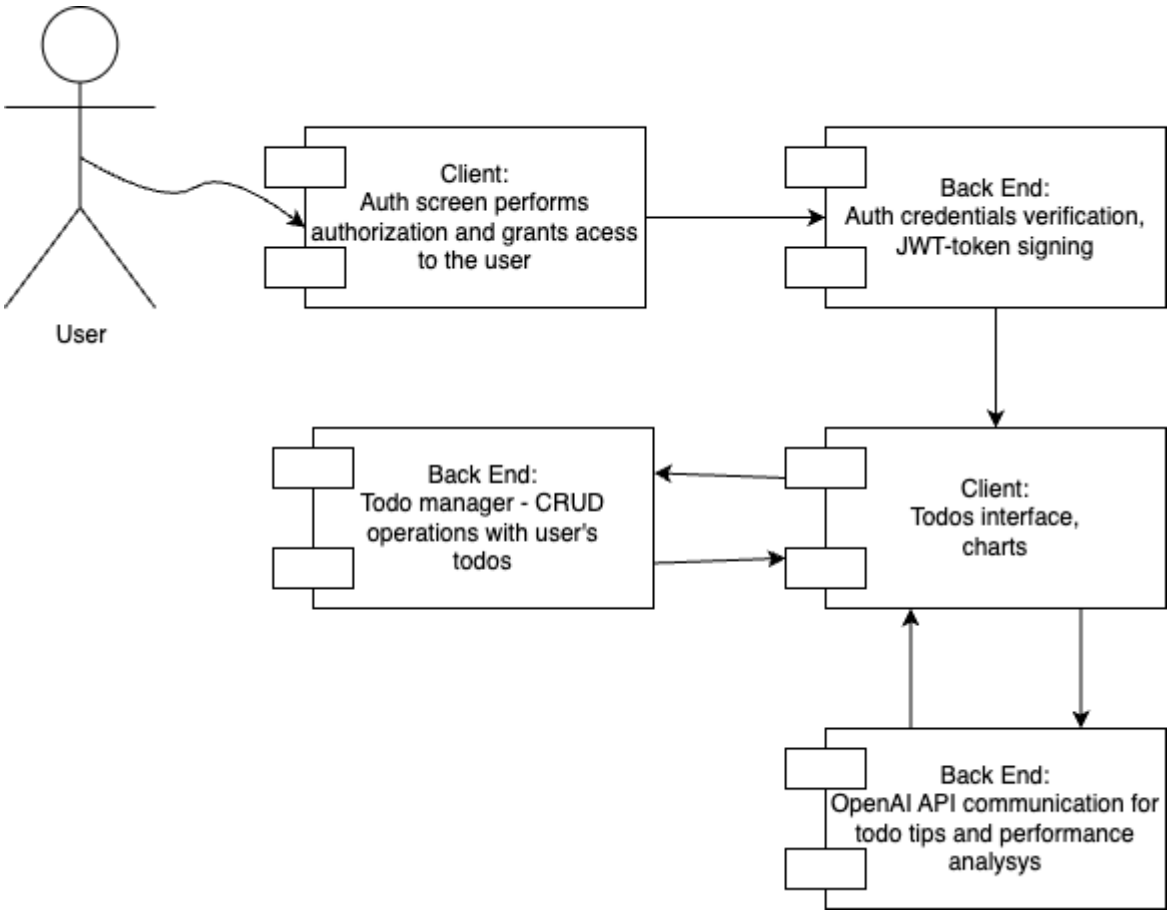


Рисунок 2.3 - Діаграма модулів системи

Переваги обраної архітектури:

- Модульність — розділення системи на незалежні компоненти спрощує її підтримку та розширення.
- Гнучкість — обраний стек технологій дозволяє швидко додавати нові функції.
- Масштабованість — використання хмарних сервісів забезпечує можливість роботи з великим обсягом даних.
- Ітеративний підхід — кожна нова функція інтегрується поступово, що дозволяє швидко реагувати на зміни.

Еволюційна модель разом із обраною архітектурою забезпечила ефективність розробки та можливість адаптації проекту до нових вимог.

2.4 Абстрактна модель розроблюваного програмного забезпечення

Для розробки цієї системи було обрано мову програмування TypeScript, що використовується в середовищі Node.js для серверної частини додатка. Враховуючи сучасні вимоги до розробки програмного забезпечення, зокрема об'єктно-орієнтовану парадигму, було проведено аналіз основних сутностей системи та їх взаємодії.

Проектування системи виконувалося з використанням типів та інтерфейсів, однак для документування архітектури та забезпечення її відповідності стандартам UML (Unified Modeling Language) було створено діаграму класів, яка представляє ключові класи, їх атрибути, методи та взаємозв’язки. UML широко використовується для проектування систем та бізнес-процесів завдяки своїй гнучкості та стандартизованій нотації. Вона дозволяє ефективно візуалізувати структуру додатка, спрощуючи комунікацію між членами команди розробників і сприяючи більш якісному проектуванню та впровадженню системи [22].

Діаграма класів представлена на рисунку 2.4, що демонструє основну архітектуру системи.

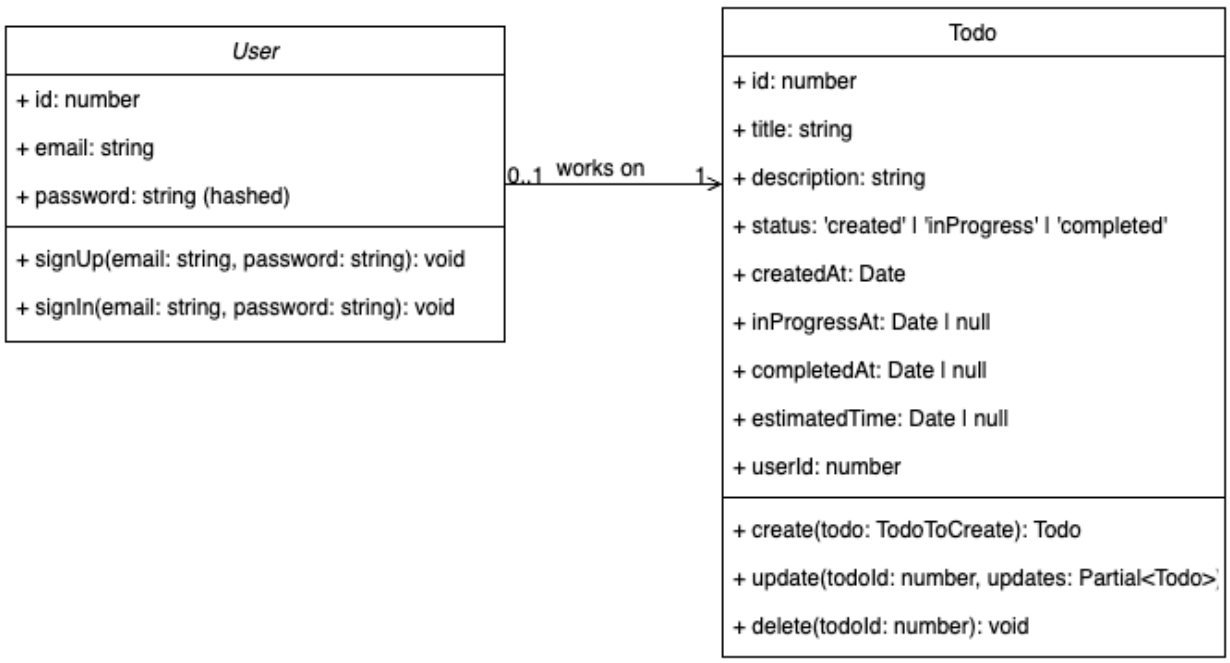


Рисунок 2.4 - UML-діаграма класів системи управління завданнями

Згідно з діаграмою, основними класами системи є User та Todo.

Клас User моделює користувачів системи і містить атрибути для зберігання унікального ідентифікатора користувача (id), його електронної пошти (email) та захешованого пароля (password). Для роботи з користувачами передбачено методи:

- signUp(email: string, password: string): void — реєстрація нового користувача.
- signIn(email: string, password: string): void — авторизація користувача в системі.

Клас Todo відповідає за зберігання завдань і включає атрибути для опису завдання: заголовок (title), опис (description), статус виконання (status), а також часові мітки (createdAt, inProgressAt, completedAt). Кожне завдання пов’язане з конкретним користувачем за допомогою поля userId. Методи класу дозволяють виконувати стандартні CRUD-операції:

- create(todo: TodoToCreate): Todo — створення нового завдання.

- update(todoId: number, updates: Partial<Todo>): void — оновлення існуючого завдання.
- delete(todoId: number): void — видалення завдання.

Зв’язок між класами User та Todo моделює відношення типу "один-до-багатьох", де один користувач може працювати з кількома завданнями.

Таким чином, представлена діаграма класів демонструє логічну структуру системи управління завданнями, відображаючи основні сутності, їхні атрибути та функціональність. Вона є фундаментом для реалізації відповідного функціоналу і допомагає структурувати розробку з урахуванням принципів об'єктно-орієнтованого проектування.

Для найпростішого відтворення процесу роботи з завданнями у системі управління завданнями можна скористатися UML-діаграмою активності. Ця діаграма крок за кроком показує сценарій взаємодії користувача із системою, починаючи з авторизації і закінчуючи виконанням різних операцій із завданнями.

На діаграмі активності присутні стартова та завершальні позиції, які відображають початок і закінчення виконання процесу. Процес завершується в одному з двох випадків: після успішного виконання дій користувача або у разі невдачі, наприклад, при неправильному введенні облікових даних.

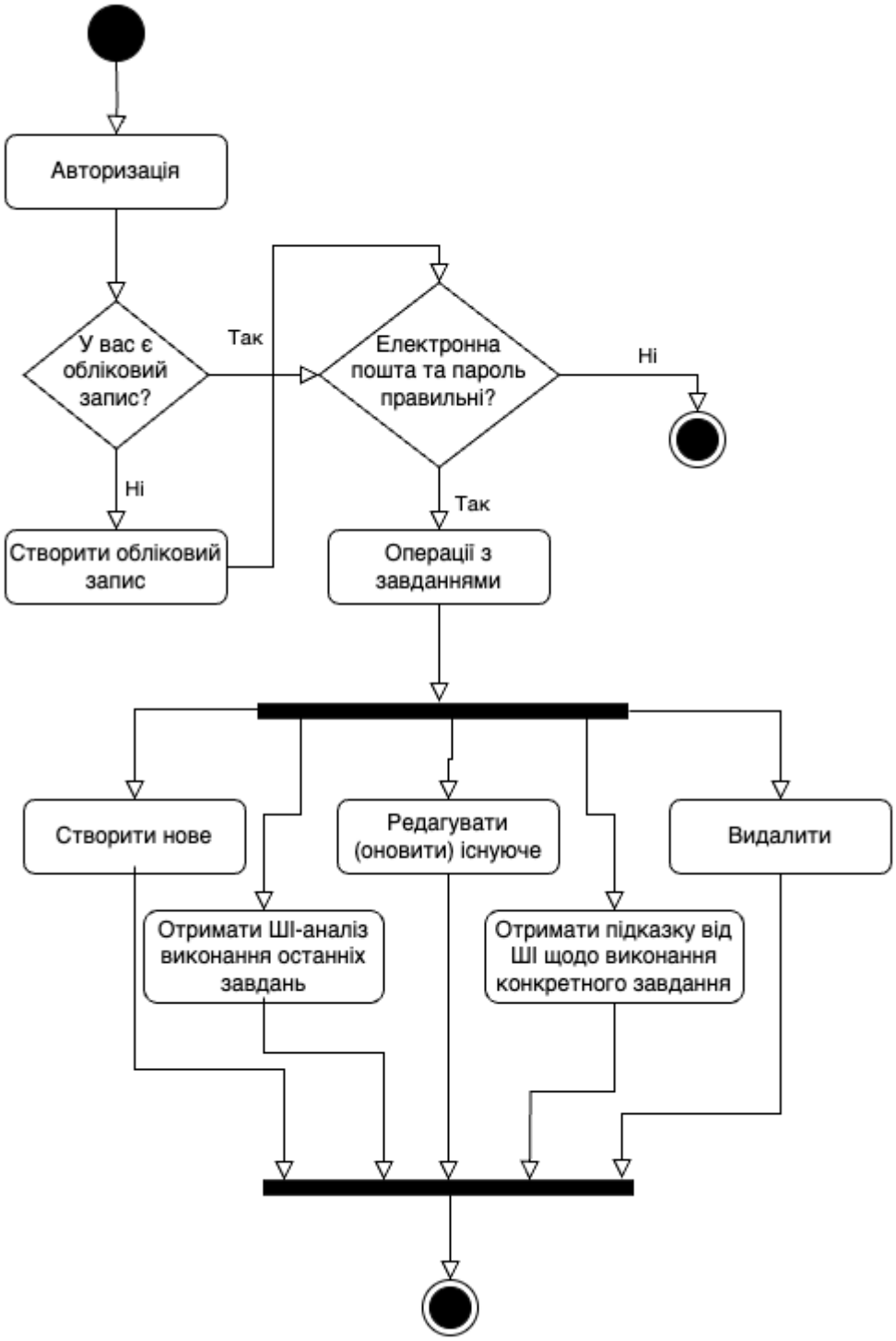


Рисунок 2.5 - UML-діаграма активностей для роботи із завданнями

На початковому етапі користувач проходить авторизацію у системі, перевіряється наявність облікового запису. У разі відсутності облікового запису пропонується його створення. Якщо обліковий запис існує, система перевіряє правильність введених облікових даних (електронної пошти та пароля).

Після успішної авторизації користувач отримує доступ до виконання операцій із завданнями. Серед доступних операцій — створення нового завдання, редагування існуючого, а також його видалення. Окрім базових дій із завданнями, система дозволяє отримувати підказки та аналіз виконання завдань за допомогою штучного інтелекту (ШІ). Наприклад:

Отримання ШІ-аналізу виконання останніх завдань, що допомагає користувачеві зрозуміти свої продуктивні показники.

Надання підказок ШІ щодо виконання конкретного завдання для підвищення ефективності його завершення.

Таким чином, діаграма активності відображає основну архітектуру процесу роботи користувача із системою, інтегруючи традиційні операції з можливостями ШІ для покращення взаємодії та продуктивності.

3 РОЗРОБКА ПРОГРАМНО-АПАРТНОГО КОМПЛЕКСУ

3.1 Конструювання і розробка системи

При розробці сучасних веб-додатків критично важливо ретельно обрати технології та засоби, які впливатимуть на гнучкість у майбутньому, а також на продуктивність та масштабованість системи. Клієнт-серверна архітектура застосунку забезпечує розділення функціональності між фронтендом і бекендом, що дозволяє незалежно оптимізувати кожен з цих шарів та спрощує обслуговування. У випадку даного ToDo-застосунку, де користувачі можуть керувати завданнями з різними станами (створене, в процесі виконання, завершене), встановлювати для них орієнтовний час виконання, а також отримувати підказки і рекомендації щодо ефективності виконання за допомогою OpenAI API, оптимальний добір технологій допомагає втілити задум із мінімальними витратами часу на підтримку та розвиток.

Для реалізації фронтенд-частини було обрано React.js у поєднанні з TypeScript [15, 23]. React.js, як бібліотека для створення інтерфейсів користувача, дає змогу компонувати інтерфейс із незалежних компонентів, що знижує складність і підвищує повторне використання коду. Використання TypeScript, який додає статичну типізацію до JavaScript, сприяє ранньому виявленню помилок, покращує читабельність та передбачуваність коду, а також полегшує взаємодію з інструментами автодоповнення. Застосування TypeScript знижує вірогідність виникнення неочікуваних поведінкових помилок під час роботи з даними завдань і станами компонентів.

Для взаємодії з користувачем та швидкої розробки візуальних елементів застосовано бібліотеку Ant Design (antd) [24]. Вона містить широкий спектр готових компонентів, як-от форми, кнопки, модальні вікна, інтерактивні списки та іконки. Це дозволяє сфокусуватись на логіці застосунку і мінімізувати час на

розробку UI, забезпечуючи водночас професійний та послідовний вигляд інтерфейсу.

На серверній стороні застосунок побудовано з використанням Node.js та Express.js з підтримкою TypeScript [25]. Node.js забезпечує асинхронну модель обробки запитів і високу продуктивність при роботі з великою кількістю одночасних підключень, а Express.js – мінімалістичну основу для створення REST API. Такий підхід полегшує процес інтеграції фронтенду та бекенду, спрощує маршрутизацію, обробку HTTP-запитів та впровадження middleware, які, наприклад, відповідають за перевірку авторизації або ведення журналів. TypeScript тут також уніфікує підхід до типізації на фронтенді та бекенді, забезпечуючи злагодженість логіки та мінімізуючи розбіжності у форматах даних між клієнтом і сервером.

Для роботи з реляційною базою даних PostgreSQL використано ORM Sequelize [26]. PostgreSQL надає потужні можливості для складних аналітичних запитів і транзакцій, а Sequelize перетворює таблиці та записи бази даних у зручні об’єкти, спрощуючи операції читання, запису та модифікації даних. Це особливо корисно для керування списками завдань, де необхідно швидко звертатися до інформації про користувачів, завдання та їхні стани, а також виконувати фільтрування, сортування та аналіз записів.

Оскільки важливою функціональною частиною застосунку є інтеграція з OpenAI API, для цього використано офіційну бібліотеку OpenAI для Node.js [5]. Вона спрощує надсилання запитів до моделей GPT, які допомагають генерувати поради щодо виконання завдань, аналізувати продуктивність користувача за останні 10 завдань та пропонувати рекомендації з підвищення ефективності. Така інтеграція розширює функціональні можливості застосунку, роблячи його більш цінним для кінцевого користувача.

Розгортання системи

При розгортанні сучасних веб-додатків важливо підібрати платформи, що забезпечуватимуть необхідний рівень надійності, швидкості доступу, а також

зручність підтримки. З метою забезпечення постійної доступності та простого оновлення застосунку, різні частини інфраструктури були розміщені на спеціалізованих хостингових платформах.

Фронтенд-частину, створену за допомогою React.js та TypeScript, розгорнуто на GitHub Pages [27]. Ця платформа пропонує просте підключення до репозиторію GitHub, автоматичне оновлення сайту після кожного push до гілки main, а також безкоштовний хостинг статичних ресурсів, що суттєво оптимізує витрати.

Бекенд-сервіс, який реалізований на Node.js та Express.js, розміщено на Render.com [21]. Цей сервіс надає безкоштовний план з автоматичним призупиненням роботи API через 15 хвилин бездіяльності, що дозволяє знизити витрати на інфраструктуру при невисокому навантаженні. Такий підхід особливо корисний на етапах розробки або для невеликих проєктів із рідшим використанням, водночас зберігаючи можливість швидко відновити роботу застосунку при нових запитах від користувачів.

Для збереження даних про користувачів і їхні завдання обрано базу даних PostgreSQL, розгорнуту на платформі Neon.tech [20]. Цей сервіс забезпечує надійну, масштабовану інфраструктуру, можливість швидкого доступу до даних та інструменти для моніторингу продуктивності. Завдяки цьому забезпечується стабільність роботи бекенду та ефективне керування даними навіть при збільшенні кількості користувачів або обсягу даних.

Система контролю версій та управління репозиторіями

Для розробки та супроводу коду застосовується система контролю версій Git. Вона дає змогу ефективно відстежувати зміни у проєкті, працювати в окремих гілках над новими функціональними можливостями, а потім інтегрувати їх у основну гілку без ризику втрати даних чи порушення цілісності коду. Окрім того, Git полегшує повернення до попередніх версій коду при необхідності аналізу або виправлення помилок, що виникли у минулому.

Для зберігання репозиторіїв використовується платформа GitHub [34]. Вона забезпечує зручний веб-інтерфейс для перегляду коду, історії змін, створення pull request’ів і управління issue. Крім того, GitHub легко інтегрується з багатьма сторонніми інструментами, наприклад сервісами безперервної інтеграції та доставки (CI/CD), що прискорює випуск оновлень і підвищує стабільність системи. Зберігання коду у хмарному середовищі GitHub також підвищує рівень надійності та безпеки, оскільки не потребує утримання власних серверів та резервного копіювання.

Загалом, використання Git та GitHub створює надійну основу для командної роботи, прозорого внесення змін та оперативного реагування на потенційні проблеми, що є критичним чинником успішного впровадження та масштабування проєкту.

3.2 Реалізація клієнтської (фронтенд) частини проєкту

Фронтенд частина була ініціалізована за допомогою інструменту Create React App з підтримкою TypeScript, що забезпечує стандартизовану структуру папок і базові налаштування для швидкого старту розробки. Окрім збірки коду та обробки ресурсів, цей підхід передбачає використання сучасних інструментів, як-от Webpack і Babel, які оптимізують продуктивність та сумісність коду з різними середовищами виконання.

Після ініціалізації проєкту були виконані наступні налаштування та інтеграції:

- Додавання бібліотеки antd: Це дає змогу швидко будувати інтерфейс користувача, використовуючи готові високоякісні UI-компоненти. Завдяки

цьому економиться час на створення базових елементів, а інтерфейс набуває сучасного та привабливого вигляду.

- Налаштування маршрутизації з React Router: Організація окремих сторінок (логін, реєстрація, список завдань, сторінка аналізу продуктивності) забезпечується за допомогою маршрутизації. На етапі розгортання на GitHub pages було використано HashRouter для коректного відображення сторінок. На Рисунку 3.1 показано лістинг компонента, що відповідає за маршрутизацію. Цей підхід забезпечує гнучкість у майбутньому – при зміні платформи розгортання можна легко перейти на BrowserRouter або інший тип маршрутизації.

src > App.tsx > ...

You, 3 weeks ago | 1 author (You)

```
1  import { HashRouter, Route, Routes } from "react-router-dom";
2  import { LoginScreen } from "./components/screens/LoginScreen";
3  import { TodosScreen } from "./components/screens/TodosScreen";
4
5  export const App: React.FC = () => (
6    <HashRouter>
7      <Routes>
8        <Route path={"/"} element={<LoginScreen />} />
9        <Route path={"/todos"} element={<TodosScreen />} />
10     </Routes>
11   </HashRouter>
12 );
13
```

Рисунок 3.1 - лістинг компонента маршрутизації

Реалізація сервісів для взаємодії з бекендом: Для спілкування фронтенду з REST API, створеним на Express.js, було написано спеціальний клієнт HTTP-запитів. На Рисунку 3.2 наведено приклад коду цього клієнта. Він додає до кожного запиту токен авторизації, отриманий при вході користувача в систему (JWT), що зберігається в Local Storage. Токен містить унікальний ідентифікатор

користувача (ID з PostgreSQL), за яким бекенд розпізнає, хто робить запит, і забезпечує захист даних користувачів від несанкціонованого доступу.

src > utils > TS fetchClient.ts > ...

```
type RequestMethod = "GET" | "POST" | "PATCH" | "DELETE";

async function request<T>(url: string, method: RequestMethod = "GET", data: any = null): Promise<T> {
  const token = getToken();
  const config: AxiosRequestConfig = {
    method,
    url: BASE_URL + url,
    headers: {},
  };
  if (data) {
    config.data = data;
    if (config?.headers) {
      config.headers["Content-Type"] = "application/json; charset=UTF-8";
    }
  }
  if (token && config.headers) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  try {
    const response = await axios(config);
    return response.data;
  } catch (error: any) {
    if (error.response) {
      if (error.response.status === 401) {
        removeToken();
        window.location.href = "/";
      }
      throw new Error(error.response.data);
    } else {
      throw new Error(error.message);
    }
  }
}
export const client = {
  get: <T>(url: string) => request<T>(url),
  post: <T>(url: string, data: any) => request<T>(url, "POST", data),
  patch: <T>(url: string, data: any) => request<T>(url, "PATCH", data),
  delete: (url: string) => request(url, "DELETE"),
};
```

Рисунок 3.2 - код клієнту роботи з API.

Реалізація компонентів авторизації, реєстрації та управління завданнями: Створено компоненти для входу до системи, створення нового облікового запису, перегляду, редагування та видалення завдань. На Рисунку 3.3 зображено JSX-код компонента для авторизації та реєстрації. Користувач може переглядати свої завдання, змінювати їхні стани та орієнтовний час виконання. За потреби, за допомогою інтегрованих сервісів, можна отримати поради щодо ефективного виконання поточних чи майбутніх завдань, а також проаналізувати свої останні

10 завдань, щоб зрозуміти, наскільки вчасно вони були виконані і як поліпшити загальну продуктивність.

src > components > LogInScreen > LogInForm.tsx > ...

```tsx
export const LogInForm: React.FC = () => {
  return (
    <form>
      <input
        type="text"
        className="todoapp__new-todo"
        placeholder="Email"
        value={email}
        onChange={(event) => setEmail(event.target.value)}
      />
      <input
        type="password"
        className="todoapp__new-todo"
        placeholder="Password"
        value={password}
        onChange={(event) => setPassword(event.target.value)}
      />
      <Footer
        signUpLoading={signUpLoading}
        signInLoading={signInLoading}
        handleSubmit={handleFormSubmit}
      />
    </form>
  );
};
```

src > components > LogInScreen > Footer.tsx > ...

You, 51 seconds ago | 1 author (You)

```tsx
import { Spin } from "antd";

type Props = {
  handleSubmit: (type: "login" | "signup") => () => void;
  signUpLoading: boolean;
  signInLoading: boolean;
};

export const Footer: React.FC<Props> = ({
  signUpLoading,
  handleSubmit,
  signInLoading
}) => (
  <footer className="todoapp__footer">
    <button
      type="button"
      className="todoapp__clear-completed"
      disabled={signUpLoading || signInLoading}
      onClick={handleSubmit("signup")}
    >
      {signUpLoading ? <Spin /> : "SIGN UP"}
    </button>
    <button
      type="submit"
      className="todoapp__clear-completed"
      disabled={signUpLoading || signInLoading}
      onClick={handleSubmit("login")}
    >
      {signInLoading ? <Spin /> : "LOG IN"}
    </button>
  </footer>
);

```

Рисунок 3.3 - Код JSX-компоненту авторизації/реєстрації

3.3 Реалізація серверної (бекенд) частини проєкту

Для бекенд-частини було створено проєкт на основі Node.js, ініціалізований командою npm init, після чого додано підтримку TypeScript шляхом встановлення необхідних залежностей (typescript, ts-node, @types/node) та налаштування файлу tsconfig.json. Такий підхід уніфікує процес розробки з фронтендом та спрощує рефакторинг коду.

Далі була налаштована інфраструктура Express.js для створення HTTP API. Сервер відповідає за обробку запитів, авторизацію користувачів, зберігання їхніх даних у базі PostgreSQL та здійснення інтеграційних запитів до OpenAI API.

Для роботи з базою даних застосовано ORM Sequelize, що дозволяє визначати моделі для таких сутностей, як користувачі (таблиця users) та завдання (таблиця todos). На Рисунку 3.4 наведено реалізацію моделі User. Модель завдання передбачає зберігання стану (створене, в роботі, завершене), орієнтовного часу виконання, назви й опису. Такий підхід до зберігання даних забезпечує гнучкість у подальшому розширенні функціоналу, наприклад, додаванні нових атрибутів або зв'язків із іншими сутностями.

src > models > TS User.ts > ...

```
11  @Table({
12    tableName: "users",
13    createdAt: true,
14    updatedAt: true,
15  })
16  export class User extends Model {
17    @PrimaryKey
18    @AutoIncrement
19    @Column({
20      type: DataTypes.INTEGER,
21    })
22    id!: number;
23
24    @AllowNull(false)
25    @Column({
26      type: DataTypes.STRING,
27    })
28    email!: string;
29
30    @AllowNull(false)
31    @Column({
32      type: DataTypes.STRING,
33    })
34    password!: string;
35  }
```

Рисунок 3.4 - реалізація моделі User

Інтеграція з OpenAI API потребує наявності API-ключа, який генерується на сайті OpenAI. Як показано на Рисунку 3.5, ключ створюється один раз і має бути збережений у змінних середовища, щоб уникнути несанкціонованого доступу та не розкривати секретну інформацію у відкритому коді. Робота з OpenAI API не є безкоштовною – вартість залежить від обраної моделі та кількості запитів. На початку проєкту було додано 5 доларів США на баланс, а інтерфейс OpenAI надає зручний дашборд для відстеження витрат і ефективності.

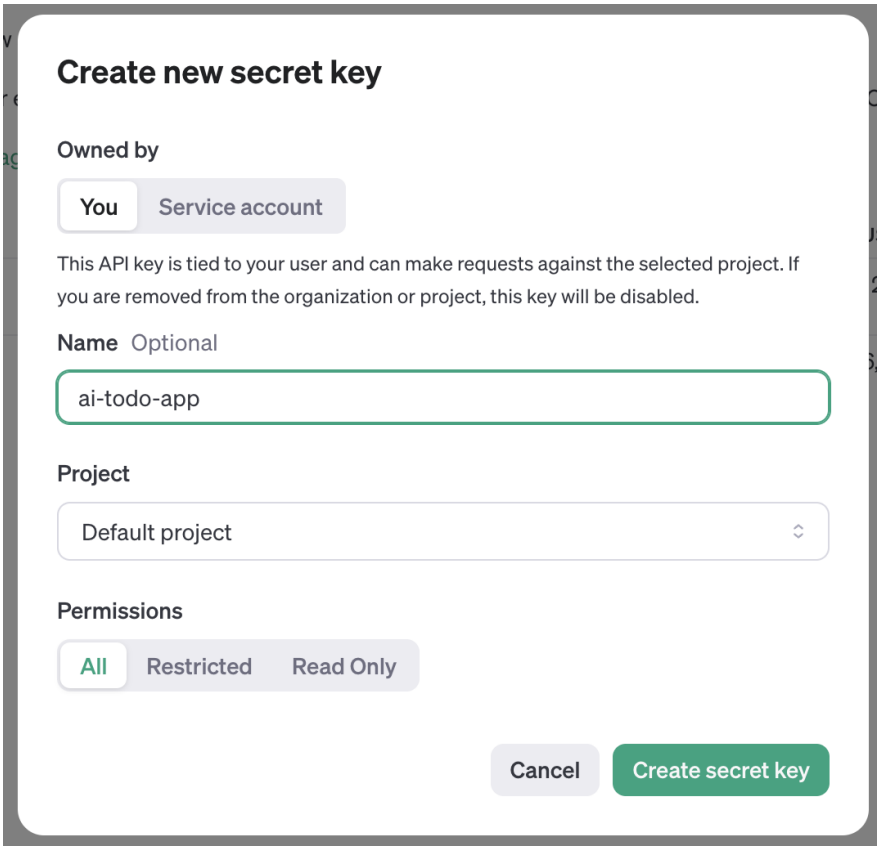


Рисунок 3.5 - модальне вікно створення API-ключа

На Рисунку 3.6 продемонстровано приклад запиту до OpenAI для аналізу останніх N завдань користувача. Серверна частина формує контекст із даних про завдання, надсилає запит до моделі GPT, а у відповіді отримує поради щодо підвищення продуктивності та ефективності роботи. Таким чином, користувач

може отримати персоналізовані рекомендації, основані на історії виконання завдань.

src > controllers > TS User.ts > analyzeLastNTasks > completion > messages > content

```ts
export const analyzeLastNTasks = async (req: Request & any, res: Response) => {
  try {
    const openai = new OpenAI();

    const completion = await openai.chat.completions.create({
      messages: [
        {
          role: "system",
          content: baseAiContext,      You, 7 days ago • added ai analysys
        },
        {
          role: "user",
          content: JSON.stringify(tasksWithTimeSpent),
        },
      ],
      model: "gpt-3.5-turbo-1106",
      response_format: { type: "text" },
    });
    if (completion.choices[0].message.content) {
      return res.status(200).json(completion.choices[0].message.content);
    }
```

src > constants > TS index.ts > ...

You, 1 second ago | 1 author (You)

```ts
export const baseAiContext = `
  You are a productivity assistant that helps users with their todos.
  Given the following todo(s), provide a tip to help the user complete it(them).
  {
    title: string;
    description?: string;
    id: number;
    status: 'created' | 'inProgress' | 'completed';
    createdAt: Date;
    inProgressAt?: Date;
    completedAt?: Date;
    estimatedTime?: number; // in hours
  }
  If the todo is completed, there will be additional field for completion time (in hours).
  If user finished it for more than he thought it would take,
  provide a tip to help him estimate better or best practises for this type of tasks.
`;
```

Рисунок 3.6 - приклад запиту для аналізу останніх N завдань користувача

Для забезпечення захисту ресурсів бекенду було створено middleware для перевірки авторизації (Рисунок 3.7). Кожен запит, окрім запитів на авторизацію та реєстрацію, проходить через це middleware, де перевіряється дійсність JWT токена. Оскільки кожен токен містить унікальний ідентифікатор користувача,

система ідентифікує, хто саме робить запит. Завдяки цьому користувачі не можуть отримати доступ до чужих даних, змінювати чи видаляти завдання інших користувачів. Такий підхід гарантує конфіденційність інформації та цілісність даних у системі.

src > middleware > TS auth.ts > ...

You, 1 second ago | 1 author (You)

```ts
import { NextFunction, Request, Response } from "express";
import jwt from 'jsonwebtoken';
import { DecodedToken } from "../types";

export const auth = (req: Request & any, res: Response, next: NextFunction) => {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) {
    return res.status(401).send('Access Denied');
  }

  try {
    const decodedData = jwt.verify(token, process.env.JWT_SECRET!) as DecodedToken;
    if ((decodedData.exp || 0) * 1000 < Date.now() || !decodedData.id) {
      return res.status(401).send('Session Expired');
    }
    req.userId = decodedData.id;
  } catch (err) {
    console.log(err);
    return res.status(401).send('Invalid Token');
  }

  return next();
};
```

Рисунок 3.7 - Код middleware перевірки авторизації

Отже, бекенд-підсистема, яка включає Express.js, Sequelize, інтеграцію з OpenAI API та middleware для авторизації, створює надійний фундамент для функціонування проєкту. Вона ефективно обробляє запити, керує даними і взаємодіє з зовнішніми сервісами, забезпечуючи комплексність і зручність використання ToDo-застосунку.

3.4. Демонстрація використання створеного програмного продукту

Демонстрацію найдоречніше почати з екрану авторизації/реєстрації. Наразі це просто форма, яка використовується для обох дій спільні поля введення електронної пошти та паролю та для виконання запитів є дві кнопки - авторизуватись (LOG IN) та зареєструватись (SIGN UP). Цей екран зображено на рисунку 3.8

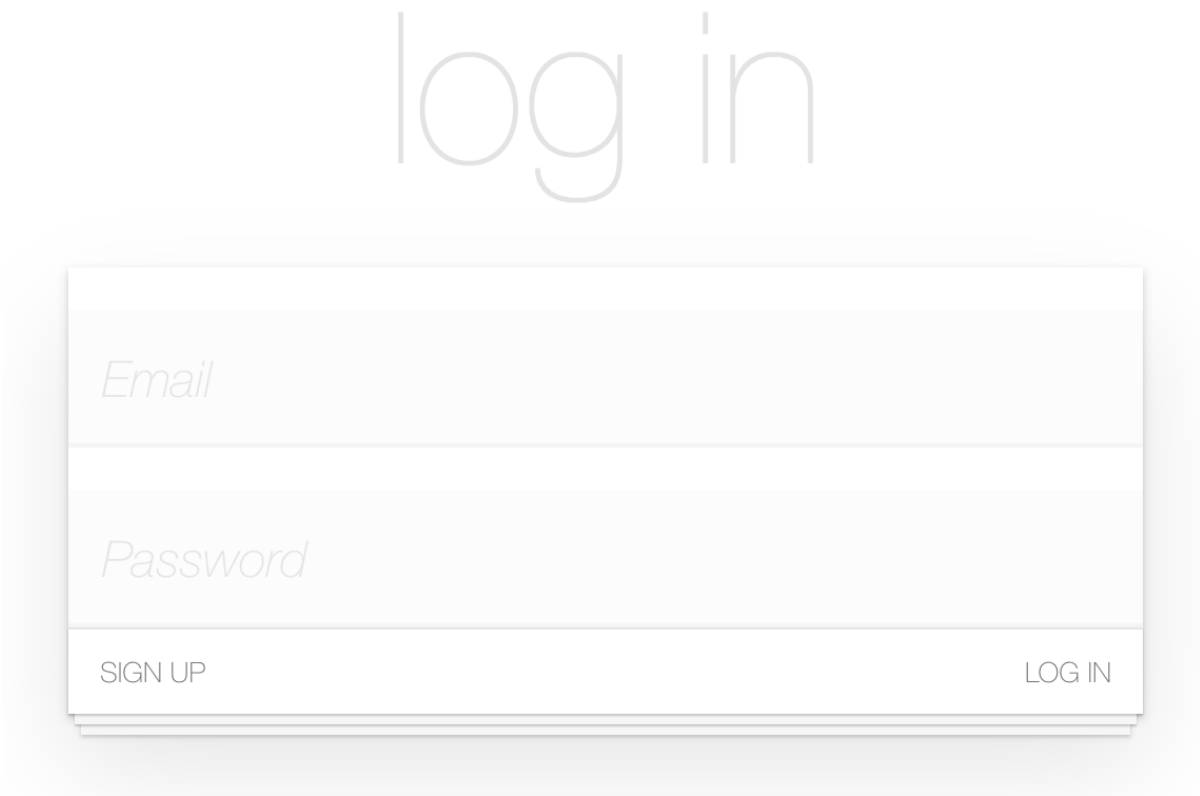


Рисунок 3.8 - Зображення екрану авторизації/реєстрації

Як я вже згадував раніше, в проекті використано бібліотеку Ant Design. Саме тут, для оповіщення користувача про будь-які помилки при авторизації/реєстрації використано сповіщення з цієї бібліотеки. Наприклад, на рискунку 3.9 зображено сповіщення, яке з’являється при спробі зареєструватись

з адресою електронної пошти, яка вже використовується. Ці сповіщення не потрібно стилізувати самостійно, є великий набір їх типів (про успішність, помилку та інші). Виклюються вони також дуже просто, немає необхідності використовувати медоти пошуку елеметів в document та змінювати їх класи, попередньо створивши для них стилі.

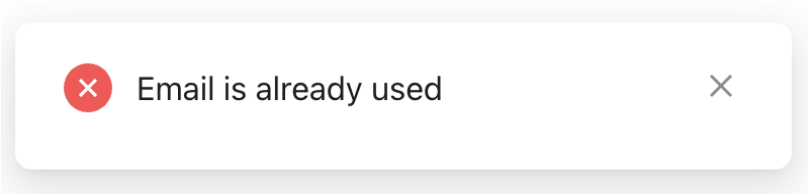
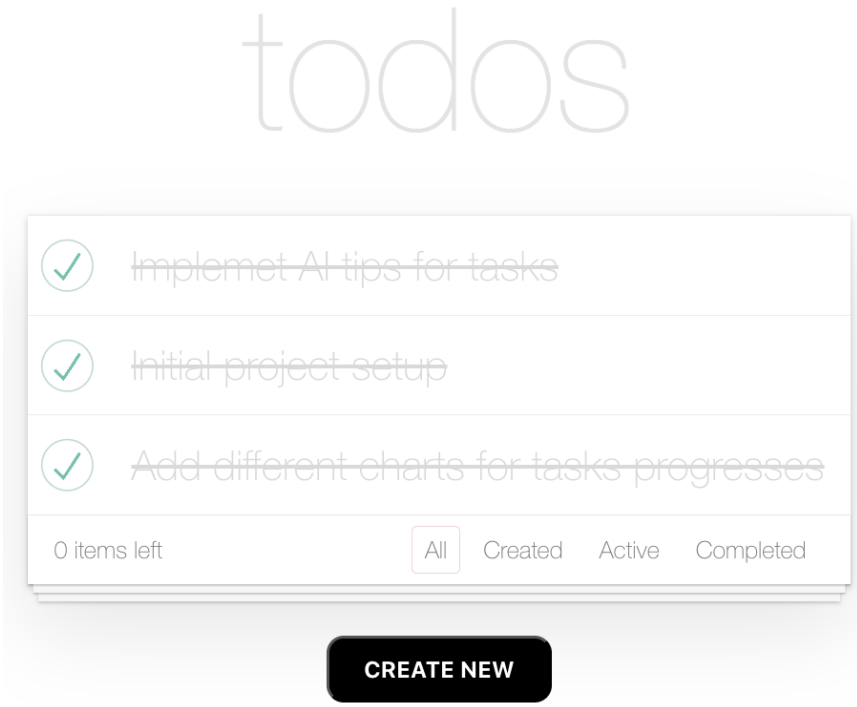
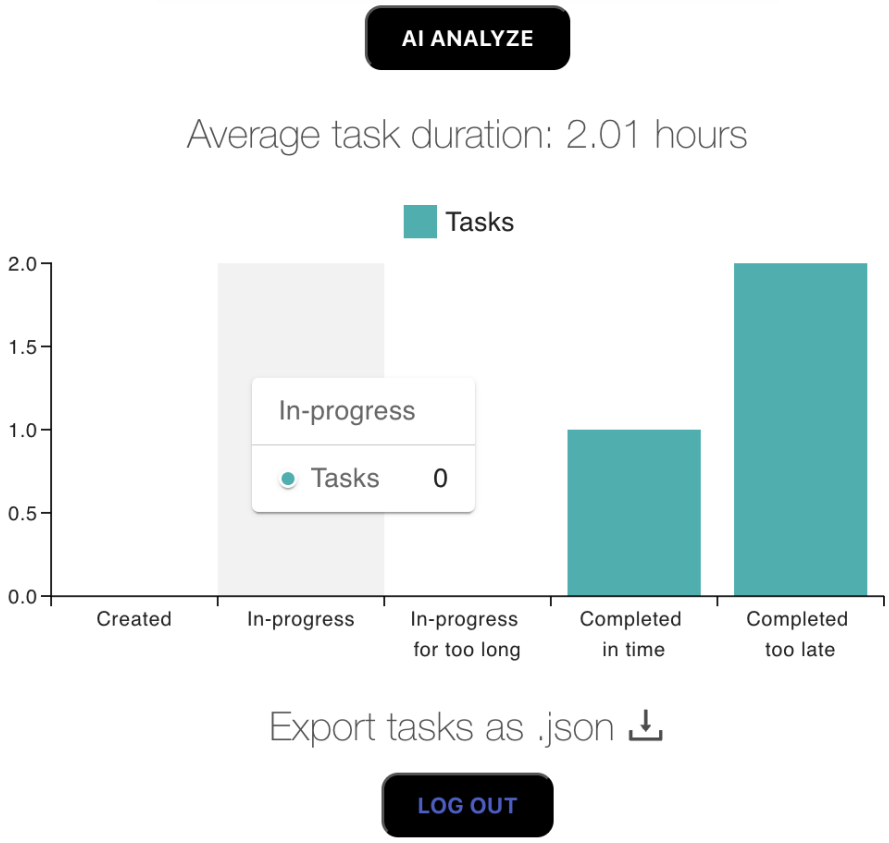


Рисунок 3.9 - Сповіщення Ant Design, використане для повідомлення про помилку при реєстрації

Після успішної реєстрації або авторизації (в першому випадку авторизація також відбується автоматично). Користувача буде переадресовано на наступний екран (рисунок 3.10).



а) Перша частина роботи з завданнями



б) Друга, аналітична частина екрану

Рисунок 3.10 - Головний екран роботи з завданнями

Отже, частина “а” рисунку 3.10 містить елементи роботи з завданнями, на ній є всі елементи необхідні для створення нових завдань, редагування та видалення існуючих, фільтрування їх за статусом. Частина “б” - аналітична, містить інформацію про середню тривалість виконання завдання, стовбчикову діаграму статусу наявних завдань за категоріями виконаних завдань, завдань, які знаходять в процесі виконання, завдань в процесі виконання, які знаходяться в такому статусі довше, ніж розрахунковий час виконання, завдань закінчених вчасно та завдань які було закінчені за довший час, ніж очікувалось користувачем. Також ця частина містить кнопку, яка генерує ШІ-аналіз останніх

десяти завдань користувача, містить кнопку експорту завдань в текстовому форматі JSON та кнопку виходу (“LOG OUT”).

Кнопка створення нового завдання (“CREATE NEW”) відкриває модальне вікно, зображене на рисунку 3.11

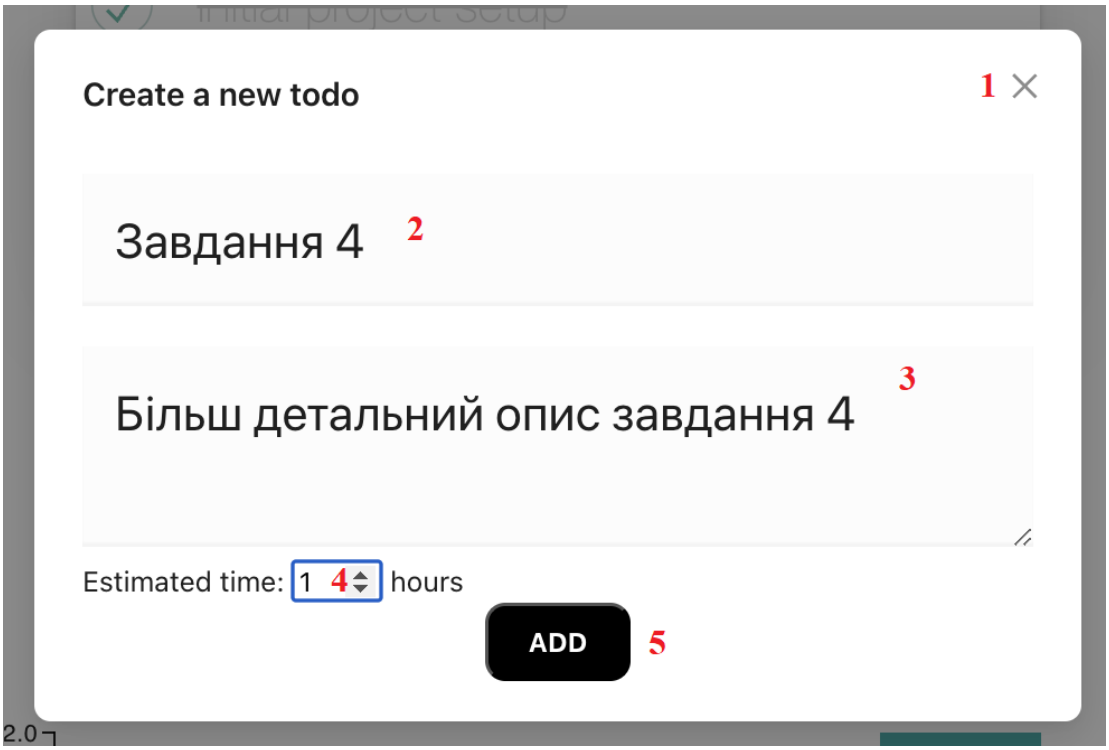


Рисунок 3.11 - Модальне вікно створення нового завдання

1 - кнопка закриття модального вікна

2 - поле введення назви завдання

3 - поле додаткового опису завдання

4 - поле для введення очікуваного часу виконання

5 - кнопка створення завдання

Після створення завдання, воно з’являється в списку зі статусом “створено” (“created”). Це зображено рисунку 3.12

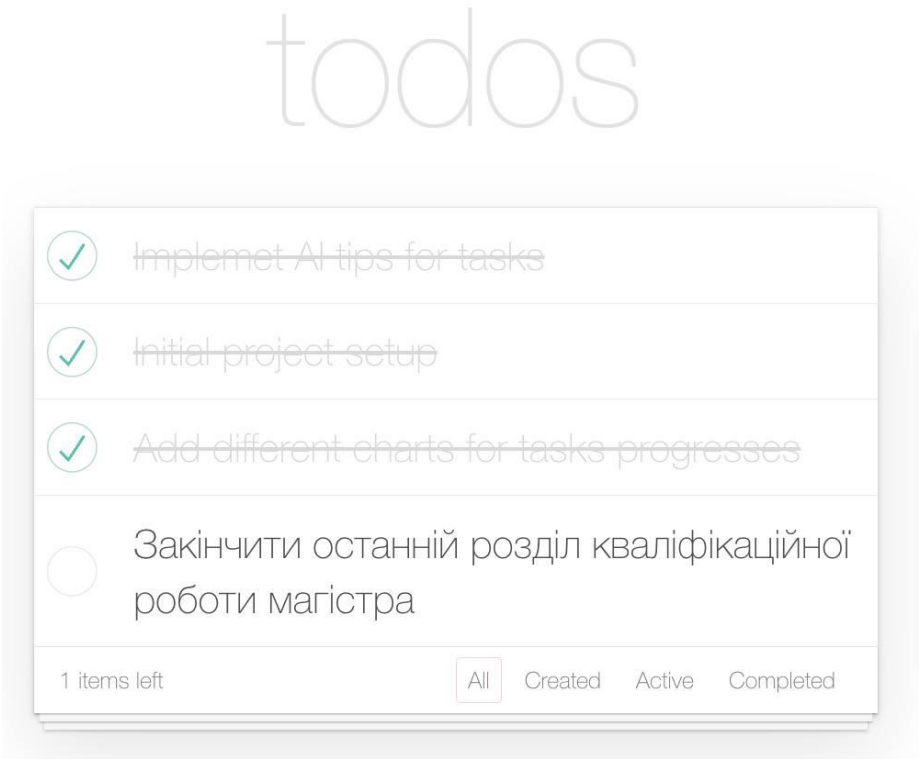


Рисунок 3.12 - Список завдань після створення нового

Подвійне натискання лівої кнопки миші або натискання кнопки редагувати в вигляді піктограми ручки відкриває модальне вікно редагування завдання, яке зображено на рисунку 3.13.

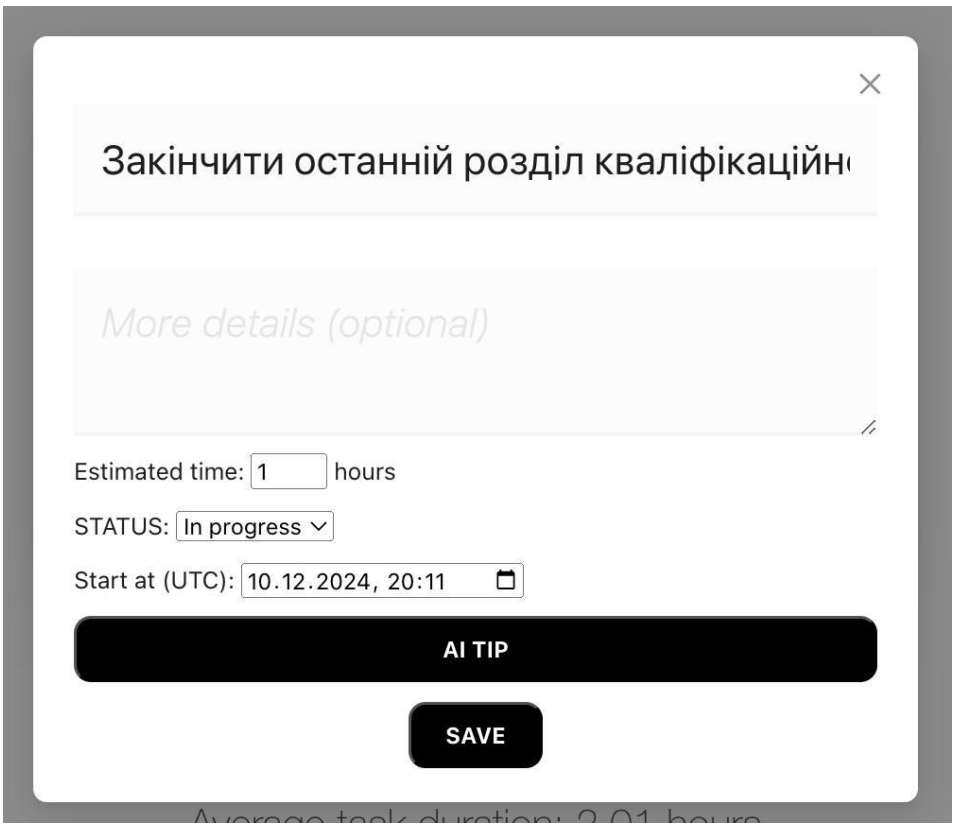


Рисунок 3.13 - Модальне вікно редагування завдання

Отже, я встановив статус завдання “виконується” (“in progress”) та встановив час початку в значення за 1.5 години до актуального часу (при розрахунковому часі виконання в 1 годину. Після цього я зберіг зміни та натиснув кнопку для отримання поради від ШІ. Ця порада зображена на рисунку 3.14.

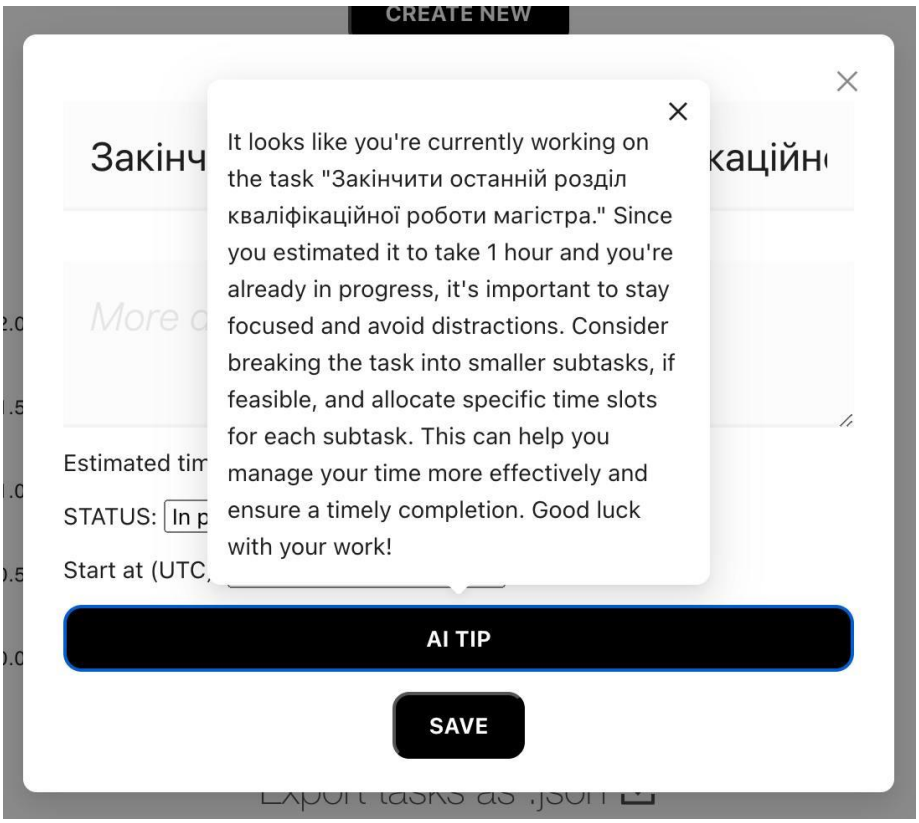


Рисунок 3.14 - Приклад поради до завдання

Так як в даному прикладі завдання виконується довше, ніж це було заплановано і досі не закінчено, ШІ рекомендує користувачеві розбивати подальші завдання на більш дрібні, щоб можна було легше розрахувати час їх виконання і він був більш точним. Враховуючи досить загальну назву і відсутній опис, ця порада здається доречною і може реально допомогти з наступними завданнями і підвищити продуктивність користувача, або хоча би допомогти йому правильно розраховувати час виконання майбутніх завдань.

В свою чергу, кнопка ШІ-аналізу на аналітичній частині головного екрану робить аналіз останніх десяти завдань користувача. Результат такого запиту зображено на рисунку 3.15.

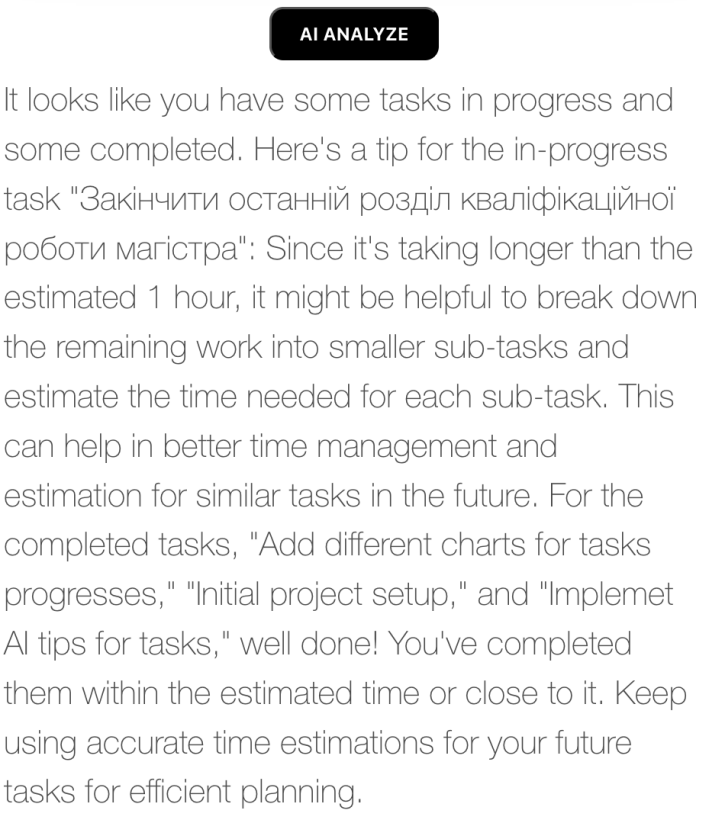


Рисунок 3.15 - Результат аналізу останніх завдань користувача

Отже, ШІ знову повторив пораду про необхідність розбиття завдань на дрібніші, щоб точніше розраховувати час виконання, та додав, що ми також мали проблеми з вчасним завершенням виконання завдань - з трьох закінчених завдань лише одне було виконане вчасно, але там прострочені завдання були близькими до вчасного виконання. Тому роботу застосунку можна вважати коректною, з поставленими завданнями він справляється успішно.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Сьогодні інформаційні технології стали невіддільною частиною нашого повсякдення, а робота за комп’ютером для багатьох є основним видом професійної діяльності. Однак комфорт та ефективність застосування комп’ютерних систем мають поєднуватися з безпекою та дотриманням вимог охорони праці. Особливо це стосується працівників, що зайняті у сфері інформаційних технологій, наприклад, фахівців, які займаються інтелектуальним аналізом продуктивності користувачів у системі управління завданнями за допомогою OpenAI API.

Під час підготовки кваліфікаційної роботи та безпосередньо у процесі розробки методів інтелектуального аналізу продуктивності користувачів системи управління завданнями на основі OpenAI API широко застосовувався персональний комп’ютер та додаткове периферійне обладнання. Тому слід наголосити на необхідності дотримання чинних вимог охорони праці з метою збереження здоров’я користувачів, які беруть участь у розробці та реалізації таких програмних рішень.

Метою даного розділу є висвітлення ключових аспектів планування робочого місця з точки зору безпеки та охорони здоров’я при роботі за комп’ютерною технікою. Розглянуто ергономічні характеристики, умови освітлення, організацію робочого простору, а також важливість дотримання регулярних перерв та фізичної активності. Окрему увагу приділено психосоціальним аспектам робочого оточення та заходам контролю за дотриманням норм безпеки.

Для визначення шкідливих і небезпечних виробничих факторів необхідно проаналізувати дотримання встановлених санітарних норм та правил [ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними

дисплейними терміналами електронно-обчислювальних машин»], що регламентують умови у виробничих приміщеннях і на робочих місцях. Робота з комп’ютером значно впливає на нервово-емоційний стан працівників, пов’язана з напруженим зоровим навантаженням, частими рухами м’язів рук при роботі з клавіатурою та інтенсивною розумовою діяльністю.

Отже, раціональне планування робочого місця сприяє зниженню втоми, підвищенню продуктивності та запобіганню дискомфорту, забезпечуючи оптимальне розташування інструментів, приладів та обладнання, які використовуються при інтелектуальному аналізі даних.

На виробництві проводиться сертифікація робочих місць для об’єктивної оцінки умов праці. Застосовується «Гігієнічна класифікація праці» на основі безпечності, шкідливості виробничого середовища, ступеня напруженості й важкості трудового процесу. Умови праці поділяються на чотири класи:

- Перший клас — оптимальні умови, що сприяють підтримці високого рівня працездатності.
- Другий клас — допустимі умови, при яких показники факторів виробничого середовища та трудового процесу не перевищують нормативів.
- Третій клас — шкідливі умови, при яких можливий негативний вплив на здоров’я працівників або їх нащадків.
- Четвертий клас — небезпечні/екстремальні умови праці.

На основі сертифікаційних даних слід оцінити інтенсивність роботи за такими напрямами:

1. Відповідність обладнання нормативно-технічним вимогам.
2. Аналіз супровідної документації та обсягів виконуваних робіт.
3. Перевірка відповідності площі та об’єму робочого місця чинним нормам.
4. Перегляд стану спеціалізованого обладнання та засобів захисту, а також їх технічного стану.

5. Врахування вимог щодо технологічного процесу, інструментів, обладнання та засобів контролю, аби відповідати стандартам безпеки й охорони праці.

Обладнання та організація робочого місця користувача візуально-дисплейних терміналів повинні відповідати ергономічним нормам, викладеним у [ДСТУ 7299:2013 «Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки»]. Раціональне планування забезпечує оптимальне розміщення робочих інструментів, зниження втоми працівників, підвищення їхньої продуктивності та мінімізацію дискомфорту.

Щоб уникнути ризику ураження електричним струмом, слід правильно розташовувати комп’ютерне обладнання та електрокабелі. Додаткові заходи електробезпеки мають відповідати загальним вимогам пожежної та електробезпеки.

Для забезпечення пожежної безпеки рекомендується прихована електропроводка, використання розеток з вогнестійких матеріалів, регулярне очищення комп’ютерів від пилу, розміщення обладнання на окремих столах та дотримання встановлених відстаней до вікон та інших конструкцій приміщення.

Робочі поверхні та стільці мають враховувати антропометричні особливості користувачів. Стілець повинен забезпечувати підтримку спини та регулюватися по висоті, щоб уникнути тиску на стегна та куприк. Важливо продумати розташування робочих місць з огляду на освітлення, відстань до вікон та використання спеціальних фільтрів для екранів ВДТ. Монітори мають розташовуватися на оптимальній відстані від очей, а лазерне випромінювання принтерів та іншого периферійного обладнання — відповідати встановленим санітарно-гігієнічним нормам.

ІТ-фахівці, що впроваджують інтелектуальні алгоритми аналізу продуктивності користувачів у системі управління завданнями із залученням

OpenAI API, постійно працюють із підключеним до електромереж обладнанням. Тому необхідно ретельно контролювати безпечність його підключення.

Нормативна база, яка регулює умови праці користувачів комп'ютерної техніки, визначає вимоги щодо створення оптимального робочого середовища. Дотримання цих норм є ключовим для збереження здоров'я при роботі за ПК. В Україні продовжується вдосконалення національних нормативних актів у цій сфері, а найповнішим на сьогодні документом є «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин» (ДСанПіН 3.3.2.007–98) [36].

Врахування усіх цих заходів забезпечить комфортні умови праці та сприятиме збереженню здоров'я осіб, залучених до інтелектуального аналізу продуктивності користувачів під час роботи із сучасними комп'ютерними системами, зокрема в рамках проєктів, де застосовуються інструменти OpenAI API для підвищення ефективності управління завданнями.

4.2 Безпека в надзвичайних ситуаціях

Забезпечення стійкості роботи систем управління завданнями в умовах надзвичайних ситуацій (НС) є критично важливим елементом для досягнення гуманітарної безпеки, підтримання соціально-економічної стабільності та захисту інформаційних ресурсів. Сучасні інформаційно-комунікаційні системи, що використовуються для керування завданнями в сфері цивільного захисту,

все частіше піддаються впливу як техногенних, так і природних загроз. Однією з ключових складових підвищення їхньої надійності та готовності до дій у складних умовах є моделювання уразливості компонентів таких систем до деструктивних факторів, зокрема пожеж та хімічного забруднення.

Моделювання уразливості системи управління завданнями до пожеж

Пожежі належать до найбільш загрозливих чинників для критичних цифрових систем, зокрема серверних приміщень, центрів обробки даних, комплексів мережевого та телекомунікаційного устаткування. Висока

температура, задимлення, а також безпосередній контакт із полум’ям створюють умови, за яких обладнання може зазнати фізичного пошкодження, виходу з ладу або повного знищення. За даними досліджень та рекомендацій, наведених у відповідних навчальних посібниках [28, 29], ефективне подолання наслідків пожежних інцидентів вимагає використання спеціальних захисних сховищ із підвищеною вогнестійкістю, систем автоматичного пожежогасіння (наприклад, газових або водяних спринклерних систем), а також сучасних комплексів димовидалення та вентиляції.

До основних елементів моделювання стійкості системи до пожеж належать:

1. Аналіз розташування системних компонентів щодо потенційних осередків займання. Визначення місць із підвищеним ризиком виникнення пожежі допомагає оптимізувати розміщення серверів, комунікаційних шаф та критичного мережевого обладнання.

2. Вибір вогнестійких матеріалів і конструкційних рішень, включно з негорючими кабельними трасами, термостійкими корпусами, теплоізоляційними панелями та покриттями, що знижують ризик поширення вогню.

3. Наявність автоматизованих систем моніторингу в реальному часі, які здатні оперативно виявляти підвищення температури, появу диму та активувати системи пожежогасіння ще до критичного поширення полум’я. Сучасні системи раннього виявлення можуть бути інтегровані у централізовані центри управління, що дає змогу персоналу реагувати миттєво.

4. Стратегії локалізації пожежі, які охоплюють розмежування приміщень на протипожежні зони та створення фізичних бар’єрів, котрі перешкоджають швидкому розповсюдженню вогню до ключових вузлів системи.

5. Резервування критичних компонентів, що дозволяє у разі пошкодження обладнання миттєво переключити навантаження на резервні системи чи віддалені майданчики, мінімізуючи час простою та втрати даних.

В Україні регулятивна база у цій сфері постійно оновлюється, впроваджуючи актуальні норми та вимоги пожежної безпеки. Зокрема, діють оновлені «Правила пожежної безпеки в України»; (НАПБ А.01.001-2014), затверджені наказом МВС № 1417 від 30 грудня 2014 року, із внесеними змінами наказом № 474 від 11 липня 2024 року [35]. Дані нормативні документи визначають вимоги до протипожежних систем, евакуаційних шляхів, встановлення систем раннього виявлення диму та пожежогасіння. Крім того, у Державних будівельних нормах ДБН В.1.1-7:2016 «Пожежна безпека об’єктів будівництва. Загальні вимоги»; [33] наведено технічні критерії щодо проєктування будівель, кімнат та приміщень з урахуванням факторів пожежної небезпеки.

Стійкість системи до хімічного забруднення

Хімічне забруднення, спричинене викидами токсичних речовин, корозійно-активних сполук чи промисловими аваріями, може критично вплинути на цілісність обладнання та функціонування систем управління завданнями. Вплив агресивних хімічних компонентів здатний зумовити руйнування зовнішніх оболонок, пошкодження друкованих плат, деградацію ізоляційних матеріалів та навіть перманентний вихід з ладу електроніки.

Для мінімізації ризиків, пов’язаних із хімічним забрудненням, рекомендується:

1. Застосування захисних покриттів на критичних компонентах систем, наприклад, антикорозійних лаків та спеціальних полімерних матеріалів, стійких до контакту з отруйними речовинами.

2. Забезпечення ефективної вентиляції та фільтрації повітря у приміщеннях, де розташовані серверні стійки, мережеве обладнання та

апаратура зв’язку. Системи фільтрації можуть утримувати більшість токсичних частинок, знижуючи їхній негативний вплив.

3. Регулярний моніторинг стану повітря на наявність небезпечних концентрацій хімічних речовин. Встановлення сенсорних систем дає змогу оперативно виявити появу загрозливих речовин, які можуть призвести до пошкодження обладнання або створити ризик для здоров’я персоналу.

4. Автоматизовані системи оповіщення та локалізації хімічних викидів, що у разі критичного перевищення допустимих концентрацій шкідливих речовин можуть активувати аварійні протоколи – перекрити вентиляцію, запустити системи очистки чи навіть здійснити переміщення обладнання у безпечніші зони.

5. Забезпечення навчання та інструктажів для персоналу, а також оснащення їх індивідуальними засобами захисту (спеціальні респіратори, захисні костюми), що дозволяють своєчасно реагувати на хімічні інциденти без ризику для життя та здоров’я.

У рамках моделювання даного виду уразливості варто оцінити:

Відстань та просторове розташування обладнання від можливих джерел забруднення (промислові підприємства, склади хімічних реагентів, транспортні магістралі).

Ступінь герметизації компонентів системи, що забезпечує мінімізацію проникнення отруйних речовин усередину пристроїв.

Рівень підготовленості персоналу, зокрема наявність протоколів екстреного реагування, готовність обслуговуючого штату до оперативних дій при хімічній аварії.

Україна імплементувала відповідні технічні регламенти та норми, зокрема Технічний регламент класифікації небезпечності, маркування та пакування хімічної продукції, що відповідає європейському Регламенту CLP та міжнародній системі GHS [34]. Також встановлено гранично допустимі концентрації шкідливих речовин у повітрі робочої зони відповідно до наказу МОЗ № 813 від 10 травня 2024 року [35]. Ці регламентні документи визначають

стандарти, яких мають дотримуватися підприємства та організації, аби забезпечити безпеку персоналу, обладнання та інформаційних систем.

Інтегровані заходи безпеки

Для забезпечення надійної та безперервної роботи систем управління завданнями у разі надзвичайних ситуацій доцільно використовувати комплексний підхід, поєднуючи протипожежні заходи із захистом від хімічних загроз. Це дозволяє створити багаторівневу систему безпеки, що мінімізує уразливість та забезпечує стійкість у найкритичніших сценаріях.

Ключові інтегровані кроки:

Розробка планів евакуації персоналу та резервного копіювання даних, що забезпечує швидку реакцію на надзвичайні події, дозволяючи мінімізувати втрати інформації та убезпечити працівників.

Встановлення автоматичних систем діагностики та реагування, здатних виявляти початкові ознаки пожежі або хімічного забруднення, активацію яких можна здійснювати як локально, так і дистанційно.

Проведення регулярних навчань і тренувань персоналу щодо дій у разі виникнення пожежі, хімічного викиду чи іншої НС. Така підготовка сприяє формуванню алгоритмічних навичок швидкого та правильного реагування.

Використання комбінованих матеріалів та інженерних рішень, що мають підвищену стійкість до вогню, корозії та хімічного впливу. Наприклад, спеціальні термостійкі та хімічно інертні композити можуть захистити корпуси обладнання, кабелі та роз’єми.

Інтеграція системи управління безпекою із зовнішніми структурами, такими як державні служби з надзвичайних ситуацій, пожежні та хімічні лабораторії, промислові підприємства-сусіди. Координація дій із цими організаціями дозволяє отримувати актуальну інформацію про можливі загрози та оперативно реагувати на них.

Моделювання уразливості систем управління завданнями до пожеж та хімічного забруднення є важливим інструментом попередження критичних

наслідків. Завдяки цьому підходу вдається своєчасно виявляти слабкі місця у системі, удосконалювати її архітектуру, впроваджувати новітні технологічні рішення та підвищувати рівень підготовки персоналу. Реалізація запропонованих заходів у комплексі із дотриманням нормативних вимог забезпечує мінімізацію ризиків втрати даних, тривалих простоїв, а також негативних впливів на здоров’я та безпеку працівників. Зрештою, це сприяє надійному функціонуванню систем управління завданнями навіть в умовах найскладніших надзвичайних ситуацій, відповідаючи сучасним вимогам гуманітарної безпеки та сталого розвитку.

Заходи та рекомендації, наведені у цьому розширеному підрозділі, ґрунтуються на положеннях навчальних посібників [28, 29] та відповідають чинним нормам і стандартам у галузі пожежної безпеки, хімічного захисту, а також технічного регулювання у сфері цивільного захисту і промислової безпеки, включно з документами [30–33].

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було створено клієнт-серверний ToDo-застосунок, у якому звичайна функціональність управління завданнями поєднується з елементами інтелектуального аналізу продуктивності користувачів. Використання технологій React.js і TypeScript для фронтенду, а також Node.js, Express.js, PostgreSQL і ORM Sequelize для бекенду, дозволило побудувати гнучку, масштабовану та продуктивну систему. Інтеграція OpenAI API забезпечила новий рівень персоналізації та аналітики, даючи змогу надавати користувачам рекомендації на основі аналізу їхньої історії виконання завдань.

Результатом роботи став повноцінний прототип системи, що дозволяє користувачам не лише планувати та відстежувати свої завдання, а й отримувати рекомендації щодо підвищення ефективності роботи, оптимізації розподілу часу та покращення прогнозів тривалості виконання завдань. Впровадження JWT-авторизації забезпечило конфіденційність даних і захист від несанкціонованого доступу, а використання хмарних платформ (GitHub Pages, Render.com, Neon.tech) дало змогу швидко розгорнути та підтримувати систему з мінімальними витратами на інфраструктуру.

Створена система є прикладом того, як сучасні веб-технології, аналітичні інструменти та штучний інтелект можуть поєднуватися для вирішення практичних завдань підвищення продуктивності. Цей підхід має потенціал до подальшого вдосконалення: можна розширювати функціонал, впроваджувати інші види інтеграцій і моделі ШІ, підвищувати точність рекомендацій та адаптувати систему під потреби різноманітних команд і організацій. Таким чином, результати, досягнуті в рамках роботи, можуть слугувати основою для розробки більш складних і ефективних інтелектуальних рішень для управління робочими процесами та підвищення загальної продуктивності користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Petterson R., Smith J. Task management systems and productivity improvement // IEEE Transactions on Systems, Man, and Cybernetics. – 2022.
2. Brown T. et al. Language Models are Few-Shot Learners // OpenAI Research Paper. – 2020.
3. Lin S., Hsieh J. A Survey of Natural Language Processing Techniques in Task Management Systems // ACM Computing Surveys. – 2021.
4. Nielsen J. Usability Engineering. – Morgan Kaufmann, 1994.
5. Official OpenAI API Documentation [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs.
6. Bernard A., Lewis D. Implementing AI-driven tools in project management // Journal of Applied Artificial Intelligence. – 2023.
7. Johnson K. AI in Productivity Tools: A Comparative Study. – Springer, 2022.
8. Стефанишин В., Стефанишин І., Пастух О., Яцишин В., Якименко І. Точність програмного та апаратного забезпечення комп’ютерних систем для взаємодії людина–машина // CEUR Workshop Proceedings. – 2024. – Т. 3842. – С. 178–183.
9. Стефанишин І., Пастух О., Стефанишин В., Баран І., Бойко І. Робастність алгоритмів штучного інтелекту для нейрокомп’ютерних інтерфейсів на основі програмно-апаратних технологій // CEUR Workshop Proceedings. – 2024. – Т. 3742. – С. 137–149.
10. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2003.
11. Ivan Dam K. M. Task Management: Theory and Practice. – Amsterdam University Press, 2016.
12. RFC 7519: JSON Web Token (JWT) // Internet Engineering Task Force. – 2015.
13. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016.

14. Wikipedia LLM [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Large_language_model.
15. React.js Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/.
16. Doyle J., Saunders P. Task Management in a Modern Workplace. – London: TechBooks, 2021.
17. Shapiro D. AI in Productivity Applications: Current Trends // Journal of Intelligent Systems. – 2023.
18. Gupta R. Enhancing User Performance Through Data Analysis // ACM Transactions. – 2022.
19. O'Reilly K. Building Scalable ToDo Applications. – New York: DevPress, 2020.
20. Neon.tech Documentation [Електронний ресурс]. – Режим доступу: https://neon.tech. – (Accessed in 2024).
21. Render.com Documentation [Електронний ресурс]. – Режим доступу: https://render.com. – (Accessed in 2024).
22. UML Wikipedia page [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Unified_Modeling_Language.
23. Microsoft. TypeScript Documentation [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/.
24. Ant Design. Ant Design Components [Електронний ресурс]. – Режим доступу: https://ant.design/.
25. Node.js. Official Website [Електронний ресурс]. – Режим доступу: https://nodejs.org/.
26. Sequelize. Sequelize ORM Documentation [Електронний ресурс]. – Режим доступу: https://sequelize.org/.
27. GitHub Pages. Documentation [Електронний ресурс]. – Режим доступу: https://pages.github.com/.
28. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання

«БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / В.С. Стручок. – Тернопіль: ФОП Паляниця В. А., 156 с. – Режим доступу: https://elartu.tntu.edu.ua/handle/lib/39196.

29. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / автор-укладач В.С. Стручок. – Тернопіль: ФОП Паляниця В. А., 156 с. – Режим доступу: http://elartu.tntu.edu.ua/handle/lib/39424.
30. НАПБ А.01.001-2014 "Правила пожежної безпеки в Україні" [Електронний ресурс]. – Останні зміни внесені наказом МВС № 474 від 11 липня 2024 року. – Режим доступу: http://zakononline.com.ua.
31. Державні будівельні норми України ДБН В.1.1-7:2016 "Пожежна безпека об'єктів будівництва. Загальні вимоги" [Електронний ресурс]. – Режим доступу: http://e-construction.gov.ua.
32. Технічний регламент класифікації небезпечності, маркування та пакування хімічної продукції [Електронний ресурс]. – Режим доступу: http://kmu.gov.ua.
33. Наказ МОЗ № 813 від 10 травня 2024 року "Гранично допустимі концентрації шкідливих речовин у повітрі" [Електронний ресурс]. – Режим доступу: http://zakononline.com.ua.
34. ДСанПІН 3.3.2.007-98 [Електронний ресурс]. – 1998. – Режим доступу: https://zakon.rada.gov.ua/rada/show/v0007282-98.

ДОДАТКИ

ДОДАТОК А

Лістинг коду

Серверна частина:

src/controllers/Todos.ts:

```
import { Request, Response } from "express";
import { TodoServices } from "../services/Todos";
import { Todo } from "../types";
import OpenAI from "openai";
import { baseAiContext } from "../constants";

export const getTodos = async (req: Request & any, res: Response)
=> {
  try {
    const id = req.userId as number;
    const todos = await TodoServices.getTodosByUserId(+id);
    res.send(todos);
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
};

export const addTodo = async (req: Request & any, res: Response) =>
{
  try {
    const todo = req.body as Todo;
    const id = req.userId as number;
    const response = await TodoServices.addTodo({ ...todo, userId:
id });
    res.send(response.dataValues);
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
```

```
};

export const deleteTodo = async (req: Request & any, res: Response)
=> {
  try {
    const id = req.params.id;
    const userId = req.userId as number;
    const todo = await TodoServices.getTodoById(+id);
    if (todo?.userId === userId) {
      await TodoServices.deleteTodoById(+id);
      return res.send("Deleted Successfully");
    }

    res.status(403).send("Forbidden");
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
};

export const patchTodo = async (req: Request & any, res: Response)
=> {
  try {
    const id = req.params.id;
    const userId = req.userId as number;
    const todo = await TodoServices.getTodoById(+id);
    if (todo?.userId === userId) {
      await TodoServices.updateTodoById(+id, req.body as
Partial<Todo>);
      const patchedTodo = await TodoServices.getTodoById(+id);
      return res.send(patchedTodo);
    }

    res.status(403).send("Forbidden");
  } catch (error) {
    console.error(error);
    res.status(500).send("Internal Server Error");
  }
```

```
};

export const getTipForTodo = async (req: Request & any, res:
Response) => {
  const id = req.params.id as number;
  const todo = await TodoServices.getTodoById(+id);
  if (!todo) {
    return res.status(404).json({ message: "Todo was not found" });
  }
  let timeTakenInHours: number | undefined;
  if (
    todo.status === "completed" &&
    todo.estimatedTime &&
    todo.inProgressAt &&
    todo.completedAt
  ) {
    timeTakenInHours =
      (new Date(todo.completedAt).getTime() -
        new Date(todo.inProgressAt).getTime()) /
      1000 /
      60 /
      60;
  }
  try {
    const openai = new OpenAI();

    const completion = await openai.chat.completions.create({
      messages: [
        {
          role: "system",
          content: baseAiContext,
        },
        {
          role: "user",
          content: JSON.stringify({ ...todo, timeTakenInHours }),
        },
      ],
      model: "gpt-3.5-turbo-1106",
```

```
            response_format: { type: "text" },
          });
          if (completion.choices[0].message.content) {
            return
res.status(200).json(completion.choices[0].message.content);
          }

          return res
            .status(500)
            .json({ message: "Something went wrong getting response" });
        } catch (error: any) {
          return res.status(500).json({
            error: error?.message || "Internal Server Error",
          });
        }
      };

      src/controllers/User.ts:
      import { Request, Response } from "express";
      import bcrypt from "bcrypt";
      import { SignUpRequest } from "../types";
      import { UserServices } from "../services/User";
      import jwt from "jsonwebtoken";
      import { TodoServices } from "../services/Todos";
      import OpenAI from "openai";
      import { baseAiContext } from "../constants";

      export const SignUp = async (
        req: Request<any, any, SignUpRequest>,
        res: Response
      ) => {
        try {
          const { email, password } = req.body;
          if (await UserServices.checkIfEmailUsed(email)) {
            return res.status(400).send("Email is already used");
          }
          const saltRounds = 10;
          const hashedPassword = await bcrypt.hash(password, saltRounds);
```

```
        await UserServices.insertUser({ email, password: hashedPassword
});
        res.sendStatus(201);
      } catch (error) {
        console.error(error);
        res.status(500).send("Internal Server Error");
      }
    };

    export const getUser = async (
      req: Request<any, any, { id: number }>,
      res: Response
    ) => {
      try {
        const { id } = req.params;
        const user = await UserServices.getUser(+id);
        res.send(user);
      } catch (error) {
        console.error(error);
        res.status(500).send("Internal Server Error");
      }
    };

    export const SignIn = async (
      req: Request<any, any, { email: string; password: string }>,
      res: Response
    ) => {
      try {
        const { email, password } = req.body;
        const user = await UserServices.getUserByEmail(email);
        if (!user) {
          return res.status(400).send("User not found");
        }
        const isPasswordCorrect = await bcrypt.compare(password,
user.password);
        if (!isPasswordCorrect) {
          return res.status(400).send("Incorrect Password");
        }
```

```
        const jwtToken = jwt.sign(
          { id: user.id },
          process.env.JWT_SECRET || "secret",
          { expiresIn: "1h" }
        );


        res.send(jwtToken);
      } catch (error) {
        console.error(error);
        res.status(500).send("Internal Server Error");
      }
    };

    export const analyzeLastNTasks = async (req: Request & any, res:
Response) => {
      const id = req.userId as number;
      const n = req.query.n as string | undefined;
      if (!n) {
        return res.status(400).send("n is required");
      }
      const tasks = await TodoServices.getLastNTasks(+n, +id);
      const tasksWithTimeSpent = tasks.map((task) => {
        const copy = JSON.parse(JSON.stringify(task));
        let timeSpent: number | undefined; // in hours

        if (task.inProgressAt && task.completedAt) {
          timeSpent =
            (new Date(task.completedAt).getTime() -
              new Date(task.inProgressAt).getTime()) /
            1000 /
            60 /
            60;
        }

        if (timeSpent) {
          copy.timeSpentInHours = timeSpent;
```

```
        }

        return copy;
      });

      try {
        const openai = new OpenAI();

        const completion = await openai.chat.completions.create({
          messages: [
            {
              role: "system",
              content: baseAiContext,
            },
            {
              role: "user",
              content: JSON.stringify(tasksWithTimeSpent),
            },
          ],
          model: "gpt-3.5-turbo-1106",
          response_format: { type: "text" },
        });
        if (completion.choices[0].message.content) {
          return
res.status(200).json(completion.choices[0].message.content);
        }

        return res
          .status(500)
          .json({ message: "Something went wrong getting response" });
      } catch (error: any) {
        return res.status(500).json({
          error: error?.message || "Internal Server Error",
        });
      }
    };
```

src/db/getDb.ts:

```
import { configDotenv } from "dotenv";
import { Sequelize } from "sequelize-typescript";
import { models } from "../models";
configDotenv();
const connectionString = process.env.POSTGRESQL_CONNECTION_STRING
|| "";

export function getDb() {
  try {
    const db = new Sequelize(connectionString, {
      models,
      dialectOptions: {
        ssl: true
      }
    });
    db.authenticate();
    console.log(
      "Connection to the database has been established
successfully."
    );
    return db;
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
}

src/db/initDb.ts:
import { getDb } from "./getDb";
import { models } from "../models";

const initDb = async () => {
  const db = getDb();

  db?.addModels(models);
  await Promise.all(models.map((model) => model.sync({ alter: true
})));
};
```

```
initDb();

src/index.ts:
import express from 'express';
import { router } from './routes';
import cors from 'cors';
import { getDb } from './db/getDb';

const main = () => {
  const app = express();
  const port = 5001;
  getDb();


  app.use(cors({
    origin: process.env.REACT_APP_FRONTEND_URL,
    credentials: true,
  }));

  app.use(express.json());
  app.use(router);

  app.listen(port, () => {
    console.log(`Server started at http://localhost:${port}`);
  });
}

main();

src/middleware/auth.ts:
import { NextFunction, Request, Response } from "express";
import jwt from "jsonwebtoken";
import { DecodedToken } from "../types";

export const auth = (req: Request & any, res: Response, next:
NextFunction) => {
  const token = req.header("Authorization")?.split(" ")[1];
  if (!token) {
```

```
        return res.status(401).send("Access Denied");
      }

      try {
        const decodedData = jwt.verify(
          token,
          process.env.JWT_SECRET!
        ) as DecodedToken;
        if ((decodedData.exp || 0) * 1000 < Date.now() ||
!decodedData.id) {
          return res.status(401).send("Session Expired");
        }
        req.userId = decodedData.id;
      } catch (err) {
        console.log(err);
        return res.status(401).send("Invalid Token");
      }

      return next();
    };

    src/models/Todo.ts:
    import { DataTypes } from "sequelize";
    import {
      AllowNull,
      Column,
      PrimaryKey,
      Table,
      Model,
      AutoIncrement,
      ForeignKey,
    } from "sequelize-typescript";
    import { User } from "./User";

    @Table({
      tableName: "todos",
      createdAt: true,
      updatedAt: true,
```

```
})
export class Todo extends Model {
  @PrimaryKey
  @AutoIncrement
  @Column({
    type: DataTypes.INTEGER,
  })
  id!: number;

  @AllowNull(false)
  @Column({
    type: DataTypes.STRING,
  })
  title!: string;

  @AllowNull(false)
  @Column({
    type: DataTypes.STRING,
  })
  description!: string;

  @AllowNull(false)
  @ForeignKey(() => User)
  @Column({
    type: DataTypes.INTEGER,
  })
  userId!: number;

  @AllowNull(false)
  @Column({
    type: DataTypes.STRING,
  })
  status!: string;

  @AllowNull(true)
  @Column({
    type: DataTypes.DATE,
  })
```

```
  inProgressAt!: Date;

  @AllowNull(true)
  @Column({
    type: DataTypes.DATE,
  })
  completedAt!: Date;

  @AllowNull(false)
  @Column({
    type: DataTypes.FLOAT,
  })
  estimatedTime!: number;
}

src/models/User.ts:
import { DataTypes } from "sequelize";
import {
  AllowNull,
  Column,
  PrimaryKey,
  Table,
  Model,
  AutoIncrement,
} from "sequelize-typescript";

@Table({
  tableName: "users",
  createdAt: true,
  updatedAt: true,
})
export class User extends Model {
  @PrimaryKey
  @AutoIncrement
  @Column({
    type: DataTypes.INTEGER,
  })
  id!: number;
```

```
    @AllowNull(false)
    @Column({
      type: DataTypes.STRING,
    })
    email!: string;

    @AllowNull(false)
    @Column({
      type: DataTypes.STRING,
    })
    password!: string;
  }

  src/models/index.ts:
  import { User } from "./User";
  import { Todo } from "./Todo";

  export const models = [User, Todo];

  src/routes/index.ts:
  import { Router } from "express";
  import { analyzeLastNTasks, getUser, SignIn, SignUp } from
"../controllers/User";
  import { addTodo, deleteTodo, getTipForTodo, getTodos, patchTodo }
from "../controllers/Todos";
  import { auth } from "../middleware/auth";

  export const router = Router();

  router.route("/signup").post(SignUp);
  router.route("/signin").post(SignIn);

  // Auth routes only:
  router.use(auth);
  router.route("/get-user-info/:id").get(getUser);
  router.route("/todos").get(getTodos).post(addTodo);
  router.route("/todos/:id").delete(deleteTodo).patch(patchTodo);
```

```
router.route("/todos/tip/:id").get(getTipForTodo);
router.route("/user/anylyzeTodos").get(analyzeLastNTasks)

src/services/Todos.ts:
import { Todo } from "../models/Todo";
import { Todo as TodoType} from "../types";

const getTodosByUserId = (userId: number) => {
  return Todo.findAll({ where: { userId }, order: [['id', 'ASC']]
});
};

const deleteTodoById = (id: number) => {
  return Todo.destroy({ where: { id } });
};

const patchTodoById = (id: number, todo: Partial<Todo>) => {
  return Todo.update(todo, { where: { id } });
};

const addTodo = (todo: TodoType) => {
  return Todo.create(todo);
};

const getTodoById = (id: number) => {
  return Todo.findOne({ where: { id } });
}

const updateTodoById = (id: number, todo: Partial<Todo>) => {
  return Todo.update(todo, { where: { id } });
}

const getLastNTasks = (n: number, userId: number) => {
  return Todo.findAll({
    where: { userId },
    order: [["createdAt", "DESC"]],
    limit: n,
  });
```

```
};

export const TodoServices = {
  getTodosByUserId,
  deleteTodoById,
  patchTodoById,
  addTodo,
  getTodoById,
  updateTodoById,
  getLastNTasks
};

src/services/User.ts:
import { SignUpRequest } from "../types";
import { User } from "../models/User";

const insertUser = (user: SignUpRequest) => {
  return User.create(user);
};

const checkIfEmailUsed = async (email: string) => {
  return !!(await User.findOne({ where: { email } }));
};

const getUser = (id: number) => {
  return User.findOne({ where: { id } });
};

const getUserByEmail = (email: string) => {
  return User.findOne({ where: { email } });
};

export const UserServices = {
  insertUser,
  checkIfEmailUsed,
  getUser,
  getUserByEmail,
};
```

```
src/types/index.ts:
export type SignUpRequest = {
  email: string;
  password: string;
};

export type TodoToCreate = {
  userId?: number;
  title: string;
  description?: string;
}

export type Todo = TodoToCreate & {
  id: number;
  status: 'created' | 'inProgress' | 'completed';
  createdAt: Date;
  inProgressAt?: Date;
  completedAt?: Date;
  estimatedTime?: number;
};

export type DecodedToken = {
  id: number;
  exp: number;
}

src/constants/index.ts:
export const baseAiContext = `
  You are a productivity assistant that helps users with their
todos.
  Given the following todo(s), provide a tip to help the user
complete it(them).
  {
    title: string;
    description?: string;
    id: number;
    status: 'created' | 'inProgress' | 'completed';
```

```
          createdAt: Date;
          inProgressAt?: Date;
          completedAt?: Date;
          estimatedTime?: number; // in hours
        }
        If the todo is completed, there will be additional field for
completion time (in hours).
        If user finished it for more than he thought it would take,
        provide a tip to help him estimate better or best practises for
this type of tasks.
      `;
```

Клієнтська частина:

src/App.tsx:

```
import { HashRouter, Route, Routes } from "react-router-dom";
import { LoginScreen } from "./components/screens/LoginScreen";
import { TodosScreen } from "./components/screens/TodosScreen";

export const App: React.FC = () => (
  <HashRouter>
    <Routes>
      <Route path={"/"} element={<LoginScreen />} />
      <Route path={"/todos"} element={<TodosScreen />} />
    </Routes>
  </HashRouter>
);
```

src/api/ai.ts:

```
import { client } from "../utils/fetchClient";

export const getTodoTipById = (id: number) =>
  client.get<string>(`/todos/tip/${id}`);

export const analyzeLastNTodos = (n: number) =>
  client.get<string>(`/user/anylyzeTodos?n=${n}`);
```

src/api/todos.ts:

```
import { Todo } from '../types/Todo';
```

```
import { TodoData } from '../types/TodoData';
import { client } from '../utils/fetchClient';

export const getTodos = () => {
  return client.get<Todo[]>(`/todos`);
};

export const addTodo = (data: TodoData) => {
  return client.post<Todo>('/todos', data);
};

export const deleteTodo = (todoId: number) => {
  return client.delete(`/todos/${todoId}`);
};

export const updateTodo = (todoId: number, data: Partial<TodoData>)
=> {
  return client.patch<Todo>(`/todos/${todoId}`, data);
};
```

src/api/user.ts:

```
import { client } from '../utils/fetchClient';

export const signIn = async(email: string, password: string,
onSuccess: () => void) => {
  const token = await client.post<string>(`/signin`, { email,
password });

  localStorage.setItem('token', token);

  onSuccess();
};

export const signUp = (email: string, password: string) => {
  return client.post<string>(`/signup`, { email, password });
};
```

src/components/ErrorMessage/ErrorMessage.tsx:

```
import cn from 'classnames';
import { memo, useEffect } from 'react';

type Props = {
  message: string;
  onDelete: () => void;
};

export const ErrorMessage: React.FC<Props> = memo(({ message,
onDelete }) => {
  useEffect(() => {
    const timerId = setTimeout(() => onDelete(), 3000);

    return () => {
      clearTimeout(timerId);
    };
  }, [message]);

  return (
    <div className={cn(
      'notification is-danger is-light has-text-weight-normal', {
        hidden: !message.length,
      },
    )}
    >
      <button
        type="button"
        className="delete"
        onClick={onDelete}
      />

      {message}
    </div>
  );
});

src/components/ErrorMessage/index.ts:
export * from './ErrorMessage';
```

src/components/LogInScreen/Footer.tsx:

```
import { Spin } from "antd";

type Props = {
  handleSubmit: (type: "login" | "signup") => () => void;
  signUpLoading: boolean;
  signInLoading: boolean;
};

export const Footer: React.FC<Props> = ({
  signUpLoading,
  handleSubmit,
  signInLoading
}) => (
  <footer className="todoapp__footer">
    <button
      type="button"
      className="todoapp__clear-completed"
      disabled={signUpLoading || signInLoading}
      onClick={handleSubmit("signup")}
    >
      {signUpLoading ? <Spin /> : "SIGN UP"}
    </button>
    <button
      type="submit"
      className="todoapp__clear-completed"
      disabled={signUpLoading || signInLoading}
      onClick={handleSubmit("login")}
    >
      {signInLoading ? <Spin /> : "LOG IN"}
    </button>
  </footer>
);
```

src/components/LogInScreen/LogInForm.tsx:

```
import { useState } from "react";
import { Footer } from "./Footer";
import { notification } from "antd";
import { signIn, signUp } from "../../api/user";
import { useNavigate } from "react-router-dom";

export const LogInForm: React.FC = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();

  const [signUpLoading, setSignUpLoading] = useState(false);
  const [signInLoading, setSignInLoading] = useState(false);

  const handleFormSubmit = (type: "login" | "signup") => async ()
=> {
    try {
      if (type === "signup") {
        setSignUpLoading(true);
        await signUp(email, password);
      } else {
        setSignInLoading(true);
      }
      await signIn(email, password, () => navigate("/todos"));
    } catch (e: any) {
      notification.error({
        message: e.message,
      });
    } finally {
      setSignUpLoading(false);
      setSignInLoading(false);
    }
  };

  return (
    <form>
```

```
      <input
        type="text"
        className="todoapp__new-todo"
        placeholder="Email"
        value={email}
        onChange={(event) => setEmail(event.target.value)}
      />
      <input
        type="password"
        className="todoapp__new-todo"
        placeholder="Password"
        value={password}
        onChange={(event) => setPassword(event.target.value)}
      />
      <Footer
        signUpLoading={signUpLoading}
        signInLoading={signInLoading}
        handleSubmit={handleFormSubmit}
      />
    </form>
  );
};
```

src/components/TodoScreen/Footer/Footer.tsx:

```
import { FilterBy } from "../../../enums/FilterBy";
import { TodoFilter } from "../TodoFilter";

type Props = {
  filter: FilterBy;
  onChange: (newFilter: FilterBy) => void;
  activeTodosNumber: number;
};

export const Footer: React.FC<Props> = ({
  filter,
  onChange,
  activeTodosNumber,
}) => (
```

```
      <footer className="todoapp__footer">
        <span className="todo-count">{`${activeTodosNumber} items
left`}</span>

        <TodoFilter filter={filter} onChange={onChange} />
      </footer>
    );

    src/components/TodoScreen/Footer/index.ts:
    export * from './Footer';

    src/components/TodoScreen/TempTodo/TempTodo.tsx:
    type Props = {
      title: string;
    };

    export const TempTodo: React.FC<Props> = ({ title }) => (
      <div className="todo">
        <label className="todo__status-label">
          <input
            type="checkbox"
            className="todo__status"
            checked
          />
        </label>

        <span className="todo__title">{title}</span>

        <button
          type="button"
          className="todo__remove"
          disabled
        >
          ×
        </button>

        <div className="modal overlay is-active">
          <div className="modal-background has-background-white-ter" />
```

```
        <div className="loader" />
      </div>
    </div>
  );

  src/components/TodoScreen/TempTodo/index.ts:
  export * from './TempTodo';

  src/components/TodoScreen/TodoFilter/TodoFilter.tsx:
  import { memo } from "react";
  import cn from "classnames";
  import { FilterBy } from "../../../enums/FilterBy";

  type Props = {
    filter: FilterBy;
    onChange: (newFilter: FilterBy) => void;
  };

  export const TodoFilter: React.FC<Props> = memo(({ filter, onChange
}) => (
    <nav className="filter">
      <a
        className={cn("filter__link", {
          selected: filter === FilterBy.All,
        })}
        onClick={() => onChange(FilterBy.All)}
      >
        All
      </a>

      <a
        className={cn("filter__link", {
          selected: filter === FilterBy.Created,
        })}
        onClick={() => onChange(FilterBy.Created)}
      >
        Created
      </a>
```

```
    <a
      className={cn("filter__link", {
        selected: filter === FilterBy.Active,
      })}
      onClick={() => onChange(FilterBy.Active)}
    >
      Active
    </a>

    <a
      className={cn("filter__link", {
        selected: filter === FilterBy.Completed,
      })}
      onClick={() => onChange(FilterBy.Completed)}
    >
      Completed
    </a>
  </nav>
));
```

src/components/TodoScreen/TodoFilter/index.ts:

```
export * from './TodoFilter';
```

src/components/TodoScreen/TodoForm/TodoForm.tsx:

```
import { memo, useCallback, useState } from "react";
import { TodoData } from "../../../types/TodoData";
import { addTodo } from "../../../api/todos";
import { Todo } from "../../../types/Todo";
import { notification } from "antd";

type Props = {
  onError: (message: string) => void;
  onChange: (newTodo: Partial<Todo> | null) => void;
  onCreate: (newTodo: Todo) => void;
  onCreated?: () => void;
};
```

```
export const TodoForm: React.FC<Props> = memo(
  ({ onError, onChange, onCreate, onCreated }) => {
    const [query, setQuery] = useState("");
    const [estimatedTime, setEstimatedTime] = useState(0.5); // in
hours
    const [description, setDescription] = useState("");
    const [isInputDisabled, setIsInputDisabled] = useState(false);

    const handleEstimatedTimeChange = (
      e: React.ChangeEvent<HTMLInputElement>
    ) => {
      const number = +e.target.value;
      if (number <= 0) {
        notification.error({
          message: "Estimated time can't be negative or zero",
        });
        return;
      }
      setEstimatedTime(number);
    };

    const handleFormSubmit = useCallback(
      async (event: React.FormEvent<HTMLFormElement>) => {
        event.preventDefault();
        onError("");

        if (!query.trim().length) {
          onError("Title can't be empty");

          return;
        }

        const todoToAdd: TodoData = {
          title: query,
          status: "created",
          description,
          estimatedTime,
        };
```

```
    onChange(todoToAdd);

    try {
      setIsInputDisabled(true);
      const newTodo = await addTodo(todoToAdd);

      onCreate(newTodo);
      setQuery("");
    } catch {
      onError("Unable to add a todo");
    } finally {
      onChange(null);
      setIsInputDisabled(false);
    }
    onCreated && onCreated();
  },
  [query, description, estimatedTime]
);

return (
  <form onSubmit={handleFormSubmit}>
    <input
      type="text"
      className="todoapp__new-todo"
      placeholder="What needs to be done?"
      value={query}
      onChange={(event) => setQuery(event.target.value)}
      disabled={isInputDisabled}
    />
    <textarea
      className="todoapp__new-todo"
      placeholder="More details (optional)"
      value={description}
      onChange={(event) => setDescription(event.target.value)}
      disabled={isInputDisabled}
    />
    <div>
```

```
          <label htmlFor="estimated-time">Estimated time: </label>
          <input
            id="estimated-time"
            type="number"
            value={estimatedTime}
            onChange={handleEstimatedTimeChange}
            step={0.5}
            style={{ width: "45px" }}
            disabled={isInputDisabled}
          />
          <span> hours</span>
        </div>
        <footer className="modal-footer">
          <button
            type="submit"
            className="todoapp__add-todo black-button"
            disabled={isInputDisabled}
          >
            ADD
          </button>
        </footer>
      </form>
    );
  }
);
```

src/components/TodoScreen/TodoForm/index.ts:

```
export * from './TodoForm';
```

src/components/TodoScreen/TodoItem/AiTip/AiTip.tsx:

```
import { notification, Popover, Spin } from "antd";
import { useState } from "react";
import { getTodoTipById } from "../../../../api/ai";
import { CloseOutlined } from "@ant-design/icons";

type Props = {
  id: number;
};
```

```
export const AiTip: React.FC<Props> = ({ id }) => {
  const [loading, setLoading] = useState(false);
  const [tip, setTip] = useState("");

  const handleTipRequest = async () => {
    setLoading(true);
    try {
      const response = await getTodoTipById(id);

      if (typeof response !== "string") {
        throw new Error("Invalid response");
      }

      setTip(response);
    } catch (e) {
      console.error(e);
      notification.error({
        message: "Failed to get AI tip",
      });
    } finally {
      setLoading(false);
    }
  };

  return (
    <Popover
      overlayStyle={{ maxWidth: 300 }}
      content={
        loading ? (
          <Spin />
        ) : (
          <div>
            <div style={{ display: "flex", justifyContent: "flex-
end" }}>
              <CloseOutlined onClick={() => setTip("")} />
            </div>
            <span>{tip}</span>
```

```
              </div>
            )
          }
          open={!!tip.length || loading}
        >
          <button type="button" className="black-button"
onClick={handleTipRequest}>
            AI TIP
          </button>
        </Popover>
      );
    };

    src/components/TodoScreen/TodoItem/TodoItem.tsx:
    import { memo, useState } from "react";
    import cn from "classnames";
    import { Todo } from "../../../types/Todo";
    import { Modal, notification } from "antd";
    import { AiTip } from "./AiTip/AiTip";
    import { EditOutlined } from "@ant-design/icons";

    type Props = {
      todo: Todo;
      onDelete: (todoToDeleteId: number) => void;
      loadingTodoIds: number[];
      handlePatchTodo: (todoId: number, data: Partial<Todo>) =>
Promise<boolean>;
    };

    export const TodoItem: React.FC<Props> = memo(
      ({ todo, onDelete = () => {}, loadingTodoIds, handlePatchTodo })
=> {
        const [isEditing, setIsEditing] = useState(false);
        const [newTitle, setNewTitle] = useState("");
        const [description, setDescription] = useState("");
        const [estimatedTime, setEstimatedTime] = useState(0.5);
        const [startAt, setStartAt] = useState(
          todo.inProgressAt ? new Date(todo.inProgressAt) : undefined
```

```
          );
          const [completedAt, setCompletedAt] = useState(
            todo.completedAt ? new Date(todo.completedAt) : undefined
          );
          const [status, setStatus] = useState(todo.status);
          const isLoading = loadingTodoIds.includes(todo.id);
          const [isDeleteModalOpen, setIsDeleteModalOpen] =
useState(false);

          const handleEdit = (oldTitle: string) => {
            setIsEditing(true);
            setNewTitle(oldTitle);
            setDescription(todo.description);
            setEstimatedTime(todo.estimatedTime);
            setStartAt(todo.inProgressAt ? new Date(todo.inProgressAt) :
undefined);
            setCompletedAt(todo.completedAt ? new Date(todo.completedAt)
: undefined);
          };

          const handleSave = async () => {
            try {
              const isUpdated = await handlePatchTodo(todo.id, {
                title: newTitle,
                description,
                estimatedTime,
                inProgressAt: startAt,
                completedAt,
                status,
              });
              if (!isUpdated) {
                throw new Error("Unable to save a todo");
              }
              notification.success({
                message: "Todo has been saved",
              });
            } catch (e) {
              notification.error({
```

```
      message: "Unable to save a todo",
    });
  } finally {
    setIsEditing(false);
  }
};

return (
  <div
    className={cn("todo", {
      completed: status === "completed",
      inProgress: status === "inProgress",
    })}
  >
    <label className="todo__status-label">
      <div className="todo__status" />
    </label>

    <span
      className="todo__title"
      onDoubleClick={() => handleEdit(todo.title)}
    >
      {todo.title}
    </span>

    <button
      type="button"
      className="todo__edit"
      onClick={() => handleEdit(todo.title)}
      disabled={isLoading}
    >
      <EditOutlined size={18}/>
    </button>

    <button
      type="button"
      className="todo__remove"
      onClick={() => setIsDeleteModalOpen(true)}
```

```
                disabled={isLoading}
              >
                x
              </button>

              <div
                className={cn("modal overlay", {
                  "is-active": isLoading,
                })}
              >
                <div className="modal-background has-background-white-
ter" />
                <div className="loader" />
              </div>
              {isEditing && (
                <Modal
                  centered
                  open={isEditing}
                  onCancel={() => setIsEditing(false)}
                  footer={
                    <div className="modal-footer">
                      <button
                        type="button"
                        className="black-button"
                        onClick={handleSave}
                        disabled={isLoading}
                      >
                        SAVE
                      </button>
                    </div>
                  }
                >
                  <div className="modal-content">
                    <input
                      type="text"
                      className="todoapp__new-todo"
                      placeholder="What needs to be done?"
                      value={newTitle}
```

```
                    onChange={(event) =>
setNewTitle(event.target.value)}
                    disabled={isLoading}
                  />
                  <textarea
                    className="todoapp__new-todo"
                    placeholder="More details (optional)"
                    value={description}
                    onChange={(event) =>
setDescription(event.target.value)}
                    disabled={isLoading}
                  />
                  <div>
                    <label htmlFor="estimated-time">Estimated time:
</label>
                    <input
                      id="estimated-time"
                      type="number"
                      value={estimatedTime}
                      onChange={(event) =>
                        setEstimatedTime(Number(event.target.value))
                      }
                      step={0.5}
                      style={{ width: "45px" }}
                      disabled={isLoading}
                    />
                    <span> hours</span>
                  </div>
                  <div>
                    <label>STATUS: </label>
                    <select
                      value={status}
                      onChange={(e) => setStatus(e.target.value as
typeof status)}
                    >
                      <option value="created">Not started</option>
                      <option value="inProgress">In progress</option>
                      <option value="completed">Completed</option>
```

```
                    </select>
                  </div>
                  {status !== "created" && (
                    <>
                      <div>
                        <label htmlFor="start-at">Start at (UTC):
</label>
                        <input
                          id="start-at"
                          type="datetime-local"
                          //@ts-ignore
                          value={startAt?.toISOString()?.slice(0, 16)}
                          onChange={(event) =>
                            setStartAt(new Date(event.target.value))
                          }
                          disabled={isLoading}
                        />
                      </div>
                      {status === "completed" && (
                        <div>
                          <label htmlFor="completed-at">Completed at
(UTC): </label>
                          <input
                            id="completed-at"
                            type="datetime-local"
                            //@ts-ignore
                            value={completedAt?.toISOString()?.slice(0,
16)}
                            onChange={(event) =>
                              setCompletedAt(new
Date(event.target.value))
                            }
                            disabled={isLoading}
                          />
                        </div>
                      )}
                    </>
                  )}
```

```
                    <AiTip id={todo.id}/>
                  </div>
                </Modal>
              )}
              {isDeleteModalOpen && (
                <Modal
                  centered
                  open={isDeleteModalOpen}
                  onCancel={() => setIsDeleteModalOpen(false)}
                  footer={
                    <div className="modal-footer">
                      <button
                        type="button"
                        className="black-button"
                        onClick={() => onDelete(todo.id)}
                        disabled={isLoading}
                      >
                        DELETE
                      </button>
                    </div>
                  }
                >
                  <div className="modal-content">
                    <h2>Are you sure you want to delete this todo?</h2>
                  </div>
                </Modal>
              )}
            </div>
          );
        }
      );
```

src/components/TodoScreen/TodoItem/index.ts:

```
export * from './TodoItem';
```

src/components/TodoScreen/TodoList/TodoList.tsx:

```
import { CSSTransition, TransitionGroup } from "react-transition-
group";
```

```
import { Todo } from "../../../types/Todo";
import { TodoItem } from "../TodoItem";
import { TempTodo } from "../TempTodo";

type Props = {
  todos: Todo[];
  onDelete: (todoToDeleteId: number) => void;
  tempTodo: Partial<Todo> | null;
  loadingTodoIds: number[];
  handlePatchTodo: (todoId: number, data: Partial<Todo>) =>
Promise<boolean>;
};

export const TodoList: React.FC<Props> = ({
  todos,
  onDelete,
  tempTodo,
  loadingTodoIds,
  handlePatchTodo,
}) => (
  <section className="todoapp__main">
    <TransitionGroup>
      {todos.map((todo) => (
        <CSSTransition key={todo.id} timeout={300}
classNames="item">
          <TodoItem
            key={todo.id}
            todo={todo}
            onDelete={onDelete}
            loadingTodoIds={loadingTodoIds}
            handlePatchTodo={handlePatchTodo}
          />
        </CSSTransition>
      ))}

      {tempTodo?.id === 0 && (
        <CSSTransition key={0} timeout={300} classNames="temp-
item">
```

```
            <TempTodo title={tempTodo.title || ""} />
          </CSSTransition>
        )}
      </TransitionGroup>
    </section>
  );
```

src/components/TodoScreen/TodoList/index.ts:

```
export * from './TodoList';
```

src/components/screens/LoginScreen.tsx:

```
import React from "react";
import { LogInForm } from "../LogInScreen/LogInForm";

export const LoginScreen: React.FC = () => (
  <div className="todoapp">
    <h1 className="todoapp__title">log in</h1>

    <div className="todoapp__content">
      <header className="todoapp__header">
        <LogInForm />
      </header>
    </div>
  </div>
);
```

src/components/screens/TodosScreen.tsx:

```
import React, { useCallback, useEffect, useMemo, useState } from
"react";
import { Todo } from "../../types/Todo";
import { FilterBy } from "../../enums/FilterBy";
import { deleteTodo, getTodos, updateTodo } from "../../api/todos";
import { TodoList } from "../TodoScreen/TodoList";
import { Footer } from "../TodoScreen/Footer";
import { ErrorMessage } from "../ErrorMessage";
import { TodoForm } from "../TodoScreen/TodoForm";
import { Modal, notification, Spin } from "antd";
```

```
      import { useLastNTodosAnalysis } from
"../../hooks/useLastNTodosAnalysis";
      import { BarChart } from "@mui/x-charts/BarChart";
      import { useDimensions } from "../../hooks/useDimensions";
      import { DownloadOutlined } from "@ant-design/icons";
      import { Link } from "react-router-dom";

      const xLabels = [
        "Created",
        "In-progress",
        "In-progress \n for too long",
        "Completed \n in time",
        "Completed \n too late",
      ];

      export const TodosScreen: React.FC = () => {
        const [isCreationModalVisible, setIsCreationModalVisible] =
useState(false);
        const [todos, setTodos] = useState<Todo[]>([]);
        const [filterBy, setFilterBy] = useState(FilterBy.All);
        const [errorMessage, setErrorMessage] = useState("");
        const [tempTodo, setTempTodo] = useState<Partial<Todo> |
null>(null);
        const [loadingTodoIds, setLoadingTodoIds] =
useState<number[]>([]);

        const { width } = useDimensions();

        const activeTodosNumber = todos.filter(
          ({ status }) => status !== "completed"
        ).length;

        const findTodoIndexById = (todoId: number) => {
          return todos.findIndex(({ id }) => id === todoId);
        };

        const createTodo = (newTodo: Todo) => {
          setTodos((prevTodos) => [...prevTodos, newTodo]);
```

```
  };

  const deleteErrorMessage = useCallback(() => {
    setErrorMessage("");
  }, []);

  const addLoadingTodoId = (todoId: number) => {
    setLoadingTodoIds((prevIds) => [...prevIds, todoId]);
  };

  const removeLoadingTodoId = (todoId: number) => {
    setLoadingTodoIds((prevIds) => prevIds.filter((id) => id !==
todoId));
  };

  const loadData = useCallback(async () => {
    try {
      const todosFromServer = await getTodos();

      setTodos(todosFromServer);
    } catch {
      setErrorMessage("Failed to load data");
    }
  }, []);

  const handleTodoDelete = useCallback(async (todoToDeleteId:
number) => {
    deleteErrorMessage();

    try {
      addLoadingTodoId(todoToDeleteId);
      await deleteTodo(todoToDeleteId);
      setTodos((prevTodos) =>
        prevTodos.filter(({ id }) => todoToDeleteId !== id)
      );
    } catch {
      setErrorMessage("Unable to delete a todo");
    } finally {
```

```
            removeLoadingTodoId(todoToDeleteId);
          }
        }, []);

        useEffect(() => {
          loadData();
        }, []);

        const filteredTodos = useMemo(() => {
          switch (filterBy) {
            case FilterBy.Created:
              return todos.filter(({ status }) => status === "created");

            case FilterBy.Active:
              return todos.filter(({ status }) => status ===
"inProgress");

            case FilterBy.Completed:
              return todos.filter(({ status }) => status ===
"completed");

            default:
              return todos;
          }
        }, [todos, filterBy]);

        const onCreated = useCallback(() => {
          setIsCreationModalVisible(false);
        }, []);

        const handlePatchTodo = useCallback(
          async (todoId: number, data: Partial<Todo>) => {
            if (data.status !== "created") {
              if (!data.inProgressAt) {
                notification.error({
                  message: "Please provide a start date and time",
                });
                return false;
```

```
        }

        if (data.status === "completed" && !data.completedAt) {
          notification.error({
            message: "Please provide a completion date and time",
          });
          return false;
        }

        if (
          data.completedAt &&
          data.inProgressAt &&
          data.inProgressAt > data.completedAt
        ) {
          notification.error({
            message:
              "Completion date and time should be greater than the
start date and time",
          });
          return false;
        }

        if (data.completedAt && data.completedAt > new Date()) {
          notification.error({
            message:
              "Completion date and time should be less than the
current date and time",
          });
          return false;
        }

        if (data.inProgressAt && data.inProgressAt > new Date()) {
          notification.error({
            message:
              "Start date and time should be less than the current
date and time",
          });
          return false;
```

```
        }
      }
      deleteErrorMessage();

      try {
        addLoadingTodoId(todoId);
        const newTodo = await updateTodo(todoId, data);

        setTodos((prevTodos) => {
          const todosCopy = [...prevTodos];
          const indexToDelete = findTodoIndexById(newTodo.id);

          todosCopy.splice(indexToDelete, 1, newTodo);

          return todosCopy;
        });
      } catch {
        setErrorMessage("Unable to update a todo");
        return false;
      } finally {
        removeLoadingTodoId(todoId);
        return true;
      }
    },
    [todos]
  );

  const uData = [
    todos.filter(({ status }) => status === "created").length,
    todos.filter(
      ({ status, inProgressAt, estimatedTime }) =>
        inProgressAt &&
        status === "inProgress" &&
        (Date.now() - new Date(inProgressAt).getTime()) / 1000 / 60
/ 60 <=
          estimatedTime
    ).length,
    todos.filter(
```

```
            ({ status, inProgressAt, estimatedTime }) =>
              inProgressAt &&
              status === "inProgress" &&
              (Date.now() - new Date(inProgressAt).getTime()) / 1000 / 60
/ 60 >
                estimatedTime
          ).length,
          todos.filter(
            ({ status, completedAt, estimatedTime, inProgressAt }) =>
              completedAt &&
              inProgressAt &&
              status === "completed" &&
              (new Date(completedAt).getTime() - new
Date(inProgressAt).getTime()) /
                1000 /
                60 /
                60 <=
                estimatedTime
          ).length,
          todos.filter(
            ({ status, completedAt, estimatedTime, inProgressAt }) =>
              completedAt &&
              inProgressAt &&
              status === "completed" &&
              (new Date(completedAt).getTime() - new
Date(inProgressAt).getTime()) /
                1000 /
                60 /
                60 >
                estimatedTime
          ).length,
        ];

        const avgTaskDuration = useMemo(() => {
          const completedTodos = todos.filter(({ status }) => status ===
"completed");

          if (completedTodos.length === 0) {
```

```
    return 0;
  }

  const totalDuration = completedTodos.reduce(
    (acc, { inProgressAt, completedAt }) =>
      acc +
      (new Date(completedAt as string | Date).getTime() -
        new Date(inProgressAt as string | Date).getTime()) /
        1000 /
        60 /
        60,
    0
  );

  return (totalDuration / completedTodos.length).toFixed(2);
}, [todos]);

const {
  handleAnalysis,
  loading: tipLoading,
  message,
} = useLastNTodosAnalysis(10);

return (
  <div className="todoapp">
    <h1 className="todoapp__title">todos</h1>

    <div className="todoapp__content">
      <TodoList
        todos={filteredTodos}
        onDelete={handleTodoDelete}
        tempTodo={tempTodo}
        loadingTodoIds={loadingTodoIds}
        handlePatchTodo={handlePatchTodo}
      />

      {todos && (
        <Footer
```

```
      filter={filterBy}
      onChange={setFilterBy}
      activeTodosNumber={activeTodosNumber}
    />
  )}
</div>

<button
  type="button"
  onClick={() => setIsCreationModalVisible(true)}
  className="black-button"
>
  CREATE NEW
</button>

<button
  type="button"
  onClick={handleAnalysis}
  className="black-button"
  disabled={todos.length === 0 || tipLoading}
>
  {tipLoading ? <Spin /> : "AI ANALYZE"}
</button>

<p>{message}</p>

{!!todos.length && (
  <span>Average task duration: {avgTaskDuration} hours</span>
)}

<BarChart
  width={width > 500 ? 600 : 300}
  height={300}
  series={[{ data: uData, label: "Tasks", type: "bar" }]}
  xAxis={[{ scaleType: "band", data: xLabels }]}
/>

<div
```

```
              onClick={() => {
                const element = document.createElement("a");
                const file = new Blob([JSON.stringify(todos, null, 2)], {
                  type: "application/json",
                });
                element.href = URL.createObjectURL(file);
                element.download = "todos.json";
                document.body.appendChild(element);
                element.click();
              }}
              style={{ cursor: "pointer" }}
            >
              <span>Export tasks as .json</span> <DownloadOutlined />
            </div>

            <button
              className="black-button"
              type="button"
              onClick={() => {
                localStorage.removeItem("token");
              }}
            >
              <Link to={"/"}>LOG OUT</Link>
            </button>

            <ErrorMessage message={errorMessage}
onDelete={deleteErrorMessage} />
            <Modal
              title="Create a new todo"
              open={isCreationModalVisible}
              onCancel={() => setIsCreationModalVisible(false)}
              footer={null}
              centered
            >
              <TodoForm
                onError={setErrorMessage}
                onChange={setTempTodo}
                onCreate={createTodo}
```

```
          onCreated={onCreated}
        />
      </Modal>
    </div>
  );
};
```

src/enums/FilterBy.ts:

```
export enum FilterBy {
  All = "all",
  Created = "created",
  Active = "active",
  Completed = "completed",
}
```

src/hooks/useDimensions.ts:

```
import { useEffect, useState } from "react";

export const useDimensions = () => {
  const [width, setWidth] = useState(window.innerWidth);
  const [height, setHeight] = useState(window.innerHeight);

  useEffect(() => {
    const handleResize = () => {
      setWidth(window.innerWidth);
      setHeight(window.innerHeight);
    };

    window.addEventListener("resize", handleResize);

    return () => {
      window.removeEventListener("resize", handleResize);
    };
  }, []);

  return { width, height };
};
```

src/hooks/useLastNTodosAnalysis.ts:

```
import { useState } from "react";
import { analyzeLastNTodos } from "../api/ai";

export const useLastNTodosAnalysis = (n: number) => {
  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState("");

  const handleAnalysis = async () => {
    setLoading(true);
    try {
      const msg = await analyzeLastNTodos(n);

      if (typeof msg !== "string") {
        throw new Error("Invalid response");
      }

      setMessage(msg);
    } catch (e) {
      console.error(e);
      setMessage("Failed to analyze todos");
    } finally {
      setLoading(false);
    }
  };

  return { loading, message, handleAnalysis };
};
```

src/index.tsx:

```
import { createRoot } from "react-dom/client";

import "bulma/css/bulma.css";
import "@fortawesome/fontawesome-free/css/all.css";
import "./styles/index.scss";

import { App } from "./App";
```

```
    createRoot(document.getElementById("root") as
HTMLDivElement).render(<App />);

    src/react-app-env.d.ts:
    /// <reference types="react-scripts" />

    src/types/Todo.ts:
    export interface Todo {
      id: number;
      userId: number;
      title: string;
      description: string;
      status: 'created' | 'inProgress' | 'completed';
      createdAt: Date;
      inProgressAt?: Date;
      completedAt?: Date;
      estimatedTime: number;
    }

    src/types/TodoData.ts:
    export interface TodoData {
      title: string;
      userId?: number;
      id?: number;
      description: string;
      status: 'created' | 'inProgress' | 'completed';
      inProgressAt?: Date;
      completedAt?: Date;
      estimatedTime: number;
    }

    src/utils/authToken.ts:
    export const saveToken = (token: string) => {
      localStorage.setItem("token", token);
    };

    export const getToken = () => {
      return localStorage.getItem("token");
```

```
};

export const removeToken = () => {
  localStorage.removeItem("token");
};

src/utils/fetchClient.ts:
import axios, { AxiosRequestConfig } from "axios";
import { getToken, removeToken } from "./authToken";

const BASE_URL = process.env.REACT_APP_BASE_URL ||
"http://localhost:5001";

type RequestMethod = "GET" | "POST" | "PATCH" | "DELETE";

async function request<T>(
  url: string,
  method: RequestMethod = "GET",
  data: any = null
): Promise<T> {
  const token = getToken();
  const config: AxiosRequestConfig = {
    method,
    url: BASE_URL + url,
    headers: {},
  };

  if (data) {
    config.data = data;
    if (config?.headers) {
      config.headers["Content-Type"] = "application/json;
charset=UTF-8";
    }
  }

  if (token && config.headers) {
    config.headers.Authorization = `Bearer ${token}`;
  }
```

```
    try {
      const response = await axios(config);
      return response.data;
    } catch (error: any) {
      if (error.response) {
        if (error.response.status === 401) {
          removeToken();
          window.location.href = "/";
        }
        throw new Error(error.response.data);
      } else {
        throw new Error(error.message);
      }
    }
  }

  export const client = {
    get: <T>(url: string) => request<T>(url),
    post: <T>(url: string, data: any) => request<T>(url, "POST",
data),
    patch: <T>(url: string, data: any) => request<T>(url, "PATCH",
data),
    delete: (url: string) => request(url, "DELETE"),
  };
```

ДОДАТОК Б

Диск з кваліфікаційною роботою магістра

ДОДАТОК В

Тези наукової конференції ІМСТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ

2024

004.41

Олег Колодій, Юрій Стоянов, к.т.н., старший викладач
Тернопільський національний технічний університет імені Івана Пулюя

Інтеграція OpenAI API для інтелектуального аналізу продуктивності в системах управління завданнями

Oleh Kolodii, Yurii Stoyanov, Ph.D.

Integration of OpenAI API for Intelligent Performance Analysis in Task Management Systems

Першими системами управління завданнями можна вважати записники та ручки, які використовувалися для створення списків справ. Сьогодні багато людей перейшли до сучасніших методів — використання мобільних або настільних додатків для нотаток. Зі впровадженням хмарних технологій ці нотатки синхронізуються між пристроями, що забезпечує доступ до них у будь-який час.

Попри зручність, такі підходи мають певні недоліки: завдання можуть бути забутими без відповідних нагадувань. Тому все більшої популярності набувають календарі та додатки для управління завданнями, які дозволяють встановлювати дедлайни та отримувати сповіщення.

З розвитком штучного інтелекту, зокрема великих мовних моделей (LLM), таких як ChatGPT, функціонал систем управління завданнями може бути значно розширений. Наприклад, за допомогою OpenAI API додатки можуть:

- Автоматично генерувати корисні підказки при створенні завдань.
- Аналізувати виконання завдань за певний період (наприклад, за тиждень) та надавати звіт про продуктивність.
- Ідентифікувати слабкі місця у підході користувача до планування [1].

OpenAI API забезпечує зручний інструментарій для інтеграції таких функцій завдяки JavaScript-бібліотеці. Робота з API організована через створення запитів (prompt) та передачу контексту моделі. Наприклад, модель можна налаштувати як персонального асистента для управління завданнями, який надає рекомендації на основі аналізу списку справ [1].

Подібні можливості можуть бути корисними для спеціалістів різних галузей, допомагаючи підвищити ефективність і точність виконання робочих завдань. Впровадження таких технологій сприяє оптимізації робочих процесів та дозволяє зробити системи управління завданнями ще більш адаптивними [2].

Література

1. OpenAI API Reference [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/api-reference/introduction.
2. GPT-4 Technical Report [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/research/gpt-4.