НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Методи аналізу показників та виявлення аномалій в інтелектуальному моніторингу веб-застосунків»

Виконав:
Студент IV курсу, групи КА-12
Чайка Микита Сергійович _____

Керівник:
Асистент каф. ММСА Древаль Максим Михайлович _____

Консультант з економічного розділу:
Доцент, к.е.н., Рощина Надія Василівна _____

Консультант з нормоконтролю:
Асистент каф. ММСА Канцедал Георгій Олегович _____

Рецензент:
Доцент каф. СП, к.т.н. Безносик Олександр Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» _____ 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Чайці Микиті Сергійовичу

1. Тема роботи «Методи аналізу показників та виявлення аномалій в інтелектуальному моніторингу веб-застосунків», керівник роботи Древаль Максим Михайлович, асистент, затверджені наказом по університету від «26» травня 2025 р. № 1759-с

2. Термін подання студентом роботи

3. Вихідні дані до роботи: Датасети з кількістю HTTP-запитів, APM-метрики з Elastic Stack.

4. Зміст роботи: 1 – Дослідження предметної області; 2 – Теоретичні основи моніторингу метрик веб-застосунків та методів виявлення аномалій у їх часових рядах; 3 – Реалізація та експериментальна перевірка методів моніторингу й виявлення аномалій; 4 – Функціонально-вартісний аналіз програмного забезпечення.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Ілюстрації роботи програмного продукту, презентація

6. Консультанти розділів роботи.

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз предметної області та формулювання завдання	01.04.2025-13.04.2025	Виконано
2.	Вивчення літератури та обробка даних за темою роботи	14.04.2025-21.04.2025	Виконано
3.	Підготовка математичних моделей, розробка програмного коду їх реалізацій	22.04.2025-10.05.2025	Виконано
4.	Розробка програми для тестування методів	11.05.2025-20.05.2025	Виконано
5.	Оформлення та узгодження пояснювальної записки і розділів роботи	21.05.2025-01.06.2025	Виконано
6.	Попередній захист дипломної роботи	02.06.2025-15.06.2025	Виконано
7.	Захист дипломної роботи	16.06.2025-22.06.2025	Виконано

Студент Микита ЧАЙКА

Керівник Максим ДРЕВАЛЬ

РЕФЕРАТ

Дипломна робота: 137 с., 8 табл., 33 рис., 3 додатки, 20 джерел.

МОНІТОРИНГ, СПОСТЕРЕЖУВАНІСТЬ, ПРОГНОЗУВАННЯ, ВИЯВЛЕННЯ АНОМАЛІЙ, ЧАСОВІ РЯДИ, ВЕБ РОЗРОБКА

Об'єкт дослідження – показники роботи веб-застосунків.

Предмет дослідження – методи аналізу часового ряду метрик та автоматичного виявлення аномалій у рамках інтелектуального моніторингу веб-застосунків.

Мета роботи – розробка та налаштування програмно-аналітичного продукту для аналізу ключових метрик веб-застосунків та автоматичного виявлення аномалій із використанням статистичних алгоритмів, моделі Prophet та платформи Elastic Stack.

Актуальність – зі зростанням складності розподілених сервісів і навантажень на них, автоматичний моніторинг і точне виявлення аномалій стають критично важливими для забезпечення безперервності роботи веб-застосунків.

Результати роботи – розроблено програмно-аналітичний продукт, що поєднує збір телеметрії через Elastic Stack, прогнозування та виявлення аномалій в роботі веб-застосунків за допомогою моделі Prophet та статистичних методів.

ABSTRACT

Diploma thesis: 137 p., 8 tables, 33 figures, 3 appendices, 20 references.

MONITORING, OBSERVABILITY, FORECASTING, ANOMALY DETECTION, TIME SERIES, WEB DEVELOPMENT

Object of research: performance indicators of web applications.

Subject of research: methods for analysing time-series metrics and automatically detecting anomalies with intelligent monitoring of web applications.

Goal of the work: to develop and configure a software-analysis product for analysing key web-application metrics and automatically detecting anomalies by means of statistical algorithms, the Prophet model and the Elastic APM platform.

Relevance of the work: with the growing complexity of distributed services and the increasing workloads they bear, automated monitoring and accurate anomaly detection have become critical for maintaining uninterrupted operation of web applications.

Results: a software-analytical solution has been created that combines telemetry collection through Elastic Stack with forecasting and anomaly detection based on the Prophet model and statistical methods.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTTP – протокол передавання гіпертексту (англ. Hypertext Transfer Protocol).

SNMP – протокол простого керування мережею (англ. Simple Network Management Protocol).

CPU – центральний процесор (англ. Central Processing Unit).

XML – розширювана мова розмітки (англ. eXtensible Markup Language).

CSV – значення, розділені комами (англ. Comma-Separated Values).

CLF – загальний формат логів (англ. Common Log Format).

JSON – нотація об’єктів JavaScript (англ. JavaScript Object Notation).

OTLP – протокол OpenTelemetry (англ. OpenTelemetry Protocol).

API – інтерфейс програмування застосунків (англ. Application Programming Interface).

SDK – набір засобів розробника (англ. Software Development Kit).

UI – користувацький інтерфейс (англ. User Interface).

MVP – мінімально життєздатній продукт (англ. Minimum Viable Product)

ВСТУП

У нинішніх реаліях будь-який сучасний веб-застосунок – від простого веб-магазину до великого складного проєкту, повинен працювати без збоїв і максимально швидко обробляти запити. Кожна хвилина простою чи затримка в роботі додатка може коштувати бізнесу дуже дорого – втрата клієнтів, падіння рейтингу, зниження доходів. Саме тому зараз одна з найактуальніших задач для будь-якого інженера чи аналітика – це побудувати ефективну систему моніторингу та спостереження за технічними та бізнесовими метриками. Ці технічні метрики та дані, які постійно генерують застосунки: час відгуку, кількість помилок, завантаження ресурсів, лог-записи, називають телеметрією. Саме вони складають основу розуміння внутрішнього стану системи. Без регулярного збору та аналізу телеметрії неможливо вчасно помітити якійсь відхилення або повноцінні збої в роботі системи.

Для організації збору, обробки, візуалізації телеметрії застосовують спеціалізовані платформи та відкриті міжнародні стандарти. Це дозволяє об’єднувати різні джерела даних і спрямувати їх у централізовані сховища чи системи візуалізації, де інженери та аналітики можуть спостерігати історичні тренди, налаштувати оповіщення та будувати інтерактивні панелі.

Водночас класичні механізми сповіщень на основі фіксованих порогів часто не враховують змінність навантажень і складність розподіленої архітектури. Також ручне відстеження аномалій вже давно перестало відповідати темпам сучасного бізнесу. Саме тому дедалі більшої популярності набувають підходи на основі статистичних та адаптивних алгоритмів, які здатні автоматично виявляти аномалії та будь-яку нестандартну поведінку в системі.

У цій роботі буде розглянуто загальні принципи збору телеметрії й обробки базових показників, а також концепції автоматизації моніторингу за допомогу статистичних методів та методів машинного навчання.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття моніторингу та спостережуваності

Моніторинг – це процес регулярного збору та аналізу заздалегідь визначених метрик, які дозволяють оцінити поточний стан системи та швидко виявити вже відомі проблеми.

Моніторинг фіксує ключові показники і повідомляє про проблеми, коли вони виникають. Наприклад, він може попередити та сигналізувати про високе завантаження CPU або про нестачу пам'яті на сервері. Проте моніторинг обмежений тільки тими параметрами, які були попередньо визначені. Зазвичай результати моніторингу виводяться у вигляді інтерактивних дашбордів для наочного контролю стану системи. Крім того, вони слугують основою для запуску сповіщень у разі перевищення встановлених порогів або виявлення нетипової поведінки. Якщо ж потрібно зрозуміти, чому саме був перевантажений CPU або чому запити до API раптово уповільнилися, одного моніторингу вже недостатньо. Тут потрібен більш глибокий підхід, такий як спостережуваність.

Спостережуваність – означає здатність аналізувати та вимірювати внутрішній стан систем на основі їхніх результатів та взаємодії між різними сервісами.

Спостережуваність дозволяє глибоко аналізувати стан системи, ставити нові, раніше не сформульовані запитання та знаходити на них відповіді – навіть у тих випадках, коли наперед невідомо, які саме параметри слід було відстежувати.

Ключові відмінності між концепцією моніторингу та спостережуваністю [1] описано в таблиці 1.1.

Таблиця 1.1 – Порівняння понять моніторингу та спостережуваності

	Моніторинг	Спостережуваність
Що це	Вимірювання конкретних метрик системи і складання звітів за ними для контролю працездатності системи.	Збір метрик, подій, журналів і трасувань для докладного вивчення проблем із працездатністю розподілених систем, що використовують архітектуру мікросервісів.
Основна задача	Збір даних для виявлення аномальних системних ефектів.	Вивчення першопричин аномальних системних ефектів.
Задіяні системи	Зазвичай належить до автономних систем.	Зазвичай відноситься до кількох розрізнених систем.
Відстежуваність	У межах кордонів системи.	Скрізь, де генеруються сигнали в будь-яких системних архітектурах.
Виявлення системних помилок	Коли і що.	Чому і як.

1.2 Актуальність проблеми

Сучасні веб-застосунки стають з кожним день все складнішими: їх архітектура стає все більш складнішою, вони складаються з багатьох сервісів, працюють у хмарних середовищах, обробляють та зберігають великі обсяги даних. Тому відслідковувати стабільну роботу застосунку стає все дедалі

складнішим. В той час, кожна хвилина простою або просто повільної роботи додатку може призвести до великих фінансових втрат сучасних IT-компаній. Тому постійний моніторинг та спостереження – це вже не просто додаткова зручність, а критично важлива частина бізнес-процесів.

Традиційного моніторингу, який просто базується на простих метриках і заздалегідь визначених порогах, стає все більш недостатньо, він не завжди дозволяє своєчасно виявити проблему та побачити всю картину разом. Тому для того, щоб мати можливість глибоко аналізувати стан систему в режимі реального часу також використовується концепція спостережуваності.

Особливо актуальним в останній час стає прогнозування метрик та автоматичне виявлення аномалій – відстеження нестандартної поведінки застосунка без необхідності вручну аналізувати мільйони логів та різних графіків. Саме такі сучасні технології, як нейронні мережі та машинне навчання, дозволяють побудувати ефективну систему для своєчасного реагування і попередження серйозних інцидентів.

Таким чином, побудова якісної системи моніторингу та механізмів для прогнозування та автоматичного виявлення аномалій на основі аналізу в режимі реального часу – це стратегічна необхідність для будь-якого бізнесу, який прагне бути конкурентоспроможним і надійним у цифрову епоху.

1.3 Історичний розвиток

Перед розглядом сучасних технологій корисно ознайомитись з історичними етапами поняття моніторингу та спостереження в інформаційних системах, щоб зрозуміти які потреби формували ті чи інші інструменти та методи, які популярні зараз.

1980-1990mi. У цей період основним способом стежити за мережею та серверами був протокол SNMP (Simple Network Management Protocol) [2]. Це

протокол для управління та моніторингу мережевих пристроїв, такі як комутатори, маршрутизатори та фаерволи. Реалізований він на моделі "менеджер-агент", де централізований менеджер періодично опитує віддалені агенти, отримуючи інформацію про використання процесора, пам’яті та мережевого трафіку. У випадку критичних подій агенти відправляє асинхронні сповіщення, що дає змогу реагувати на проблему. З часом, протокол був розвинутий до версій v2c та v3 з поліпшеною продуктивністю передачі даних та вбудованими механізмами шифрування.

2000-2015 роки. Пізніше з розвитком клієнт-сервісних і веб-архітектур виникла потреба в моніторингу прикладних сервісів, такі як HTTP-ендпоінти, бази даних, бек-процеси. Для цього з’явились такі рішення як Nagios – одна з перших систем систем моніторингу з вихідним кодом. Основними можливостями були підтримка кастомним плагінів, які можуть перевіряти стан процесів, доступність HTTP-ендпоінтів або навантаження бази даних. Далі у розподілених системах з’явилась складність у трасуванні запитів через мікросервіси. Так з’явились Google Dapper [3] та Zipkin [4], які започаткували концепцію розподіленого трасування, що дало змогу корелювати час виконання між сервісами.

Так сформувалася сучасна концепція спостережуваності, яка побудована на трьох стовпах: метрики, логи та трейси. А також, методи виявлення аномалій пройшли шлях від класичних статистичних методів до алгоритмів машинного навчання. Сьогодні універсальні фреймворки по типу OpenTelemetry надають спільні API для роботи з всіма телеметричними видами даних, а сучасні системи для моніторингу та спостережуваності забезпечують масштабоване зберігання, аналіз, візуалізацію та потужні механізми сповіщень. Завдяки цьому організації отримують можливість не лише збирати й відображати дані, а й прогнозувати проблеми до того, як вони вплинуть на кінцевих користувачів, що дозволяє заздалегідь вживати заходів для запобігання простоїв, оптимізувати ресурси та підтримувати стабільну роботу сервісів.

1.4 Основні компоненти спостережуваності

Три стовпи спостережуваності – метрики, логи і трейси [5], які використовуються для отримання уявлення про поведінку системи, виявлення і вирішення проблем з продуктивністю і безпекою для підвищення надійності системи, а також для забезпечення видимості складних систем, щоб допомогти виявити вузькі місця і збої в роботі.

1.4.1 Метрики

Метрики – це числове представлення даних.

Вони посилаються на вимірювання, які проводяться під час виконання програми. Будь-які значення, що мають відношення до аналізу продуктивності та поведінки програми, можна відстежувати за допомогою відповідних інструментів. Наприклад, метрикою може бути відсоток завантаження процесора, загальна кількість або час відповіді HTTP-запитів, або різні бізнесові показники. Метрики також можна використовувати для вимірювання відповідних значень в інфраструктури програми (так звані системні метрики)

Існують різні типи/види метрик, які впливають на спосіб вимірювання значень. Номенклатура цих типів може змінюватися залежно від специфікації інструменту моніторингу, що використовується, але загалом вони складаються з наступних елементів.

1. Лічильник (Counter*)* – кумулятивна метрика, яка тільки зростає з часом. Такий тип зазвичай використовується для кількість запитів до API, кількість помилок, кількість продуктових продажів.
2. Датчик (Gauge) – метрика, яка представляє одне числове значення, що може довільно збільшуватися або зменшуватися. Наприклад,

використання CPU, завантаження пам’яті, кількість активних підключень.

3. Гістограма (Histogram*)* – метрика, яка збирає значення та розподіляє їх по діапазонах. Наприклад, розподіл часу відповіді HTTP-запитів.

1.4.2 Логи

Логи – фрагмент тексту, який генерується системою при виконанні певної частини коду.

У контексті спостережуваності під логуванням зазвичай розуміють централізоване логування, коли логи з усіх компонентів системи надсилаються до серверу спостережуваності, де їх можна шукати та аналізувати. Зазвичай логи були неструктурованими, тобто вони були створені для читання людьми, а не комп'ютерами. Наприклад:

2025-04-18T10:33:21.456Z INFO [auth-service] User login successful: user_id=12345

2025-04-18T10:33:24.002Z ERROR [db] Failed to connect to database: timeout

Оскільки ці логи не мають уніфікованого формату, комп’ютерам важко їх обробляти, що стає проблемою, коли логів надто багато, щоб людина могла їх перечитати вручну. Щоб комп’ютер міг здійснювати пошук і аналіз великої кількості логів, вони мають бути структурованими. Однак не існує єдиного загальноприйнятого стандарту для структурування логів. Різні системи спостережуваності, які підтримують централізоване логування, зазвичай здатні розпізнавати декілька форматів.

1. XML – Формат представлення структурованих даних у вигляді тегів, подібно до HTML.

2. CSV – Простий табличний формат, де дані розділені комами або іншими роздільниками.
3. CLF – Стандартний формат логів для веб-серверів (Apache, Nginx)
4. JSON – Легкий формат обміну даними, який представляє інформацію у вигляді пар ключ-значення. Є найгнучкішим і найпоширенішим у сучасних системах observability.

1.4.3 Трейси

Трейси – це тип телеметричних даних, який дозволяє відстежити шлях виконання конкретного запиту або транзакції через усі компоненти розподіленої системи.

Розподілене трасування допомагає зрозуміти, що сталося під час розподіленої транзакції, наприклад запиту, ініційованого користувачем, і оцінити його вплив на всі зачеплені downstream-мікросервіси та сховища даних. Трейси зазвичай являють собою послідовності "точок даних", або спанів (span), і візуалізуються за допомогою діаграми Ганта. Приклад трейсу в вигляді діаграми Ганта можна побачити на рисунку 1.1.

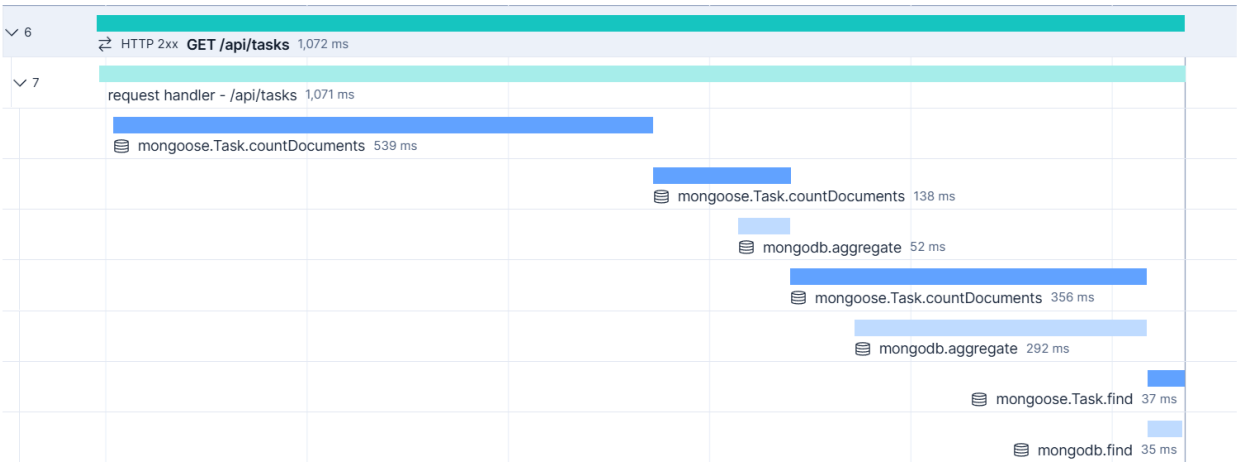


Рисунок 1.1 – Приклад трейсу

Спани сильно прив'язані до контексту. Крім того, спани записують інформацію про батьківський спан, який їх ініціював. Це дає змогу простежити причинно-наслідкові зв'язки між різними елементами розподіленої системи, такими як сервіси, черги, бази даних тощо. Спан повинен містити наступну інформацію.

1. Trace ID: Унікальний ідентифікатор для трейсу, який створюється кореневим проміжком і поширюється на дочірні проміжки.
2. Span ID: Унікальний ідентифікатор для всіх створених окремих спанів.
3. Parent ID: Span ID батьківського елементу або батьків, які створили цей спан. Відсутній, якщо це кореневий проміжок.
4. Start time: Відмітка часу (Timestamp) початку роботи.
5. End time: Мітка часу (Timestamp) завершення роботи.

Більшість реальних реалізацій трасування містять додаткову інформацію в проміжках, наприклад: опис, назва сервісу та різні додаткові теги.

У системах спостережуваності трейси часто пов’язуються з логами та метриками через загальні ідентифікатори (наприклад, trace_id, span_id). Це дозволяє швидко переходити від загального огляду системи до аналізу конкретного запиту, і навіть до перегляду відповідних логів, створених у межах цього запиту. Завдяки цьому досягається контекстність і повнота спостереження.

Трасування може бути реалізовано з будь-яким типом архітектури додатків (від монолітів до складних сервісних завдяки механізму поширення контексту, який дозволяє передавати важливу інформацію про виконання поточного запиту від компонента до компонента, незалежно від інфраструктури, що його підтримує, і, зрештою, об'єднати всі відрізки в єдиний трейс.

Таким чином, трейси є невід’ємною складовою сучасної системи спостережуваності, забезпечуючи глибоке розуміння того, як система поводиться в реальному часі та які елементи впливають на її продуктивність.

1.5 Висновки до розділу 1

У даному розділі було розібрано, що таке моніторинг і спостережуваність, виокремлено основні поняття та значущі складники. Було підкреслено актуальність теми: сучасні системи складаються з багатьох сервісів, сучасна розподілена архітектура суттєво ускладнює виявлення збоїв та аномалій в системи, що може призводити до великих проблем та фінансових втрат серед IT-компаній. В той час як налагоджена система для моніторингу та спостережуваності дозволяє оперативно реагувати на будь-які відхилення від норми в роботі додатка, та зменшувати час вирішення інцидентів, або взагалі запобігти їх настанню.

Детально було розглянуто три ключових джерела даних – метрики, логи та трейси.

1. Метрики відображають кількісні показники продуктивності.
2. Логи фіксують послідовність дій і контекст виконання запитів, що сприяє детальному розслідуванню технічних збоїв.
3. Трейси забезпечують розгорнуто картину маршруту запитів між компонентами.

Завдяки цьому теоретичному вступу далі можна переходити до конкретних методів аналізу даних та методів автоматичного виявлення аномалій у роботі веб-системи.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ МОНІТОРИНГУ МЕТРИК ВЕБ-ЗАСТОСУНКІВ ТА МЕТОДІВ ВИЯВЛЕННЯ АНОМАЛІЙ У ЇХ ЧАСОВИХ РЯДАХ

2.1 Основи інструментації та телеметрії у веб-застосунках

Налаштування повноцінної системи спостережуваності потребує відповідних інструментів для збору та аналізу телеметричних даних. До таких інструментів належать як відкриті стандарти інструментації, так і комерційні рішення класу APM (Application Performance Monitoring).

2.1.1 OpenTelemetry

OpenTelemetry (OTLP) – це незалежний, vendor-agnostic компонент, що виконує роль шлюзу між застосунками та системами спостережуваності [6]. Його ключа задача – отримувати, обробляти та експортувати дані трейсів, метрик і логів. Це дозволяє організаціям централізувати управління телеметрією, зменшити накладні витрати на окремі послуги та оптимізувати збір даних з різних джерел. Тобто його основна мета – зробити уніфікований підхід незалежно від мови програмування чи платформи. Він надає готові API, SDK та інструменти для генерації, збору, обробки та експорту.

Основними перевагами використання OpenTelemetry можна назвати:

1) єдина стандартизована платформа для метрик, трейсів та логів;

2) гнучкість розгортання;

3) відкритість та сумісність з існуючими рішеннями;

4) масштабованість та розширюваність завдяки модульній архітектурі.

Архітектура складається з п’яти основних блоків.

1. Receivers – приймають телеметрію в різних протоколах (OTLP/gPRC, HTTP, Jaeger, Prometheus).
2. Processors – виконують попередню обробку даних. На цьому етапі відбувається модифікація або оптимізація телеметричних даних.
3. Exporters – відправляють оброблені дані до бекендів для зберігання та візуалізації.
4. Extensions – розширення, що надають додаткову функціональність. Наприклад, моніторинг стану, автентифікацію, або профілювання продуктивності. Вони не беруть безпосередньої участі в обробці даних, але розширюють функціонал.
5. Service Pipelines – визначають потік телеметричних даних від приймачів, через процесори, і нарешті до експортерів. Можна визначати різні pipelines для різних типів телеметричних даних.

Завдяки такій модульній архітектурі, OpenTelemetry є універсальним інструментом для збору, обробки та експорту телеметричних даних.

2.1.2 Сучасні системи моніторингу та спостережуваності

Створення системи моніторингу та спостережуваності вимагає наявності серверної частини, де можна зберігати всю телеметрію, отриману під час виконання програми. Сучасні інструменти також забезпечують інтеграцію методів візуалізації даних таких як зручні дашборди, графіки і діаграми в режимі реального часу.

Серед основних обраних критеріїв для вибору інструментів, особливо важливими для розгляду були такі як сумісність з OTLP, можливості розгортання та конфігурації, та популярність. Основні, які найбільш часто використовуються в комерційних проєктах можна виділити [7].

1. Prometheus: система для збору та зберігання різних метрик. Має власну мову запитів PromQL і тісно інтегрується з екосистемою Kubernetes. Дозволяє отримувати дані з бекенд-сервісу через регулярні проміжки часу.
2. Grafana: Платформа для візуалізації даних метрик. Дозволяє створювати різноманітні динамічні дашборди з діаграмами та графіками для моніторингу у режимі реального часу. Є можливість налаштування різного виду альортів або навіть комбінувати дані з кількох систем. Часто застосовується разом з Prometheus.
3. Elastic stack (Elasticsearch, Logstash, Kibana): комбінація різних інструментів з відкритим вихідним кодом [8]. Сам по собі є пошуковим движком, але має окрему компоненту Elastic APM, яка є системою спостережуваності для відслідковування трейсів, метрік, логів.

Також, варто зазначити, що багато інструментів також пропонують готові рішення на основі штучного інтелекту, які допомагають в аналізі. Вони автоматично виявляють аномалії або формують аналітичні звіти. З одного боку це зручно, але існує досить багато причин, чому використання лише таких рішень не завжди є оптимальним. Основні які можна виділити – це те, що більшість подібних інструментів є платними та вимагають підписок, що може бути надмірно дорогим для невеликих команд чи дослідницьких проєктів. І також так як ми не має повного доступу до алгоритмів, ми не можемо впливати на їх логіку роботи, їх кастомізацію або вдосконалення під специфіку власного додатку. Саме цьому в цій роботі будуть розглянути власні методи автоматичного виявлення аномалій, як і звичайні статистичні так і з використанням методів машинного навчання. Також будь-який власний модуль легко потім можна інтегрувати вже в існуючі системи.

2.2 Поняття аномалії в часових рядах

Виявлення аномалій означає ідентифікацію точок даних, шаблонів або спостережень, які значно відхиляються від очікуваної поведінки в наборі даних. Аналіз часових рядів дає змогу виявляти відхилення у поведінці системи, які можуть свідчити про збої, перевантаження або інші порушення нормального функціонування.

Аномалії в часових рядах класифікуються за їх характером і контекстом виникнення. Найчастіше виділяють три основні типи [9].

1. Точкові аномалії – окремі точки даних, які суттєво відрізняються від решти. Прикладом такої аномалії може бути різке зростання часу відповіді на якийсь конкретний запит до серверу, тоді як усі інші залишаються в межах норми.
2. Контекстуальні аномалії – точки даних, які є аномальними в певному контексті, але можуть здаватися нормальними в інших умовах. Такі аномалії часто виникають у даних із сезонністю або трендом. Наприклад, зниження кількості створених задач у вихідні дні є абсолютно нормальним, але таке саме зниження в робочий день – це вже аномалія.
3. Колективні аномалії – група точок даних, які в сукупності відрізняються від очікуваного шаблону загального набору даних. Наприклад, плавне, але постійне зростання часу відповіді запиту протягом 10 хвилин вже може свідчити про проблеми з навантаженням, хоча кожна окрема точка знаходиться в допустимих межах.

2.3 Типи часових рядів у веб-додатках та аномалії, що в них виявляються

У веб-додатках часові ряди формуються з метрик, які є результатом взаємодії користувача із системою або внутрішніх процесів самої системи. Серед типових прикладів можна виділити:

1) кількість HTTP-запитів на одиницю часу;
2) середній час відповіді запитів (Latency);
3) кількість помилок (клієнтських і серверних);
4) навантаження CPU та Memory Usage.

Також крім технічних метрик це можуть також і звичайні бізнесові (використання функціоналу, створення транзакцій тощо). Для більшості таких рядів характерна певна циклічність: добова (наприклад, спад активності вночі), тижнева (менша активність на вихідних), або навіть сезонна (збільшена активність в святкові дні та періоди акцій).

Розуміння цих типів часових рядів є критично важливим для якісного моніторингу системи. Особлива увага приділяється виявленню таких типів аномалій:

1) раптове зростання або падіння кількості запитів – це може свідчити про DDoS атаку або падіння трафіку;
2) збільшення часу відповіді;
3) поява великої кількості помилок (4xx або 5xx);
4) будь-які відхилення в бізнесових метриках.

Всі ці аномалії можуть свідчити про різні технічні несправності, тому своєчасно їх виявлення є критично важливим для підтримки стабільної роботи системи.

2.4 Методи виявлення аномалій

Методи виявлення аномалій поділяються на дві великі категорії: статистичні та методи з використанням машинного навчання [10]. Перші базуються на аналізі розподілу значень і не складні в реалізації, але вони обмежені в здатності працювати з контекстом. Другі – здатні розпізнавати складні патерни та показують кращі результати, але потребують навчання і більш складні в реалізації. У цьому розділі розглянуто обидва підходи, їхні переваги та недоліки в рамках використання для побудови системи виявлення аномалій у телеметрії веб-додатків.

2.4.1 Статистичні методи

Статистичні методи є одними з найпростіших методів виявлення аномалій і покладаються на математичні пороги для виявлення відхилень. Такі методи можуть застосовуватися у базовому моніторингу, коли необхідно швидко реагувати на якісь відхилення телеметрії без складного аналізу контексту. Найбільш поширеними статистичними підходами можна назвати метод Z-оцінки та міжквартильного розмаху (IQR).

Z-оцінка: даний метод вимірює, наскільки точка віддалена від середнього значення у вимірі стандартного відхилення [11]. Тобто метод дозволяє визначити, наскільки аномальним є значення у межах загального розподілу. Z-оцінка вираховується за формулою (2.1):

$$z = \frac{x - \mu}{\sigma}, \tag{2.1}$$

де x – значення метрики;

μ – середнє значення ряду;

σ – стандартне відхилення.

Якщо абсолютне значення перевищує певний заданий поріг (наприклад |z| > 3), значення вважається аномальним. Цей метод може бути ефективний лише у випадках, коли дані мають приблизно нормальний розподіл.

Міжквартильний розмах (IQR): це діапазон між першим квартилем (Q1) і третім квартилем (Q3) даних [12]. Точки, які виходять за межі певного діапазону за межами Q1 і Q3, вважаються викидами. Тобто IQR – це як середня частина даних. Точка, яка знаходиться вище або нижче цієї середини вважається аномалією. IQR розраховується за формулою (2.2).

$$IQR = Q3 - Q1, \tag{2.2}$$

Нижня межа = Q1 - k * IQR, Верхня межа = Q3 + k * IQR

де k – це константа (зазвичай береться 1.5)

Метод добре працює коли дані не мають нормального розподілу або містять одиничні викиди, на які не слід реагувати.

2.4.2 Методи з застосуванням машинного навчання: Prophet

Prophet – це статистична модель прогнозування часових рядів. Вона добре працює з даними, які мають чітку сезонність (наприклад, по днях тижня або годинах) і тренд, який змінюється з часом.

У центрі Prophet лежить адитивна модель, в якій загальне значення сигналу складається з кількох незалежних частин: тренду, сезонності та випадкової складової [13].

Загальне рівняння задається за формулою (2.3) (також ілюстраційний приклад можна побачити на рисунку 2.1):

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t, \tag{2.3}$$

де g(t) – довготривалий тренд (лінійний або логістичній, може змінюватися в різні моменти часу);

s(t) – сезонна складова (наприклад, щотижнева або щоденна циклічність);

h(t) – вплив окремих подій (свята, рекламні кампанії тощо);

ε_t – залишкова похибка, або шум, що відображає невраховані впливи.

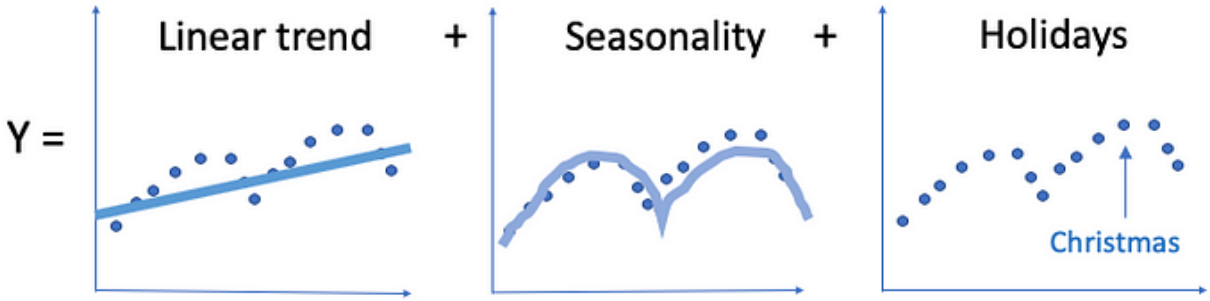


Рисунок 2.1 – Адитивна модель часових рядів у Prophet

Prophet був обраний, тому що він інтуїтивно зрозумілий та простий у використанні. У Prophet автоматизовано багато гіперпараметрів. Він спроектований так, що автоматично вибирає найкращу модель. Також модель має зручний API для моделювання та прогнозування та легко інтегрується з іншими Python бібліотеками, такими як NumPy, Pandas та Matplotlib.

У Prophet передбачено дві альтернативні моделі тренду, які охоплюють більшість кейсів: насичувальне (S-подібне) зростання та кусочно-лінійний тренд із точками зламу.

Для процесів, що прямують до стелі місткості C (carrying capacity), Prophet застосовує узагальнену логістичну криву вигляду (рис. 2.2): $g(t) = \frac{C}{1+exp[-k(t-m)]}$. Логістична функція дозволяє моделювати зріст з насиченням, коли при збільшенні показника зменшується темп його росту (наприклад, кількість реєстрацій, підписників або замовників, що росте S-подібно й рано чи пізно впирається у "стелю" ринку)

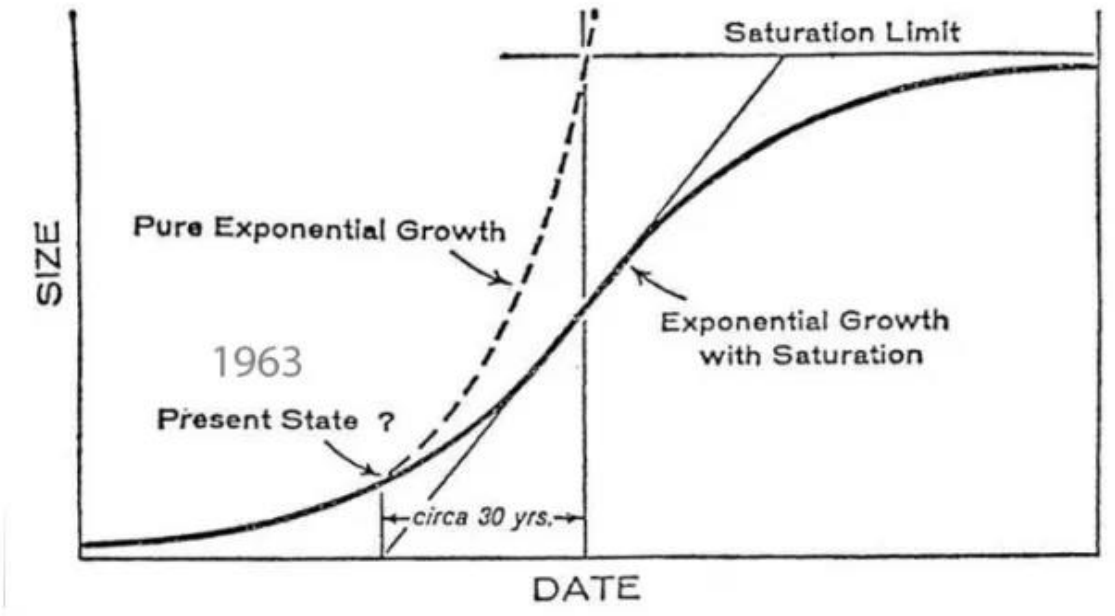


Рисунок 2.2 – Порівняння чистого експоненціального зростання та експоненціального зростання з насиченням

Для задач прогнозування, які не демонструють насичувального зросту, доцільною моделлю є кусочно-лінійний тренд. У цьому випадку тренд задається рівнянням (2.4):

$$g(t) = (k + a(t)^T \delta)t + (m + a(t)^T \gamma), \quad (2.4)$$

де k – базовий коефіцієнт зростання

a(t) – індикаторна функція для точок зламу

δ,γ – зміни швидкості зростання у певні моменти часу

m – початкове значення

Бізнес-ряди завжди мають повторювані шаблони, які пов’язані з поведінкою людей. Наприклад, 5-денний робочий тиждень може мати вплив на часовий ряд, який повторюються щотижня, тоді як графіки відпусток і шкільних канікул можуть спричиняти ефекти, які повторюються щороку.

Сезонність описується через розклад в ряд Фур'є, що дозволяє моделювати плавні та періодичні коливання за формулою (2.5):

$$s(t) = \sum_{n=1}^{N} (a_n cos(\frac{2\pi nt}{P}) + b_n sin(\frac{2\pi nt}{P})), \quad (2.5)$$

де P – період;

N – кількість гармонік;

an, bn – вагові коефіцієнти.

Порядок фур'є визначає, наскільки швидко сезонна складова може змінюватися (за замовчуванням – 10 для річної сезонності та 3 для тижневої). Збільшення цього порядку дозволяє моделі підлаштовуватися під більш стрімкі цикли, але надто великі значення може спричинити перенавчання.

Оцінювання сезонної компоненти потребує 2N параметрів $\beta = [a_1 b_1, \ldots, a_n b_n]^T$. Це робиться за формулою (2.6): для кожного моменту часу t у наших історичних та майбутніх даних будується вектор сезонних ознак, на їх основі формується матриця, по якій розраховуються параметри.

$$X(t) = [cos(\frac{2\pi t}{P}), sin(\frac{2\pi t}{P}), \ldots, cos(\frac{2\pi N t}{P}), sin(\frac{2\pi N t}{P})], \qquad (2.6)$$

Тоді сезонність $s(t) = X(t)\beta$.

Свята і окремі події, які не мають конкретної періодичності, спричиняють суттєві стрибки у багатьох рядах. Тому їх вплив погано описується плавною сезонною хвилею.

У моделі Prophet їх вплив описується окремою адитивною складовою h(t). Нехай є L подій і для кожної події j задано множину її дат D_j. Тоді вводиться бінарний індикатор $Z_j(t) = 1_{\{t \in D_j\}}$, що дорівнює 1 у ті дні, коли подія j триває, й дорівнює 0 решту часу. Зібравши індикатори всіх подій у вектор $Z(t) = [Z_1(t), Z_2(t), \ldots, Z_L(t)]$ описуємо внесок свят як лінійну комбінацію $h(t) = Z(t)\kappa = \sum_{j=1}^{L} \kappa_j Z_j(t)$. Так само як і з сезонністю використовується пріор $\kappa \sim N(0, v^2)$

Модель дає для кожного моменту часу t не просто точковий прогноз $\hat{y}_t$, а й довірчий інтервал прогнозу $[y_t^{lower}, y_t^{upper}]$. Спостереження y_t вважається аномальним, якщо воно виходить за межі свого довірчого інтервалу.

2.5 Метрики оцінки якості прогнозування

У цьому розділі розглядаються основні метрики, які будуть застосовуватись для оцінки точності прогнозу нашої моделі Prophet. Вони відіграють ключу роль, адже саме від точності самої прогнозної моделі залежить коректність виявлення аномалій. Низькі значення MAE і MSE свідчать про невеликі абсолютні та квадратичні відхилення прогнозу від реальних даних, а невисокий MAPE підтверджує, що помилка прогнозу є відносно малою стосовно масштабу спостережень. Оскільки Prophet будує довірчі інтервали навколо прогнозу, чим точніше модель описує тренд і сезонність, тим вужчі виявляються ці інтервали, і тим надійніше стає механізм автоматичного виявлення аномалій, тому що лише справді нетипові відхилення будуть виходити за межі прогнозованого діапазону.

2.5.1 Абсолютна помилка

Абсолютна помилка (MAE) – це середнє абсолютне відхилення прогнозу від фактичних значень. Чим менше значення MAE, тим "ближче" в середньому знаходиться прогноз до реальних даних. Вираховується за формулою (2.7)

$$MAE = \frac{1}{n}\sum_{t=1}^{n} |y_t - \hat{y}_t|, \tag{2.7}$$

де n – загальна кількість прогнозних точок;

y_t – фактичне значення метрики в момент часу t;

$\hat{y}_t$ – прогнозне значення метрики в момент часу t.

2.5.2 Середня квадратична помилка

Середня квадратична помилка (MSE) – це середнє значення квадратів різниці між прогнозованими та фактичними значеннями. За рахунок того, що помилку прогнозу ми зводимо в квадрат, це робить MSE чутливою до окремих великих помилок (викидів). Вираховується за формулою (2.8)

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2, \tag{2.8}$$

де n – загальна кількість прогнозних точок;

y_t – фактичне значення метрики в момент часу t;

$\hat{y}_t$ – прогнозне значення метрики в момент часу t.

2.5.3 Середня абсолютна відносна помилка

Середня абсолютна відносна помилка (MAPE) – це середнє абсолютне відхилення прогнозу від фактичних значень, виражене у відсотках від самим фактичних величин. Вираховується за формулою (2.9)

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|\%, \tag{2.9}$$

де n – загальна кількість прогнозних точок;

y_t – фактичне значення метрики в момент часу t;

$\hat{y}_t$ – прогнозне значення метрики в момент часу t.

2.6 Висновки до розділу 2

У цьому розділі спочатку було розглянуто основні компоненти інструментації та телеметрії у веб-застосунках на прикладі OpenTelemetry - фреймворку, що узагальнює підходи до збору даних у вигляді метрик, логів та трейсів. Також розглянули, які сучасні платформи найчастіше використовуються для подальшої обробки, аналізу та візуалізації цих даних.

Також було розглянуто поняття аномалій в часових рядах та наведено типові сценарії аномалій у веб-середовищі: зміни обсягу HTTP-трафіку, збільшення часу обробки запитів, різке збільшення помилок або навантаження на систему. Методи для виявлення аномалій було поділено на дві групи: класичні статистичні методи (Z-score та IQR), що базуються на розрахунку порогів відхилення від середнього, та методи з використанням машинного навчання (Prophet), які моделюють тренд і сезонність для автоматичного розпізнавання нестандартних відхилень.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРЕМЕНТАЛЬНА ПЕРЕВІРКА МЕТОДІВ МОНІТОРИНГУ Й ВИЯВЛЕННЯ АНОМАЛІЙ

3.1 Опис MVP для експерименту

Для проведення практичного експерименту був реалізований простий веб-застосунок для введення списку задач та цілей (Daily Planner). Для нього буде реалізована система моніторингу технічних, а також протестовані описані в минулому розділі методи для виявлення аномалій в його роботі.

3.1.1 Архітектура веб-додатка

Проєкт було побудовано за схемою клієнт-серверної архітектури. Клієнтська частина була розроблена на базі UI-бібліотеки React.js [14], що відповідає за інтерфейс та взаємодію з API. Основні причини, чому була обрана ця бібліотека.

1. Швидка та проста розробка.
2. Популярність серед великих IT-компаній. React активно використовують такі компанії як Meta, Netflix, Spotify та інші популярні в IT компанії.
3. Гнучка інтеграція з моніторингом. React має вже готові пакети для зручної інтеграції OpenTelemetry в свій проект.

Серверна частина була написана на базі Node.js + Express [15]. Основні переваги.

1. Легкість у розробці API. Express має мінімалістичний синтаксис і дозволяє швидко створювати маршрути (routes), обробники помилок, middleware.
2. Так само має вже готові агенти та пакети для повноцінної інтеграції з моніторингом.
3. Можливість масштабування у майбутньому.

Базу даних було обрано MongoDB [16] – документоорієнтовану NoSQL-базу, яка зберігає інформацію у вигляді JSON-подібних документів. Основною причиною чому був зроблений цей вибір, це те що застосунок не передбачає складної взаємозалежності між таблицями чи сутностями. Структура даних є відносно простою, тому використання класичної реляційної СУБД є недоцільним.

3.1.2 Опис функціональних модулів

Для проведення експерименту були розроблено декілька модулів.

Модуль аутентифікації та авторизації (рис. 3.1). Перше, що пропонується зробити користувачу – це зареєструватися та увійти в свій аккаунт, щоб почати створювати власні задачі. Після входу користувачу видається спеціальний токен, який прикріплюється до кожного подальшого запиту. Це дозволяє впевнено сказати, хто саме виконує ту чи іншу дію.

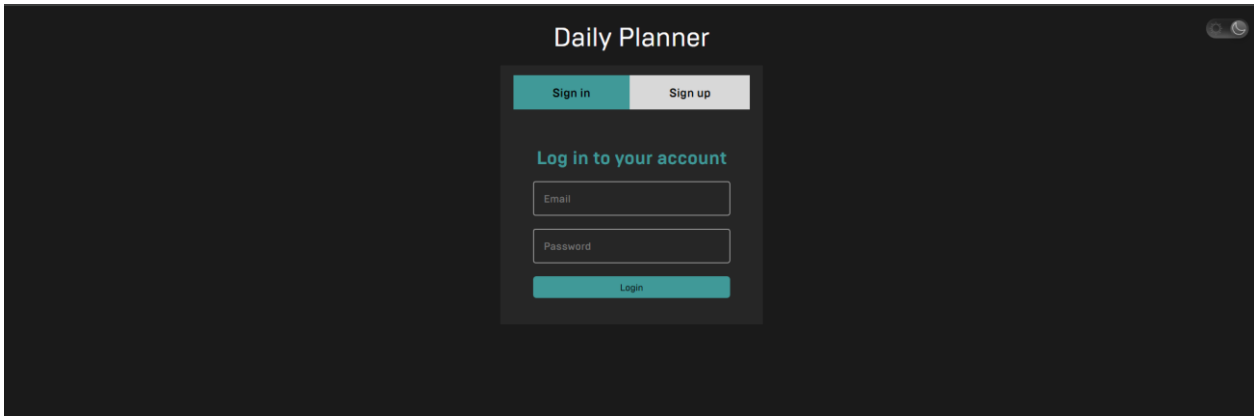


Рисунок 3.1 – Модуль аутентифікації та авторизації

Створені endpoints, які використовуються в цьому модулі.

1. POST api/user/registration – створює нового користувача. Виконує валідацію, перевірку на співпадіння email та додає інформацію в базу даних.
2. POST api/user/login – приймає облікові дані користувача (email та пароль) і виконує перевірку на відповідність. У разі успішної відповіді генерує токен для подальшого доступу.

В рамках цього модуля ми будемо відслідковувати успішні та неуспішні виконання запитів, кількість зареєстрованих користувачів, кількість зареєстрованих або авторизованих протягом останньої доби.

Другий модуль відповідає за *роботу із задачами* (рис 3.2). Він є основним модулем, який є головною частиною додатку. Він відповідає за можливість створення, редагування або видалення задачі. Всі ці дії обробляються сервером та зберігаються в базі даних.

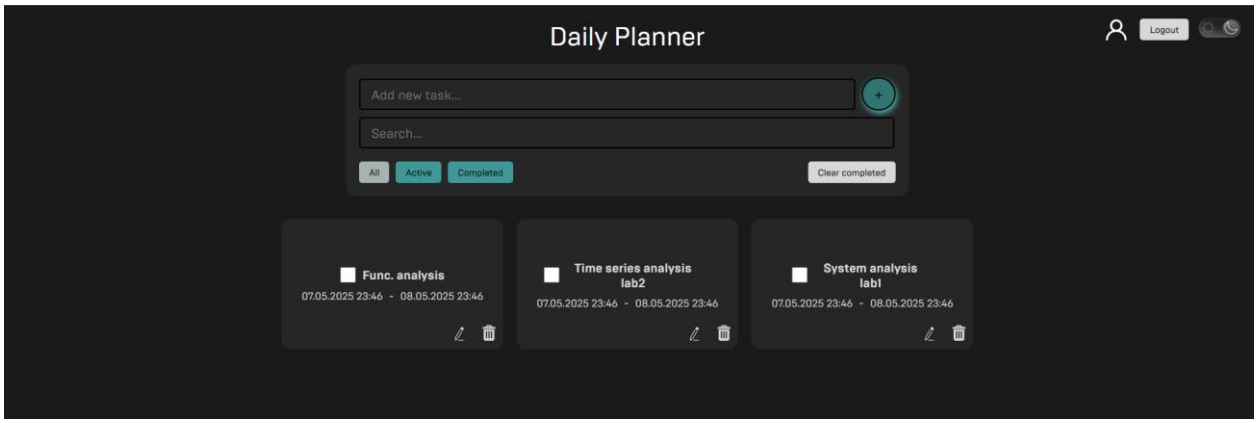


Рисунок 3.2 – Модуль роботи із задачами

Створені endpoints, які використовуються в цьому модулі.

1. GET api/tasks – отримує список задач, які належать конкретному користувачу. Запит перевіряє наявність токену та віддає записи, які зв’язані з цим користувачем.
2. POST api/tasks – додає нову задачу до списку користувача.
3. PUT api/tasks/taskId – дозволяє редагувати існуючі задачі.
4. DELETE api/tasks/taskId – видялає конкретну задачу.

В цьому модулі ми будемо так перевіряти кількість успішних та неуспішних запитів, а також вести детально статистику про кількість створених/видалених/відредагованих задач.

Третій модуль – це модуль *профіля користувача*. Даний модуль дозволяє користувачу на окремій сторінці побачити свою персональну інформацію (рис. 3.3), а також є можливість змінити пароль (рис 3.4).

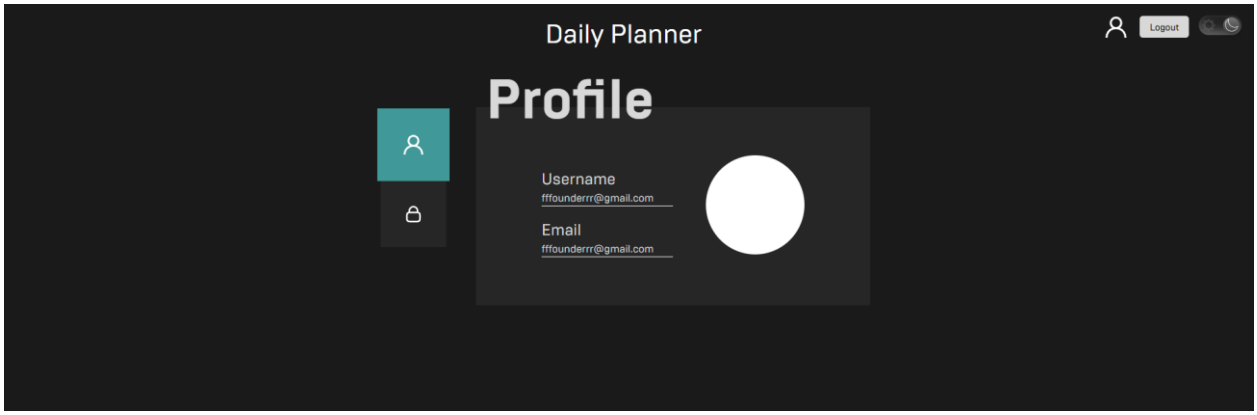


Рисунок 3.3 – Модуль профіля користувача

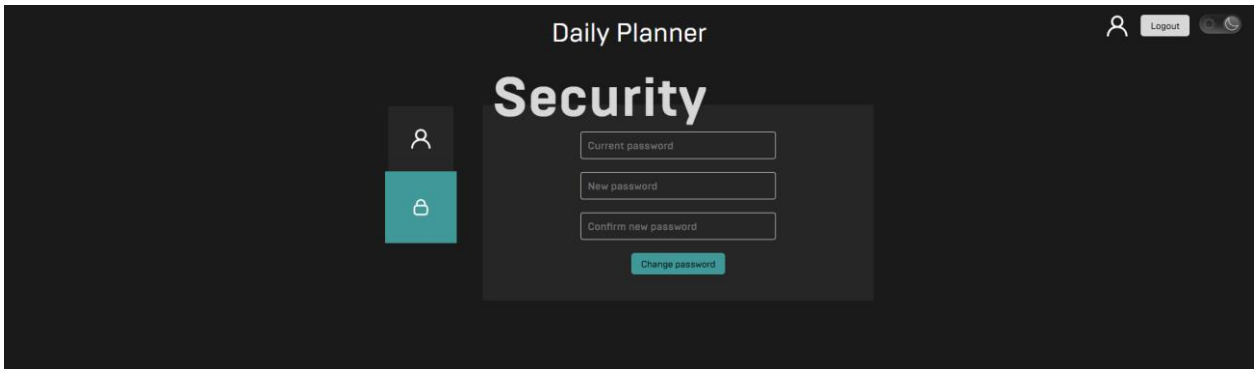


Рисунок 3.4 – Модуль профіля користувача для зміни пароля

Створені endpoints, які використовуються в цьому модулі.

1. GET api/user/profile – повертає інформацію про поточного користувача (username та email).
2. POST api/user/change-password – виконує зміну пароля користувача. Приймає новий та старий паролі. Запит перевіряє правильність актуального паролю, та у разі відповідності змінює на новий.

В рамках даного модуля буде перевіритися успішність та неуспішність виконання запитів.

3.2 Інтеграція OpenTelemetry

Для зборку телеметрії з серверної частини застосунка було використано офіційний SDK OpenTelemetry. Для його інтеграції в проєкт спочатку обираються базові компоненти: інтерфейси API для створення трейсів та метрик, ядро SDK, яке відповідає за обробку подій, інструментатори для автоматичної інтеграції з HTTP-запитами, а також експортери, що передають зібрану інформацію до зовнішніх систем. Далі вже налаштовується сам модуль, який зазвичай називають otel та зберігають в корні проєкта. Там відбувається імпорт необхідних компонентів, налаштування інструментування HTTP-запитів і бази даних, вибір процесора та експортера, а потім ініціалізація та запуск самого SDK для збору й передачі трас та метрик.

3.3 Налаштування системи моніторингу

В якості системи моніторингу для експерименту було обрано Elastic Stack. Основна його перевага полягає, що він є відкритим, активно підтримуваним та універсальним проєктом, який здатен одночасно обробляти

та зберігати в одному середовищі метрики, логи і трейси [17]. Також він має повну підтримку стандартну OTLP (OpenTelemetry Protocol), що дозволяє легко інтегрувати його використовуючи OpenTelemetry SDK без створення додаткових конекторів або іншої логіки. Це зменшує точку відмови й зменшує затримки при передачі даних.

Також Elastic Stack має низький поріг входу, що є важливим аргументом для швидкого впровадження в MVP. Він легко запускається локально або в container середовищі (наприклад Docker), або є опція розгорнення в офіційній хмарній версії платформи Elastic Cloud, не витрачаючи час на налаштування серверів, що і було зроблено в рамках цієї роботи. Обидва сценарії добре документовані та підтримуються спільнотою.

До складу стеку, який буде використовуватись в цій роботі входять такі основні компоненти.

1. ElasticSearch – високопродуктивна пошукова та аналітична система, яка зберігає та індексує телеметрію.
2. LogStash або OpenTelemetry Collector – для збору та попередньої обробки даних.
3. Kibana – для візуалізації та створення графіків, діаграм та загальних дашбордів.
4. Elastic APM – для автоматичного моніторингу додатка.

3.3.1 Огляд APM-сервісів

Після налаштування базового налаштування всіх APM-агентів, основна вкладка для початку роботи – це Services (рис. 3.5). Тут ми можемо побачити всі приєднані та активні мікросервіси, додатки підключених до нашої системи моніторингу та спостереження. Тут ми бачимо список всіх зареєстрованих

сервісів, в нашому випадку це один сервіс daily-planner-backend, для якого ми будемо відстежувати роботу всіх наших описаних вище ендпоінтів. Поруч з кожним сервісом відображаються основні показники у вигляді середнього часу відповіді (Latency), пропускної здатності (Throughput) і відсоток невдалих транзакцій (Failed transaction rate). За допомогою пошуку та фільтрації можна швидко знайти потрібний нам сервіс та одразу отримати одна з найважливіших показників.

Також опція порівняння з попереднім періодом дозволяє миттєво оцінити динаміку змін продуктивності та помилок.

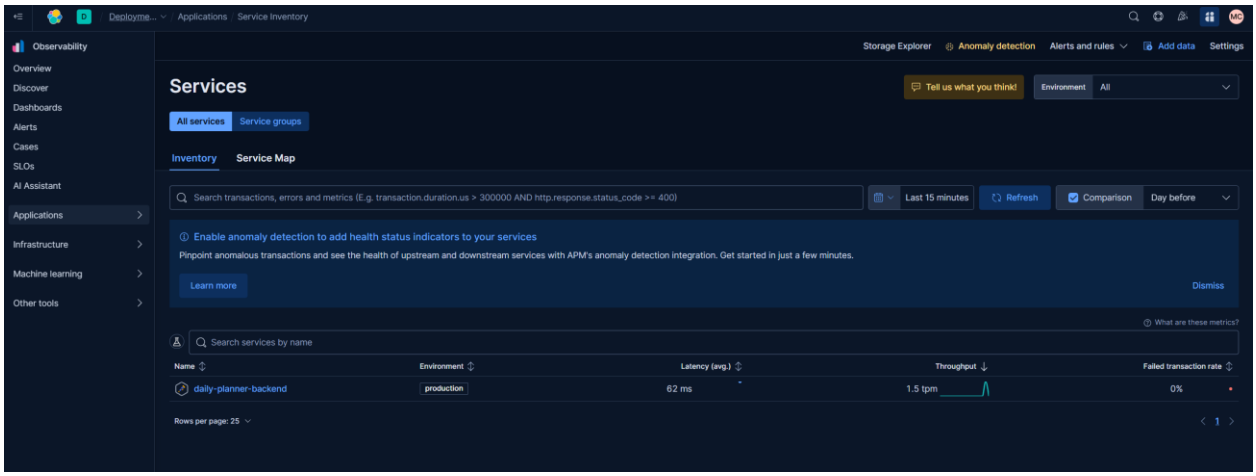


Рисунок 3.5 – Вкладка services

При натисканні на назву сервісу відкривається детальна сторінка обраного мікросервісу (рис. 3.6), де ми можемо детально продивитись всю його поведінку. В розділі Overview розміщуються візуалізація всіх основних показників з переліком всіх активних API маршрутів для даного сервісу (Transactions).

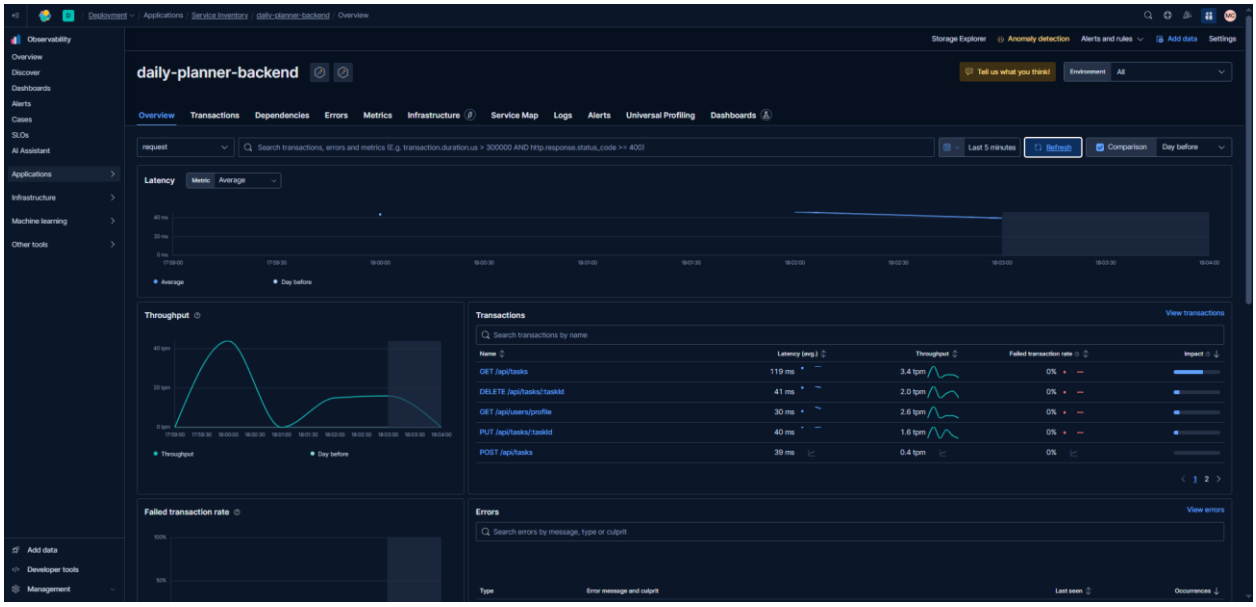


Рисунок 3.6 – Сторінка обраного мікросервісу

Також звідси ми можемо детально продивитись всю наявну інформацію по транзакціями, помилкам, логам тощо. А в розділі Service Map (рис. 3.7) можемо відкрити інтерактивну карту зв’язків між нашими компонентами. Клік на будь-який елемент миттєво відкриває відповідні трейси й метрики, що дозволяє локалізувати вузькі місця та аномальні ланцюжки запитів без переходу до інших розділів. Завдяки фільтрам часу й середовища Service Map швидко перебудовується, показуючи актуальну картину взаємодій навіть у пікові періоди навантаження.

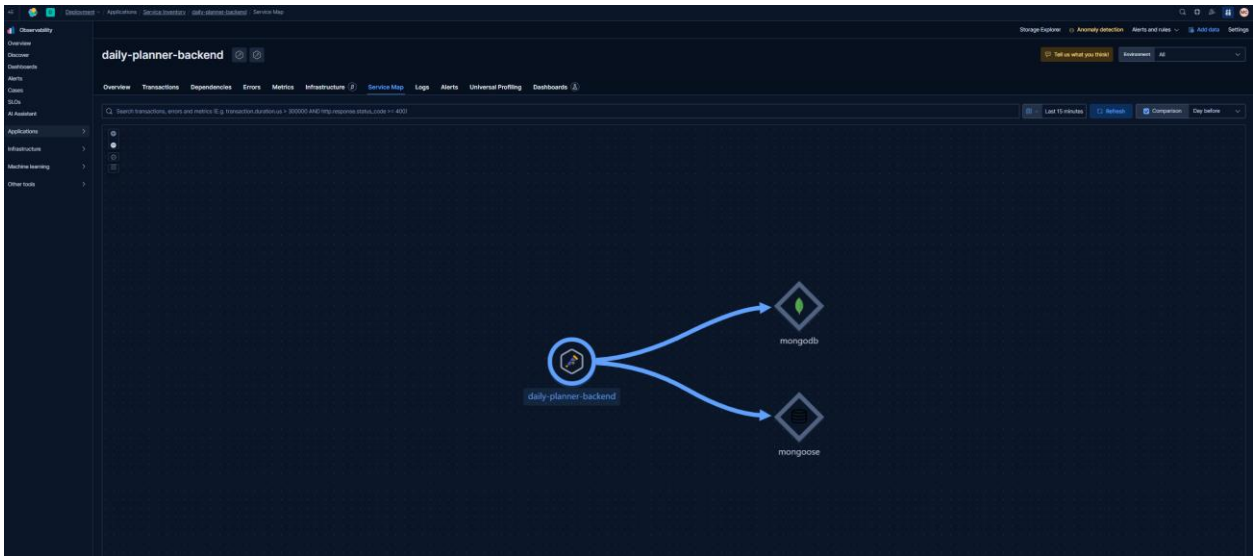


Рисунок 3.7 – Розділ Service Map

3.3.2 Пошуковий аналіз телеметрії

Вкладка Discover (рис. 3.8) – це універсальний для швидкого дослідження сирих даних, які були індексовані. Тут ми маємо можливість заглибитись в потік потрібних для нас логів, метрик чи трасувань. Для швидкого вибору даних можна вводити запити мовою KQL та Lucene, наприклад, відфільтрувати всі помилки 500-го статусу для конкретного сервісу. Якщо виникає потреба переглянути на записи з певним значенням поля, наприклад, за IP-адресою – достатньо клікнути по цьому значенню в бічній панелі, і Kibana миттєво відфільтрує всю таблицю.

Рисунок 3.8 – Вкладка Discover

3.3.3 Створення власних дашбордів

Для спрощення спостереження, візуалізації та порівняння різноманітних метрик, ми маємо можливість створювати власні дашборди. Це дає змогу в одному вікні зібрати всі необхідні графіки, таблиці, діаграми, які детально

будуть описувати метрики, які нас цікавлять. Кожен дашбордів можна приєднати для конкретного сервісу. Це забезпечує швидкий доступ для ключових даних, що дозволяє швидко та зручно відслідковувати стан додатку.

Графік на рисунку 3.9 показує кількість HTTP-запитів до сервера за хвилину. Він дозволяє оцінити загальне навантаження та активність системи.

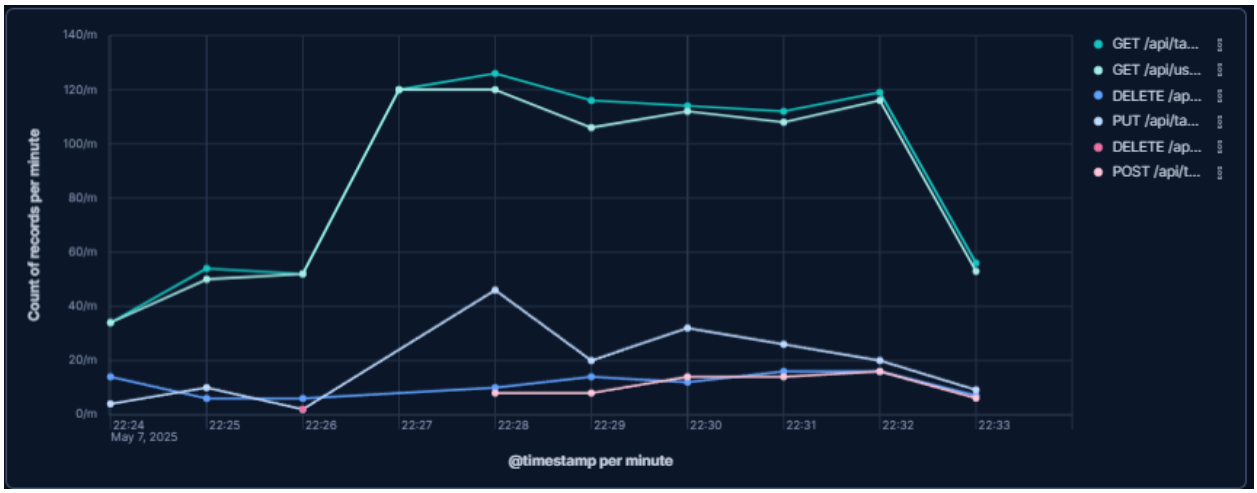


Рисунок 3.9 – Графік кількості HTTP-запитів на хвилину

Графік на рисунку 3.10 відображає середній час відповіді сервера на HTTP-запити, що допомагає виявити затримки та проблеми з продуктивністю. Наприклад, поступове зростання часу відповіді може свідчити про перевантаження серверу або затримку в роботі бази даних. Різкі стрибки вказують на можливі аномалії та проблеми з мережею.

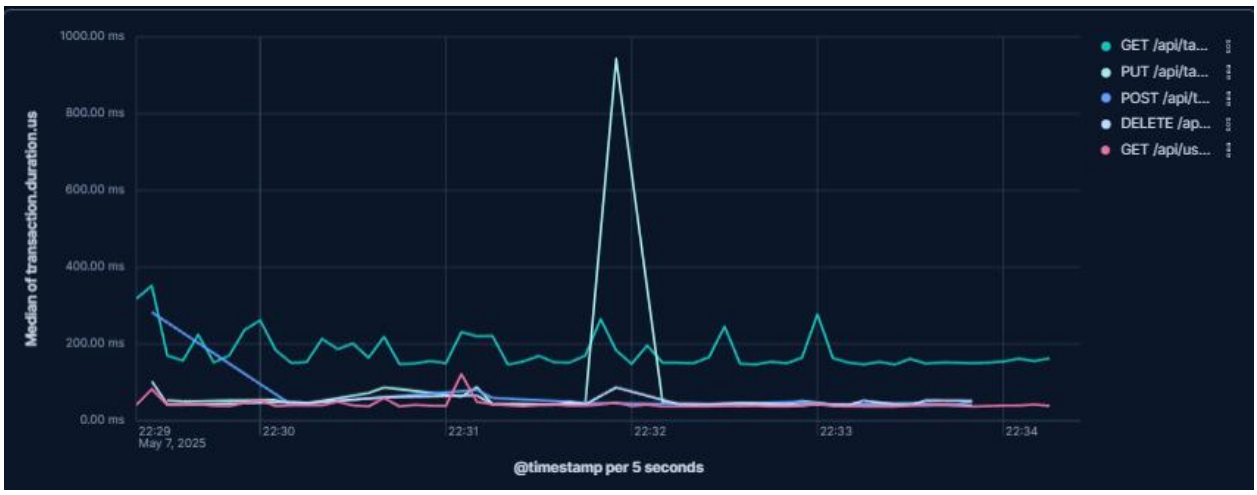


Рисунок 3.10 – Графік середнього часу відповіді HTTP-запитів

На рисунку 3.11 зображено індикатор, який показує кількість HTTP-запитів розподілених за статусами (2xx – успішні, 3xx – перенаправлення, 4xx – клієнтські помилки). Такий формат дозволяє швидко оцінити стан системи. Велика кількість помилок за останню годину сигналізує про якійсь проблеми в роботі.

Рисунок 3.11 – Індикатор кількості HTTP-відповідей за типами статус-кодів за останню годину

Графік на рисунку 3.12 показує рівень використання CPU – центрального процесора на хості, де безпосередньо розміщено веб-застосунок. Є одним з ключових показників стабільності інфраструктури. Постійно великі значення можуть вказувати або на неефективність коду, витоки ресурсів або велику кількість запитів.

Рисунок 3.12 – Графік використання CPU

Графік на рисунку 3.13 показує використання оперативної пам’яті, тобто скільки пам’яті займають запущені процеси на хості. Так само, ця метрика є критично важливою для стабільної роботи інфраструктури. Недостатня кількість доступної оперативної пам’яті може призвести до зниження продуктивності роботи додатка, уповільнення роботи серверу, або взагалі в найгіршому випадку – до крашу процесів.

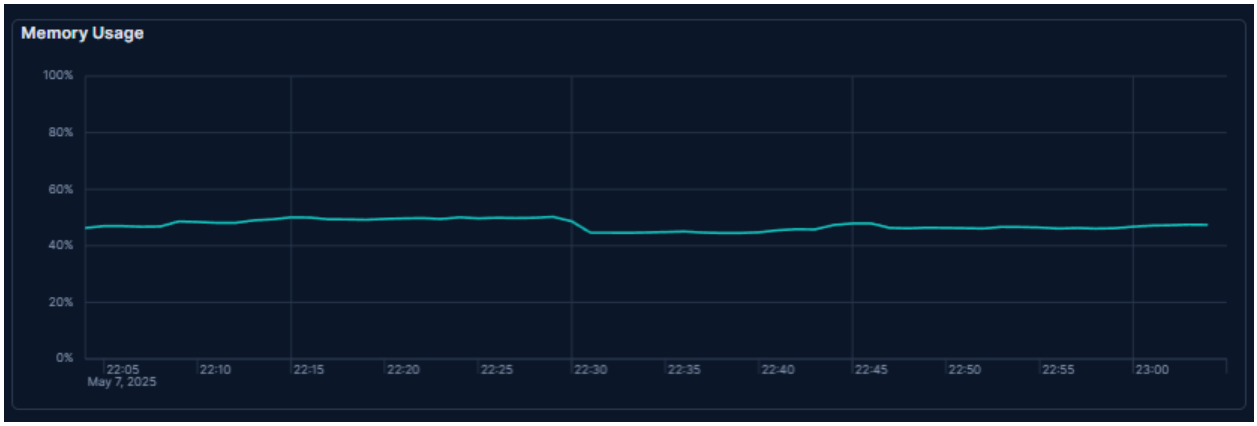


Рисунок 3.13 – Графік використання пам’яті

3.3.4 Налаштування автоматичних сповіщень

Після того, як ключові метрики та графіки зібрані в єдиному дашборді, наступним кроком є налаштування автоматичних сповіщень – алертів. Потрібно це, щоб у разі відхилення показників від очікуваних значень, команда отримувала негайне сповіщення та могла оперативно відреагувати.

У Kibana для цього є механізм Alerts, де ми можемо налаштувати наші "правила" для відпрацювання різних сповіщень. Для цього нам потрібно обрати якусь потрібну нам метрику та задати певний поріг (Threshold). Наприклад, середня затримка відповіді сервісу daily-planner-backend за останні 5 хвилин вища 300 мс, або error_rate більше 5%. Після цього система буде автоматично перевіряти всі наші вказані "правила" в заданий період часу, і у разі спрацьовування надсилати нам сповіщення в саму Kibana. Або додатково

можемо налаштувати відправку сповіщення на пошту, робочий канал в Slack, Telegram тощо.

В рамках експерименту було створено сповіщення на випадок, якщо кількість 4xx (клієнтських) помилок за останні 5 хвилин перевищило 50 штук, з можливістю відправки додаткового сповіщення на пошту. Для цього було налаштовано подібне "правило" на рисунку 3.14.

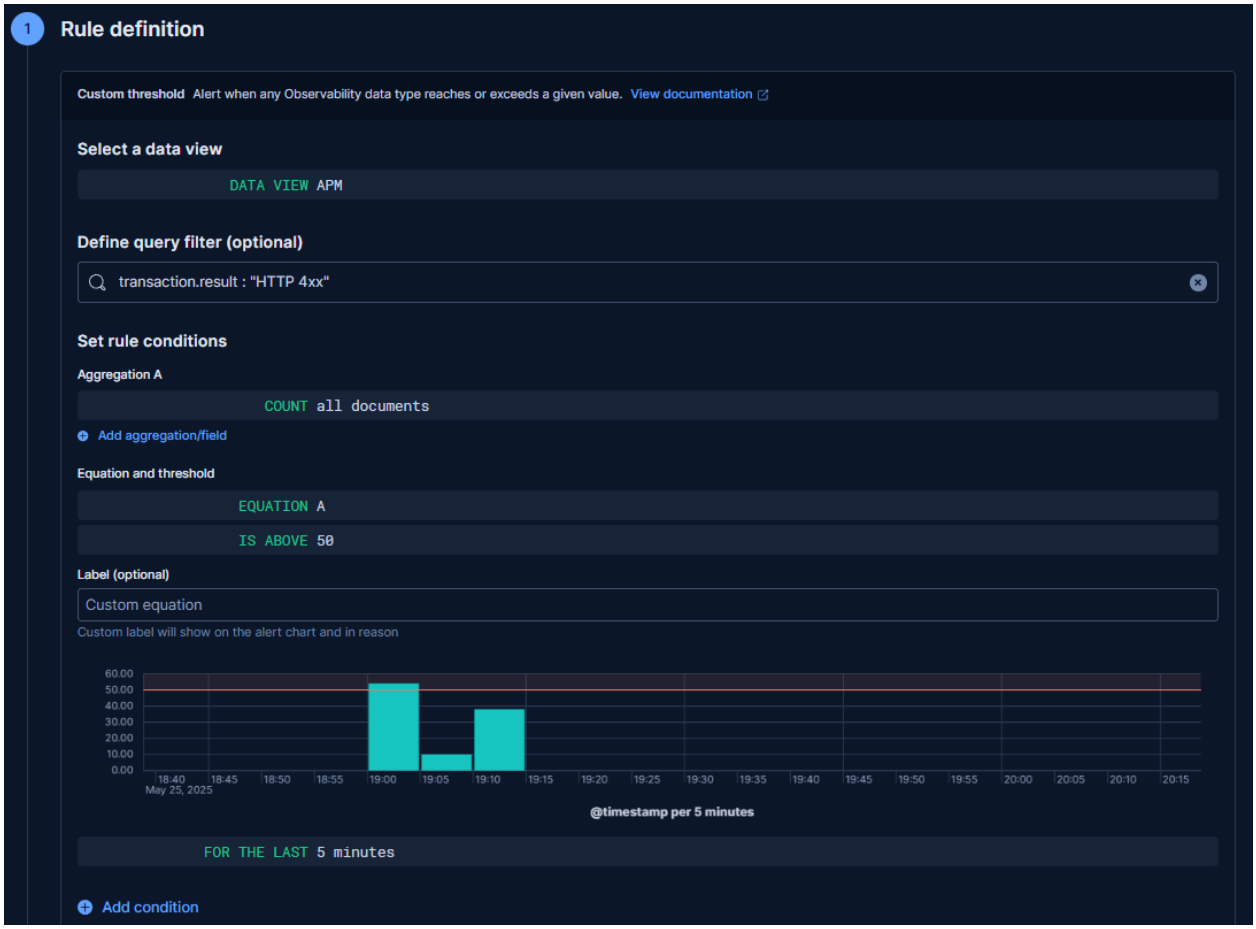


Рисунок 3.14 – Налаштування сповіщення про велику кількість клієнстьких помилок за останні 5 хвилин

Також нижче на рисунку 3.15 показано, що в вкладці Alerts ми можемо подивитися кількість активних сповіщень, та кількість тих, що вже повернулися до норми (Recovered)

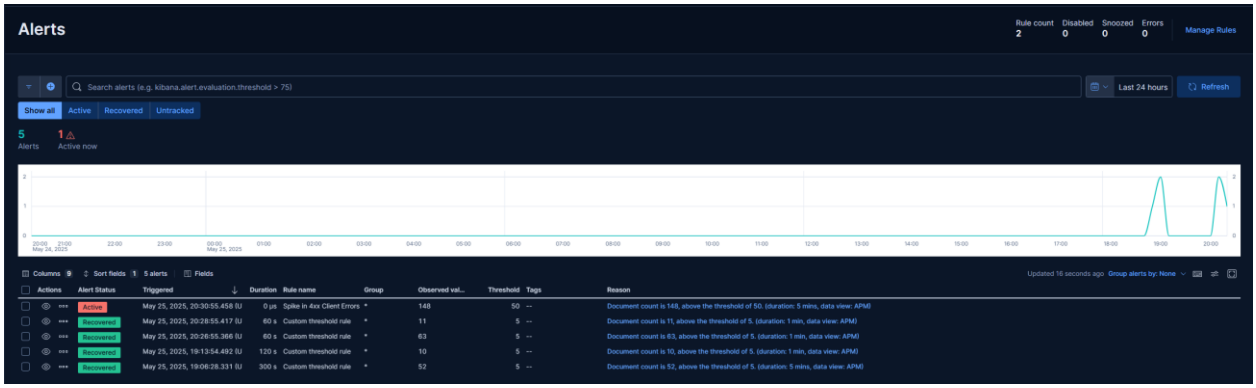


Рисунок 3.15 – Кількість активних сповіщень

Також нижче на рисунку 3.16 показано приклад письма, який Elastic надсилає у випадку виявлення проблеми.

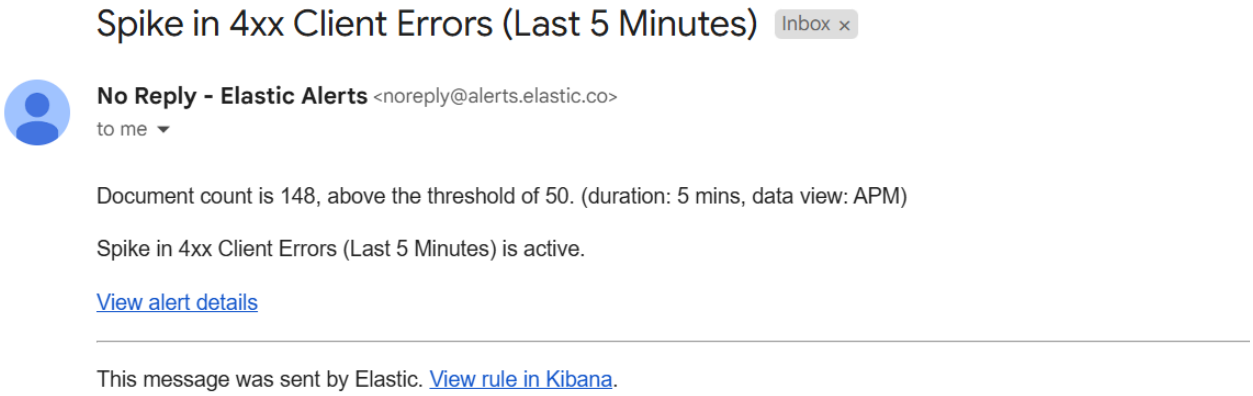


Рисунок 3.16 – Приклад письма у випадка виявлення проблеми

3.4 Реалізація методів виявлення аномалій

3.4.1 Підготовка даних

Зазвичай всі телеметричні дані збираються безпосередньо з бекенд-сервісів моніторингу, які були описані в минулих розділах. Ці сервіси допомагають збирати, фільтрувати, зберігати та візуалізовувати великі обсяги даних, які подальшому будуть використовуватись для аналізу у режимі

реального часу. Однак, для спрощення експерименту та перевірки ефективності алгоритмів, у рамках роботи використовувались вручну згенеровані датасети, які імітують кількість HTTP-запитів та аномалії.

3.4.2 Статистичні методи

При виявленні аномалій за допомогою статистичних методів таких як Z-score та IQR дуже важливим є розмір "локального" інтервалу, за яким обчислюються середні, стандартні відхилення або перцентили. Було обрано трьохгодинне скользяче вікно.

Z-score: на рисунку 3.17 показано графік, на якому Z-score визначив як аномалії тільки найрізкіші зміни: два яскравих спайки трафіку вгору і два глибоких провали, при цьому не реагуючи на звичайні коливання. Це означає, що метод дуже добре відфільтровує "фоновий" шум і не спрацьовує на дрібні відхилення, але водночас може пропускати середньо-інтенсивні спайки.

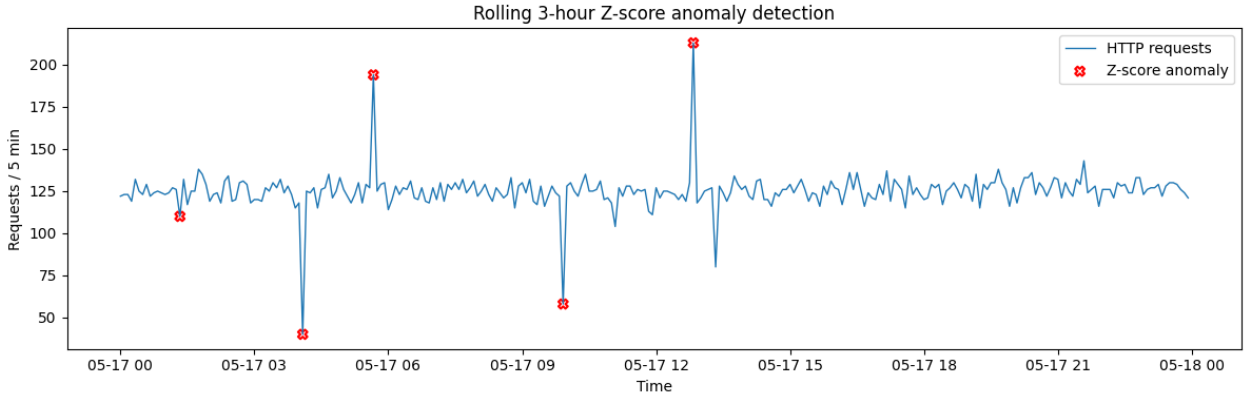


Рисунок 3.17 – Графік результату метода Z-score

IQR: на рисунку 3.18 показано графік вже з використанням IQR-метода, на якому видно, що він позначає не лише найгрубіші сплески, а й помірні відхилення, які Z-score пропустив. На самому початку багато точок позначаються як аномальні через те, що для перших спостережень Rolling IQR

рахує квартилі на дуже малій вибірці, через що межі виявляються надто вузькими.

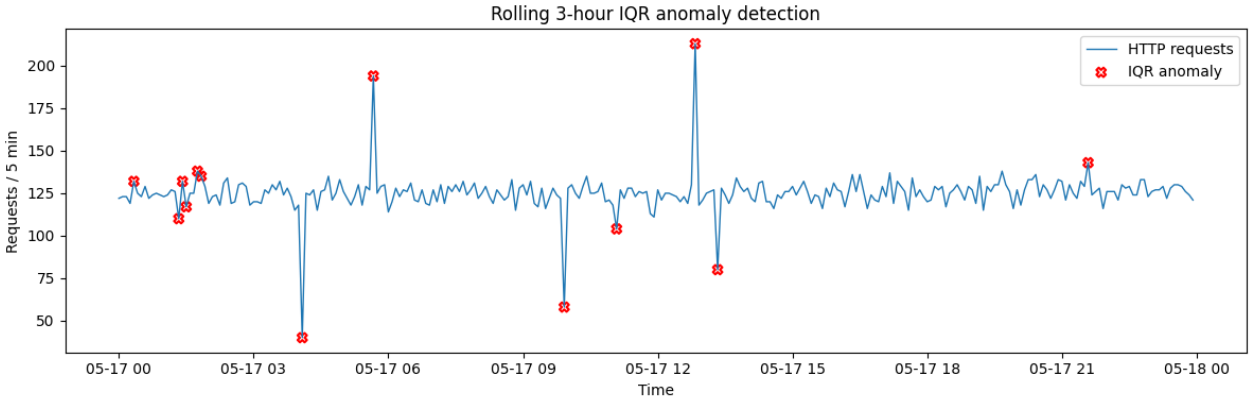


Рисунок 3.18 – Графік результату метода IQR

3.4.3 Реалізація моделі Prophet

Prophet – це інструмент для роботи з одновимірними часовими рядами. Він потребує двох стовпчиків з іменами "ds" та "y". Для цього експерименту використовувався місячний датасет, який показує кількість HTTP запитів з інтервалом 30 хвилин і включає добову та тижневу сезонність у вигляді підвищеного трафіку на вихідних. Він був розділений на навчальну (перші 2 тижня) та тестову частини. Графік зображено на рисунку 3.19.

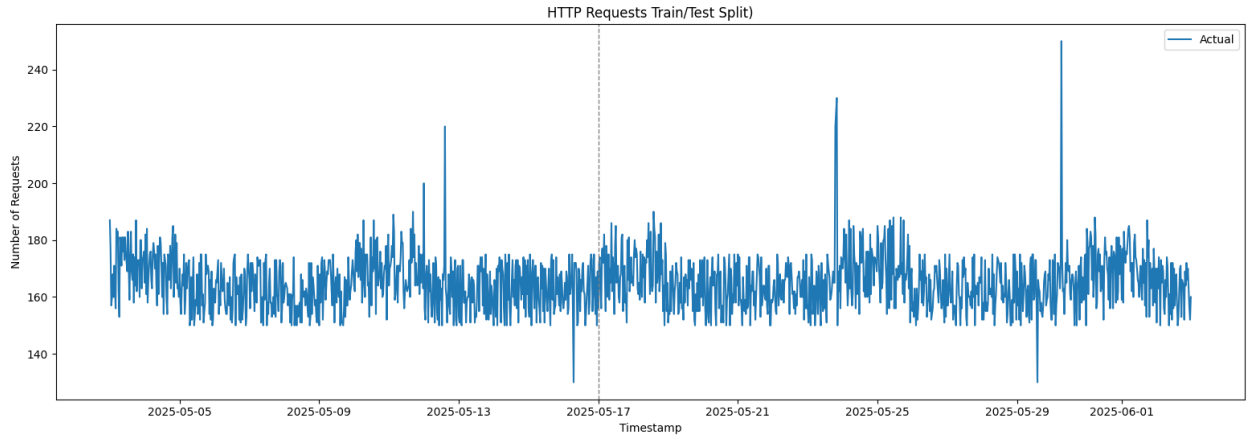


Рисунок 3.19 – Графік тестового місячного датасету

Для побудови моделі прогнозу та подальшого виявлення аномалій відбувається цикл із трьох етапів: ініціалізація, навчання та прогнозування.

На моменті ініціалізації моделі створюється екземпляр Prophet із набором базових параметрів [18].

1. daily_seasonality та weekly_seasonality вмикають гармоніки для 24-годинних та 7-денних циклів.
2. changepoint_prior_scale контролює, наскільки тренд "підлаштовується" під короткострокові зміни.
3. interval_width задає ширину довірчого інтервалу, який необхідний для виявлення аномалій.

Під нас етапу навчання даних відбувається навчання моделі згідно з параметрами. Тут оцінюються задані вище параметри тренду, сезонності та ефектів свят, якщо вони були включені.

На етапі генерації майбутніх даних відбувається створення майбутнього датафрейму з календарними ознаками, наприклад, подовженням періоду та позначення майбутніх свят.

І вже на етапі самого прогнозування вже відбувається саме застосування моделі для передбачення майбутніх значень. Результатом є DataFrame зображений на рисунку 3.20.

	ds	yhat	yhat_upper	yhat_lower
0	2025-05-03 00:00:00	167.717364	187.901310	146.937882
1	2025-05-03 00:30:00	168.355863	191.511251	146.908828
2	2025-05-03 01:00:00	169.026058	190.920303	147.975565
3	2025-05-03 01:30:00	169.618736	191.682037	147.185738
4	2025-05-03 02:00:00	170.050708	195.437065	148.508152
...	...	...	...	...
1483	2025-06-02 21:30:00	163.794364	184.152697	141.935535
1484	2025-06-02 22:00:00	163.326841	185.863739	139.513846
1485	2025-06-02 22:30:00	162.987362	184.652087	142.070787
1486	2025-06-02 23:00:00	162.860822	185.164105	141.730029
1487	2025-06-02 23:30:00	162.980505	183.539901	138.029478

Рисунок 3.20 – Датасет прогнозованих значень

Наведений вище прогноз вихідних містить всі компоненти, необхідні Prophet для візуалізації результатів (рис. 3.21):

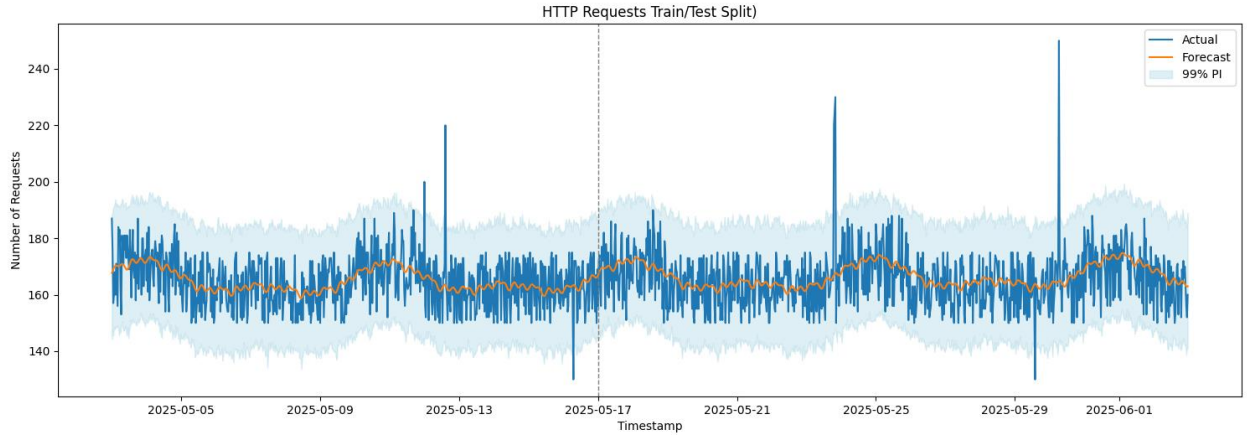


Рисунок 3.21 – Графік прогнозованих значень

Компонентний аналіз прогнозу. Щоб зрозуміти, як сезонність відображаються в моделі і як генеруються прогнози достатньо одного рядка коду model.plot_components(forecast). В результаті ми отримуємо компоненти моделі, оцінені на нашому датасеті [19].

Тижнева сезонність (weekly): графік на рисунку 3.22 показує циклічні коливання, які повторюються щотижня. Бачимо, що в нашому випадку найвищий підйом спостерігається в неділю (+5-6 запитів), а найнижчий у вівторок (-4 запити). Це знову підтверджує, що сценарій з більшим трафіком у вихідні дні був коректно виявлений моделлю.

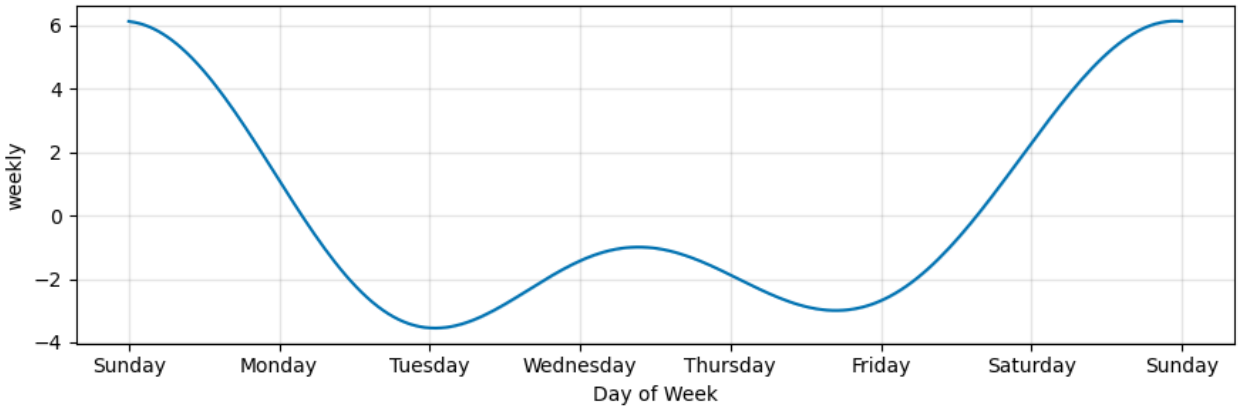


Рисунок 3.22 – Графік тижневої сезонності

Добова сезонність (daily): графік на рисунку 3.23 показує добову сезонність, яка відображає регулярні коливання трафіку протягом 24 годин, тобто як активність користувача змінюється протягом доби. Бачимо, що найнижчий рівень трафіку припадає на ранок, тоді як пік ближче до вечора.

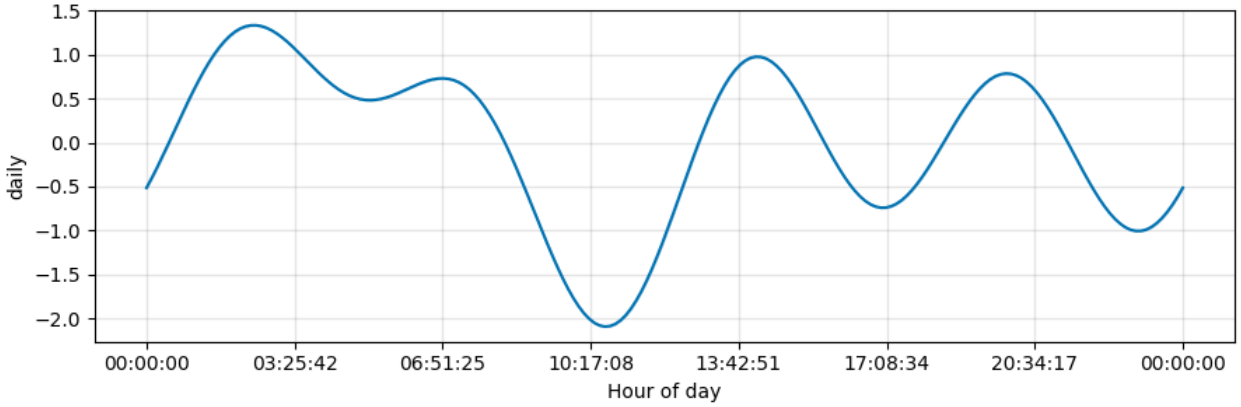


Рисунок 3.23 – Графік добової сезонності

Для оцінки точності побудованого прогнозу були використані такі показники як MAE, MSE та MAPE. За підрахунками модель показала MAE на рівні 7.45 та MSE = 97.77, що означає помірне середнє відхилення від фактичних значень. Відносна помилка MAPE = 4.53%, що свідчить про високу адекватність прогнозу. Узагальнені результати занесені нижче до таблиці 3.1

Таблиця 3.1 – Метрики точності прогнозу моделі Prophet

MAE	MSE	MAPE
7.45	97.77	4.53%

Оскільки якість прогнозу виявилося високою, ми можемо бути впевнені, що модель якісно буде виявляти аномальні точки, а саме ті які вийшли за межі довірчого інтервалу нашого прогнозу [20]. В даному випадку, був використаний 99% довірчий інтервал для того, щоб зменшити кількість хибних спрацьовувань та відслідковувати тільки критичні та неочікувані сплески або провали у трафіку, які зображені на рисунку 3.24.

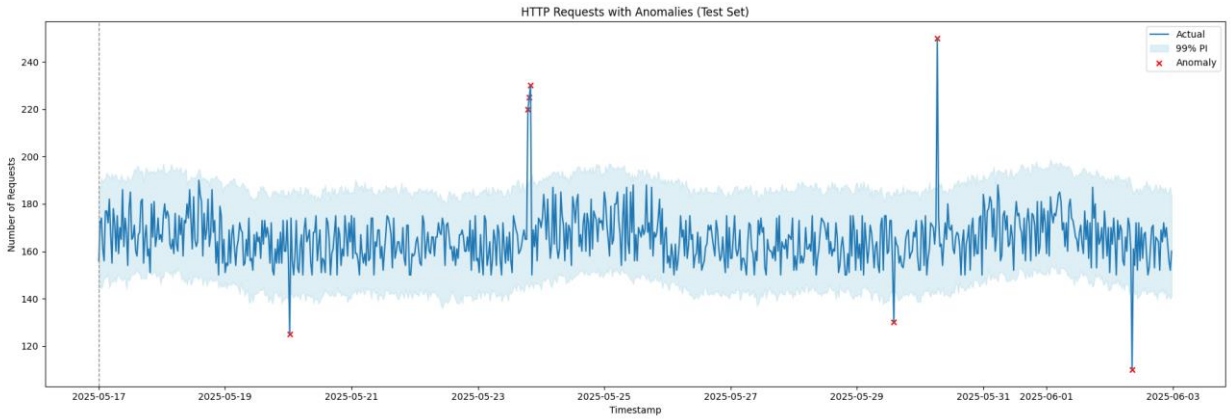


Рисунок 3.24 – Графік з виявленими аномальними точками

Бачимо, що аномальними було позначено 7 точок, які зображені на рисунку 3.25.

```
Detected anomalies at the following timestamps:
2025-05-20 00:30:00
2025-05-23 19:00:00
2025-05-23 19:30:00
2025-05-23 20:00:00
2025-05-29 14:00:00
2025-05-30 06:30:00
2025-06-02 08:30:00
```

Рисунок 3.25 – Виявлені аномальні точки

3.5 Висновки до розділу 3

У цьому розділі була представлена програмна реалізація інтелектуального моніторингу веб-застосунка на прикладі розробленого MVP "Daily Planner". Для нього був налаштовано збір телеметрії за допомогою OpenTelemetry та Elastic APM-агентів, з подальшим налаштуванням всередині Elastic інтерактивних дашбордів для відображення ключових технічно важливих метрик (кількість HTTP-запитів, затримка відповіді сервера, статус

коди відповідей від сервера, навантаження процесора та пам’яті). Також були налаштовані та протестовані прості види сповіщень базуючись на певних порогових значеннях для отримання інформації про конкретні критичні події в роботі застосунка.

Для автоматичного виявлення аномалій були реалізовані власні методи: два статистичних підходи – ковзний Z-score та IQR у трьох годинному вікні - які успішно виявляють раптові стрибки трафіку та короткочасні провали, а також модель Prophet, яка виявляла аномалії на основі побудованого прогнозу. Модель показала високу якість прогнозу на тестовому наборі даних (MAE = 7.45, MSE = 97.77, MAPE = 4.53%), що забезпечує якісний прогноз та мінімальну кількість хибних спрацьовувань при виявленні аномалій. Крім цього, Prophet розкладає часовий ряд на тренд, добову та тижневу сезонність, автоматично підбирає місця змін тренду.

Такий підхід створює гнучку та надійну систему для ефективного моніторингу та спостереження показників веб-застосунків.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі виконується комплексна оцінка ключових характеристик майбутнього програмного забезпечення, що спеціалізується на дослідженні демографічного стану.

Запропонована реалізація забезпечить можливість проведення повних досліджень не лише в Україні, а й у міжнародному масштабі, що дозволить отримати порівняльні висновки щодо демографічних процесів у різних регіонах.

У роботі розглядається різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору для майбутнього продукту. Для цього застосовано метод функціональна-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це підхід, який дозволяє встановити реальну собівартість продукту чи послуги незалежно від організаційної структури. Він проводиться для того щоб виявити резерви зниження витрат ра рахунок оптимізації процесів, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм ФВА включає опис послідовності етапів розробки, розрахунок загальних річних витрат та трудових ресурсів, ідентифікацію основних статей витрат і остаточний підрахунок собівартості програмного продукту.

4.1 Постановка задачі проєктування

У роботі застосовується метод ФВА для технічно-економічної оцінки розробки системи прогнозування стійкості фінансових показників. Оскільки архітектурні рішення та вибір реалізаційних компонентів безпосередньо впливають на ефективність і надійність усієї системи, важливо оцінити кожну підсистему на відповідність загальним вимогам та функціональним завданням.

Фактичний фокус спрямовано на аналіз функцій програмного продукту, який відповідальний за: збір даних з різних джерел, їх попередню обробку та подальшу аналітику показників компанії.

До основних технічних вимог до програмного продукту відносяться:

1. Сумісність із стандартним обладнанням користувача. Програмне забезпечення має коректно працювати на персональних комп’ютерах без необхідності додаткових апаратних модифікацій.
2. Зручність та зрозумілість для користувача. Користувачі повинні легко орієнтуватися в меню програми.
3. Висока продуктивність у реальному часу.
4. Масштабованість та підтримка. Архітектура повинна передбачати просте додавання нових модулів, оновлення та обслуговування без зупинки роботи.
5. Мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. На її основі, можна виділити наступні:

1) F_1 – вибір мови програмування;

2) F_2 – вибір реалізацій базових функцій;

3) F_3 – вибір програмного середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації.

Функція F_1:

1) Python;

2) R.

Функція F_2:

1) Застосування допоміжних бібліотек;

2) Створення власних функцій.

Функція F_3:

1) Visual Studio Code;

2) Jupyter Notebook.

Варіанти реалізацій основних функцій узагальнено та подано у вигляді морфологічної карти системи (рис. 4.1).

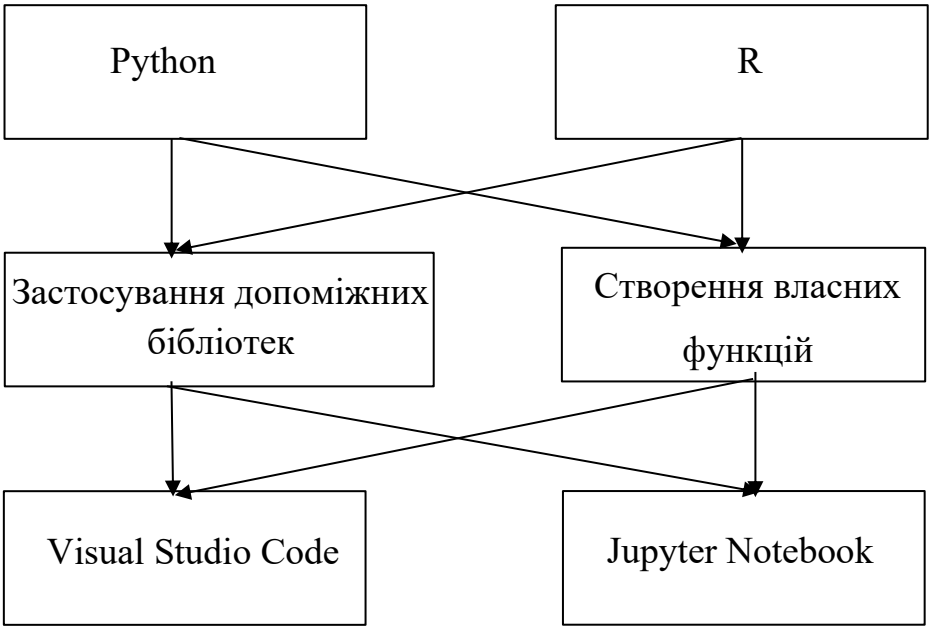


Рисунок 4.1 – Морфологічна карта

Морфологічна карта демонструє варіанти реалізації ключових функцій системи. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Велика екосистема бібліотек, висока читабельність та легкість написання коду	Менша продуктивність у чисельних обчисленнях, потребує оптимізації для великих даних
	Б	Вбудовані статистичні та візуалізаційні можливості, широка підтримка аналітичних пакетів	Менш зручний для побудови складних програмних архітектур, менш універсальний
F_2	А	Швидка розробка із готових рішень, регулярні оновлення та підтримка	Залежність від сторонніх пакетів, ризик невідповідності версій, ліцензійні обмеження
	Б	Повний контроль над кодом, мінімальні зовнішні залежності, можливості оптимізації	Високі трудові затрати, ризик помилок, необхідність власного супроводу коду
F_3	А	Інтерактивність, зручна візуалізація коду та результатів	Неоптимально для масштабних проектів, складна організація структури коду
	Б	Кросплатформна IDE з розширеними можливостями через плагіни, зручне управління проектом	Вимагає початкового налаштування плагінів та конфігурацій для специфічних потреб проекту

За результатом аналізу позитивно-негативної матриці ми встановили, що певні з розглянутих варіантів функцій необхідно відкинути, оскільки вони не відповідають сформульованим для програмного продукту цілям і технічним вимогам. Ці варіанти відзначені на морфологічний карті.

Функція F_1: Перевагу віддаємо мові програмування Python, варіант R відкидаємо як менш відповідний завданням проекту.

Функція F_2: Реалізація першого варіанту є сприйнятливою для поставленої задачі. Це варіант А.

Функція F_3: Припустимі обидва варіанти. Можливо використати варіанти А чи Б

Таким чином, будемо розглядати такі варіанти реалізації ПП.

1. F_1а – F_2а – F_3а.

2. F_1а – F_2а – F_3б.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) X1 – швидкодія мови програмування;

2) X2 – об’єм пам’яті, необхідний для обчислень та збереження даних;

3) X3 – час навчання даних;

4) X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	60	70	120
Об’єм пам’яті	X2	мб	64	32	16
Час навчання даних	X3	мс	1000	400	200
Потенційний об’єм програмного коду	X4	кількість рядків коду	3000	2500	1500

За даними таблиці 4.2 будуються графічні характеристики параметрів – (див. рисунок 4.2 – рисунок 4.5).

Рисунок 4.2 – X1, швидкодія мови програмування

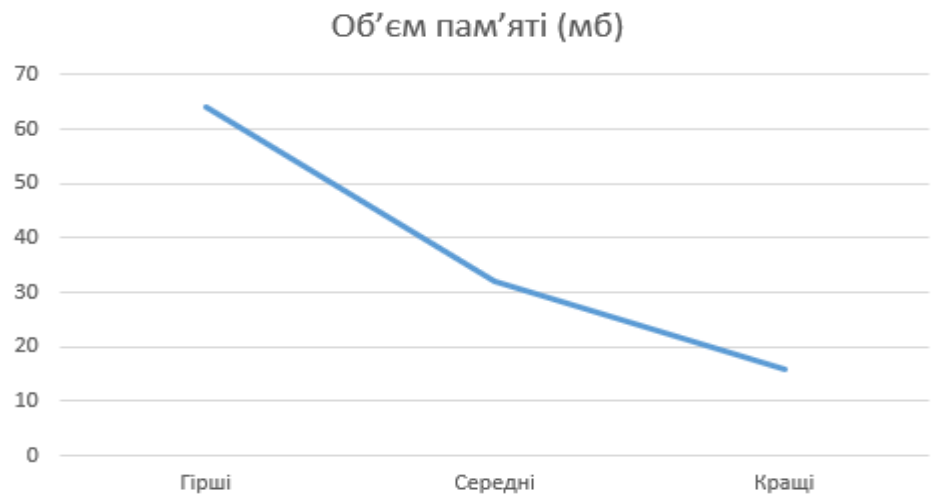


Рисунок 4.3 – X2, об’єм пам’яті

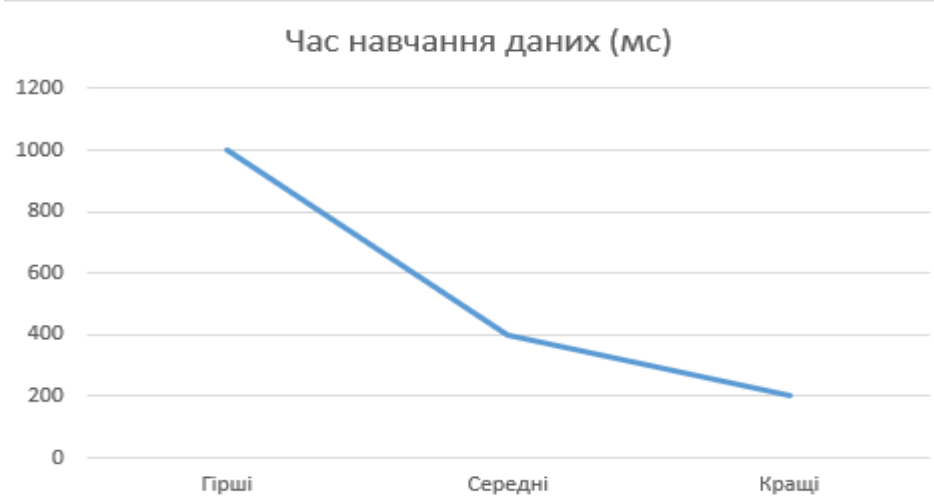


Рисунок 4.4 – X3, час навчання даних

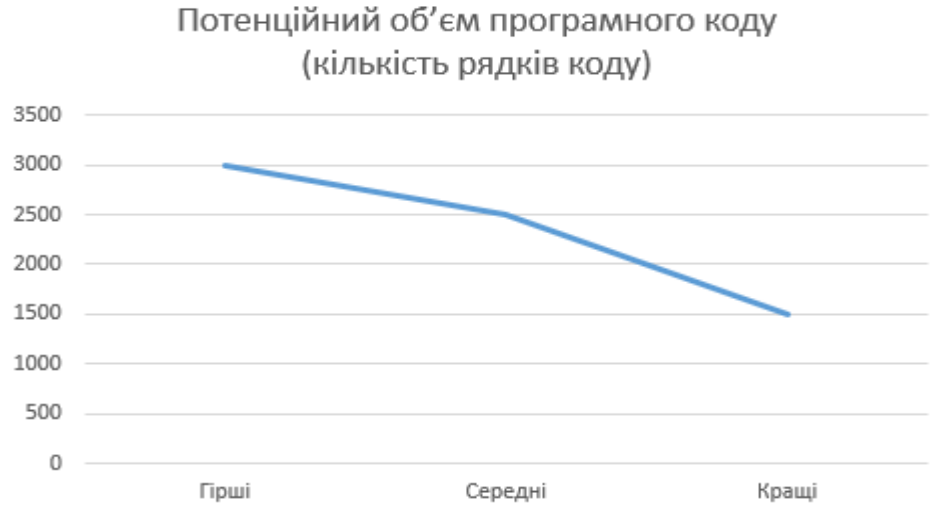


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – створення системи моніторингу показників веб-застосунка та виявлення аномалій в його роботі

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;
2) перевірку придатності експертних оцінок для подальшого використання;
3) визначення оцінки попарного пріоритету параметрів;
4) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	3	4	4	4	3	3	4	25	7.5	56.25
X2	Об’єм пам’яті, необхідний для обчислень	мб	2	2	1	2	1	1	2	11	-6.5	42.25

Продовження таблиці 4.3

X3	Час навчання даних	мс	4	3	3	3	4	4	3	24	6.5	42.25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	1	1	2	1	2	2	1	10	-7.5	56.25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

1) сума рангів кожного з параметрів і загальна сума рангів за формулою (4.1).

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів;

n – кількість параметрів;

2) середня сума рангів за формулою (4.2)

$$T = \frac{1}{n} R_{ij} = 17.5 \quad (4.2)$$

3) відхилення суми рангів кожного параметра від середньої суми рангів за формулою (4.3).

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

4) загальна сума квадратів відхилення за формулою (4.4)

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197 \quad (4.4)$$

Порахуємо коефіцієнт узгодженості за формулою (4.5).

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0.804 > W_k = 0.67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1.5
X1 і X3	<	>	>	>	<	<	>	>	1.5
X1 і X4	>	>	>	>	>	>	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	>	>	<	>	<	<	>	>	1.5
X3 і X4	>	>	>	>	>	>	>	>	1.5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається за формулою (4.6).

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами (4.7) та (4.8).

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На

другому і наступних кроках відносні оцінки розраховуються за наступними формулами (4.9) та (4.10).

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	1.5	1.5	1.5	5.5	0.34	21.25	0.36	77.875	0.36
X2	0.5	1	0,5	1.5	3.5	0.22	12.25	0.21	44.875	0.21
X3	0.5	1.5	1	1.5	4.5	0.28	16.25	0.27	59.125	0.27
X4	0.5	0.5	0.5	1	2.5	0.16	9.25	0.16	34.125	0.16
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 (Швидкодія мови програмування), X2 (Об’єм пам’яті, необхідний для обчислень та збереження даних) та X3 (Час навчання даних) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X4 (потенційний об’єм програмного коду) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою (4.11)

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \tag{4.11}$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості і–го параметра;

B_i – оцінка і–го параметра в балах.

За даними з таблиці 4.6 та за формулою (4.12) визначаємо рівень якості кожного з варіантів.

$$K_K = K_{ТУ}[F_{1k}] + K_{ТУ}[F_{2k}] + \ldots + K_{ТУ}[F_{zk}], \tag{4.12}$$

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	120	7	0.36	2.52
F2	А	X2	16	8	0.21	1.68
F3	А	X3	200	6	0.27	1.62
	Б	X4	2500	5	0.16	0.8

$$К_{К1} = 2.52 + 1.68 + 1.62 = 5.82;$$

$$К_{К2} = 2.52 + 1.68 + 0.8 = 5$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості. Всі варіанти включають в себе два окремих завдання:

1) розробка проекту програмного продукту;

2) розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється за формулою (4.13):

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 50 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це

за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 50 \cdot 1.7 \cdot 0.9 = 76.5 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =30 людино-днів, $К_П$ =0.8, $К_{СК}$ = 1, $К_{СТ}$ = 0.8:

$$Т_2 = 30 \cdot 0.8 \cdot 0.8 = 19.2 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (76.5 + 19.2 + 5.82) \cdot 8 = 812.16 \text{ людино-годин.}$$

$$Т_{II} = (122.4 + 19.44 + 5) \cdot 8 = 1174.72 \text{ людино-годин.}$$

У розробці бере участь один програміст з окладом 30000 грн та один аналітик даних з окладом 25000. Визначимо середню зарплату за годину за формулою (4.14).

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.,} \qquad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{30000 + 25000}{2 \cdot 21 \cdot 8} = 163.69 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою (4.15).

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д , \qquad (4.15)$$

де $С_ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 163.69 \cdot 812.16 \cdot 1.2 = 159530.96$ грн.

II. $С_{ЗП} = 163.69 \cdot 1174.72 \cdot 1.2 = 230747.90$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 159530.96 \cdot 0.22 = 35096.81$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 230747.90 \cdot 0.22 = 50764.53$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0.2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0.2 = 72000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 72000 \cdot (1 + 0.2) = 86400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 86400 \cdot 0.22 = 19008 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 30000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 30000 = 8625 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 30000 \cdot 0.05 = 1725 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.8 =$$

$$= 1516.8 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1516.8 \cdot 0.2 \cdot 0.8 \cdot 9.43 = 2288.54 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 30000 \cdot 0.67 = 20100 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть обчислюватись за формулою (4.16):

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.16)$$

$$С_{ЕКС} = 86400 + 19008 + 8625 + 1725 + 2288.54 + 20100 =$$

$$= 138146.54 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 138146.54 / 1516.8 = 91.07 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу обчислюється за формулою (4.17).

$$С_М = С_{М-Г} \cdot Т, \quad (4.17)$$

I. $С_М = 91.07 \cdot 812.16 = 73963.41$ грн.

II. $С_М = 91.07 \cdot 1174.72 = 106981.75$ грн.

Накладні витрати складають 67% від заробітної плати і обчислюються за формулою (4.18):

$$С_Н = С_{ЗП} \cdot 0.67, \quad (4.18)$$

I. $С_Н = 159530.96 \cdot 0.67 = 106885.74$ грн.

II. $С_Н = 230747.90 \cdot 0.67 = 154601.09$ грн.

Отже, вартість розробки ПП обчислюється за формулою (4.19).

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.19)$$

Отже, для нашого варіанту становить:

I. $С_{ПП} = 159530.96 + 35096.81 + 73963.41 + 106885.74 = 375476.92$ грн.

II. $С_{ПП} = 230747.90 + 50764.53 + 106981.75 + 154601.09 = 543095.27$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою (4.20).

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (4.20)$$

$$К_{ТЕР1} = 5.82 / 375476.92 = 15.5 \cdot 10^{-6},$$

$$К_{ТЕР2} = 5 / 543095.27 = 9.206 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 15.5 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, які залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР1} = 15.5 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

1) вибір мови програмування – Python;

2) вибір реалізацій базових функцій – застування допоміжних бібліотек;

3) вибір програмного середовища розробки – Visual studio code.

Даний варіант виконання програмного комплексу дає користувачу зручну реалізацію програми, зрозумілий інтерфейс та широкий і доступний функціонал для роботи. Гнучкі можливості масштабування дозволяють легко адаптувати рішення під майбутні потреби та інтегрувати його з іншими сервісами.

4.8 Висновки до розділу 4

У цьому розділі було виконано функціонально-вартісний аналіз програмного продукту, а також проведено оцінювання його ключових функцій та характеристик. На основі отриманих результатів визначено основні параметри, що описують роботу системи, і підібрано оптимальний варіант реалізації.

Таким чином, після детального дослідження функцій продукту та їх вартості отримано обґрунтовані рекомендації для подальшої розробки.

ВИСНОВКИ

У сучасних веб-застосунках будь-який несподіваний стрибок навантаження, падіння продуктивності чи настання якоїсь помилки в роботі системи може призвести до простоїв сервісів, втрати клієнтів і мільйонних збитків. У рамках роботи було проаналізоване та розроблено комплексне рішення для інтелектуального моніторингу та спостережуваності з автоматичним виявленням аномальних поведінок.

На початку роботи було розглянуто відмінності між традиційним моніторингом та сучасною спостережуваністю, а також визначено три стовпи останньої – метрики, логи та трейси – як основу для повноти діагностики системи. Було проведено теоретичне дослідження еволюції підходів до спостережуваності – від класичного SNMP до сучасного поєднання OpenTelemetry і Elastic Stack.

Практична реалізація включала налаштування Elastic APM-агентів та Kibana для створення адаптивних дашбордів для візуалізації важливих технічних метрик. Для автоматичного виявлення нетипових відхилень у часі було реалізовані статистичні методи Z-score та IQR у трьох годинному вікні, а також прогнозну модель Prophet, що продемонструвала високу оцінку якості прогнозу. Завдяки своїй адитивній конструкції Prophet автоматично розкладає часовий ряд на довгостроковий тренд, добову та тижневі сезонності та залишкові коливання, а також виявляє точки перелому тренду. Це дозволило створювати високоточний прогноз на прикладі обсягу HTTP-запитів, який враховує як регулярні добові піки, так і підвищений трафік у вихідні.

Отримана система, яка об’єднала збір телеметрії, статистичну обробку, прогнозування Prophet і можливості Elastic Stack, створює надійну й масштабовану платформу для активного моніторингу та своєчасного виявлення інцидентів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What’s the Difference Between Observability and Monitoring*? Amazon Web Services*. URL: https://aws.amazon.com/compare/the-difference-between-monitoring-and-observability/ (last accessed: 02.04.2025).
2. Simple Network Management Protocol (SNMP). *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/simple-network-management-protocol-snmp/ (last accessed: 04.04.2025).
3. Insights from Paper: Google Dapper - A Large-Scale Distributed Systems Tracing Infrastructure. *Medium*. URL: https://medium.com/100paperschallenge/insights-from-paper-google-dapper-a-large-scale-distributed-systems-tracing-infrastructure-1f5a448ca000 (last accessed: 04.04.2025).
4. What Is Zipkin and How Does It Work? *Medium*. URL: https://greyamp.medium.com/what-is-zipkin-and-how-does-it-work-86a628e56a2f (last accessed: 04.04.2025).
5. Three Pillars of Observability. *CrowdStrike*. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/observability/three-pillars-of-observability/ (last accessed: 07.45.2025).
6. OpenTelemetry Documentation. *OpenTelemetry*. URL: https://opentelemetry.io/docs/ (last accessed: 07.04.2025).
7. Majors C., Fong-Jones L., Miranda G. Observability engineering: achieving production excellence. Sebastopol: O’Reilly Media, 2022. 318 p.
8. Collier R., Azarmi B., Montonen C. Installing the Elastic Stack with Machine Learning. Birmingham, UK: Packt Publishing, 2021. 450 p.
9. Anomaly Detection Techniques: How to Uncover Risks, Identify Patterns, and Strengthen Data Integrity. *MindBridge Blog*. URL: https://www.mindbridge.ai/blog/anomaly-detection-techniques-how-to-

uncover-risks-identify-patterns-and-strengthen-data-integrity/ (last accessed: 07.04.2025).

10. Anomaly Detection for Time Series. *Spot Intelligence*. URL: https://spotintelligence.com/2023/03/18/anomaly-detection-for-time-series/ (last accessed: 10.05.2025).
11. Real-Time Anomaly Detection: Use Cases and Code Examples. *Tinybird Blog*. URL: https://www.tinybird.co/blog-posts/anomaly-detection (last accessed: 10.05.2025).
12. Why 1.5 Is Used in the IQR Rule for Outlier Detection. *Built In.* URL: https://builtin.com/articles/1-5-iqr-rule (last accessed: 15.04.2025).
13. Time Series Analysis Using Prophet in Python — Part 1: Math Explained. *Medium*. URL: https://medium.com/analytics-vidhya/time-series-analysis-using-prophet-in-python-part-1-math-explained-5936509c175c (last accessed: 20.04.2025).
14. React – The Library for Web and Native User Interfaces. *React*. URL: https://react.dev/ (last accessed: 10.05.2025).
15. Node.js — Run JavaScript Everywhere. *Node.js*. URL: https://nodejs.org/en (last accessed: 10.05.2025).
16. MongoDB Atlas Documentation. *MongoDB*. URL: https://www.mongodb.com/docs/atlas/ (last accessed: 10.05.2025).
17. Observability with the Elastic Stack. *Elastic Blog*. URL: https://www.elastic.co/blog/observability-with-the-elastic-stack (last accessed: 20.05.2025).
18. Prophet | Forecasting at scale. *Facebook Open Source*. URL: https://facebook.github.io/prophet/ (last accessed: 25.05.2025).
19. Prophet for Business Forecasting. *Medium*. URL: https://dataman-ai.medium.com/prophet-for-business-forecasting-24fda0c7401c (last accessed: 20.05.2025).

20. Anomaly Detection in Time Series. *Neptune.ai*. URL: https://neptune.ai/blog/anomaly-detection-in-time-series (last accessed: 20.05.2025).

ДОДАТОК А ЛІСТИНГ КОДУ ТЕСТОВОГО MVP

client\src\main.tsx

```
import ReactDOM from "react-dom/client";
import { Provider } from "react-redux";
import { BrowserRouter } from "react-router-dom";

import { ThemeProvider } from "./context/ThemeContext";
import { store } from "./redux/store";
import { App } from "./App";

import "./scss/index.scss";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <Provider store={store}>
    <BrowserRouter>
      <ThemeProvider>
        <App />
      </ThemeProvider>
    </BrowserRouter>
  </Provider>
);
```

client\src\App.tsx

```
import { FC } from "react";
import { Routes, Route } from "react-router-dom";

import { routes } from "@/core/routes";
import Layout from "@/components/Layout";
import PrivateRoute from "./components/PrivateRoute";

export const App: FC = () => {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        {routes.map((route) => {
          if (route.private) {
            return (
              <Route
                key={`route-${route.path}`}
                path={route.path}
                element={
                  <PrivateRoute>
                    <route.Element />
                  </PrivateRoute>
                }
              />
            );
          }
          return (
            <Route
              key={`route-${route.path}`}
              path={route.path}
              element={<route.Element />}
            />
          );
        })}
      </Route>
    </Routes>
  );
```

```
};
```

client\src\pages\HomePage.tsx

```
import { FC } from "react";
import Home from "@/components/Home";
import { useFetchUserTasks } from "@/hooks/use-fetch-user-tasks.hook";

export const HomePage: FC = () => {
  useFetchUserTasks();

  return <Home />;
};
```

client\src\pages\LoginPage.tsx

```
import { FC, useEffect } from "react";
import { useNavigate } from "react-router-dom";

import { useAuth } from "@/hooks/useAuth";
import { ROUTE_PATHS } from "@/consts/routePaths";
import { Auth } from "@/components/Auth/Auth";

export const LoginPage: FC = () => {
  const navigate = useNavigate();
  const { isAuth } = useAuth();

  useEffect(() => {
    if (isAuth) {
      navigate(ROUTE_PATHS.HOME);
    }
  }, [isAuth, navigate]);

  return <Auth/>
};
```

client\src\pages\ProfilePage.tsx

```
import { FC, useEffect } from "react";
import { useAppDispatch } from "@/hooks/useAppDispatch";
import { fetchUserProfile } from "@/redux/thunks/auth.thunks";
import Profile from "@/components/Profile";

export const ProfilePage: FC = () => {
  const dispatch = useAppDispatch();

  useEffect(() => {
    dispatch(fetchUserProfile());
  }, [dispatch]);

  return <Profile />;
};
```

client\src\components\Home\Home.tsx

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

import TodoInput from "../TodoComponents/TodoInput";
import ToDoDashboard from "../TodoComponents/TodoDashboard";
import AddTodoButton from "../TodoComponents/AddTodoButton";
import AddTodoModal from "../TodoComponents/AddTodoModal";
import FilterButtons from "../FilterButtons";
import SearchInput from "../SearchInput";
import { selectTodoItems } from "@/redux/selectors/tasks.selectors";
import { getCurrentDateISO, getEndDateISO } from "@/helpers/getDate";
```

```
import { useAppSelector } from "@/hooks/useAppSelector";
import { ITodoItem } from "@/core/api/todo-list/tasks/dto/task.dto";
import { useModal } from "@/hooks/useModal";
import { TodoType } from "@/core/api/todo-list/tasks/dto/task.dto";

import "./Home.scss";

export const Home: FC = () => {
  const { isModalOpen, openModal, closeModal } = useModal();
  const [todo, setTodo] = useState<TodoType>({
    title: "",
    createdDate: getCurrentDateISO(),
    expiredDate: getEndDateISO(),
  });
  const [modalTitle, setModalTitle] = useState("");
  const [validationMessage, setValidationMessage] = useState("");
  const [modalValidationMessage, setModalValidationMessage] = useState("");
  const [editItem, setEditItem] = useState<ITodoItem | null>(null);
  const todoItems = useAppSelector(selectTodoItems);

  const updateTodo = (updates: Partial<typeof todo>) => {
    setTodo((prevState) => ({ ...prevState, ...updates }));
  };

  const resetData = () => {
    updateTodo({
      title: "",
      createdDate: getCurrentDateISO(),
      expiredDate: getEndDateISO(),
    });
    setModalTitle("");
    setValidationMessage("");
    setModalValidationMessage("");
  };

  const handleModalOpen = () => {
    setEditItem(null);
    openModal();
    updateTodo({
      createdDate: getCurrentDateISO(),
      expiredDate: "",
    });
    setModalTitle(todo.title);
  };

  const handleOpenEditModal = (item: ITodoItem) => {
    setEditItem(item);
    openModal();
    updateTodo({
      createdDate: item.createdDate,
      expiredDate: item.expiredDate,
    });
    setModalTitle(item.title);
  };

  return (
    <>
      <div className="enter-block">
        <div className="input-container">
          <TodoInput
            todo={todo}
            updateTodo={updateTodo}
            setValidationMessage={setValidationMessage}
            resetData={resetData}
          />
          <AddTodoButton onAddTodoButtonClick={handleModalOpen} />
        </div>
        {validationMessage && (
```

```
        <p className="validation-message">{validationMessage}</p>
       )}
       <SearchInput/>
       <FilterButtons/>
      </div>
      <ToDoDashboard
       items={todoItems}
       handleOpenEditModal={handleOpenEditModal}
      />
      {isModalOpen && (
       <AddTodoModal
        onClose={closeModal}
        todo={todo}
        modalTitle={modalTitle}
        setModalTitle={setModalTitle}
        updateTodo={updateTodo}
        modalValidationMessage={modalValidationMessage}
        setModalValidationMessage={setModalValidationMessage}
        editItem={editItem}
        resetData={resetData}
       />
      )}
     </>
  );
};
```

client\src\components\Home\Home.scss

```
@import "@/scss/index.scss";

.main {
  display: flex;
  align-items: center;
  flex-direction: column;
  justify-content: center;
}

.enter-block {
  padding: 20px;
  background: var(--background-secondary);
  border-radius: 15px;
}

.input-container {
  display: flex;
  align-items: center;
  gap: 10px;
}

.validation-message {
  margin-top: 5px;
  color: $error;
}
```

client\src\components\Home\index.ts

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

export default Home;
```

client\src\components\TodoComponents\TodoInput\TodoInput.tsx

```
import { FC, KeyboardEvent } from "react";

import { TodoType } from "@/core/api/todo-list/tasks/dto/task.dto";
import { MAX_INPUT_LENGTH } from "@/consts/inputLength";
```

```
import { useAppDispatch } from "@/hooks/useAppDispatch";
import { useSwitchCompletedFilter } from "@/hooks/useCompletedSwitch";
import { isValid } from "@/helpers/isValid";
import { ERROR_MESSAGES } from "@/consts/Messages";
import { KEYS } from "@/consts/keys";
import { addUserTask } from "@/redux/thunks/tasks.thunks";

import "./TodoInput.scss";

interface TodoInputProps {
  todo: TodoType;
  updateTodo: (updates: Partial<TodoType>) => void;
  setValidationMessage: (message: string) => void;
  resetData: () => void;
}

export const TodoInput: FC<TodoInputProps> = ({
  todo,
  updateTodo,
  setValidationMessage,
  resetData,
}) => {
  const dispatch = useAppDispatch();
  const { switchCompletedFilter } = useSwitchCompletedFilter();

  const handleKeyDown = (e: KeyboardEvent<HTMLInputElement>) => {
    if (e.key === KEYS.ENTER && todo.title.trim()) {
      if (isValid(todo.title)) {
        dispatch(
          addUserTask({
            title: todo.title.trim(),
            createdDate: todo.createdDate,
            expiredDate: todo.expiredDate,
          })
        )
        switchCompletedFilter();
        resetData();
      } else {
        setValidationMessage(ERROR_MESSAGES.INVALID_SYMBOLS);
      }
    }
  };

  return (
    <input
      className="todo-input"
      type="text"
      placeholder="Add new task..."
      value={todo.title}
      onChange={(e) => updateTodo({ title: e.target.value })}
      onKeyDown={handleKeyDown}
      maxLength={MAX_INPUT_LENGTH}
    />
  );
};
```

client\src\components\TodoComponents\TodoInput\TodoInput.scss

```
@import "@/scss/index.scss";

.todo-input {
  width: 760px;
  padding: 1rem;
  max-height: 3rem;
  border: 2px solid var(--border-primary);
  border-radius: 5px;
  background-color: var(--input-background);
```

```
  color: var(--text-primary);
  font-size: 1.25rem;

  @include onBig-tablet {
    width: 500px;
  }

  @include onMobile {
    width: 260px;
  }
}
```

client\src\components\TodoComponents\TodoInput\index.ts

```
import { TodoInput } from "./TodoInput"

export default TodoInput;
```

client\src\components\TodoComponents\TodoItem\TodoItem.tsx

```
import { FC } from "react";
import clsx from "clsx";
import parse from "date-fns/parse";

import ConfirmationModal from "../../ConfirmationModal";
import { ITodoItem } from "@/core/api/todo-list/tasks/dto/task.dto";
import { useAppDispatch } from "@/hooks/useAppDispatch";
import { DATE_FORMAT } from "@/consts/dateFormats";
import { useModal } from "@/hooks/useModal";
import TrashIcon from "@/assets/images/trash.svg?react";
import EditIcon from "@/assets/images/edit.svg?react";
import { CONFIRMATION_MESSAGES } from "@/consts/Messages";
import { deleteTask, editTask } from "@/redux/thunks/tasks.thunks";
import { formatDate } from "@/helpers/getDate";

import "./TodoItem.scss";

export interface TodoItemProps {
  item: ITodoItem;
  handleOpenEditModal: (item: ITodoItem) => void;
}

export const TodoItem: FC<TodoItemProps> = ({
  item: { _id, title, createdDate, expiredDate, completed },
  handleOpenEditModal,
}) => {
  const dispatch = useAppDispatch();

  const { isModalOpen, openModal, closeModal } = useModal();

  const handleToggleDone = () => {
    dispatch(editTask({ taskId: _id, taskData: { completed: !completed } }));
  };

  const handleConfirmDelete = () => {
    dispatch(deleteTask(_id));
    closeModal();
  };

  const endDateParsed = parse(expiredDate, DATE_FORMAT, new Date());

  const titleClass = clsx("todo-item__title", { completed });
  const dateClass = clsx("todo-item__date", { completed });
  const todoItemClass = clsx("todo-item", {
    expired: !completed && endDateParsed < new Date(),
```

```
  });
  const editButtonClass = clsx("todo-item__button", { disabled: completed });

  return (
   <>
    <div className={todoItemClass}>
     <div className="todo-item__title-container">
      <input
       type="checkbox"
       checked={completed}
       onChange={handleToggleDone}
       className="todo-item__checkbox"
      />
      <h3 className={titleClass}>{title}</h3>
     </div>
     <p className="todo-item__dates">
      <span className={dateClass}>{formatDate(createdDate)}</span>-
      <span className={dateClass}>{formatDate(expiredDate)}</span>
     </p>
     <div className="todo-item__buttons">
      <button
       className={editButtonClass}
       disabled={completed}
       onClick={() =>
        handleOpenEditModal({
         _id,
         title,
         createdDate,
         expiredDate,
         completed,
        })
       }
      >
       <EditIcon className="todo-item__button-image" />
      </button>
      <button className="todo-item__button" onClick={openModal}>
       <TrashIcon className="todo-item__button-image" />
      </button>
     </div>
    </div>
    {isModalOpen && (
     <ConfirmationModal
      message={`${CONFIRMATION_MESSAGES.DELETE_ONE_TASK} "${title}"?`}
      onConfirm={handleConfirmDelete}
      onClose={closeModal}
     />
    )}
   </>
  );
};
```

client\src\components\TodoComponents\TodoItem\TodoItem.scss

```
@import "@/scss/colors.scss";

.todo-item {
 position: relative;
 display: flex;
 flex-direction: column;
 align-items: center;
 justify-content: center;
 width: 340px;
 height: 200px;
 padding: 10px;
 border-radius: 15px;
 background: var(--background-secondary);
```

```
&.expired {
  border: 2px solid $error;
}

&__title-container {
  display: flex;
  align-items: center;
  max-width: 100%;
  gap: 10px;
  margin-bottom: 10px;
}

&__title {
  max-width: 80%;
  display: inline-block;
  text-align: center;
  word-wrap: break-word;

  &.done {
    text-decoration: line-through;
  }
}

&__dates {
  display: flex;
  gap: 10px;
}

&__date {

  &.done {
    text-decoration: line-through;
  }

}

&__buttons {
  position: absolute;
  bottom: 10px;
  right: 10px;
  display: flex;
  align-items: center;
  gap: 20px;
}

&__button {
  background-color: initial;
  border: none;
  cursor: pointer;

  &.disabled {
    opacity: 0.3;
    cursor: initial;
  }

}

&__button-image {
  width: 24px;
  height: 24px;
  fill: var(--text-primary);
}

&__checkbox {
  width: 24px;
  height: 24px;
  cursor: pointer;
}
```

```
}
```

client\src\components\TodoComponents\TodoItem\index.ts

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

export default TodoItem;
```

client\src\components\TodoComponents\TodoDashboard\TodoDashboard.tsx

```
import { FC } from "react";

import TodoItem from "../TodoItem";
import NoTodoFound from "../NoTodoFound";
import { ITodoItem } from "@/core/api/todo-list/tasks/dto/task.dto";
import { useAppSelector } from "@/hooks/useAppSelector";
import {
  selectError,
  selectIsLoading,
  selectSearchQuery,
} from "@/redux/selectors/tasks.selectors";

import "./TodoDashboard.scss";

interface ToDoDashboardProps {
  items: ITodoItem[];
  handleOpenEditModal: (item: ITodoItem) => void;
}

export const ToDoDashboard: FC<ToDoDashboardProps> = ({
  items,
  handleOpenEditModal,
}) => {
  const searchQuery = useAppSelector(selectSearchQuery);
  const isLoading = useAppSelector(selectIsLoading);
  const error = useAppSelector(selectError);

  if (isLoading) {
    return (
      <div className="todo-dashboard">
        <div>Loading...</div>
      </div>
    );
  }

  if (error) {
    return (
      <div className="todo-dashboard">
        <div>Error: {error}</div>
      </div>
    );
  }

  if (!isLoading && !items.length) {
    return (
      <div className="todo-dashboard">
        {searchQuery ? (
          <p className="nothing-found-message">
            Nothing found for "{searchQuery}"
          </p>
        ) : (
          <NoTodoFound />
        )}
      </div>
    );
  }
```

```
 return (
  <div className="todo-dashboard">
   <ul className="todo-dashboard__list">
    {items?.map((item) => (
     <li key={item._id}>
      <TodoItem item={item} handleOpenEditModal={handleOpenEditModal} />
     </li>
    ))}
   </ul>
  </div>
 );
};
```

client\src\components\TodoComponents\TodoDashboard\TodoDashboard.scss

```
@import "@/scss/index.scss";

.todo-dashboard {
 margin: 15px 0;

 &__list {
  display: flex;
  flex-wrap: wrap;
  margin: 20px 0;
  padding: 0 10px;
  gap: 20px;

  @include onBig-desktop {
   justify-content: center;
   align-items: center;
  }

  @include onMobile {
   flex-direction: column;
  }

 }

}

.nothing-found-message {
 font-weight: bold;
}
```

client\src\components\TodoComponents\TodoDashboard\index.ts

```
import { ToDoDashboard } from "./TodoDashboard";

export default ToDoDashboard;
```

client\src\components\TodoComponents\NoTodoFound\NoTodoFound.tsx

```
import { FC } from "react";

import { NOT_FOUND_MESSAGES } from "@/consts/Messages";
import { useAppSelector } from "@/hooks/useAppSelector";
import { selectFilter } from "@/redux/selectors/tasks.selectors";
import todoIcon from "@/assets/images/todo.png";

import "./NoTodoFound.scss";

export const NoTodoFound: FC = () => {
 const currentFilter = useAppSelector(selectFilter)
 return (
```

```
    <div className="no-todo-found">
      <img className="no-todo-found__image" src={todoIcon} alt="todo Icon" />
      <p>{NOT_FOUND_MESSAGES[currentFilter]}</p>
    </div>
  );
};
```

client\src\components\TodoComponents\NoTodoFound\NoTodoFound.scss

```
.no-todo-found {
  padding: 20px;
  text-align: center;
  margin: 0 auto;
  font-weight: bold;

  &__image {
    width: 100px;
    height: 100px;
  }

}
```

client\src\components\TodoComponents\NoTodoFound\index.ts

```
.no-todo-found {
  padding: 20px;
  text-align: center;
  margin: 0 auto;
  font-weight: bold;

  &__image {
    width: 100px;
    height: 100px;
  }

}
```

client\src\components\TodoComponents\AddTodoModal\AddTodoModal.tsx

```
import {
  FC,
  MouseEvent,
  KeyboardEvent,
  FormEvent,
  ChangeEvent,
  useState,
} from "react";
import { isToday, addMinutes } from "date-fns";
import DatePicker from "react-datepicker";

import CustomButton from "@/components/UI/CustomButton";
import { ITodoItem } from "@/core/api/todo-list/tasks/dto/task.dto";
import { TodoType } from "@/core/api/todo-list/tasks/dto/task.dto";
import { DATE_FORMAT, TIME_FORMAT, TIME_INTERVAL } from "@/consts/dateFormats";
import { MAX_INPUT_LENGTH } from "@/consts/inputLength";
import { getMinDate, getMaxDate } from "@/helpers/getDate";
import { isValid } from "@/helpers/isValid";
import { useAppDispatch } from "@/hooks/useAppDispatch";
import { useSwitchCompletedFilter } from "@/hooks/useCompletedSwitch";
import { ERROR_MESSAGES } from "@/consts/Messages";
import { KEYS } from "@/consts/keys";
import { ButtonTypes, ButtonVariants } from "@/types/buttonTypes";
import { formatDate } from "@/helpers/getDate";

import "react-datepicker/dist/react-datepicker.css";
```

```
import "./AddTodoModal.scss";
import { addUserTask, editTask } from "@/redux/thunks/tasks.thunks";
import { FIVE_MINUTES } from "@/consts/timeFrames";

interface AddTodoModalProps {
  todo: TodoType;
  modalTitle: string;
  editItem: ITodoItem | null;
  modalValidationMessage: string;
  updateTodo: (updates: Partial<TodoType>) => void;
  setModalValidationMessage: (message: string) => void;
  setModalTitle: (modalTitle: string) => void;
  resetData: () => void;
  onClose: () => void;
}

export const AddTodoModal: FC<AddTodoModalProps> = ({
  todo,
  modalTitle,
  updateTodo,
  modalValidationMessage,
  setModalValidationMessage,
  setModalTitle,
  editItem,
  resetData,
  onClose,
}) => {
  const dispatch = useAppDispatch();
  const { switchCompletedFilter } = useSwitchCompletedFilter();

  const [isTodayDateSet, setIsTodayDateSet] = useState<boolean>(false);

  const handleDateChange = (selectedDate: Date | null) => {
    if (selectedDate && isToday(selectedDate) && !isTodayDateSet) {
      selectedDate = addMinutes(new Date(), FIVE_MINUTES); // Setting the current time + 5min only on the first click on today's date
      setIsTodayDateSet(true);
    } else if (selectedDate && !isToday(selectedDate)) {
      setIsTodayDateSet(false);
    }

    if (selectedDate) {
      const isoDate = selectedDate.toISOString();
      updateTodo({
        expiredDate: isoDate,
      });
    } else {
      updateTodo({
        expiredDate: "",
      });
    }
  };

  const expirationDate = todo.expiredDate ? new Date(todo.expiredDate) : null;

  // Prevent click propagation within the modal (to close modal when user clicks outside it)
  const handleModalClick = (event: MouseEvent<HTMLDivElement>) => {
    event.stopPropagation();
  };

  const preventKeyDownSubmit = (e: KeyboardEvent<HTMLInputElement>) => {
    if (e.key === KEYS.ENTER) {
      e.preventDefault();
    }
  };

  const handleSubmit = (e: FormEvent) => {
    e.preventDefault();
    if (modalTitle.trim()) {
```

```
    handleSave();
  } else {
    setModalValidationMessage(ERROR_MESSAGES.EMPTY_TITLE);
  }
};

const handleSave = () => {
  if (isValid(modalTitle)) {
    const newAddTodo = {
      title: modalTitle.trim(),
      createdDate: todo.createdDate,
      expiredDate: todo.expiredDate,
      completed: false,
    };
    if (editItem) {
      dispatch(editTask({ taskId: editItem._id, taskData: newAddTodo }));
    } else {
      dispatch(addUserTask(newAddTodo));
    }
    switchCompletedFilter();
    onClose();
    resetData();
  } else {
    setModalValidationMessage(ERROR_MESSAGES.INVALID_SYMBOLS);
  }
};

const handleModalClose = () => {
  onClose();
  resetData();
};

// preventing data entry from the keyboard
const handleRawChange = (e: ChangeEvent<HTMLInputElement>) => {
  e.preventDefault();
};

return (
  <div className="modal-overlay" onClick={handleModalClose}>
    <div className="modal" onClick={handleModalClick}>
      <h2 className="modal__title">
        {editItem ? "Edit Task" : "Add New Task"}
      </h2>
      <form onSubmit={handleSubmit}>
        <label className="modal__label">Title</label>
        <input
          className="modal__input"
          type="text"
          value={modalTitle}
          onChange={(e) => setModalTitle(e.target.value)}
          onKeyDown={preventKeyDownSubmit}
          maxLength={MAX_INPUT_LENGTH}
          required
        />
        <label className="modal__label">Creation date</label>
        <input
          className="modal__input"
          type="text"
          value={formatDate(todo.createdDate)}
          readOnly
        />
        <label className="modal__label">Expiration date</label>
        <DatePicker
          className="modal__input"
          showIcon
          selected={expirationDate}
          onChange={handleDateChange}
          timeFormat={TIME_FORMAT}
```

```
            dateFormat={DATE_FORMAT}
            timeIntervals={TIME_INTERVAL}
            placeholderText="Select expiration date"
            showTimeSelect
            minDate={new Date()}
            minTime={getMinDate(expirationDate)}
            onChangeRaw={handleRawChange}
            maxTime={getMaxDate(new Date())}
            required
          />
          <div className="modal__buttons">
            <CustomButton
              variant={ButtonVariants.SECONDARY}
              onClick={handleModalClose}
            >
              Cancel
            </CustomButton>
            <CustomButton
              variant={ButtonVariants.PRIMARY}
              type={ButtonTypes.SUBMIT}
            >
              Save
            </CustomButton>
          </div>
          {modalValidationMessage && (
            <p className="validation-message">{modalValidationMessage}</p>
          )}
        </form>
      </div>
    </div>
  );
};
```

client\src\components\TodoComponents\AddTodoModal\AddTodoModal.scss

```
@import "@/scss/index.scss";

.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: rgba(0, 0, 0, 0.5);
}

.modal {
  background: var(--background-secondary);
  border-radius: 5px;
  padding: 1.2rem;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
  width: 600px;
  height: 400px;

  @include onMobile {
    width: 350px;
  }

  &__title {
    text-align: center;
    margin-bottom: 1rem;
  }

  &__label {
```

```
    display: block;
    margin-bottom: 10px;
    font-weight: bold;
  }

  .react-datepicker__view-calendar-icon input {
    padding: 0.75rem;
  }

  &__input {
    width: 100%;
    padding: 0.75rem;
    margin-bottom: 1.2rem;
    background: var(--input-background);
    color: var(--text-primary);
    border: 1px solid #ccc;
    border-radius: 5px;
  }

  .react-datepicker-wrapper {
    width: 100%;
  }

  .react-datepicker__calendar-icon {
    top: 12px;
    right: 12px;
    fill: var(--text-primary);
    padding: 0;
  }

  &__buttons {
    display: flex;
    gap: 5px;
    justify-content: flex-end;
  }

  &__button {
    padding: 0.625rem 1.25rem;
    border: none;
    border-radius: 5px;
    background-color: var(--button-primary);
    color: $black;
    cursor: pointer;
  }

  &__button.cancel {
    background-color: var(--button-secondary);
  }
}
```

client\src\components\TodoComponents\AddTodoModal\index.ts

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

export default AddTodoModal;
```

client\src\components\TodoComponents\AddTodoButton\AddTodoButton.tsx

```
import { FC } from "react";

import "./AddTodoButton.scss";

interface AddTodoButtonProps {
  onAddTodoButtonClick: () => void;
}
```

```
export const AddTodoButton: FC<AddTodoButtonProps> = ({
  onAddTodoButtonClick,
}) => {
  return (
    <button className="add-todo" onClick={onAddTodoButtonClick}>
      <span className="add-todo__plus">+</span>
    </button>
  );
};
```

client\src\components\TodoComponents\AddTodoButton\AddTodoButton.scss

```
@import "@/scss/index.scss";

.add-todo {
  display: flex;
  align-items: center;
  justify-content: center;
  background-color: darken($green, 10%);
  border: 2px solid var(--border-primary);
  color: $black;
  font-size: 1.5rem;
  height: 3.2rem;
  width: 3.2rem;
  border-radius: 50%;
  box-shadow: 2px 4px 10px lighten($green, 10%);
  cursor: pointer;

  &__plus {
    font-size: 1.5rem;
  }

}
```

client\src\components\TodoComponents\AddTodoButton\index.ts

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

export default AddTodoButton;
```

client\src\components\ThemeSwitcher\ThemeSwitcher.tsx

```
import Sun from "@/assets/images/Sun.svg?react";
import Moon from "@/assets/images/Moon.svg?react";
import { useTheme } from "@/hooks/useTheme";
import { Theme } from "@/types/themeTypes";

import "./ThemeSwitcher.scss";

export const ThemeSwitcher: React.FC = () => {
  const { theme, toggleTheme } = useTheme();

  return (
    <div className="theme-switcher">
      <input
        className="theme-switcher__input"
        type="checkbox"
        id="theme-switcher__toggle"
        checked={theme === Theme.DARK}
        onChange={toggleTheme}
      />
      <label className="theme-switcher__label" htmlFor="theme-switcher__toggle">
        <Sun />
        <Moon />
```

```
    </label>
   </div>
  );
};
```

client\src\components\ThemeSwitcher\ThemeSwitcher.scss

```
@import "@/scss/index.scss";

.theme-switcher {
  position: absolute;
  top: 0;
  right: 20px;

  @include onMobile {
    right: 10px;
    top: -10px;
  }

  &__label {
    width: 65px;
    height: 30px;
    position: relative;
    display: block;
    background: $switcher-bg-light;
    border-radius: 200px;
    box-shadow: inset 0px 5px 15px $switcher-shadow-light,
      inset 0px -5px 15px $switcher-shadow-white;
    cursor: pointer;
    transition: 0.3s;

    &:after {
      content: "";
      width: 25px;
      height: 25px;
      position: absolute;
      top: 3px;
      left: 3px;
      background: linear-gradient(
        180deg,
        $switcher-toggle-gradient-light-start,
        $switcher-toggle-gradient-light-end
      );
      border-radius: 180px;
      box-shadow: 0px 5px 10px $switcher-toggle-shadow;
      transition: 0.3s;
    }

    &:active:after {
      width: 30px;
    }

    svg {
      position: absolute;
      width: 20px;
      top: 5px;
      z-index: 100;

      &.sun {
        left: 5px;
        fill: $icon-sun-color;
        transition: 0.3s;
      }

      &.moon {
        left: 40px;
        fill: $icon-moon-color;
```

```
      transition: 0.3s;
    }
   }
  }

  &__input {
   width: 0;
   height: 0;
   visibility: hidden;

   &:checked + .theme-switcher__label {
    background: $switcher-bg-dark;

    &:after {
     left: 62px;
     transform: translateX(-100%);
     background: linear-gradient(
      180deg,
      $switcher-toggle-gradient-dark-start,
      $switcher-toggle-gradient-dark-end
     );
    }

    svg.sun {
     fill: $icon-moon-color;
    }

    svg.moon {
     fill: $icon-sun-color;
    }
   }
  }
}
```

client\src\components\ThemeSwitcher\index.ts

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

export default ThemeSwitcher;
```

client\src\components\SearchInput\SearchInput.tsx

```
import { FC, useState, useEffect, useRef, useCallback, ChangeEvent} from "react";

import { debounce } from "@/helpers/debounce";
import { SEARCH_DELAY } from "@/consts/debounceDelays";
import { VALID_INPUT_REGEX } from "@/consts/searchRegex";
import ClearIcon from "@/assets/images/clearIcon.svg?react";

import "./SearchInput.scss";
import { useAppSelector } from "@/hooks/useAppSelector";
import { useAppDispatch } from "@/hooks/useAppDispatch";
import { selectSearchQuery } from "@/redux/selectors/tasks.selectors";
import { setSearchQuery } from "@/redux/reducers/tasks.reducer";

export const SearchInput: FC = () => {
  const dispatch = useAppDispatch();
  const inputRef = useRef<HTMLInputElement | null>(null);
  const searchQuery = useAppSelector(selectSearchQuery);
  const [internalValue, setInternalValue] = useState(searchQuery);

  const onChange = (value: string) => {
    dispatch(setSearchQuery(value));
  };

  useEffect(() => {
    setInternalValue(searchQuery);
```

```
  }, [searchQuery]);

  const updateSearchValue = useCallback(
    debounce((str) => {
      onChange(str);
    }, SEARCH_DELAY),
    []
  );

  const handleInputChange = (e: ChangeEvent<HTMLInputElement>) => {
    const value = e.target.value;
    const filteredValue = value.replace(VALID_INPUT_REGEX, "");

    setInternalValue(filteredValue);
    updateSearchValue(filteredValue);
};

  const onClear = () => {
    setInternalValue("");
    onChange("");
    inputRef.current?.focus();
  };

  return (
    <div className="search">
      <input
        ref={inputRef}
        className="search__input"
        type="text"
        value={internalValue}
        onChange={handleInputChange}
        placeholder="Search..."
      />
      {internalValue && (
        <ClearIcon className="search__clear-icon" onClick={onClear} />
      )}
    </div>
  );
};
```

client\src\components\SearchInput\SearchInput.scss

```
@import "@/scss/index.scss";

.search {
  position: relative;

  &__input {
    width: 820px;
    max-height: 3rem;
    margin-top: 10px;
    padding: 1rem;
    border: 2px solid var(--border-primary);
    color: var(--text-primary);
    border-radius: 5px;
    background-color: var(--input-background);

    font-size: 1.25rem;

    @include onBig-tablet {
      width: 560px;
    }

    @include onMobile {
      width: 320px;
    }

  }
```

```
  &__clear-icon {
    position: absolute;
    right: 15px;
    top: 25px;
    opacity: 0.3;
    width: 18px;
    height: 18px;
    fill: var(--text-primary);
    cursor: pointer;

    &:hover {
      opacity: 0.8;
    }

  }

}
```

client\src\components\SearchInput\index.ts

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

export default SearchInput;
```

client\src\components\Profile\Profile.tsx

```
import { useState, FC } from "react";
import ChangePasswordBlock from "./ChangePasswordBlock";
import ProfileBlock from "./ProfileBlock";
import profileIcon from "@/assets/images/profile.svg";
import lockIcon from "@/assets/images/lock.svg";

import "./Profile.scss";

export const Profile: FC = () => {
  const [activeTab, setActiveTab] = useState("Profile");

  return (
    <div className="profile">
      <ul className="profile__navigation">
        <li
          className={`profile__navigation-item ${
            activeTab === "Profile" ? "active" : ""
          }`}
          onClick={() => setActiveTab("Profile")}
        >
          <img className="profile__icon" src={profileIcon} alt="Profile icon" />
        </li>
        <li
          className={`profile__navigation-item ${
            activeTab === "Security" ? "active" : ""
          }`}
          onClick={() => setActiveTab("Security")}
        >
          <img className="profile__icon" src={lockIcon} alt="Profile icon" />
        </li>
      </ul>
      <div className="profile__tabs">
        <h1 className="profile__title">{activeTab}</h1>
        <div className="profile__info-block">
          {activeTab === "Profile" && <ProfileBlock />}
          {activeTab === "Security" && <ChangePasswordBlock />}
        </div>
      </div>
    </div>
```

```
  );
};
```

client\src\components\Profile\Profile.scss

```
.profile {
  display: flex;
  gap: 40px;

  &__icon {
    width: 30px;
    height: 30px;
  }

  &__navigation {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 70px;
  }

  &__navigation-item {
    display: flex;
    align-items: center;
    justify-content: center;
    width: 100px;
    height: 100px;
    background: var(--background-secondary);
    cursor: pointer;

    &.active {
       width: 110px;
       height: 110px;
       background: var(--background-active);
    }

  }

  &__title {
    font-size: 84px;
    margin-bottom: -35px;
    margin-left: 15px;
    color: var(--text-primary);
  }

  &__info-block {
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 45px;
    width: 600px;
    height: 300px;
    background: var(--background-secondary);
  }

  &__list {
    display: flex;
    flex-direction: column;
    gap: 20px;
    margin-top: 20px;
  }

  &__key {
    font-size: 24px;
    margin-bottom: 5px;
  }
```

```
  &__value {
    padding-bottom: 3px;
    width: 200px;
    border-bottom: 1px solid white;
  }

  &__image {
    width: 150px;
    height: 150px;
    border-radius: 50%;
    background: white;
  }

}
```

client\src\components\Profile\index.ts

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

export default Profile;
```

client\src\components\Profile\ProfileBlock\ProfileBlock.tsx

```
import { FC } from "react";
import { useAppSelector } from "@/hooks/useAppSelector";

import "./ProfileBlock.scss";

export const ProfileBlock: FC = () => {
  const user = useAppSelector((state) => state.auth.user);

  return (
    <div className="profile__block">
      <ul className="profile__list">
        <li className="profile__list-item">
          <div>
            <p className="profile__key">Username</p>
            <p className="profile__value">{user?.username}</p>
          </div>
        </li>
        <li className="profile__list-item">
          <div>
            <p className="profile__key">Email</p>
            <p className="profile__value">{user?.email}</p>
          </div>
        </li>
      </ul>
      <div className="profile__image"></div>
    </div>
  );
};
```

client\src\components\Profile\ProfileBlock\ProfileBlock.scss

```
.profile__block {
  display: flex;
  gap: 50px;
}
```

client\src\components\Profile\ProfileBlock\index.ts

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

export default ProfileBlock;
```

client\src\components\Profile\ChangePasswordBlock\ChangePasswordBlock.tsx

```
import { useState } from "react";
import { useForm } from "react-hook-form";

import { yupResolver } from "@hookform/resolvers/yup";
import { changeUserPassword } from "@/redux/thunks/auth.thunks";
import { useAppDispatch } from "@/hooks/useAppDispatch";
import CustomButton from "@/components/UI/CustomButton";
import { ButtonTypes } from "@/types/buttonTypes";
import { changePasswordSchema } from "@/schemas/authSchemas";

import "./ChangePasswordBlock.scss";
import { CHANGE_PASSWORD_FIELDS } from "@/consts/changePasswordForm";

interface ChangePasswordFormData {
  currentPassword: string;
  newPassword: string;
  confirmPassword: string;
}

export const ChangePasswordBlock = () => {
  const [message, setMessage] = useState("");
  const dispatch = useAppDispatch();
  const {
    register,
    handleSubmit,
    formState: { errors },
    reset,
  } = useForm<ChangePasswordFormData>({
    resolver: yupResolver(changePasswordSchema),
  });

  const onSubmit = async (data: ChangePasswordFormData) => {
    const { currentPassword, newPassword } = data;
    try {
      await dispatch(changeUserPassword({ currentPassword, newPassword }));
      setMessage("Password successfully changed.");
      reset();
    } catch (error) {
      setMessage("Failed to change password. Please try again.");
    }
  };

  return (
    <form className="change-password" onSubmit={handleSubmit(onSubmit)}>
      <div className="change-password__input-container">
        <input
          className="change-password__input"
          type={CHANGE_PASSWORD_FIELDS.currentPassword.type}
          {...register(CHANGE_PASSWORD_FIELDS.currentPassword.name)}
          placeholder={CHANGE_PASSWORD_FIELDS.currentPassword.placeholder}
        />
        <p>{errors.currentPassword?.message}</p>
      </div>
      <div className="change-password__input-container">
        <input
          className="change-password__input"
          type={CHANGE_PASSWORD_FIELDS.newPassword.type}
          {...register(CHANGE_PASSWORD_FIELDS.newPassword.name)}
          placeholder={CHANGE_PASSWORD_FIELDS.newPassword.placeholder}
        />
        <p>{errors.newPassword?.message}</p>
      </div>
      <div className="change-password__input-container">
        <input
          className="change-password__input"
          type={CHANGE_PASSWORD_FIELDS.confirmPassword.type}
          {...register(CHANGE_PASSWORD_FIELDS.confirmPassword.name)}
          placeholder={CHANGE_PASSWORD_FIELDS.confirmPassword.placeholder}
```

```
        />
        <p>{errors.confirmPassword?.message}</p>
      </div>
      {message && <p>{message}</p>}
      <CustomButton className="myButton" type={ButtonTypes.SUBMIT}>
        Change password
      </CustomButton>
    </form>
  );
};
```

client\src\components\Profile\ChangePasswordBlock\ChangePasswordBlock.scss

```
.change-password {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
  max-width: 300px;
  width: 100%;
  margin: 15px 0;

  &__input-container {
    width: 100%;
  }

  &__input {
    width: 100%;
    background: var(--input-background);
    border: 1px solid #ccc;
    outline: none;
    padding: 10px;
    border-radius: 4px;
    color: var(--text-primary);
    font-weight: 500;
    font-size: 1em;
  }

}
```

client\src\components\Profile\ChangePasswordBlock\index.ts

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

export default ChangePasswordBlock;
```

client\src\components\PrivateRoute\PrivateRoute.tsx

```
import { FC, PropsWithChildren } from "react";

import { Navigate } from "react-router-dom";
import { useAuth } from "@/hooks/useAuth";
import { ROUTE_PATHS } from "@/consts/routePaths";

export const PrivateRoute: FC<PropsWithChildren> = ({ children }) => {
  const { isAuth } = useAuth();

  if (!isAuth) {
    return <Navigate to={ROUTE_PATHS.SIGN_IN} replace />;
  }

  return <>{children}</>;
};
```

client\src\components\PrivateRoute\index.ts

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

export default PrivateRoute;
```

client\src\components\Header\Header.tsx

```
import { FC } from "react";
import { Link, useNavigate } from "react-router-dom";

import { useAppDispatch } from "@/hooks/useAppDispatch";
import CustomButton from "../UI/CustomButton";
import { useAuth } from "@/hooks/useAuth";
import { ROUTE_PATHS } from "@/consts/routePaths";
import { useModal } from "@/hooks/useModal";
import ConfirmationModal from "../ConfirmationModal";
import { ButtonVariants } from "@/types/buttonTypes";
import { removeRefreshToken, removeToken } from "@/helpers/tokenHelpers";
import logo from "@/assets/images/logo.svg";
import profileIcon from "@/assets/images/profile.svg";

import "./Header.scss";
import { logout } from "@/redux/reducers/auth.reducer";
import { resetTodoState } from "@/redux/reducers/tasks.reducer";

export const Header: FC = () => {
  const dispatch = useAppDispatch();
  const navigate = useNavigate();
  const { isAuth } = useAuth();
  const { isModalOpen, openModal, closeModal } = useModal();

  const handleLogout = () => {
    removeToken();
    removeRefreshToken();
    dispatch(logout());
    dispatch(resetTodoState());
    closeModal();
    navigate(ROUTE_PATHS.SIGN_IN);
  };

  return (
    <>
      <header className="header">
        <Link className="header__title" to="/">
          <h1 className="header__text">Daily Planner</h1>
        </Link>
        <div className="header__nav-block">
          {isAuth && (
            <Link className="header__title" to="/profile">
              <img
                className="header__img"
                src={profileIcon}
                alt="Profile icon link"
              />
            </Link>
          )}
          {isAuth && (
            <CustomButton
              variant={ButtonVariants.SECONDARY}
              className="header__logout"
              onClick={openModal}
            >
              Logout
            </CustomButton>
          )}
        </div>
      </header>
      {isModalOpen && (
```

```
      <ConfirmationModal
        message={`Are you sure that you want to logout?`}
        onConfirm={handleLogout}
        onClose={closeModal}
      />
    )}
  </>
 );
};
```

client\src\components\Header\Header.scss

```
@import "@/scss/index.scss";

.header {
 display: flex;
 align-items: center;
 justify-content: center;
 padding: 20px 0;
 color: $white;

 @include onMobile {
  padding: 40px 0;
 }

 &__title {
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 10px;

  @include onMobile {
   flex-direction: column;
  }
 }

 &__text {
  font-size: 40px;
  font-weight: 400;
  color: $white;
 }

 &__nav-block {
  position: absolute;
  display: flex;
  align-items: center;
  gap: 20px;
  top: 20px;
  right: 100px;
 }

 &__img {
  width: 32px;
  height: 32px;
 }

}
```

client\src\components\Header\index.ts

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

export default Header;
```

client\src\components\FilterButtons\FilterButtons.tsx

```
import { FC } from "react";
```

```
import ConfirmationModal from "../ConfirmationModal";
import CustomButton from "../UI/CustomButton";
import { useAppDispatch } from "@/hooks/useAppDispatch";
import { useAppSelector } from "@/hooks/useAppSelector";
import { FILTER_TYPES } from "@/consts/filterOptions";
import {
  deleteCompletedTasks,
} from "@/redux/thunks/tasks.thunks";
import { selectCompletedCount, selectFilter } from "@/redux/selectors/tasks.selectors";
import { useModal } from "@/hooks/useModal";
import { CONFIRMATION_MESSAGES } from "@/consts/Messages";
import { ButtonVariants } from "@/types/buttonTypes";

import "./FilterButtons.scss";
import { setFilter } from "@/redux/reducers/tasks.reducer";

export const FilterButtons: FC = () => {
  const dispatch = useAppDispatch();
  const currentFilter = useAppSelector(selectFilter)

  const handleFilterChange = (filterValue: string) => {
    dispatch(setFilter(filterValue));
  };

  const { isModalOpen, openModal, closeModal } = useModal();
  const completedTasksCount = useAppSelector(selectCompletedCount);

  const handleConfirmDelete = () => {
    dispatch(deleteCompletedTasks());
    closeModal();
  };

  return (
    <>
      <div className="filter-container">
        <div className="filter-buttons">
          {FILTER_TYPES.map(({ key, label }) => (
            <CustomButton
              key={key}
              variant={ButtonVariants.PRIMARY}
              onClick={() => handleFilterChange(key)}
              isDisabled={currentFilter === key}
            >
              {label}
            </CustomButton>
          ))}
        </div>
        <div className="clear-buttons">
          <CustomButton
            onClick={openModal}
            variant={ButtonVariants.SECONDARY}
            isDisabled={!completedTasksCount}
          >
            Clear completed
          </CustomButton>
        </div>
      </div>
      {isModalOpen && (
        <ConfirmationModal
          message={CONFIRMATION_MESSAGES.DELETE_ALL_COMPLETED}
          onConfirm={handleConfirmDelete}
          onClose={closeModal}
        />
      )}
    </>
  );
};
```

client\src\components\FilterButtons\FilterButtons.scss

```
@import "@/scss/index.scss";

.filter-container {
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  justify-content: space-between;
  margin-top: 20px;

  @include onMobile {
    justify-content: center;
    gap: 20px;
  }
}

.filter-buttons {
  display: flex;
  gap: 10px;
}
```

client\src\components\FilterButtons\index.ts

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

export default FilterButtons;
```

client\src\components\Container\Container.tsx

```
import { FC, PropsWithChildren } from "react";

import "./Container.scss";

export const Container: FC<PropsWithChildren> = ({ children }) => {
  return <div className="container">{children}</div>;
};
```

client\src\components\Container\Container.scss

```
.container {
  display: flex;
  flex-direction: column;
  margin: 0 auto;
  padding: 0 20px;
  max-width: 1480px;
  width: 100%;
  min-height: 100vh;

}
```

client\src\components\Container\index.ts

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

export default Container;
```

client\src\components\ConfirmationModal\ConfirmationModal.tsx

```
import { FC } from "react";

import CustomButton from "../UI/CustomButton";
import { ButtonVariants } from "@/types/buttonTypes";

import "./ConfirmationModal.scss";
```

```
interface ConfirmationModalProps {
  message: string;
  onConfirm: () => void;
  onClose: () => void;
}

export const ConfirmationModal: FC<ConfirmationModalProps> = ({
  message,
  onConfirm,
  onClose,
}) => {
  return (
    <div className="confirm-modal-overlay">
      <div className="confirm-modal">
        <p className="confirm-modal__message">{message}</p>
        <div className="confirm-modal__buttons">
          <CustomButton variant={ButtonVariants.PRIMARY} onClick={onConfirm}>
            Confirm
          </CustomButton>
          <CustomButton variant={ButtonVariants.SECONDARY} onClick={onClose}>
            Cancel
          </CustomButton>
        </div>
      </div>
    </div>
  );
};
```

client\src\components\ConfirmationModal\ConfirmationModal.scss

```
@import "@/scss/index.scss";

.confirm-modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: rgba(0, 0, 0, 0.5);
  z-index: 1;
}

.confirm-modal {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  text-align: center;
  background: var(--background-secondary);
  border-radius: 5px;
  padding: 1.2rem;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
  width: 400px;
  height: 200px;

  @include onMobile {
    width: 350px;
  }


  &__message {
    font-weight: bold;
  }

  &__buttons {
```

```
    margin-top: 20px;
    display: flex;
    gap: 15px;
  }
}
```

client\src\components\ConfirmationModal\index.ts

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

export default ConfirmationModal;
```

client\src\components\Auth\Auth.tsx

```
import { FC, useState } from "react";
import LoginForm from "@/components/Auth/LoginForm";
import RegistrationForm from "@/components/Auth/RegistrationForm";

import "./Auth.scss";

const TABS = {
  LOGIN: "login",
  REGISTER: "register",
};

export const Auth: FC = () => {
  const [activeTab, setActiveTab] = useState(TABS.LOGIN);

  return (
    <div className="auth">
      <div className="auth__tab-buttons">
        <button
          className={`auth__button ${
            activeTab === TABS.LOGIN ? "active" : ""
          }`}
          onClick={() => setActiveTab(TABS.LOGIN)}
        >
          Sign in
        </button>
        <button
          className={`auth__button ${
            activeTab === TABS.REGISTER ? "active" : ""
          }`}
          onClick={() => setActiveTab(TABS.REGISTER)}
        >
          Sign up
        </button>
      </div>
      {activeTab === TABS.LOGIN ? <LoginForm /> : <RegistrationForm />}
    </div>
  );
};
```

client\src\components\Auth\Auth.scss

```
.auth {
  background: var(--background-secondary);

  &__tab-buttons {
    display: flex;
    justify-content: center;
  }

  &__button {
    background: none;
    border: none;
    background: var(--button-secondary);
    padding: 15px 60px;
```

```
    margin: 15px 0;
    font-size: 18px;
    font-weight: 500;
    cursor: pointer;
    transition: 0.3s linear;

    &:hover {
      background: var(--button-secondary-hover);
    }

    &.active {
      background: var(--button-primary);
    }

  }

}
```

client\src\components\Auth\RegistrationForm\RegistrationForm.tsx

```
import { useForm, SubmitHandler } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import { useNavigate } from "react-router-dom";

import { useAppDispatch } from "@/hooks/useAppDispatch";
import { ButtonTypes } from "@/types/buttonTypes";
import CustomButton from "@/components/UI/CustomButton";
import { useAppSelector } from "@/hooks/useAppSelector";
import {
  selectAuthError,
  selectAuthStatus,
} from "@/redux/selectors/auth.selectors";
import { ROUTE_PATHS } from "@/consts/routePaths";
import { AUTH_INPUT_FIELDS } from "@/consts/authInputs";
import { registerUser } from "@/redux/thunks/auth.thunks";
import { signUpSchema } from "@/schemas/authSchemas";
import { IRegisterForm } from "@/types/authTypes";

import "./RegistrationForm.scss";
import { LoadingStatus } from "@/types/loading-status";

export const RegistrationForm = () => {
  const dispatch = useAppDispatch();
  const authError = useAppSelector(selectAuthError);
  const status = useAppSelector(selectAuthStatus);
  const navigate = useNavigate();

  const isLoading = status === LoadingStatus.PENDING;

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<IRegisterForm>({
    resolver: yupResolver(signUpSchema),
  });

  const onSubmit: SubmitHandler<IRegisterForm> = async (data) => {
    await dispatch(registerUser(data));
    navigate(ROUTE_PATHS.HOME);
  };

  return (
    <div className="register">
      <div className="register__container">
        <h2 className="register__title">Create your account</h2>
        <form onSubmit={handleSubmit(onSubmit)}>
          <div className="register__input-container">
```

```
          <input
            className="register__input"
            {...register(AUTH_INPUT_FIELDS.username.name)}
            placeholder={AUTH_INPUT_FIELDS.username.placeholder}
            type={AUTH_INPUT_FIELDS.username.type}
          />
          <p className="register__error">{errors.username?.message}</p>
        </div>
        <div className="register__input-container">
          <input
            className="register__input"
            {...register(AUTH_INPUT_FIELDS.email.name)}
            placeholder={AUTH_INPUT_FIELDS.email.placeholder}
            type={AUTH_INPUT_FIELDS.email.type}
          />
          <p className="register__error">{errors.email?.message}</p>
        </div>
        <div className="register__input-container">
          <input
            className="register__input"
            {...register(AUTH_INPUT_FIELDS.password.name)}
            placeholder={AUTH_INPUT_FIELDS.password.placeholder}
            type={AUTH_INPUT_FIELDS.password.type}
          />
          <p className="register__error">{errors.password?.message}</p>
        </div>
        <div className="register__input-container">
          <input
            className="register__input"
            {...register(AUTH_INPUT_FIELDS.confirmPassword.name)}
            placeholder={AUTH_INPUT_FIELDS.confirmPassword.placeholder}
            type={AUTH_INPUT_FIELDS.confirmPassword.type}
          />
          <p className="register__error">{errors.confirmPassword?.message}</p>
        </div>
        <CustomButton type={ButtonTypes.SUBMIT} isDisabled={isLoading}>
          {isLoading ? "Loading..." : "Sign Up"}
        </CustomButton>
        {authError && <p className="register__error">{authError}</p>}
      </form>
    </div>
  </div>
 );
};
```

client\src\components\Auth\RegistrationForm\RegistrationForm.scss

```
@import "@/scss/colors";

.register {
 flex: 1;
 display: flex;
 align-items: center;
 justify-content: center;

 &__title {
   font-size: 28px;
   font-weight: 800;
   color: $green;
 }

 &__container {
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: center;
   width: 400px;
   height: auto;
```

```
    padding: 40px;
    border-radius: 4px;
    background: var(--background-secondary);
    gap: 20px;

    form {
      display: flex;
      flex-direction: column;
      gap: 20px;
      max-width: 300px;
      width: 100%;
    }
  }

  &__input-container {
    position: relative;
    width: 100%;
  }

  &__input {
    position: relative;
    width: 100%;
    background: var(--input-background);
    border: 1px solid #ccc;
    outline: none;
    padding: 15px;
    border-radius: 4px;
    color: var(--text-primary);
    font-weight: 500;
    font-size: 1em;
  }

  &__error {
    color: $error;
    font-size: 14px;
    font-weight: 700;
  }
}
```

client\src\components\Auth\RegistrationForm\index.ts

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

export default RegistrationForm;
```

client\src\components\Auth\LoginForm\LoginForm.tsx

```
import { useForm, SubmitHandler } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import { useNavigate } from "react-router-dom";

import { useAppDispatch } from "@/hooks/useAppDispatch";
import { ButtonTypes } from "@/types/buttonTypes";
import { signInSchema } from "@/schemas/authSchemas";
import { ILoginForm } from "@/types/authTypes";
import { login } from "@/redux/thunks/auth.thunks";
import CustomButton from "@/components/UI/CustomButton";
import { useAppSelector } from "@/hooks/useAppSelector";
import {
  selectAuthError,
  selectAuthStatus,
} from "@/redux/selectors/auth.selectors";
import { ROUTE_PATHS } from "@/consts/routePaths";
import { AUTH_INPUT_FIELDS } from "@/consts/authInputs";

import "./Login.scss";
import { LoadingStatus } from "@/types/loading-status";
```

```
export const LoginForm = () => {
  const navigate = useNavigate();
  const dispatch = useAppDispatch();
  const authError = useAppSelector(selectAuthError);
  const status = useAppSelector(selectAuthStatus);
  const isLoading = status === LoadingStatus.PENDING;

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<ILoginForm>({
    resolver: yupResolver(signInSchema),
  });

  const onSubmit: SubmitHandler<ILoginForm> = async (data) => {
    await dispatch(login(data));
    navigate(ROUTE_PATHS.HOME);
  };

  return (
    <div className="login">
      <div className="login__container">
        <h2 className="login__title">Log in to your account</h2>
        <form onSubmit={handleSubmit(onSubmit)}>
          <div className="login__input-container">
            <input
              className="login__input"
              {...register(AUTH_INPUT_FIELDS.email.name)}
              placeholder={AUTH_INPUT_FIELDS.email.placeholder}
            />
            <p className="login__error">{errors.email?.message}</p>
          </div>
          <div className="login__input-container">
            <input
              className="login__input"
              type={AUTH_INPUT_FIELDS.password.type}
              {...register(AUTH_INPUT_FIELDS.password.name)}
              placeholder={AUTH_INPUT_FIELDS.password.placeholder}
            />
            <p className="login__error">{errors.password?.message}</p>
          </div>
          <CustomButton type={ButtonTypes.SUBMIT} isDisabled={isLoading}>
            {isLoading ? "Loading..." : "Login"}
          </CustomButton>
          {authError && <p className="login__error">{authError}</p>}
        </form>
      </div>
    </div>
  );
};
```

client\src\components\Auth\LoginForm\Login.scss

```
@import "@/scss/colors";

.login {
  flex: 1;
  display: flex;
  align-items: center;
  justify-content: center;

  &__title {
    font-size: 28px;
    font-weight: 800;
    color: $green;
  }
```

```
 &__container {
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: center;
   width: 400px;
   height: auto;
   padding: 40px;
   border-radius: 4px;
   background: var(--background-secondary);
   gap: 20px;

   form {
     display: flex;
     flex-direction: column;
     gap: 20px;
     max-width: 300px;
     width: 100%;
   }
 }

 &__input-container {
   position: relative;
   width: 100%;
 }

 &__input {
   position: relative;
   width: 100%;
   background: var(--input-background);
   border: 1px solid #ccc;
   outline: none;
   padding: 15px;
   border-radius: 4px;
   color: var(--text-primary);
   font-weight: 500;
   font-size: 1em;
 }

 &__error {
   color: $error;
   font-size: 14px;
   font-weight: 700;
 }

}
```

client\src\components\Auth\LoginForm\index.ts

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

export default LoginForm;
```

client\src\context\ThemeContext.tsx

```
import { FC, useEffect, createContext, useState } from "react";
import { Theme } from "@/types/themeTypes";

interface ThemeContextProps {
  theme: Theme;
  toggleTheme: () => void;
}

interface ThemeProviderProps {
  children: React.ReactNode;
}

export const ThemeContext = createContext<ThemeContextProps | undefined>(
```

```
  undefined
);

export const ThemeProvider: FC<ThemeProviderProps> = ({ children }) => {
  const prefersDarkScheme = window.matchMedia(
    "(prefers-color-scheme: dark)"
  ).matches;

  const initialTheme =
    (localStorage.getItem(Theme.KEY) as Theme) ||
    (prefersDarkScheme ? Theme.DARK : Theme.LIGHT);

  const [theme, setTheme] = useState<Theme>(initialTheme);

  const toggleTheme = () => {
    setTheme((currentTheme) =>
      currentTheme === Theme.LIGHT ? Theme.DARK : Theme.LIGHT
    );
  };

  useEffect(() => {
    if (theme === Theme.DARK) {
      document.body.classList.add(Theme.DARK);
    } else {
      document.body.classList.remove(Theme.DARK);
    }

    localStorage.setItem(Theme.KEY, theme);
  }, [theme]);

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

client\src\core\routes.ts

```
import { FC } from 'react';

import { HomePage } from '@/pages/HomePage';
import { LoginPage } from '@/pages/LoginPage';
import { ROUTE_PATHS } from '@/consts/routePaths';
import { ProfilePage } from '@/pages/ProfilePage';

interface RouteItem {
    path: string;
    Element: FC;
    private?: boolean;
}

export const routes: RouteItem[] = [
    {
        path: ROUTE_PATHS.HOME,
        Element: HomePage,
        private: true,
    },
    {
        path: ROUTE_PATHS.SIGN_IN,
        Element: LoginPage,
    },
    {
        path: ROUTE_PATHS.PROFILE,
        Element: ProfilePage,
        private: true,
    },
];
```

client\src\core\api.ts

```
import axios, { AxiosInstance } from 'axios';
import {
  getToken,
  setToken,
  getRefreshToken,
  setRefreshToken,
  removeToken,
  removeRefreshToken,
} from '@/helpers/tokenHelpers';
import { ROUTE_PATHS } from '@/consts/routePaths';

let onUnauthorized: (() => void) | null = null;

export const setUnauthorizedHandler = (handler: () => void) => {
  onUnauthorized = handler;
};

export const getAxios = (): AxiosInstance => {
  const instance = axios.create();

  instance.interceptors.request.use(
    (config) => {
      const token = getToken();
      if (token) {
        config.headers.Authorization = `Bearer ${token}`;
      }
      return config;
    },
    (error) => Promise.reject(error)
  );

  const refreshAccessToken = async (): Promise<string | null> => {
    try {
      const refreshToken = getRefreshToken();
      const { data } = await instance.post('users/refresh', { refreshToken });
      const { newAccessToken, newRefreshToken } = data;
      setToken(newAccessToken);
      setRefreshToken(newRefreshToken);
      return newAccessToken;
    } catch (error) {
      console.error('Failed to refresh token:', error);
      removeToken();
      removeRefreshToken();
      window.location.href = ROUTE_PATHS.SIGN_IN;
      return null;
    }
  };

  instance.interceptors.response.use(
    (response) => response,
    async (error) => {
      const originalRequest = error.config;
      if (
        error.response &&
        error.response.status === 401 &&
        !originalRequest._retry
      ) {
        originalRequest._retry = true;
        const newAccessToken = await refreshAccessToken();
        if (newAccessToken) {
          originalRequest.headers['Authorization'] = `Bearer ${newAccessToken}`;
          return instance(originalRequest);
        } else {
          if (onUnauthorized) {
            onUnauthorized();
```

```
                }
                return Promise.reject(error);
            }
        }
        return Promise.reject(error);
    }
  );

  return instance;
};
```

client\src\core\api\todo-list\todo-list-api.ts

```
import { AuthApi } from "./auth/auth.api";
import { TasksApi } from "./tasks/tasks.api";

class TodoListApi {
  public auth: AuthApi;
  public tasks: TasksApi;

  constructor() {
    const apiURL = 'http://localhost:4000/api';
    this.auth = new AuthApi(apiURL);
    this.tasks = new TasksApi(apiURL);
  }
}

const todoListApi = new TodoListApi();
export default todoListApi;
```

client\src\core\api\todo-list\tasks\dto\task.dto.ts

```
import { actionTypes } from "../../../../../types/actionTypes";

export interface TodoType {
  title: string;
  createdDate: string;
  expiredDate: string;
}

export interface ITodoItem {
  _id: string;
  title: string;
  createdDate: string;
  expiredDate: string;
  completed: boolean;
}

export interface IUserTasksResponse {
  tasks: ITodoItem[];
  totals: {
    all: number;
    completed: number;
    active: number;
  };
}

export interface IDeleteCoompletedAction {
  type: actionTypes.DELETE_COMPLETED;
}

export interface IFetchTasksRequestAction {
  type: actionTypes.FETCH_TASKS_REQUEST;
}

export interface IFetchTasksSuccessAction {
  type: actionTypes.FETCH_TASKS_SUCCESS;
  payload: {
```

```
      tasks: ITodoItem[];
      totals: {
        all: number;
        completed: number;
        active: number;
      };
    };
}

export interface IFetchTasksFailureAction {
  type: actionTypes.FETCH_TASKS_FAILURE;
  payload: string;
}

export interface IAddTaskRequestAction {
  type: actionTypes.ADD_TASK_REQUEST;
}

export interface IAddTaskSuccessAction {
  type: actionTypes.ADD_TASK_SUCCESS;
  payload: ITodoItem;
}

export interface IAddTaskFailureAction {
  type: actionTypes.ADD_TASK_FAILURE;
  payload: string;
}

export interface IEditTaskRequestAction {
  type: actionTypes.EDIT_TASK_REQUEST;
}

export interface IEditTaskSuccessAction {
  type: actionTypes.EDIT_TASK_SUCCESS;
  payload: ITodoItem;
}

export interface IEditTaskFailureAction {
  type: actionTypes.EDIT_TASK_FAILURE;
  payload: string;
}

export interface IDeleteTaskRequestAction {
  type: actionTypes.DELETE_TASK_REQUEST;
}

export interface IDeleteTaskSuccessAction {
  type: actionTypes.DELETE_TASK_SUCCESS;
  payload: string;
}

export interface IDeleteTaskFailureAction {
  type: actionTypes.DELETE_TASK_FAILURE;
  payload: string;
}

export interface IDeleteCompletedTasksRequestAction {
  type: actionTypes.DELETE_COMPLETED_TASKS_REQUEST;
}

export interface IDeleteCompletedTasksSuccessAction {
  type: actionTypes.DELETE_COMPLETED_TASKS_SUCCESS;
}

export interface IDeleteCompletedTasksFailureAction {
  type: actionTypes.DELETE_COMPLETED_TASKS_FAILURE;
  payload: string;
}
```

```
export interface ISetSearchQueryAction {
   type: actionTypes.SET_SEARCH_QUERY;
   payload: string;
}

export interface ISetFilterAction {
   type: actionTypes.SET_FILTER;
   payload: string;
}

export interface IResetTodoStateAction {
   type: actionTypes.RESET_TODO_STATE;
}

export type ITodoListAction =
   IDeleteCoompletedAction |
   IFetchTasksRequestAction |
   IFetchTasksSuccessAction |
   IFetchTasksFailureAction |
   IAddTaskRequestAction |
   IAddTaskSuccessAction |
   IAddTaskFailureAction |
   IEditTaskRequestAction |
   IEditTaskSuccessAction |
   IEditTaskFailureAction |
   IDeleteTaskRequestAction |
   IDeleteTaskSuccessAction |
   IDeleteTaskFailureAction |
   IDeleteCompletedTasksRequestAction |
   IDeleteCompletedTasksSuccessAction |
   IDeleteCompletedTasksFailureAction |
   ISetSearchQueryAction |
   ISetFilterAction |
   IResetTodoStateAction;
client\src\core\api\todo-list\tasks\tasks.api.ts

import { AxiosInstance } from 'axios';
import { getAxios } from '@/core/api';
import { ITodoItem, IUserTasksResponse } from '@/core/api/todo-list/tasks/dto/task.dto';

export class TasksApi {
   private axios: AxiosInstance;
   private apiURL: string;

   constructor(apiURL: string) {
      this.axios = getAxios();
      this.apiURL = apiURL;
   }

   public async fetchTodos(search: string, status: string): Promise<IUserTasksResponse> {
      const { data } = await this.axios.get(`${this.apiURL}/tasks?status=${status}&search=${search}`);
      return data;
   }

   public async createTodo(taskData: { title: string; createdDate: string; expiredDate: string }): Promise<ITodoItem> {
      const { data } = await this.axios.post(`${this.apiURL}/tasks`, taskData);
      return data;
   }

   public async updateTodo(
      taskId: string,
      updates: Partial<{ title: string; createdDate: string; expiredDate: string; completed: boolean }>
   ): Promise<ITodoItem> {
      const { data } = await this.axios.put(`${this.apiURL}/tasks/${taskId}`, updates);
      return data;
   }
```

```
    public async deleteTodo(taskId: string): Promise<void> {
      await this.axios.delete(`${this.apiURL}/tasks/${taskId}`);
    }

    public async deleteCompletedTodos(): Promise<void> {
      await this.axios.delete(`${this.apiURL}/tasks/completed`);
    }
}
```

client\src\core\api\todo-list\auth\dto\auth.dto.ts

```
export interface ILoginRequestDto {
    email: string;
    password: string;
}

export interface ILoginResponseDto {
    username: string;
    email: string;
    accessToken: string;
    refreshToken: string;
}

export interface IRegisterRequestDto {
    username: string;
    email: string;
    password: string;
    confirmPassword: string;
}

export interface IChangePasswordRequestDto {
    currentPassword: string;
    newPassword: string;
}
```

client\src\core\api\todo-list\auth\auth.api.ts

```
import { AxiosInstance } from 'axios';
import { getAxios } from '@/core/api';
import {
  ILoginRequestDto,
  ILoginResponseDto,
  IRegisterRequestDto,
  IChangePasswordRequestDto,
} from '@/core/api/todo-list/auth/dto/auth.dto';

export class AuthApi {
  private axios: AxiosInstance;
  private apiURL: string;

  constructor(apiURL: string) {
    this.axios = getAxios();
    this.apiURL = apiURL;
  }

  public async fetchProfile() {
    const { data } = await this.axios.get(`${this.apiURL}/users/profile`);
    return data;
  }

  public async login(params: ILoginRequestDto): Promise<ILoginResponseDto> {
    const { data } = await this.axios.post(`${this.apiURL}/users/login`, params);
    return data;
  }

  public async register(params: IRegisterRequestDto) {
```

```
    const { data } = await this.axios.post(`${this.apiURL}/users/register`, params);
    return data;
  }

  public async changePassword(params: IChangePasswordRequestDto) {
    await this.axios.put(`${this.apiURL}/users/change-password`, params);
  }
}
```

server\src\index.ts

```
import './otel';

import express, { Request, Response, Application } from 'express';
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import cors from 'cors';

import userRoutes from './routes/userRoutes';
import tasksRoutes from './routes/taskRoutes';

dotenv.config();

const { DB_URL, PORT = 8000 } = process.env;

if (!DB_URL) {
    console.log("DB URL is not defined");
    process.exit(1);
}

// DB connection
mongoose
    .connect(DB_URL)
    .then(() => console.log('DB ok'))
    .catch((err) => console.log('DB error', err));

const app: Application = express();
app.use(cors());
app.use(express.json());

app.get('/', (req: Request, res: Response) => {
    res.send('Hello world');
});

app.use('/api/users', userRoutes);
app.use('/api/tasks', tasksRoutes)

app.listen(PORT, () => {
    console.log(`Server OK, running on port ${PORT}`);
});
```

server\src\validations\taskValidations.ts

```
import { param, body } from 'express-validator';

export const createUserTaskValidation = [
    body('title').notEmpty().withMessage('Title is required'),
    body('createdDate').notEmpty().withMessage('Created date is required'),
    body('expiredDate').notEmpty().withMessage('Expired date is required')
];

export const editUserTaskValidation = [
    param('taskId').isMongoId().withMessage('Invalid task ID'),

    body('title').optional(),
    body('createdDate').optional(),
    body('expiredDate').optional(),
```

```
  body('completed').optional(),
];
```

server\src\types\tasksTypes.ts

```
import { Request } from 'express';
import { JwtPayload } from 'jsonwebtoken';

export interface UserPayload extends JwtPayload {
  userId: string;
}

export interface AuthRequest extends Request {
  user?: UserPayload;
}

export interface AddTaskRequest extends Request {
  user?: UserPayload;
  body: {
    title: string;
    createdDate: string;
    expiredDate: string;
  };
}

export interface EditTaskRequest extends Request {
  user?: UserPayload;
  body: {
    title?: string;
    createdDate?: string;
    expiredDate?: string;
    completed?: boolean;
  };
  params: {
    taskId: string;
  };
}

export interface DeleteTaskRequest extends Request {
  user?: UserPayload;
  params: {
    taskId: string;
  };
}

export interface DeleteCompletedTasksRequest extends Request {
  user?: UserPayload;
}

export interface TaskQuery {
  userId: string;
  title?: {
    $regex: string;
    $options: string;
  };
  completed?: boolean;
}
```

server\src\routes\taskRoutes.ts

```
import express from 'express';

import { getUserTasks, createUserTask, editUserTask, deleteUserTask, deleteUserCompletedTasks } from '../controllers/taskControllers';
import { checkAuth } from "../middlewares/checkAuth";
import { createUserTaskValidation, editUserTaskValidation } from '../validations/taskValidations';
import { handleValidationErrors } from '../middlewares/handleValidationErrors';

const tasksRouter = express.Router();
```

```
tasksRouter.get('/', checkAuth, getUserTasks);
tasksRouter.post('/', checkAuth, createUserTaskValidation, handleValidationErrors, createUserTask);
tasksRouter.delete('/completed', checkAuth, deleteUserCompletedTasks);
tasksRouter.put('/:taskId', checkAuth, editUserTaskValidation, handleValidationErrors, editUserTask);
tasksRouter.delete('/:taskId', checkAuth, deleteUserTask);

export default tasksRouter;
```

server\src\routes\userRoutes.ts

```
import express from 'express';
import { changePassword, getUserProfile, login, refreshToken, register } from '../controllers/userControllers';
import { handleValidationErrors } from '../middlewares/handleValidationErrors';
import { checkAuth } from '../middlewares/checkAuth';

const usersRouter = express.Router();

usersRouter.post('/register', handleValidationErrors, register)
usersRouter.post('/login', handleValidationErrors, login);
usersRouter.post('/refresh', handleValidationErrors, refreshToken);
usersRouter.get('/profile', checkAuth, handleValidationErrors, getUserProfile);
usersRouter.post('/change-password', checkAuth, changePassword);

export default usersRouter;
```

server\src\models\task.ts

```
import mongoose from "mongoose";

const taskSchema = new mongoose.Schema({
   userId: {
      type: String,
      select: false,
   },
   title: {
      type: String,
      required: true
   },
   createdDate: {
      type: String,
   },
   expiredDate: {
      type: String,
   },
   completed: {
      type: Boolean,
      default: false
   }
});

export const Task = mongoose.model('Task', taskSchema);
```

server\src\models\user.ts

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
   username: {
      type: String,
      required: true,
      unique: true
   },
   email: {
      type: String,
      require: true,
      unique: true,
   },
```

```
    password: {
        type: String,
        required: true
    },
    refreshToken: {
        type: String,
        default: ''
    }
});

export const User = mongoose.model('User', userSchema);
```

server\src\middlewares\checkAuth.ts

```
import { Request, Response, NextFunction } from 'express';
import jwt, { JwtPayload } from 'jsonwebtoken';

interface AuthRequest extends Request {
    user?: JwtPayload | string;
}

const { JWT_SECRET } = process.env;

export const checkAuth = (req: AuthRequest, res: Response, next: NextFunction) => {
    const token = req.headers.authorization?.split(' ')[1];

    if (!token) {
        return res.status(401).json({ message: "Authentication token is missing" });
    }

    try {
        const decoded = jwt.verify(token, JWT_SECRET || 'secretcode111') as JwtPayload;
        req.user = decoded;
        next();
    } catch (error) {
        if (error instanceof jwt.TokenExpiredError) {
            return res.status(401).json({ message: "Token has expired" });
        }
        res.status(401).json({ message: "Invalid or expired token" });
    }
};
```

server\src\middlewares\handleValidationErrors.ts

```
import { Request, Response, NextFunction } from 'express';
import { validationResult } from 'express-validator';

export const handleValidationErrors = (req: Request, res: Response, next: NextFunction) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json(errors.array());
    }

    next();
};
```

server\src\controllers\taskControllers.ts

```
import { Response } from 'express';

import { Task } from '../models/task';
import {
    AddTaskRequest,
    AuthRequest,
    DeleteCompletedTasksRequest,
    DeleteTaskRequest,
    EditTaskRequest,
```

```
    TaskQuery
} from '../types/tasksTypes';
import { TASK_STATUSES } from '../constants/filterStatuses';

export const getUserTasks = async (req: AuthRequest, res: Response) => {
    try {
        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;
        const search = req.query.search as string | undefined;
        const status = req.query.status as string | undefined;

        let baseQuery: TaskQuery = { userId };
        let currentQuery: TaskQuery = { ...baseQuery };

        if (search) {
            currentQuery.title = { $regex: search, $options: 'i' };
        }

        if (status) {
            if (status === TASK_STATUSES.ACTIVE) {
                currentQuery.completed = false;
            } else if (status === TASK_STATUSES.COMPLETED) {
                currentQuery.completed = true;
            }
        }

        const totalTasks = await Task.countDocuments(baseQuery);
        const completedTasks = await Task.countDocuments({ ...baseQuery, completed: true });
        const activeTasks = await Task.countDocuments({ ...baseQuery, completed: false });
        const tasks = await Task.find(currentQuery).sort({ _id: -1 });

        res.status(200).json({
            tasks,
            totals: {
                all: totalTasks,
                completed: completedTasks,
                active: activeTasks
            }
        });
    } catch (error) {
        res.status(500).json({ message: "Failed to get user's tasks" });
    }
};

export const createUserTask = async (req: AddTaskRequest, res: Response) => {
    try {

        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;
        const { title, createdDate, expiredDate } = req.body;

        const newTask = new Task({
            userId,
            title,
            createdDate,
            expiredDate,
            completed: false
        });

        await newTask.save();

        const taskObject = newTask.toObject();
```

```
        delete taskObject.userId;

        res.status(201).json(taskObject);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: "Failed to add a task" });
    }
};

export const editUserTask = async (req: EditTaskRequest, res: Response) => {
    try {
        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;
        const taskId = req.params.taskId;
        const { title, createdDate, expiredDate, completed } = req.body;

        const updatedTask = await Task.findOneAndUpdate(
            { _id: taskId, userId },
            { title, createdDate, expiredDate, completed },
            { new: true, runValidators: true }
        );

        if (!updatedTask) {
            return res.status(404).json({ message: "Task not found" });
        }

        const taskObject = updatedTask.toObject();
        delete taskObject.userId;

        res.status(200).json(taskObject);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: "Failed to edit task" });
    }
};

export const deleteUserTask = async (req: DeleteTaskRequest, res: Response) => {
    try {
        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;
        const taskId = req.params.taskId;

        const task = await Task.findOneAndDelete({ _id: taskId, userId });

        if (!task) {
            return res.status(404).json({ message: "Task not found" });
        }

        res.status(200).json({ message: "Task successfully deleted" });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: "Failed to delete task" });
    }
};

export const deleteUserCompletedTasks = async (req: DeleteCompletedTasksRequest, res: Response) => {
    try {
        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;
```

```
        await Task.deleteMany({ userId, completed: true });

        res.status(200).json({ message: "Completed tasks successfully deleted" });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: "Failed to delete completed tasks" });
    }
};
```

server\src\controllers\userControllers.ts

```
import { Request, Response } from 'express';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { JwtPayload } from 'jsonwebtoken';
import { User } from '../models/user';
import { AuthRequest } from '../types/tasksTypes';

export const register = async (req: Request, res: Response): Promise<void> => {
    try {
        const { username, email, password, confirmPassword } = req.body;

        if (password !== confirmPassword) {
            res.status(400).send("Passwords do not match");
            return;
        }

        const existingUser = await User.findOne({ email });
        if (existingUser) {
            res.status(400).send("Email already in use");
            return;
        }

        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);

        const newUser = new User({
            username,
            email,
            password: hashedPassword,
        });

        const accessToken = jwt.sign({ userId: newUser._id }, "secretcode111", { expiresIn: '30m' });
        const refreshToken = jwt.sign({ userId: newUser._id }, "secretcode111", { expiresIn: '45m' });

        newUser.refreshToken = refreshToken;
        await newUser.save();

        res.status(201).json({
            username: newUser.username,
            email: newUser.email,
            accessToken,
            refreshToken
        });
    } catch (error) {
        console.error(error);
        res.status(500).send("Something went wrong...");
    }
};

export const login = async (req: Request, res: Response): Promise<void> => {
    try {
        const { email, password } = req.body;
        const user = await User.findOne({ email });

        if (!user || !await bcrypt.compare(password, user.password)) {
            res.status(401).send("Email/Password mismatch");
```

```
            return;
        }

        const accessToken = jwt.sign({ userId: user._id }, "secretcode111", { expiresIn: '15m' });
        const refreshToken = jwt.sign({ userId: user._id }, "secretcode111", { expiresIn: '1d' });

        user.refreshToken = refreshToken;
        await user.save();

        res.status(200).json({
            username: user.username,
            email: user.email,
            accessToken,
            refreshToken
        });
    } catch (error) {
        res.status(500).send("Something went wrong...");
    }
};

export const refreshToken = async (req: Request, res: Response): Promise<void> => {
    const { refreshToken } = req.body;

    if (!refreshToken) {
        res.status(401).send("Refresh Token is required");
        return;
    }

    try {
        const decoded = jwt.verify(refreshToken, "secretcode111") as JwtPayload;
        const user = await User.findOne({ _id: decoded.userId, refreshToken });

        if (!user) {
            res.status(403).send("Invalid refresh token");
            return;
        }

        const newAccessToken = jwt.sign({ userId: user._id }, "secretcode111", { expiresIn: '15m' });
        const newRefreshToken = jwt.sign({ userId: user._id }, "secretcode111", { expiresIn: '1d' });

        user.refreshToken = newRefreshToken;
        await user.save();

        res.status(200).json({ newAccessToken, newRefreshToken });
    } catch (error) {
        res.status(403).send("Invalid or expired refresh token");
    }
};

export const getUserProfile = async (req: AuthRequest, res: Response) => {
    try {

        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;

        const user = await User.findById(userId).select('username email');

        if (!user) {
            res.status(404).send("User not found");
            return;
        }

        res.status(200).json({ username: user.username, email: user.email });
    } catch (error) {
        res.status(500).send("Something went wrong...");
```

```
    }
};

export const changePassword = async (req: AuthRequest, res: Response) => {
    try {

        if (!req.user || !req.user.userId) {
            return res.status(400).json({ message: "User ID is missing" });
        }

        const userId = req.user.userId;
        const { currentPassword, newPassword } = req.body;

        if (!currentPassword || !newPassword) {
            res.status(400).send("Current and new password are required.");
            return;
        }

        const user = await User.findById(userId);

        if (!user) {
            res.status(404).send("User not found.");
            return;
        }

        const isMatch = await bcrypt.compare(currentPassword, user.password);
        if (!isMatch) {
            res.status(401).send("Incorrect current password.");
            return;
        }

        const isSame = await bcrypt.compare(newPassword, user.password);
        if (isSame) {
            res.status(400).send("New password must be different from the current password.");
            return;
        }

        const salt = await bcrypt.genSalt(10);
        user.password = await bcrypt.hash(newPassword, salt);

        await user.save();

        res.status(200).send("Password successfully changed.");
    } catch (error) {
        console.error(error);
        res.status(500).send("Internal server error.");
    }
};
```

server\src\constants\filterStatuses.ts

```
export const TASK_STATUSES = {
    ACTIVE: 'active',
    COMPLETED: 'completed'
};
```

server\src\otel.ts

```
import 'dotenv/config';

import { NodeSDK } from '@opentelemetry/sdk-node';
import {
    PeriodicExportingMetricReader,
    ConsoleMetricExporter,
} from '@opentelemetry/sdk-metrics';
import { resourceFromAttributes } from '@opentelemetry/resources';
import {
    ATTR_SERVICE_NAME,
```

```
  ATTR_SERVICE_VERSION,
} from '@opentelemetry/semantic-conventions';
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-proto';
import { OTLPMetricExporter } from '@opentelemetry/exporter-metrics-otlp-proto';

const OTEL_ENDPOINT = process.env.OTEL_EXPORTER_OTLP_ENDPOINT

const sdk = new NodeSDK({
  resource: resourceFromAttributes({
    [ATTR_SERVICE_NAME]: 'todo-app',
    [ATTR_SERVICE_VERSION]: '1.0',
  }),
  traceExporter: new OTLPTraceExporter({
    url: `${OTEL_ENDPOINT}/v1/traces`,
  }),
  metricReader: new PeriodicExportingMetricReader({
    exporter: new OTLPMetricExporter({
      url: `${OTEL_ENDPOINT}/v1/metrics`,
    }),
  }),
  instrumentations: [getNodeAutoInstrumentations({
    '@opentelemetry/instrumentation-pino': {
      logHook: (span, record) => {
        // adding elastic fields
        record['trace.id'] = span.spanContext().traceId;
        record['span.id'] = span.spanContext().spanId;

        // deleting otel auto-instrumented fields to avoid duplication
        delete record['trace_id'];
        delete record['span_id'];
        delete record['trace_flags'];
      },
    },
    '@opentelemetry/instrumentation-fs': {
      enabled: false,
    },
  })],
});

sdk.start();
…
```

ДОДАТОК Б ЛІСТИНГ КОДУ МЕТОДІВ ВИЯВЛЕННЯ АНОМАЛІЙ

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def load_and_prepare(file_path: str) -> pd.DataFrame:
    df = pd.read_csv(file_path, parse_dates=['timestamp'])
    df = (
        df.set_index('timestamp')
          .sort_index()
          .resample('5T')
          .sum()
          .rename(columns={'http_requests': 'y'})
    )
    return df

def detect_z_score(df: pd.DataFrame, window: str = '3h', z_thresh: float = 3.0) -> pd.DataFrame:
    mean = df['y'].rolling(window, min_periods=1).mean()
    std  = df['y'].rolling(window, min_periods=1).std(ddof=0).replace(0, np.nan)
    z_score = (df['y'] - mean) / std

    df_z = df.copy()
    df_z['z_score'] = z_score
    df_z['z_anomaly'] = z_score.abs() > z_thresh
    return df_z

def detect_iqr(df: pd.DataFrame, window: str = '3h', iqr_factor: float = 1.5) -> pd.DataFrame:
    q25 = df['y'].rolling(window, min_periods=1).quantile(0.25)
    q75 = df['y'].rolling(window, min_periods=1).quantile(0.75)
    iqr = q75 - q25
    lower = q25 - iqr_factor * iqr
    upper = q75 + iqr_factor * iqr

    df_iqr = df.copy()
    df_iqr['iqr_lower'] = lower
    df_iqr['iqr_upper'] = upper
    df_iqr['iqr_anomaly'] = (df['y'] < lower) | (df['y'] > upper)
    return df_iqr

def plot_anomalies(df: pd.DataFrame, anomaly_col: str, title: str):
    fig, ax = plt.subplots(figsize=(12, 4))
    ax.plot(df.index, df['y'], label='HTTP requests', linewidth=1)
    anomalies = df[df[anomaly_col]]
    ax.scatter(
        anomalies.index, anomalies['y'],
        marker='X', facecolors='none', edgecolors='red',
        s=40, label=anomaly_col
    )
    ax.set_title(title)
    ax.set_xlabel('Time')
    ax.set_ylabel('Requests / 5 min')
    ax.legend()
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    file_path = 'http_requests_1day_spikes.csv'
    df = load_and_prepare(file_path)

    df_z = detect_z_score(df)
    plot_anomalies(df_z, 'z_anomaly', 'Rolling 3-hour Z-score Anomaly Detection')
```

```
    df_iqr = detect_iqr(df)
    plot_anomalies(df_iqr, 'iqr_anomaly', 'Rolling 3-hour IQR Anomaly Detection')

import pandas as pd
from prophet import Prophet
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error

FILE_PATH = "prophet.csv"
FREQ = "30min"
INTERVAL_WIDTH = 0.99

def load_data() -> pd.DataFrame:
    df = pd.read_csv(FILE_PATH, parse_dates=["timestamp"])
    df = (
        df.set_index("timestamp")["http_requests"]
          .resample(FREQ).sum()
          .reset_index()
          .rename(columns={"timestamp":"ds", "http_requests":"y"})
          .sort_values("ds")
    )
    return df

def main():
    data = load_data()

    split_point = data["ds"].min() + pd.Timedelta(days=14)
    train_data = data[data["ds"] < split_point]
    test_data = data[data["ds"] >= split_point]

    model = Prophet(
        daily_seasonality=True,
        weekly_seasonality=True,
        changepoint_prior_scale=0.05,
        interval_width=INTERVAL_WIDTH
    )
    model.fit(train_data)

    future = model.make_future_dataframe(periods=len(test_data), freq='30min', include_history=False)
    forecast = model.predict(future)

    model.plot_components(forecast)
    plt.show()

    forecast_df = forecast[['ds','yhat','yhat_upper','yhat_lower']]

    print(forecast_df)

    df = data.merge(
        forecast[["ds","yhat","yhat_lower","yhat_upper"]],
        on="ds", how="right"
    )

    test = df[df["ds"] >= split_point]
    mae = mean_absolute_error(test["y"], test["yhat"])
    mse = mean_squared_error(test["y"], test["yhat"])
    mask = test["y"] != 0
    mape = mean_absolute_percentage_error(test.loc[mask,"y"], test.loc[mask,"yhat"])
    print(f"MAE: {mae:.2f}")
    print(f"MSE: {mse:.2f}")
    print(f"MAPE: {mape:.2%}")

    df["anomaly"] = (df["y"] < df["yhat_lower"]) | (df["y"] > df["yhat_upper"])
    anoms_test = df[(df["anomaly"]) & (df["ds"] >= split_point)]

    if not anoms_test.empty:
        print("\nDetected anomalies at the following timestamps:")
        for ts in anoms_test["ds"]:
```

```
            print(ts.strftime("%Y-%m-%d %H:%M:%S"))
    else:
        print("\nNo anomalies detected in the test set.")

    plt.figure(figsize=(14,5))
    plt.plot(df["ds"], df["y"], label="Actual")
    plt.plot(df["ds"], df["yhat"], label="Forecast")
    plt.fill_between(df["ds"], df["yhat_lower"], df["yhat_upper"],
                     color="lightblue", alpha=0.4, label=f"{int(INTERVAL_WIDTH*100)}% PI")
    plt.axvline(split_point, color="grey", linestyle="--", linewidth=1)
    plt.title("HTTP Requests Train/Test Split)")
    plt.xlabel("Timestamp")
    plt.ylabel("Number of Requests")
    plt.legend()
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(14,5))
    plt.plot(df["ds"], df["y"], label="Actual")
    plt.fill_between(df["ds"], df["yhat_lower"], df["yhat_upper"],
                     color="lightblue", alpha=0.4, label=f"{int(INTERVAL_WIDTH*100)}% PI")
    plt.scatter(anoms_test["ds"], anoms_test["y"],
                color="red", marker="x", s=30, label="Anomaly")
    plt.axvline(split_point, color="grey", linestyle="--", linewidth=1)
    plt.title("HTTP Requests with Anomalies (Test Set)")
    plt.xlabel("Timestamp")
    plt.ylabel("Number of Requests")
    plt.legend()
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

ДОДАТОК В ПРЕЗЕНТАНЦІЯ

1

Методи аналізу показників та виявлення аномалій в інтелектуальному моніторингу веб-застосунків

Виконав:

Студент IV курсу, групи КА-12
Чайка Микита Сергійович

Керівник:

Асистент каф. ММСА
Древаль Максим Михайлович

2

АКТУАЛЬНІСТЬ РОБОТИ

Зростаюча складність веб-застосунків – сучасні сервіси працюють у вигляді мікросервісної архітектури з розподіленими компонентами, що ускладнює виявлення та діагностику несправностей.

Фінансові ризики простоїв – кожна хвилина простою або погіршення продуктивності може призводити до мільйонних втрат.

Швидкі зміни в навантаженні та поведінці користувачів – зростання трафіку, пікові навантаження та зміни користувацьких патернів вимагають прогнозування й адаптивного реагування в режимі реального часу.

3

ОБ’ЄКТ, ПРЕДМЕТ ТА МЕТА ДОСЛІДЖЕННЯ

Об'єкт дослідження – показники роботи веб-застосунків.

Предмет дослідження – методи аналізу часового ряду метрик та автоматичного виявлення аномалій у рамках інтелектуального моніторингу веб-застосунків.

Мета роботи – розробка та налаштування програмно-аналітичного продукту для аналізу ключових метрик веб-застосунків та автоматичного виявлення аномалій із використанням статистичних алгоритмів, моделі Prophet та платформи Elastic Stack.

4

ПОРІВНЯННЯ КОНЦЕПЦІЙ МОНІТОРИНГУ ТА СПОСТЕРЕЖУВАНОСТІ

Моніторинг

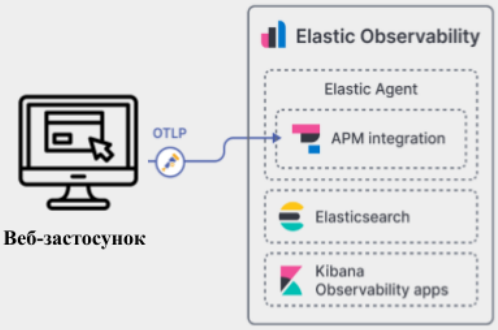
Безперервне спостерігання за системою, щоб виявляти та повідомляти про аномальну поведінку.

Спостережуваність

Розуміння, що відбувається всередині системи, та прогнозування її поведінку в майбутньому.

5

ОГЛЯД ВИКОРИСТАНИХ МЕТОДІВ



+

PROPHET

Статистичні методи:
Z-score та IQR

6

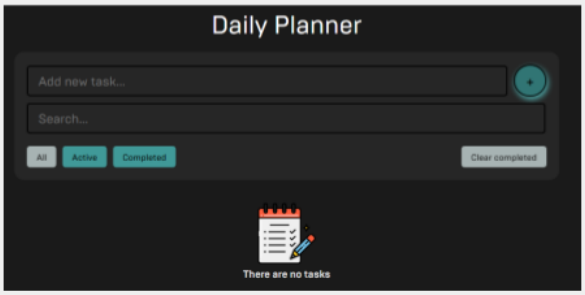
РОЗРОБКА ЗАСТОСУНКА ДЛЯ ЕКСПЕРИМЕНТУ

У якості мінімально життєздатного проєкту (MVP) був обраний простий Daily planner, який складається з трьох базових модулів:

Авторизація: реєстрація користувача, вхід/вихід, захищені сесії (JWT).

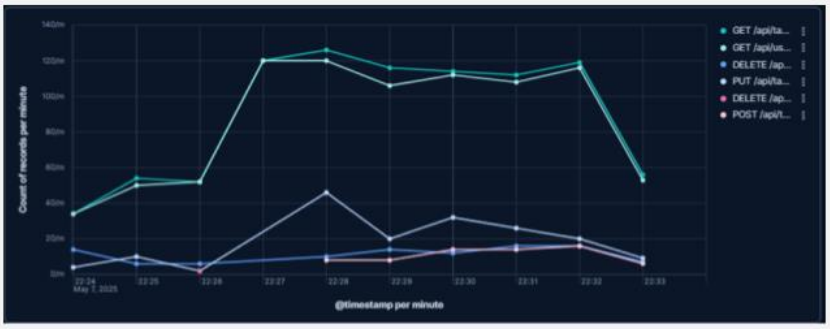
Робота з задачами: створення, перегляд, редагування та видалення записів;

Особистий кабінет: можливість подивитися дані профілю (ім'я, електронну пошту), змінити пароль.



7

НАЛАШТУВАННЯ СИСТЕМИ МОНІТОРИНГУ



Графік відображає зміну кількості HTTP-запитів за хвилину до шести API-ендпоїнтів протягом інтервалу 22:24 – 22:33 7 травня 2025 р. Кожна лінія показує окремий метод (GET, POST, PUT, DELETE) для ресурсів tasks та users, що дозволяє швидко оцінити динаміку навантаження на сервіси.

8

НАЛАШТУВАННЯ СИСТЕМИ МОНІТОРИНГУ

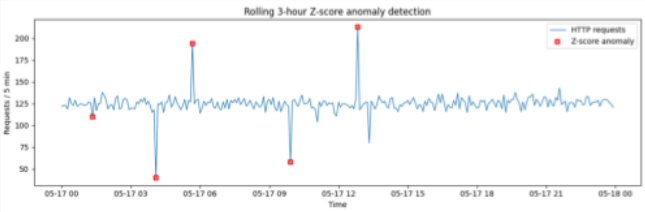
Статуси відповідей запитів

Затримка відповіді запитів (latency)

Навантаження центрального процесора

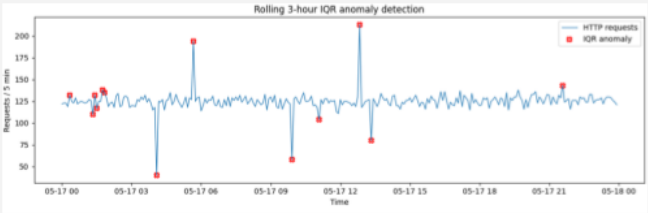
9

МЕТОДИ ВИЯВЛЕННЯ АНОМАЛІЙ



$$z = \frac{x - \mu}{\sigma},$$

Якщо абсолютне значення перевищує певний заданий поріг (наприклад $|z| > 3$), значення вважається аномальним.



$$IQR = Q3 - Q1,$$

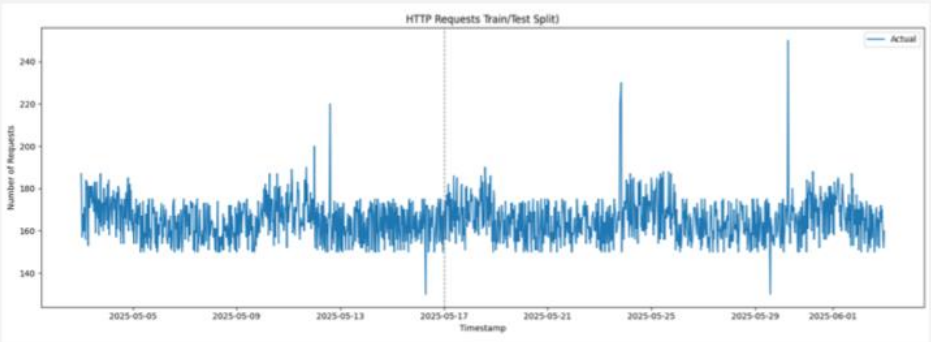
Нижня межа = Q1 - k * IQR, Верхня межа = Q3 + k * IQR

Точки, які виходять за межі певного діапазону за межами Q1 і Q3, вважаються викидами.

Для перевірки цих методів було використано згенерований набір даних, що охоплює часовий проміжок HTTP-запитів за період 17-18 травня 2025 р з інтервалом 5 хвилин.

10

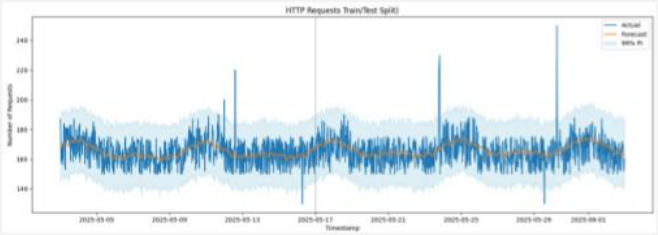
ВИБІР ТА ОПИС ДАТАСЕТУ ДЛЯ МОДЕЛІ PROPHET



Для цього експерименту використовувався місячний датасет, який показує кількість HTTP запитів з інтервалом 30 хвилин за період з 3 травня по 3 червная 2025 року, і включає добову та тижневу сезонність у вигляді підвищеного трафіку на вихідних. Він був розділений на навчальну (перші 2 тижня) та тестову частини.

ПРОГНОЗУВАННЯ ЗА ДОПОМОГОЮ МОДЕЛІ PROPHET

11



Графік прогнозованих значень

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t,$$

g(t) – довготривалий тренд (лінійний або логістичний, може змінюватися в різні моменти часу);

s(t) – сезонна складова (наприклад, щотижнева або щоденна цикличність);

h(t) – вплив окремих подій (свята, рекламні кампанії тощо);

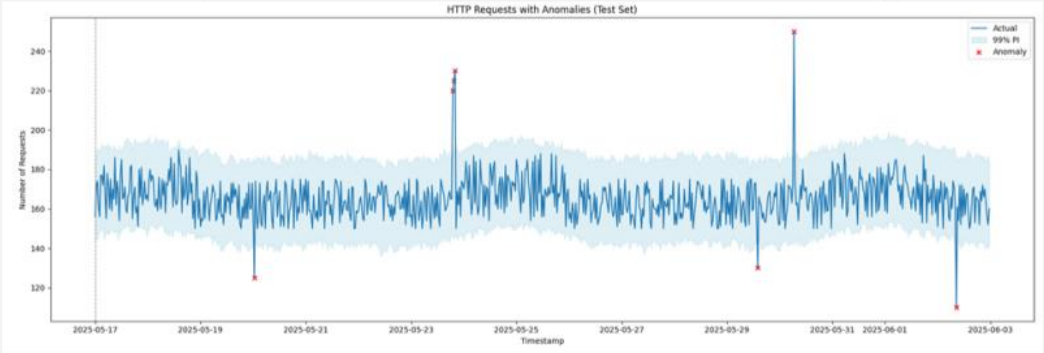
ε_t – залишкова похибка, або шум, що відображає невраховані впливи.

MAE	MSE	MAPE
7.45	97.77	4.53%

Метрики точності прогнозу моделі Prophet

ВИЯВЛЕННЯ АНОМАЛІЙ ЗА ДОПОМОГОЮ МОДЕЛІ PROPHET

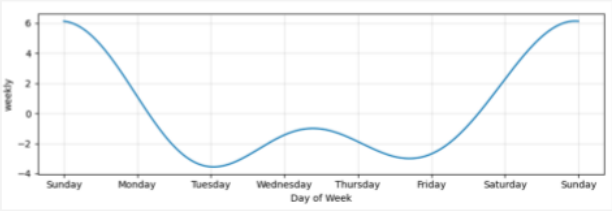
12



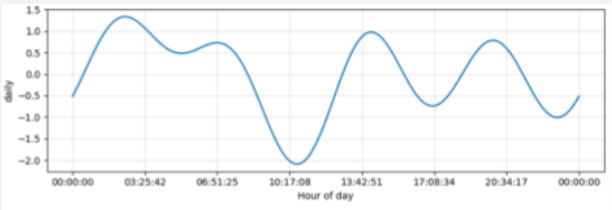

```
Detected anomalies at the following timestamps:
2025-05-20 00:30:00
2025-05-23 19:00:00
2025-05-23 19:30:00
2025-05-23 20:00:00
2025-05-29 14:00:00
2025-05-30 06:30:00
2025-06-02 08:30:00
```

КОМПОНЕНТНИЙ АНАЛІЗ ПРОГНОЗУ

13



Тижнева сезонність (weekly): графік показує циклічні коливання, які повторюються щотижня. Бачимо, що в нашому випадку найвищий підйом спостерігається в неділю (+5-6 запитів), а найнижчий у вівторок (-4 запити).



Добова сезонність (daily): графік показує добову сезонність, яка відображає регулярні коливання трафіку протягом 24 годин, тобто як активність користувача змінюється протягом доби. Бачимо, що найнижчий рівень трафіку припадає на ранок, тоді як пік ближче до вечора.

ВИСНОВКИ

14

У рамках роботи створено програмно-аналітичний продукт для інтелектуального моніторингу веб-застосунків та перевірено його ефективність на розробленому MVP "Daily Planner". Для моніторингу дані збиралися за допомогою OpenTelemetry в Elastic Stack, де їх візуалізували на інтерактивних дашбордах, а для автоматичного виявлення аномалій застосували два статистичні підходи й модель Prophet:

- **Статистичні методи (ковзний Z-score та IQR) –** Швидко й надійно фіксують різкі «сплески» або «провали» – класичні викиди, що тривають кілька хвилин.

- **Модель Prophet** – Будує комплексний прогноз із урахуванням трендів і сезонностей, тому краще ловить добові та тижневі коливання й мінімізує хибні спрацювання; на тестових даних модель досягла MAPE = 4,53 % і виявила 7 аномалій (4 піків та 3 просідання) у період 17 травня – 3 червня 2025 р.

15

Дякую за увагу