ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут заочно-дистанційної освіти
Форма навчання заочна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА САЙТУ ДЛЯ УПРАВЛІННЯ СВОЇМ РОЗКЛАДОМ ТА ЗАВДАННЯМИ»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Загоруйко Яна Олександрівна

_____ «___»_____2025 р.
(підпис)

Науковий керівник к., ф.-м. н., доц., Черненко Оксана Олексіївна

_____ «___»_____2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «Розробка сайту для управління своїм розкладом та завданнями»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Загоруйко Яна Олександрівна
Затверджена наказом ректора № 181-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні тренажери в дистанційних курсах з комп'ютерних наук.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
   2.1. Класифікація сайтів для управління розкладом
   2.2. Переваги та недоліки сучасних цифрових планувальників
   2.3. Огляд схожих цифрових планувальників
   2.4. Необхідність та актуальність теми роботи
3. ТЕОРЕТИЧНА ЧАСТИНА
   3.1. Опис матеріалу з теми
   3.2. Опис структури сайту
   3.3. Алгоритм роботи сайту управління розкладом
   3.4. UML-діаграма
   3.5. Обґрунтування вибору мов програмування
4. ПРАКТИЧНА ЧАСТИНА
   4.1. Опис розробки програмного забезпечення сайту
   4.2. Опис роботи сайту для управління розкладом
   4.3. Аналіз роботи планувальника на основі тестового використання
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А
Перелік графічного матеріалу: 1 UML-діаграма, інші необхідні ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Черненко О. О.		
Інформаційний огляд	Черненко О. О.		
Теоретична частина	Черненко О. О.		
Практична реалізація	Черненко О. О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.

Здобувач вищої освіти Яна Загоруйко

Науковий керівник к. ф.-м. н., доц. Оксана ЧЕРНЕНКО

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.

Секретар ЕК_____ _____

(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

Погоджено
Науковий керівник _____
к. ф.-м. н., доц. Оксана ЧЕРНЕНКО
«____» _____ 202_ р.

План

кваліфікаційної роботи ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
Загоруйко Яна Олександрівна
Прізвище, ім'я, по батькові

на тему «Розробка сайту для управління своїм розкладом та завданнями»

Здобувач вищої освіти _____ Яна ЗАГОРУЙКО
«____» _____ 202_ р.

АНОТАЦІЯ

Записка: 94 сторінки, в тому числі з яких: основна частина – 45 сторінок та 49 сторінок додатку, літературних джерел – 17 назв, рисунків – 21, UML-діаграма – 1 сторінка.

Мета роботи – проєктування та програмна реалізація інтерфейсу веб-планувальника для організації щоденних завдань з можливістю встановлення до п’яти ключових цілей.

Об’єкт розробки – веб-сайт для управління особистим розкладом та завданнями.

Методи дослідження та інформаційне забезпечення – аналіз сучасних підходів до планування часу, дослідження методів концентрації на ключових цілях, UX/UI-дизайну; використання інструментів фронтенд і бекенд-розробки (React, Next.js, NestJS, TypeScript, MongoDB). У розробці реалізовано темну/світлу тему, двомовність (українська/англійська), систему автентифікації, drag-and-drop переміщення завдань, відображення щомісячної та річної статистики виконання завдань відповідно до встановлених цілей.

Результати дослідження. Розроблено веб-планувальник, що дозволяє користувачеві створювати до п’яти ключових цілей та необмежену кількість завдань, що до них належать. Такий підхід забезпечує глибоку концентрацію на найважливішому, зберігаючи при цьому гнучкість у плануванні щоденної активності. Інтерфейс містить режим темної та світлої теми, підтримку двох мов, статистику. Планувальник дозволяє гнучко переглядати щоденні, тижневі та цільові завдання, підтримує сортування, редагування та взаємодію через перетягування.

Рекомендації щодо використання результатів дослідження. Розроблений веб-планувальник рекомендовано використовувати як інструмент для особистого планування, підвищення продуктивності та досягнення цілей. Він може бути корисним студентам, фахівцям різних галузей, творчим людям та всім, хто прагне структурувати своє життя, фокусуючись на важливому.

Результати дослідження можуть бути використані як основа для подальшої розробки мобільного додатку, інтеграції з календарями або платформами ментального здоров’я.

Ключові слова: МОВА ПРОГРАМУВАННЯ, ВЕБ-ПЛАНУВАЛЬНИК, ПРОГРАМА-АСИСТЕНТ, УПРАВЛІННЯ ЧАСОМ, JAVASCRIPT, TYPESCRIPT, NEXT.JS, NEST.JS, MONGODB

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	**Пояснення умовних позначень, скорочень, символів**
API	Інтерфейс програмування додатків (Application Programming Interface). Механізм для взаємодії клієнта з сервером.
JWT	JSON Web Token. Стандарт для передачі даних між клієнтом та сервером у вигляді токену.
HTTP	HyperText Transfer Protocol. Протокол для передачі даних в Інтернеті.
Token	Токен. Ключ для доступу до захищених ресурсів після авторизації.
Interceptor	Перехоплювач. Механізм для додавання токенів до запитів або обробки помилок перед тим, як передати запит на сервер.
Middleware	Проміжне програмне забезпечення. Функції, що виконуються між отриманням запиту та його обробкою сервером.
Drag-and-drop	Функціональність перетаскування елементів в інтерфейсі.

ВСТУП

У сучасному світі, де кількість щоденних завдань невпинно зростає, а обсяг інформації постійно збільшується, ефективне управління власним часом і розкладом перетворюється на життєву необхідність [1]. Завдяки стрімкому розвитку Інтернет-технологій з’явилися цифрові інструменти, зокрема інтерактивні веб-планувальники, які дозволяють не лише слідкувати за виконанням завдань, а й ефективно організовувати день, відстежувати прогрес і підвищувати особисту продуктивність. Такі платформи надають користувачам зручний доступ до управління завданнями з будь-якої точки світу, забезпечуючи структуровану підтримку у досягненні особистих і професійних цілей.

Планування – це не просто розподіл часу, а стратегічний підхід до досягнення довгострокових цілей [2]. Застосування комп’ютерних технологій у сфері самоменеджменту сприяє покращенню якості життя, а також може бути використане для підвищення ефективності в освітньому та професійному середовищі.

Одне з ключових завдань сучасного планування – забезпечити людині інструменти, які враховують її інтереси, здібності та потреби, підтримуючи особистісний розвиток і збереження ментального балансу [3].

Метою роботи є розробка веб-сайту для управління особистим розкладом і завданнями, який дозволяє користувачам зосереджуватися на досягненні до п’яти ключових цілей одночасно, з гнучким налаштуванням щоденних завдань, статистикою прогресу та інтерактивними візуальними елементами.

Об’єктом розробки цієї роботи є програмне забезпечення у вигляді веб-застосунку для організації особистого планування.

Предметом розробки є веб-платформа з теми «Управління своїм розкладом та завданнями».

Методи, які були використані при розробці: аналітичний підхід до вивчення користувацьких потреб і моделей поведінки, методи UX/UI-дизайну, застосування сучасних фреймворків і мов програмування (TypeScript, React, Next.js, NestJS), а також використання принципів адаптивної верстки, клієнт-серверної архітектури та баз даних MongoDB.

Новизна даної роботи полягає у поєднанні кількох інноваційних аспектів, що сприяють підвищенню ефективності та зручності використання планувальника. По-перше, застосовано фокусний підхід до цілей, що обмежує кількість активних завдань до 5 одночасно, що дозволяє користувачам зберігати концентрацію та уникати перевантаження, а також дає змогу краще організувати час. По-друге, інтерактивний інтерфейс з drag-and-drop функціональністю забезпечує гнучкість у плануванні та зручність для користувача, дозволяючи змінювати пріоритети завдань за допомогою простих перетягувань. Крім того, додаток підтримує багатомовність, що дозволяє використовувати його користувачам різних мовних груп, та надає можливість вибору між темною і світлою темами, що підвищує зручність роботи в різних умовах освітлення. Важливим аспектом є вбудована статистика виконання завдань, що дозволяє користувачам відслідковувати свій прогрес по кожній цілі, а також отримувати огляд на виконані завдання за місяць і рік, що сприяє кращому плануванню та мотивації. Все це разом робить планувальник універсальним і персоналізованим інструментом для управління часом, відповідно до потреб кожного користувача.

Практична значимість роботи полягає у створенні інструменту, який може бути використаний широким колом користувачів – студентами, фахівцями, фрилансерами, творчими людьми – для ефективного управління особистим часом і досягнення стратегічних життєвих або професійних цілей.

Пояснювальна записка до кваліфікаційної роботи складається з чотирьох розділів:

- перший розділ містить постановку задачі, визначення мети, об’єкта та предмета дослідження, обґрунтування актуальності теми;

- другий розділ присвячено інформаційному аналізу: досліджено переваги та недоліки вже існуючих інструментів планування, проведено огляд аналогічних веб-платформ, виявлено труднощі у їх використанні, підкреслено потребу у створенні нової системи;
- третій розділ містить технічне обґрунтування, UML-діаграми, опис структури сайту, алгоритмів його функціонування, а також вибір інструментів і технологій;
- четвертий розділ охоплює опис реалізації програмного продукту, з прикладами інтерфейсу та описом функціональних можливостей веб-планувальника.

Обсяг пояснювальної записки до кваліфікаційної роботи складає: 94 сторінки, в тому числі з яких: основна частина – 45 сторінок та 49 сторінок додатку, літературних джерел – 17 назв, рисунків – 21, UML-діаграми – 1 сторінка.

1. ПОСТАНОВКА ЗАДАЧІ

Задачею кваліфікаційної роботи є проектування та розробка веб-сайту з теми «Управління своїм розкладом та завданнями».

Для досягнення цієї мети визначимо такі основні завдання:

- проаналізувати переваги та недоліки сучасних цифрових планувальників;
- провести огляд і аналіз існуючих рішень, реалізованих у подібних програмних продуктах. Зокрема, виокремити функціональні переваги та труднощі, що виникають під час їхнього використання;
- розглянути теоретичні аспекти ефективного тайм-менеджменту. Виділити ключові концепції, на основі яких буде сформовано алгоритм роботи веб-платформи;
- розробити логічну та технічну структуру майбутнього веб-сайту;
- сформувати алгоритм роботи цифрового планувальника;
- створити UML-діаграми, що ілюструють архітектуру та логіку функціонування системи;
- обґрунтувати вибір мов програмування, фреймворків та інструментів для реалізації програмного забезпечення;
- докладно описати процес розробки веб-додатку, включаючи етапи верстки, логіки та тестування;
- описати основну функціональність готового планувальника, включно з авторизацією, створенням, редагуванням та відстеженням завдань і цілей.

Для розробки планувальника висунемо деякі вимоги:

- інтерфейс веб-сайту повинен бути інтуїтивно зрозумілим, зручним у використанні та адаптованим для різних пристроїв;
- сервіс повинен забезпечувати збереження даних, базові заходи безпеки для авторизованих користувачів;

- програмний продукт має допомагати користувачеві в досягненні особистих цілей шляхом структурування задач та фокусування на пріоритетних напрямах;
- в теоретичній частині планується включення типових прикладів, що демонструють процеси: створення цілей і завдань, авторизації та реєстрації, редагування існуючих даних;
- у практичній частині, для закріплення матеріалу, будуть представлені приклади реального використання платформи, тести на функціональність і юзабіліті, а також етапи налаштування та розгортання.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Класифікація сайтів для управління розкладом

У сучасному цифровому світі, де темп життя стрімко зростає, а кількість завдань та інформації перевищує можливості звичайного паперового обліку, цифрові планувальники стали невід'ємним інструментом для організації повсякденних справ. Ці платформи дозволяють користувачам ефективно планувати свій час, ставити цілі, відстежувати прогрес і підвищувати особисту продуктивність. Однак, незважаючи на численні переваги, цифрові планувальники мають також і певні недоліки, які слід враховувати при виборі або розробці такого програмного продукту.

Цифрові планувальники забезпечують зручність завдяки доступу з будь-якого пристрою та в будь-який час. Завдяки синхронізації даних між пристроями, користувач завжди має актуальну інформацію щодо своїх завдань. Більшість сервісів надають можливість налаштування нагадувань, що допомагає уникнути забутих дедлайнів та забезпечує постійний контроль над виконанням завдань. Крім того, сучасні інструменти для планування містять функції аналітики та візуалізації, що дозволяють користувачу оцінити ефективність своєї роботи протягом певного періоду.

Гнучкість таких систем полягає у широких можливостях персоналізації – від вибору темної або світлої теми до налаштування власних категорій та пріоритетів. Багато сервісів також підтримують функції командної роботи, що дозволяє планувати спільні проєкти або розподіляти завдання між учасниками.

У свою чергу, цифрові планувальники можна класифікувати за кількома основними критеріями, що допомагає користувачам визначити оптимальний інструмент для їх потреб:

- за типом користувачів:

1) індивідуальні планувальники: спрямовані на організацію особистих завдань, цілей та подій. Вони зазвичай прості у використанні та не потребують складних налаштувань;
2) корпоративні планувальники: призначені для командної роботи, управління проектами, розподілу завдань та спільного виконання задач. Такі сервіси часто включають інтеграцію з іншими корпоративними системами, що підвищує ефективність спільної роботи;

- за функціональністю:
  1) базові планувальники: пропонують основні функції, такі як список завдань, календар, нагадування. Ці сервіси підходять для користувачів, яким необхідна простота й мінімалізм;
  2) розширені планувальники: включають додаткові функції, такі як аналітика виконаних завдань, статистика, можливість встановлення пріоритетів, дедлайнів, налаштування цілей та задач на різні періоди часу. Вони підходять для більш організованих користувачів, які хочуть мати контроль над детальним плануванням свого часу;
  3) інтерактивні планувальники: включають можливості drag-and-drop для змінюваних завдань і цілей. Це дозволяє користувачам маніпулювати завданнями візуально і зручніше адаптувати планування до їхніх потреб;
- за ціною:
  1) безкоштовні планувальники;
  2) платні планувальники;
- за рівнем доступу:
  1) веб-сайти: планувальники, що доступні через браузер. Вони дозволяють користувачам працювати з плануванням на будь-якому пристрої, що має доступ до Інтернету;

2) мобільні додатки: планувальники, що можна завантажити на смартфони або планшети, що забезпечує доступ до функцій навіть без наявності Інтернету.

Кожен тип цифрових планувальників має свої особливості, які можуть підходити різним категоріям користувачів в залежності від їхніх вимог та специфіки діяльності. Тому при виборі платформи для управління розкладом необхідно враховувати не тільки функціональність, але й особисті уподобання та цілі користувача.

2.2 Переваги та недоліки сучасних цифрових планувальників

Водночас цифрові планувальники мають і недоліки. Залежність від інтернет-з'єднання або конкретного пристрою може призвести до неможливості доступу до системи в критичний момент, що обмежує гнучкість у використанні. Існують також ризики втрати або витоку персональних даних через хакерські атаки, технічні помилки або неналежну безпеку платформи, що ставить під загрозу конфіденційність користувачів. Деякі інструменти можуть бути занадто складними для новачків, що потребує часу на навчання і адаптацію до нових функцій. Крім того, цифрові пристрої самі по собі є джерелом відволікаючих чинників, таких як повідомлення, сповіщення з інших додатків та соціальних мереж, що може значно знижувати концентрацію і продуктивність користувача.

Порівняльний аналіз переваг та недоліків сучасних цифрових планувальників наведено у таблиці нижче:

Таблиця 2.1 – Цифрові планувальники: переваги і недоліки

№	Перевага (+)	Недолік (–)
1	Доступ з будь-якого пристрою та з будь-якої точки світу.	Залежність від інтернету та техчнічних засобів.
2	Синхронизація даних між пристроями.	Можливість витоку персональних даних або їх втрати.

№	Перевага (+)	Недолік (–)
3	Автоматичні нагадування про дедлайни та події.	Перевантажений інтерфейс або складність у використанні для новачків.
4	Вбудовані елементи аналітики та статистики.	Залежність від сервісу або його підтримки.
5	Персоналізація інтерфейсу та гнучке налаштування.	Необхідність дисципліни для регулярного використання.
6	Підтримка командної роботи та колаборації.	Можливі відволікаючи фактори при роботі з пристроями.
7	Екологічність – зменшення використання паперу.	Часткова заміна живого планування, яке для деяких людей є ефективнішим.

Таким чином, цифрові планувальники є потужним інструментом для організації часу, але їх ефективність залежить від грамотного вибору, рівня цифрової грамотності користувача, а також здатності підтримувати дисципліну. При розробці власного планувальника важливо врахувати як позитивні аспекти, так і можливі обмеження наявних рішень, щоб створити інструмент, що максимально відповідає потребам цільової аудиторії.

2.3 Огляд схожих цифрових планувальників

При розробці власного цифрового планувальника щоденних завдань важливо врахувати функціональність, дизайн і користувацький досвід існуючих рішень, що вже зарекомендували себе на ринку. Це дозволяє виявити сильні сторони конкурентів, визначити недоліки, які можна уникнути, а також запозичити найкращі практики для підвищення якості та ефективності майбутнього програмного продукту.

Серед найбільш популярних платформ для управління завданнями та організації часу варто виокремити **Trello** та **Todoist**. Обидва сервіси широко використовуються як для особистого, так і для командного планування, але реалізують різні підходи до структурування інформації.

Trello – це візуальний інструмент для управління проєктами та завданнями, що базується на методології канбан. Trello широко використовується як для особистого планування, так і для командної роботи (інформація взята з офіційного сайту Trello) [4]. Робоча область має простий та зрозумілий інтерфейс (рис. 2.1).

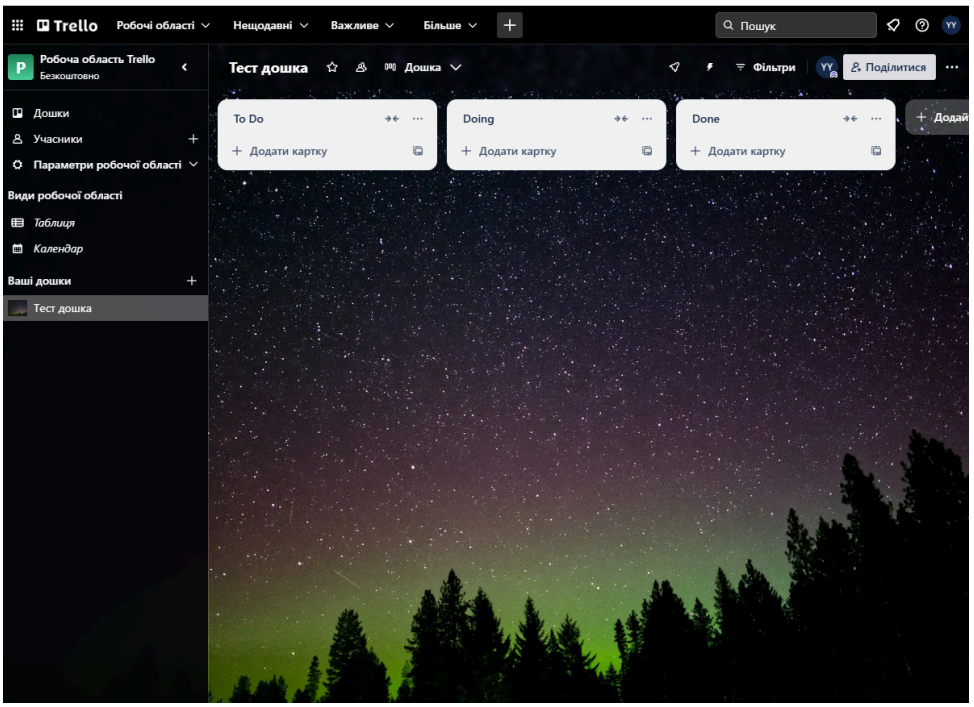

Рисунок 2.1 – Робоча область Trello

Основні характеристики Trello:

- візуалізація процесу: Trello дозволяє організовувати завдання у вигляді дощок, списків і карток, що допомагає користувачам наочно бачити, на якому етапі знаходиться кожна задача;
- гнучкість налаштувань: користувачі можуть налаштовувати картки для різних типів завдань, додавати коментарі, дедлайни, вкладення та навіть автоматизувати рутинні дії;

- колаборація та спільний доступ: Trello ідеально підходить для роботи в команді, оскільки дозволяє декільком користувачам спільно працювати над завданнями, призначати відповідальних, залишати коментарі та ділитися файлами;
- інтеграції з іншими інструментами: Trello має інтеграцію з багатьма іншими сервісами, такими як Slack, Google Drive та інші, що робить його гнучким для різних робочих процесів.

Недоліки Trello:

- надлишкова складність для особистих цілей: для щоденних індивідуальних завдань інтерфейс Trello може здаватися перевантаженим;
- відсутність акценту на пріоритети;
- відсутність глибокої аналітики: Trello не надає статистики про виконання завдань без додаткових плагінів.

Todoist – це популярний додаток для управління завданнями, який дозволяє структурувати список справ, встановлювати пріоритети і відстежувати прогрес (інформація взята з офіційного сайту Todoist) [5]. Робоча область досить проста та зрозуміла (рис. 2.2).

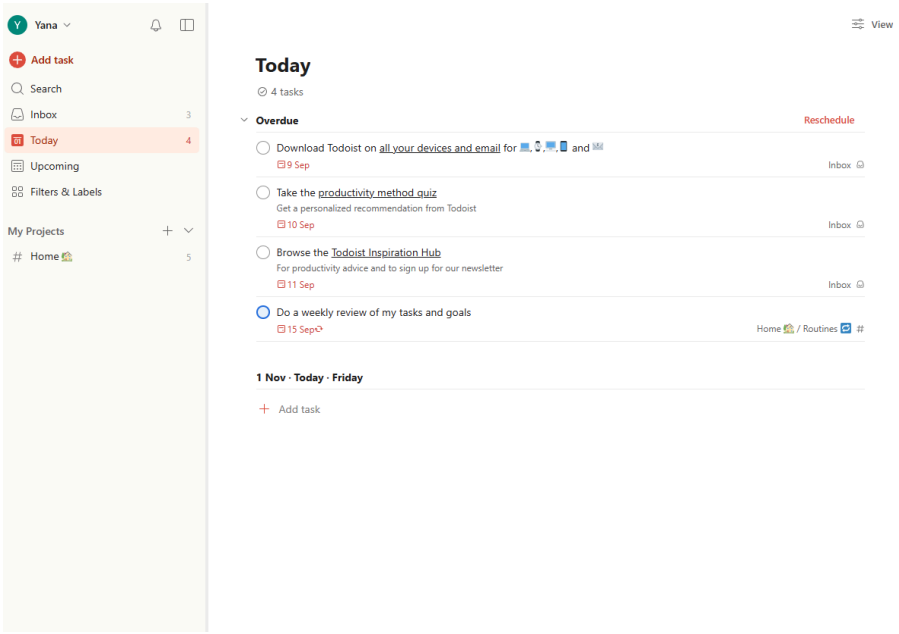

Рисунок 2.2 – Робоча область Todoist

Основні функції Todoist включають:

- фокус на ключових завданнях**:** Todoist дозволяє створювати окремі списки завдань на кожен день, тиждень або місяць, а також виділяти пріоритетні завдання, що допомагає користувачам зосередитись на найважливішому;
- статистика та аналітика: Todoist пропонує огляд виконаних завдань та показує, як користувач виконує поставлені цілі, що є важливим для мотивації;
- інтуїтивний інтерфейс: Todoist має простий та зрозумілий дизайн, що дозволяє користувачам швидко додавати або редагувати завдання;
- синхронізація та доступ на різних пристроях: дані автоматично зберігаються, і користувачі можуть отримати доступ до свого розкладу з комп'ютера, планшета або телефону;
- Todoist особливо зручний для тих, хто шукає простий спосіб керувати завданнями з мінімальними налаштуваннями.

Недоліки Todoist:

- мінімальна візуалізація: користувачі не отримують наочного представлення свого прогресу чи загальної картини завдань;
- складно встановлювати зв’язок завдань із довгостроковими цілями.

Обидва інструменти – Trello і Todoist – демонструють високий рівень ефективності, але орієнтовані на різні типи користувачів і стилі роботи. Todoist краще підходить для користувачів, які шукають простоту, швидкість і мінімалістичний дизайн для особистого планування. Натомість Trello забезпечує більше можливостей для візуалізації та командної взаємодії, що є важливим при реалізації багатоступеневих проєктів.

Для розробки власного цифрового планувальника доцільно враховувати найкращі практики цих сервісів, зокрема: інтуїтивність інтерфейсу, гнучкість налаштувань, можливість аналітики. Це дозволить створити конкурентоспроможний продукт, що відповідає сучасним очікуванням користувачів.

2.4 Необхідність та актуальність теми роботи

У сучасних умовах стрімкого розвитку суспільства, цифровізації, постійного зростання обсягів інформації та зобов’язань дедалі більше людей зіштовхуються з труднощами в організації власного часу. Багатозадачність, численні відволікання, розмиті життєві цілі та відсутність чіткої структури дня призводять до втрати фокусу, хронічної втоми й почуття незадоволеності від виконаної роботи. Люди часто відчувають, що день минув, але нічого важливого зроблено не було. Усе це є проявами кризи особистої ефективності, яка потребує дієвих, але не обтяжливих рішень.

Хоч на ринку представлено безліч цифрових інструментів для планування – від простих to-do списків до комплексних систем управління проєктами – вони не завжди відповідають реальним потребам користувачів. Одні інструменти занадто мінімалістичні, не дають змоги систематизувати роботу та відстежувати прогрес. Інші, навпаки, перенасичені функціями: складна навігація, безліч полів для введення, зайві сповіщення – усе це лише посилює відчуття навантаженості замість того, щоб допомогти користувачу впорядкувати свої справи.

У результаті люди витрачають більше часу на налаштування і ведення планувальника, ніж на виконання самих завдань. Такі інструменти не розв’язують головної проблеми – нездатності сконцентруватися на пріоритетах, а лише маскують її під виглядом "псевдоорганізованості".

Саме тому виникає потреба у створенні нового цифрового планувальника, орієнтованого не лише на фіксацію завдань, а й на досягнення цілей. Основна ідея – допомогти користувачам зосередитися на справді важливому, позбутися інформаційного шуму та перетворити щоденні дії на кроки до довгострокових результатів.

Запропонований у цій роботі цифровий планувальник вирізняється такими особливостями:

- фокус на цілях: користувач може одночасно мати не більше ніж 5 активних цілей, що дозволяє зберігати концентрацію на дійсно важливому;
- необмежена кількість завдань до кожної з цілей – це створює гнучку структуру планування, яка дозволяє поступово досягати довгострокових результатів через щоденні кроки;
- відсутність зайвого функціоналу – інтерфейс залишається лаконічним і інтуїтивним;
- статистика успішності відображається за кожною ціллю – як у місячному, так і у річному розрізі, що дозволяє оцінити прогрес та ефективність власних зусиль;
- адаптивність до потреб користувача: підтримка світлої та темної теми, а також двох мов інтерфейсу – української та англійської – робить застосунок зручним для ширшого кола користувачів.

Зростаюча зацікавленість у темі особистої ефективності свідчить про актуальність створення саме такого планувальника. Все більше людей прагнуть впорядкувати своє життя, знизити рівень стресу, відчути більший контроль над часом та досягти внутрішньої гармонії. Водночас прості й зрозумілі інструменти значно легше інтегруються в щоденне користування, що робить їх особливо привабливими для широкого кола користувачів.

Багато користувачів відмовляються від складних систем саме через їхню громіздкість. З іншого боку, паперові щоденники втрачають свою актуальність через обмежену функціональність та відсутність можливості гнучко аналізувати статистику. Саме цифрові рішення з продуманою структурою, але простим і приємним інтерфейсом, мають найбільший потенціал до впровадження в життя широкого кола людей.

Отже, розробка мінімалістичного, але функціонального цифрового планувальника, орієнтованого на досягнення цілей, відповідає запиту сучасного користувача.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Опис матеріалу з теми

Управління завданнями є важливою складовою сучасного особистісного та професійного розвитку. Ефективне планування дозволяє оптимізувати час, знизити рівень стресу та зосередитися на досягненні стратегічних цілей. Саме тому виникає потреба у створенні цифрових інструментів, які допомагають не лише фіксувати завдання, а й управляти фокусом, мотивацією та продуктивністю.

Основним матеріалом для розробки планувальника стали концепції з таких галузей знань:

- **теорія цілепокладання** (goal-setting theory): дослідження Едвіна Локка та Гаррі Латема доводять, що постановка конкретних, обмежених за кількістю і складністю цілей підвищує продуктивність та мотивацію. У планувальнику цю ідею реалізовано через ліміт у 5 активних цілей, що дозволяє уникнути розфокусу та зберегти чіткість у плануванні [6];
- **когнітивна психологія**: численні дослідження вказують, що перевантаження інтерфейсу великою кількістю інформації або функцій спричиняє когнітивне навантаження, що негативно впливає на концентрацію [7]. Саме тому інтерфейс планувальника реалізовано у форматі мінімалізму, де користувач взаємодіє лише з ключовими елементами;
- **методи тайм-менеджменту**: теорії Девіда Аллена (метод GTD), техніка Pomodoro Франческо Чірілло, принцип Парето (80/20), вказують на необхідність структурування задач і фокусування на пріоритетних діях [8]. У розробленому додатку завдання

структуруються навколо цілей, які формують фокус користувача на найважливішому;

- **UX/UI-дизайн**: сучасні принципи дизайну (керованість, контрастність, доступність) вимагають адаптації інтерфейсів до потреб користувачів, включаючи темну та світлу тему, а також багатомовність [9]. У планувальнику реалізовано підтримку української та англійської мов, що розширює аудиторію проєкту;
- **візуальна аналітика**: з точки зору користувацької мотивації важливо мати змогу відстежувати власні досягнення. Тому в планувальнику реалізовано щомісячну та річну статистику, де видно кількість виконаних завдань у розрізі кожної цілі. Це сприяє як самоаналізу, так і покращенню особистої ефективності [10].

Отже, у межах теоретичного дослідження було враховано сучасні підходи до планування, психології продуктивності, користувацького досвіду та аналізу даних. Це дозволило сформувати основу для створення цифрового планувальника, який поєднує простоту, функціональність і гнучкість.

3.2 Опис структури сайту

Сайт-планувальник побудований за принципами мінімалізму, фокусуючись на продуктивності та зручності користувача. Його структура складається з таких основних розділів:

а) головна сторінка сайту:
   1) коротка інформація про проєкт;
   2) кнопка для авторизації користувача;
   3) контактна інформація та посилання на соцмережі або додаткові ресурси;

б) сторінка авторизації:
   1) можливість входу за допомогою електронної пошти або Google-акаунту;

2) форма реєстрації нового користувача;

в) головна сторінка планувальника:

1) можливість створення до 5 активних цілей;
2) додавання необмеженої кількості завдань до кожної цілі;
3) відображення поточних завдань та короткої тижневої статистики виконання;

г) тижневе планування:

1) інтерфейс для планування завдань на поточний тиждень;
2) можливість перетягування завдань між днями тижня (drag-and-drop);
3) кожне завдання пов’язане з конкретною ціллю, що дозволяє зберігати фокус;

д) статистика:

1) доступна щомісячна та річна статистика;
2) виводиться кількість виконаних завдань у розрізі цілей;
3) можна простежити активність користувача, продуктивні дні та прогрес;

е) налаштування:

1) зміна мови інтерфейсу (українська / англійська);
2) перемикання між темною і світлою темами;
3) оновлення електронної пошти, паролю, аватару користувача.

3.3 Алгоритм роботи сайту управління розкладом

Крок 1. Відкриття сайту.

Користувач ознайомлюється з основною інформацією про проєкт на головній сторінці та має змогу перейти до авторизації для подальшого використання планувальника.

Крок 2. Реєстрація або вхід у систему.

Користувач створює обліковий запис або входить за допомогою електронної пошти чи Google-акаунту. Після успішної автентифікації система зчитує відповідні дані та перенаправляє користувача на головну сторінку планувальника.

Крок 3. Створення цілей.

На головній сторінці планувальника користувач може створити до 5 активних цілей, а також за потреби редагувати або видаляти їх.

Крок 4. Додавання та керування завданнями.

До кожної цілі можна додати необмежену кількість завдань, що допомагає детально розпланувати етапи досягнення поставленої мети. Завдання можна редагувати, видаляти або змінювати їх дату виконання за допомогою функціональності drag-and-drop, що забезпечує гнучкість і зручність у їх управлінні. Це дозволяє легко адаптувати план до змінних обставин.

Крок 5. Перегляд статистики.

У бічному меню доступна сторінка статистики, де користувач може переглядати інформацію про свій прогрес. Тут відображається річна та помісячна статистика у форматі "ціль — кількість виконаних завдань", що дозволяє оцінити ефективність роботи і допомагає визначити сильні та слабкі сторони в процесі виконання цілей. Цей розділ мотивує користувача до подальших досягнень і допомагає вчасно коригувати стратегію планування.

Крок 6. Налаштування облікового запису.

Користувач має змогу змінити електронну пошту, пароль, обрати мову інтерфейсу (українська або англійська), а також перемикати світлу або темну тему оформлення сайту.

Крок 7. Ознайомлення з функціоналом.

На сторінці "Як це працює" представлена детальна інформація про особливості користування планувальником, що допомагає швидко адаптуватися до інтерфейсу та функцій системи.

3.4 UML-діаграма

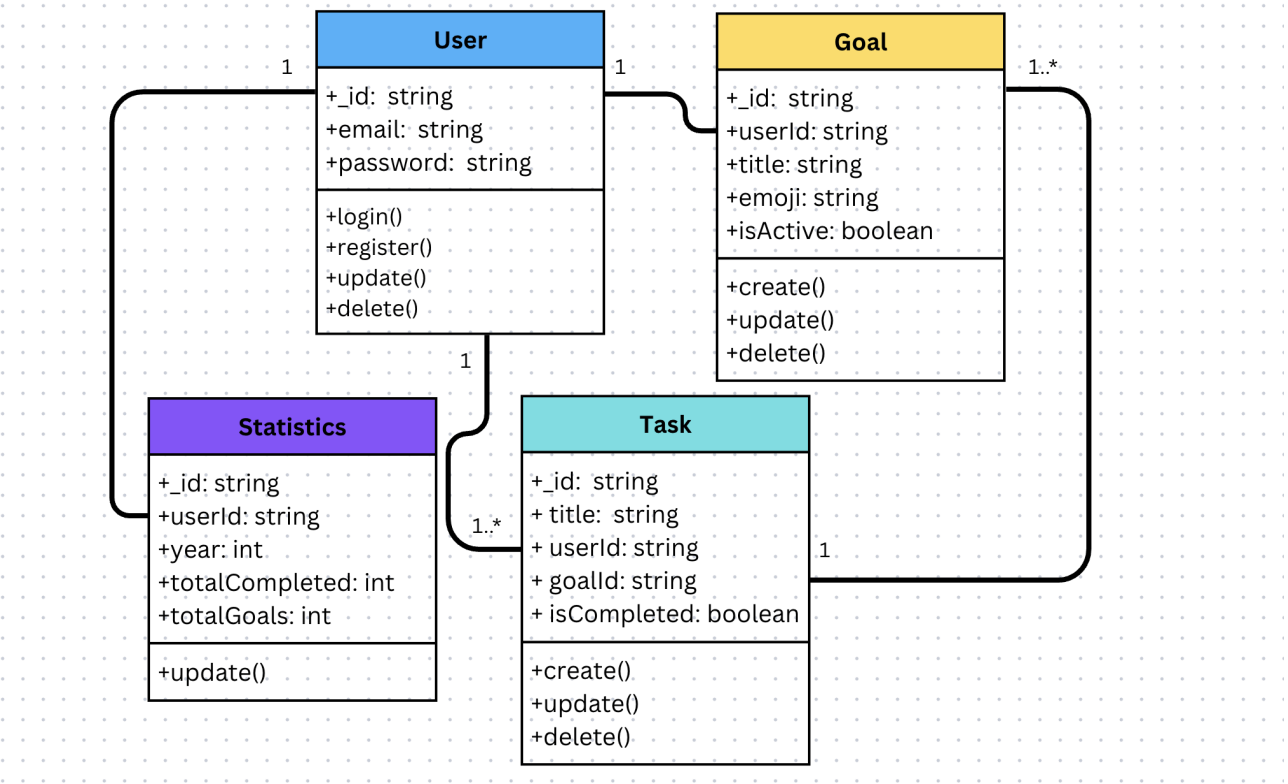

Рисунок 3.1 – UML-діаграма

- зв’язок між User та Goal. Тип зв’язку: 1 → 0..*. Один користувач може створювати багато цілей. Кожна ціль належить лише одному користувачу. Таким чином, між сутностями User і Goal реалізовано зв’язок типу "один до багатьох";
- зв’язок між User та Task. Тип зв’язку: 1 → 0..*. Один користувач може мати багато завдань. Кожне завдання належить одному користувачу. Це відображає зв’язок "один до багатьох" між користувачем та його завданнями;
- зв’язок між User та Statistics. Тип зв’язку: 1 → 0..*. Один користувач може мати декілька записів статистики — за різні роки або періоди відстеження. Таким чином, між User і Statistics існує зв’язок типу "один до багатьох";

- зв’язок між Goal та Task. Тип зв’язку: 1 → 0..* Кожна ціль може включати багато завдань, які сприяють її досягненню. Завдання належить одній конкретній цілі. Це зв’язок "один до багатьох" між Goal і Task.

3.5 Обґрунтування вибору мов програмування

Для створення елементів планувальника щоденних завдань використовуються мови програмування TypeScript [11] та JavaScript [12] разом з фреймворками Next.js і Nest.js. Середовище розробки Visual Studio Code забезпечує зручність написання коду, базу даних MongoDB.

JavaScript та TypeScript як основи фронтенд-розробки. JavaScript – це популярна мова програмування для створення динамічних і інтерактивних веб-сторінок, яка широко використовується у веб-розробці завдяки своїй простоті, гнучкості та кросплатформенності. TypeScript, у свою чергу, розширює можливості JavaScript, додаючи строгий типізований синтаксис, що допомагає розробникам уникати помилок і створювати більш стабільний код (інформація взята з документації TypeScript). Основні характеристики JavaScript і TypeScript**:**

- простота та зручність: завдяки інтуїтивному синтаксису, ці мови є зрозумілими та легкими для вивчення;
- типізація в TypeScript: строгий контроль типів допомагає запобігти помилкам на етапі компіляції, що сприяє створенню надійного та стабільного ПЗ;
- об’єктно-орієнтованість: TypeScript підтримує об’єктно-орієнтоване програмування, що дозволяє створювати модульні й масштабовані програми;
- сумісність з JavaScript: TypeScript компілюється у звичайний JavaScript, що дозволяє використовувати його у всіх сучасних браузерах.

Next.js для створення інтерфейсу планувальника. Next.js – це фреймворк для React, який дозволяє створювати серверний рендеринг і статичні веб-додатки (інформація взята з документації Next.js) [13]. Основні особливості Next.js:

- серверний рендеринг та генерація статичних сторінок: завдяки серверному рендерингу (SSR) та генерації статичних сторінок (SSG), Next.js дозволяє значно підвищити продуктивність додатка, оскільки веб-сторінки завантажуються швидше. Наприклад, сторінки з даними, що часто оновлюються, можуть бути статично згенеровані для покращення швидкості доступу;
- роутинг за замовчуванням: Next.js автоматично створює маршрути на основі структури файлів, що значно спрощує роботу над навігацією в додатку;
- гнучкість та масштабованість: Next.js надає розробникам можливості створювати динамічні й багатофункціональні додатки;
- автоматична оптимізація для SEO: завдяки серверному рендерингу та статичним сторінкам Next.js автоматично оптимізує SEO, що важливо для підвищення видимості планувальника в пошукових системах. Це дозволяє залучати нових користувачів та забезпечує кращу доступність додатку.

Nest.js для бекенд-розробки. Nest.js – це прогресивний фреймворк для Node.js, який використовує TypeScript і дозволяє створювати масштабовані серверні додатки (інформація взята з документації Nest.js) [14]. Основні особливості Nest.js:

- модульна структура: Nest.js забезпечує організацію коду у вигляді модулів, що полегшує роботу над великими проєктами;
- підтримка REST і GraphQL: дозволяє створювати гнучкі API для обміну даними з клієнтською частиною додатка;

- висока продуктивність: Nest.js побудований на основі Express.js, що забезпечує високу швидкість обробки запитів і стабільну роботу додатка.

MongoDB для зберігання данних. MongoDB – це документно-орієнтована база даних NoSQL, яка чудово підходить для зберігання неструктурованих і гнучких даних, таких як списки завдань [15]. Основні переваги MongoDB:

- гнучкість та масштабованість: MongoDB зберігає дані у форматі документів BSON, що дозволяє легко масштабувати дані горизонтально та додавати нові поля без суттєвих змін у структурі бази;
- підтримка JSON-подібного формату: це спрощує інтеграцію з JavaScript, що особливо зручно для роботи з фреймворками на кшталт Next.js та Nest.js;
- висока продуктивність: MongoDB забезпечує швидке зберігання та витяг даних, що важливо для роботи планувальника, де потрібна оперативна обробка інформації про завдання користувача;
- гнучкий підхід до обробки даних: завдяки використанню MongoDB дані про завдання користувачів можуть легко оновлюватися і масштабуватися відповідно до потреб додатка.

Середовище розробки Visual Studio Code. Visual Studio Code (VS Code) [16] – це безкоштовне середовище розробки з відкритим кодом, що забезпечує зручність написання і тестування коду завдяки підтримці численних розширень, інструментів для відлагодження та інтеграції з Git. Основні можливості:

- підтримка розширень: дозволяє інтегрувати додаткові інструменти для роботи з JavaScript, TypeScript, Next.js і Nest.js;
- інструменти для рефакторингу та автозавершення коду**:** полегшують процес розробки, допомагаючи уникати помилок та підвищуючи продуктивність;

- інтеграція з Git**:** спрощує управління версіями, дозволяючи контролювати зміни та робити зручні комміти.

Загальний технологічний стек:

- **TypeScript та JavaScript** – мови для створення інтерфейсу та логіки додатка;
- **Next.js** – фреймворк для розробки інтерфейсу планувальника із зручним роутингом та серверним рендерингом;
- **Nest.js** – бекенд-фреймворк для створення API і обробки даних;
- **MongoDB** – база даних для зберігання інформації про завдання, користувачів та прогрес;
- додаткові бібліотеки для реалізації функціоналу (react-hook-form – для управління формами з високою продуктивністю , yup - бібліотека для схем валідації форм, next-intl – багатомовність і локалізація контенту, @tanstack/react-query – керування асинхронними запитами, кешування, рефетчінг та інші).

4. ПРАКТИЧНА ЧАСТИНА

4.1 Опис розробки програмного забезпечення сайту

Для початку роботи запускаємо середовище розробки Visual Studio Code та одразу створюємо дві папки: planner-be – для реалізації серверної частини проєкту та planner-fe - для реалізації клієнтської частини. Далі приступаємо до побудови архітектури бекенду, використовуючи фреймворк NestJS.

Структура бекенду

Проєкт організовано за модульним принципом. Виділено основні модулі: auth, user, tasks, statistics, goals, date.

Кожен модуль містить три основні частини:

- controller – відповідає за обробку HTTP-запитів (GET, POST, PUT, DELETE) та викликає відповідні методи сервісу;
- service – містить основну бізнес-логіку модуля (створення, оновлення, валідація, зв'язки з базою даних);
- module – слугує як конфігураційний елемент, який реєструє контролери та сервіси модуля та забезпечує їхню взаємодію.

Модуль авторизації auth. Модуль авторизації відповідає за:

- реєстрацію нового користувача;
- вхід у систему (login) з перевіркою облікових даних;
- оновлення та валідацію токенів доступу (JWT);
- інтеграцію з Google OAuth.

Модуль включає:

- Auth Controller – приймає запити від клієнта (/register, /login, /refresh-token, тощо) (код А.1);
- Auth Service – обробляє логіку перевірки користувача, генерації токенів, хешування паролів (код А.2);

- Auth Module – підключає залежності, реєструє потрібні провайдери та імпортує інші модулі (наприклад, UserModule для доступу до користувачів).

Модуль користувача user. Модуль користувача відповідає за:

- отримання інформації про користувача;
- оновлення особистих даних (ім’я, email, аватар, тощо);
- видалення облікового запису.

Модуль включає:

- User Controller – приймає запити від клієнта;
- User Service – обробляє логіку додавання та зміни данних користувача;
- User Module – підключає залежності.

Модуль створення та редагування завдань tasks. Модуль відповідає за створення, оновлення, видалення завдань. Модуль включає:

- Tasks Controller – приймає запити від клієнта для додавання, оновлення завдань (код А.3);
- Tasks Service – обробляє логіку додавання та зміни данних завдань (код А.4);
- Tasks Module – підключає залежності.

Модуль створення та редагування цілей goals. Модуль відповідає за отримання, створення, оновлення, видалення цілей. Модуль включає:

- Goals Controller – приймає запити від клієнта для отримання, додавання, оновлення цілей;
- Goals Service – обробляє логіку отримання цілей, додавання та зміни цілей (код А.5);
- Goals Module – підключає залежності.

Модуль статистики statistics. Модуль відповідає за отримання, створення статистики. Модуль включає:

- Statistics Controller – приймає запити від клієнта для отримання статистики;

- Statistics Service – обробляє логіку отримання, розрахунку та оновлення статистики(код А.6);
- Statistics Module – підключає залежності.

Допоміжний модуль **date** допомагає отримати дати для обчислень (код А.7).

Загальні принципи:

- кожен модуль ізольований, відповідає за одну область відповідальності (SRP);
- взаємодія між модулями відбувається через сервіси, які експортуються через @Module({ exports: [...] });
- дані зберігаються в MongoDB, моделі створюються за допомогою Typegoose схем;
- усі змінні оточення зберігаються у файлі .env.

Також для забезпечення стабільності, безпеки й зрозумілості API в проєкті реалізовані додаткові технічні шари: DTO, валідація та guard-и.

Реалізація клієнтської частини

Після встановлення фреймворку Next.js створюємо архітектуру проєкту, орієнтуючись на модульність та масштабованість, а також встановлюємо необхідні пакети для роботи з маршрутизацією, стилями, станом, автентифікацією, запитами до API та іншими функціями, які забезпечують повноцінну роботу клієнтської частини.

Проєктна структура реалізована за принципами features-based архітектури, яка забезпечує логічну організацію коду, покращену модульність та легкість у підтримці й масштабуванні застосунку. Цей підхід передбачає поділ логіки не за технічними аспектами (наприклад, компоненти, стилі, утиліти), а за функціональними одиницями або «фічами» – тобто окремими можливостями застосунку. Кожна така фіча розміщується в окремій директорії, яка містить усі пов’язані з нею файли: компоненти інтерфейсу, стилі, типи, сервіси, API-запити, локальний стан тощо. Це дозволяє уникнути тісного зв’язку між модулями,

спрощує тестування, повторне використання коду та забезпечує прозору структуру проєкту.

У даному застосунку уся логіка розділена на чітко визначені функціональні блоки, кожен з яких має відповідність у backend-частині системи. Основними модулями є:

- auth – відповідальний за автентифікацію користувача, реєстрацію, вхід, вихід та обробку токенів;
- user – зберігає та керує даними профілю користувача, такими як email, мова інтерфейсу, фото тощо;
- tasks – відповідає за створення, редагування, видалення та переміщення завдань у межах обраних днів чи цілей;
- goals – реалізує логіку додавання та контролю за особистими цілями користувача, включно з лімітом на кількість активних цілей;
- statistics – формує статистику успішності виконаних завдань за цілями та періодами (місяць, рік), дозволяючи оцінити прогрес у досягненні цілей.

Оскільки застосунок передбачає роботу з авторизованими користувачами, важливо забезпечити захищену та стабільну взаємодію між frontend і backend частинами. Для цього реалізовано декілька ключових механізмів:

- interceptor – спеціальний обробник HTTP-запитів, який автоматично додає access-токени до кожного запиту користувача. У разі їхнього закінчення interceptor виконує оновлення токена (refresh) без участі користувача, що покращує користувацький досвід і підвищує безпеку (код А.8);
- middleware у Next.js – цей серверний модуль інтегровано у frontend-частину та використовується для попередньої перевірки запитів до сторінок, що вимагають автентифікації. Middleware аналізує HTTP cookie, в яких зберігається токен доступу, та вирішує, чи дозволяти доступ до ресурсу. У разі відсутності або недійсного токена користувач перенаправляється на сторінку входу (код А.9).

Завдяки такій архітектурі забезпечується не лише безпека, а й узгодженість у роботі між усіма частинами застосунку – як на рівні даних, так і з точки зору користувацького інтерфейсу. Розглянемо детально кожен модуль.

Модуль авторизації відіграє ключову роль у забезпеченні безпечного доступу користувача до функціоналу платформи. Він охоплює реалізацію форм входу та реєстрації (код А.10), логіку отримання токенів, а також завантаження профілю автентифікованого користувача (код А.11), що є необхідним для коректної роботи персоналізованих розділів сайту. Наявність авторизації дозволяє розмежувати загальнодоступний та захищений функціонал. Для реалізації цієї логіки у фронтенді створено компонент **AuthProvider** (код А.12), який відповідає за збереження стану автентифікації та передачу відповідного контексту в усі частини застосунку, де це необхідно. Таким чином, система може динамічно реагувати на наявність або відсутність авторизованого користувача, змінюючи доступ до окремих сторінок або компонентів.

Особливу увагу в реалізації модуля приділено валідації форм. Кожна форма входу та реєстрації містить логіку перевірки коректності введених даних ще до їх надсилання на сервер (код А.13). Це дозволяє зменшити кількість некоректних запитів, знизити навантаження на сервер і, що найважливіше, забезпечити користувачеві зворотний зв’язок у реальному часі. Завдяки валідації можна уникнути поширених помилок (наприклад, неправильний формат електронної пошти або слабкий пароль), підвищити довіру до сервісу та покращити загальну якість взаємодії користувача із застосунком. Валідація є важливим елементом безпеки й зручності, адже вона забезпечує початковий рівень контролю над даними, які потрапляють у систему, і допомагає уникнути потенційних вразливостей або помилок у подальшій роботі.

Модуль планувальника (код А.14) є центральним функціональним елементом застосунку, оскільки саме він забезпечує реалізацію ключових можливостей управління особистою ефективністю користувача. Основною метою цього модуля є організація щоденної активності через постановку цілей, додавання конкретних завдань до них і відстеження результатів у вигляді

статистики. У його межах реалізована повноцінна підтримка CRUD-операцій для цілей та завдань: користувач може додавати нові цілі, редагувати наявні, видаляти ті, що вже втратили актуальність, а також змінювати структуру пов’язаних із ними завдань (код А.15), переглядати тижневу статистику (код А.16).

Для підвищення наочності та зручності використання модуль планувальника включає сервіси, що відповідають за візуалізацію щотижневої активності користувача. Зокрема, компонент активних тижнів (код А.17) дає змогу переглядати поточний тиждень і автоматично відкриває доступ до наступного тижня щонеділі, що створює безперервний цикл планування без необхідності ручного оновлення. Це дозволяє користувачу залишатися у постійному робочому ритмі, своєчасно переходити до планування нових завдань та уникати застою.

Компонент виводу завдань (код А.18) адаптовано до структури тижня та обраної цілі, і він відображає актуальні завдання на кожен день, згруповані відповідно до поставлених цілей. Такий підхід забезпечує швидкий доступ до пріоритетної інформації, дозволяючи користувачеві чітко бачити, які саме завдання заплановано на сьогодні, і які з них пов’язані з довгостроковими намірами.

Ще однією важливою особливістю модуля є підтримка drag-and-drop взаємодії (код А.19), яка значно підвищує гнучкість у плануванні. Завдяки цій системі користувачі можуть інтуїтивно перетягувати завдання між днями тижня або цілями, що створює відчуття живого, гнучкого плану, який легко адаптувати до реальних обставин. Впровадження такої інтерактивної взаємодії робить застосунок не лише зручнішим, а й мотивує користувача активніше взаємодіяти зі своїми планами, відчуваючи більший контроль над щоденними діями.

Загалом, модуль планувальника поєднує в собі логіку, гнучкість і зручність візуального подання інформації, що є ключовим чинником для досягнення користувачем своїх довгострокових цілей через щоденну дисципліну та усвідомлене планування.

Модуль статистики реалізує важливу функціональність збору (код А.20) та обробки даних про виконання завдань користувача. Інформація групується за роками та місяцями, що дозволяє простежити динаміку досягнення цілей у довгостроковій перспективі, виявити періоди більшої чи меншої активності та проаналізувати загальний рівень особистої ефективності. Оброблені результати подаються у зручному для користувача форматі через інтерфейс (код А.21), де одразу видно, скільки кроків було виконано в межах тієї чи іншої цілі. Це сприяє кращому візуальному сприйняттю власного прогресу. Такий підхід не лише допомагає сформувати більш усвідомлену стратегію досягнення результатів, а й підтримує мотивацію завдяки наочному підтвердженню поступу та відчуттю реального впливу щоденних зусиль.

Модуль налаштувань забезпечує зручний і безпечний доступ до управління особистими даними користувача. Він включає функції для оновлення основної інформації профілю – зокрема, імені, електронної пошти, аватару та паролю. Крім того, модуль передбачає можливість видалення облікового запису, що гарантує повний контроль користувача над власними даними. Усі форми в модулі налаштувань супроводжуються валідацією введених даних, що підвищує надійність роботи системи та запобігає введенню некоректної інформації. Завдяки цьому інтерфейс налаштувань залишається простим і водночас функціональним, відповідаючи базовим вимогам безпеки й користувацького комфорту.

Уся логіка, яка може використовуватись у різних частинах проєкту, згрупована в папці shared. Вона включає:

- константи – загальні значення для API-запитів (код А.22), маршрутизації (код А.23) та інших конфігурацій, що використовуються в усьому проєкті;
- компоненти – повторно використовувані елементи інтерфейсу, такі як кнопки (код А.24), поля вводу (код А.25), випадаючі списки (код А.26), елементи вибору (код А.27), а також загальні частини макету, як Header (код А.28) та Footer;

- утіліти – допоміжні функції, наприклад, для перевірки шляху у роутингу (код А.29), обробки помилок, що надходять із сервера, або форматування даних;
- hooks – кастомні React-хуки, зокрема, для обробки кліків за межами елементів (код А.30);
- іконки – набір react-icons, які використовуються в інтерфейсі для підвищення зручності та візуальної ідентифікації дій.

Окрім цього, у проєкті реалізовано багатомовність інтерфейсу через систему перекладів, що дозволяє адаптувати застосунок для користувачів з різних мовних середовищ. Повідомлення про успішні або помилкові дії виводяться через модуль сповіщень, що забезпечує зворотний зв’язок у взаємодії з користувачем. Для уніфікації зовнішнього вигляду застосунку та забезпечення консистентності дизайну створено окремі файли зі стилями, які охоплюють як глобальні налаштування, так і стилі для окремих компонентів.

4.2 Опис роботи сайту для управління розкладом

Розроблений сайт дозволяє користувачеві створювати, переглядати та редагувати свій щоденний розклад із фокусом до 5 цілей. Користувач взаємодіє з інтуїтивним інтерфейсом, у якому всі елементи взаємопов’язані: завдання можна створювати, сортувати, редагувати, перетягувати, позначати як виконані та зв’язувати з довгостроковими цілями.

Після переходу на сайт користувач потрапляє на головну сторінку планувальника (рис 4.1).

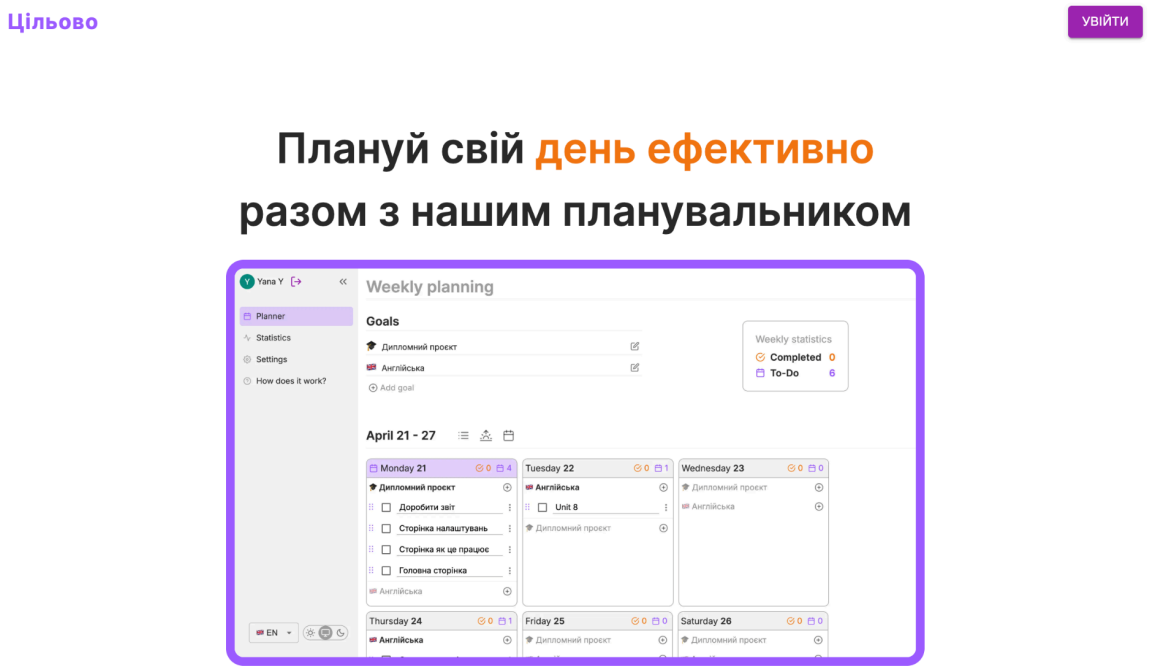

Рисунок 4.1 – Головна сторінка планувальника

Сайт підтримує дві мови – українську та англійську. Мова інтерфейсу визначається автоматично відповідно до локалізації браузера, але користувач завжди може змінити її вручну (рис. 4.2).

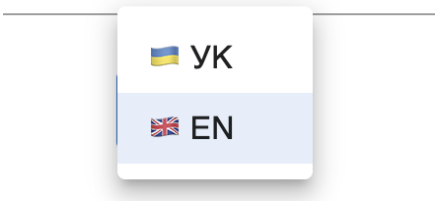

Рисунок 4.2. – Вибір мови

Також користувач може обрати світлу або темну тему оформлення. За замовчуванням застосовується системна тема, але її можна змінити вручну в налаштуваннях (рис. 4.3).

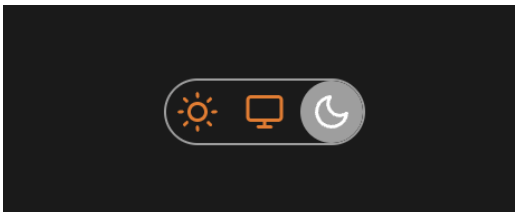

Рисунок 4.3. – Вибір теми

Щоб почати планування, необхідно увійти або зареєструватися в системі. Це можна зробити за допомогою пароля або через Google-акаунт (рис. 4.4). Обидві форми містять валідацію, яка перевіряє правильність введених даних. (рис. 4.5).

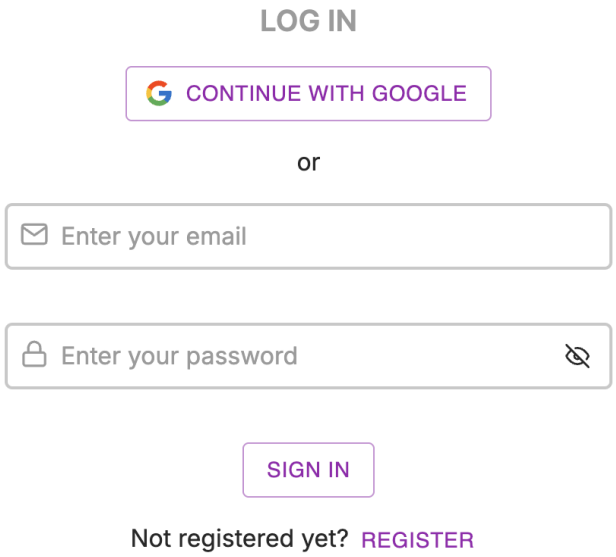

Рисунок 4.4 – Сторінка входу

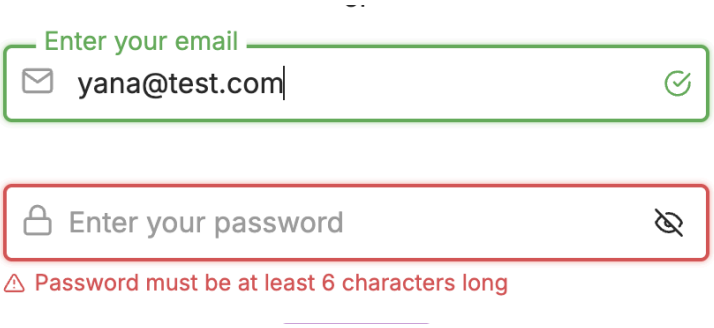

Рисунок 4.5 – Валідація введних даних

Ось так виглядає авторизація через Google (рис. 4.6)

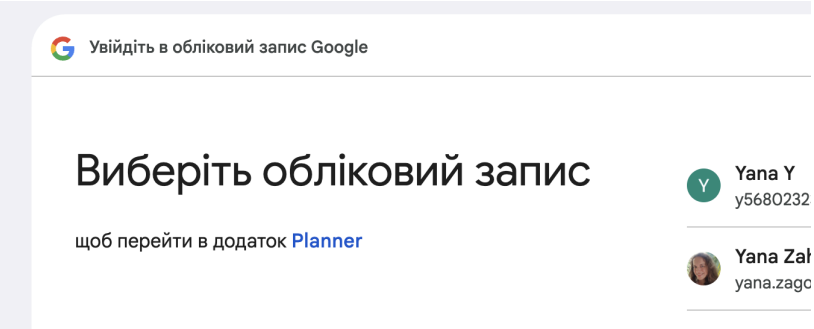

Рисунок 4.6 – Google авторизація

Після успішної авторизації або реєстрації користувач перенаправляється на головну сторінку планувальника (рис. 4.7), де за допомогою бокового меню можна перейти до потрібного розділу, змінити мову та тему, а також згорнути меню при необхідності (рис. 4.8).

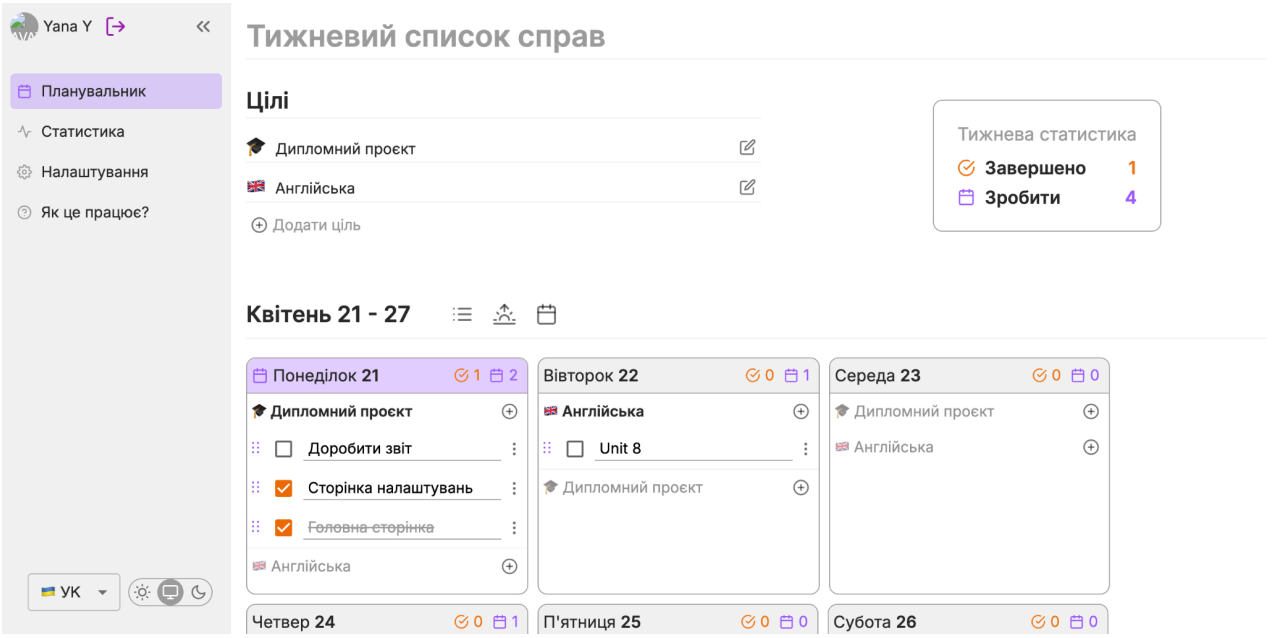

Рисунок 4.7 – Головна сторінка планувальника

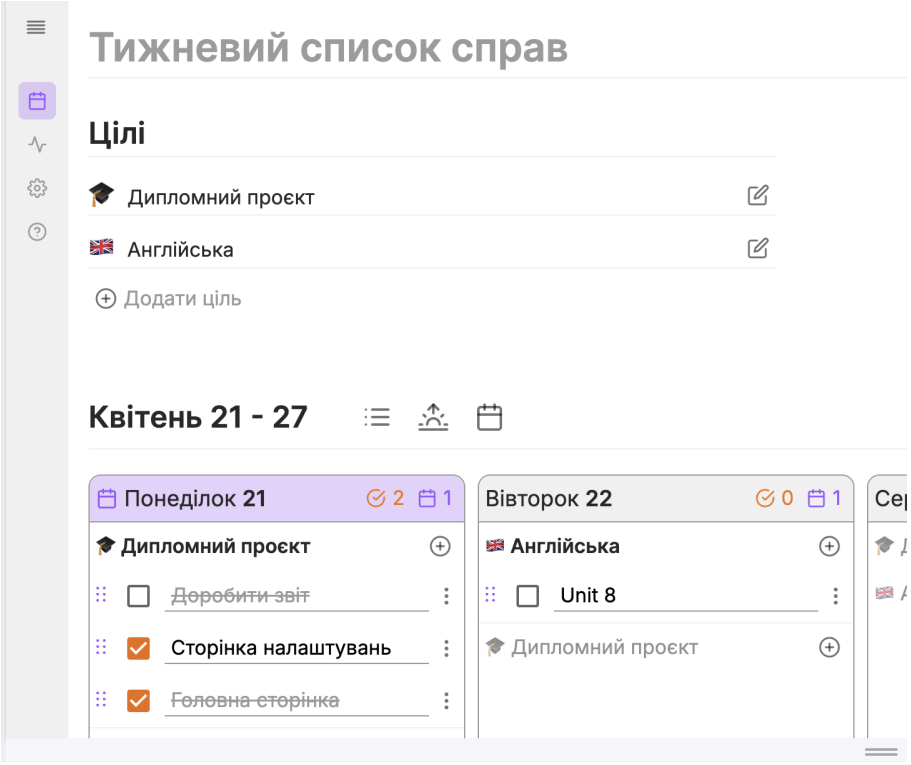

Рисунок 4.8 – Згорнуте бокове меню

Перед додаванням завдань необхідно визначити цілі, а потім приступати до планування. Користувач може працювати з до 5 цілями одночасно. Цілі можна редагувати (рис. 4.9) або видаляти, але ціль з активними завданнями не можна видалити, це буде показано в повідомленні (рис. 4.10).

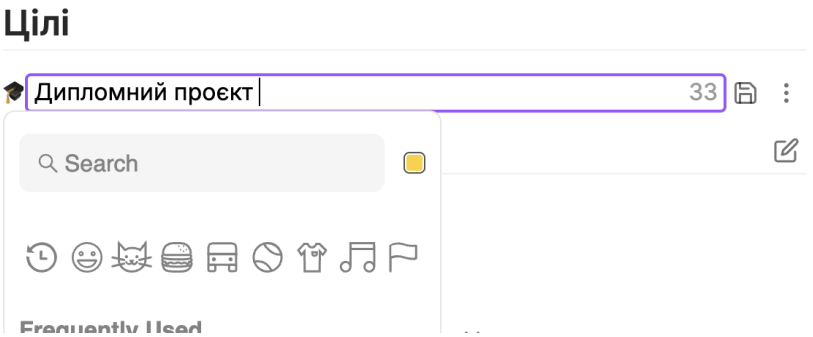

Рисунок 4.9 – Редагування цілі

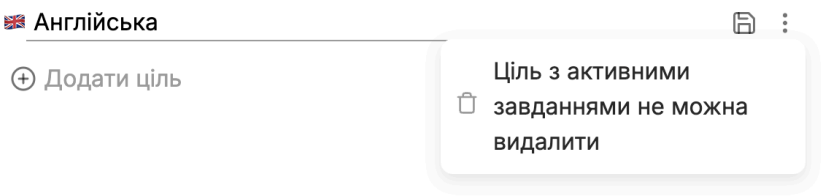

Рисунок 4.10 – Ціль з активними завданнями

Поряд із цілями користувач може переглянути статистику за тиждень (рис. 4.11).

Рисунок 4.11 – Тижнева статистика

Завдання можна додавати на поточний тиждень, а щонеділі відкривається можливість планування завдань на наступний тиждень. Завдання за замовченням відображаються у вигляді блоків, можна переглядати у вигляді списку (рис. 4.12), є можливість спершу показати дні з наявними завданнями. Також можна обрати лише поточний день, заголовок якого підсвічується.

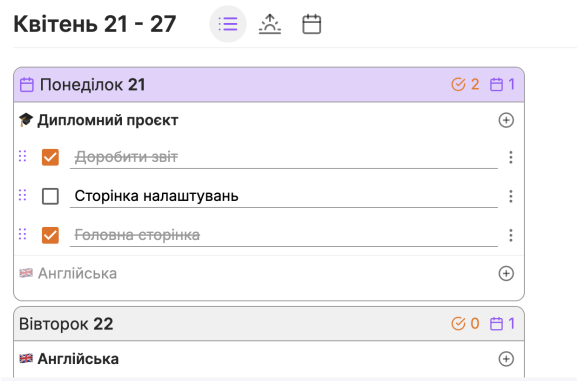

Рисунок 4.12 – Завдання у вигляді списку

Після додавання цілей вони автоматично відображаються в кожному дні тижня. До кожної цілі можна додавати завдання, редагувати їх, видаляти, а також перетягувати між днями за допомогою системи drag and drop. У правому верхньому куті відображається денна статистика (рис. 4.13).

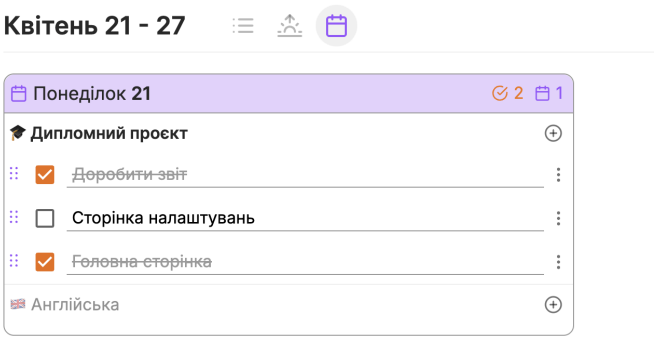

Рисунок 4.13 – Планування на день

При переході на сторінку статистики користувач може ознайомитися зі статистикою за рік. Із випадаючого меню можна обрати попередні роки. Також відображається статистика по місяцях, де видно перелік цілей та кількість виконаних завдань для кожної з них (рис. 4.14). Якщо користувач ще не виконав ніяких завдань, він побачить повідомлення про відсутність статистики (рис. 4.15).

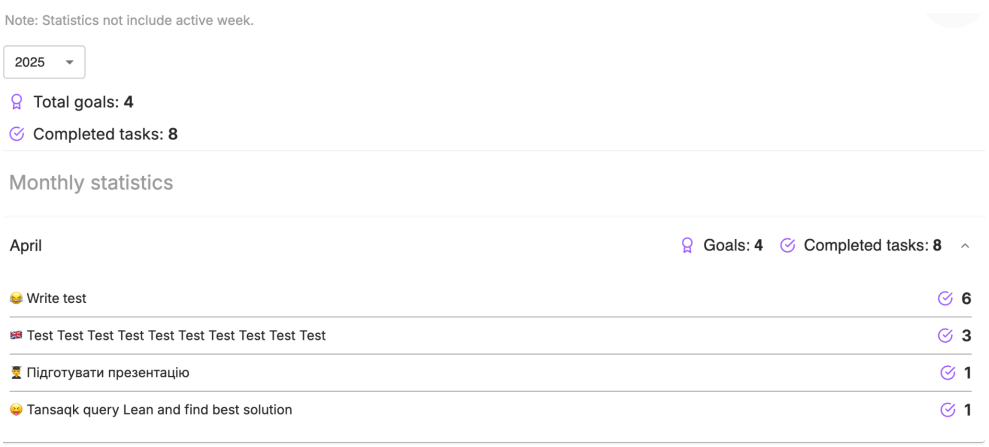

Рисунок 4.14 – Сторінка статистики

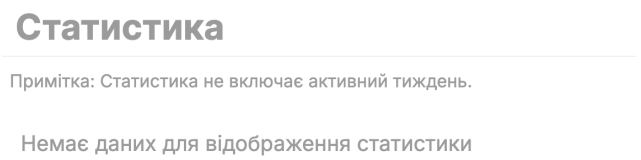

Рисунок 4.15 – Повідомлення про відсутність статистики

На сторінці налаштувань, можливо змінити персональні дані (рис. 4.16).

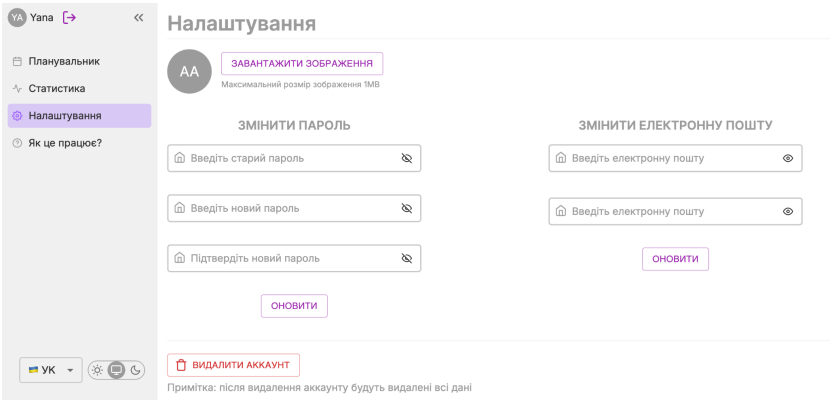

Рисунок 4.16. – Повідомлення про відсутність статистики

На сторінці «Як це працює» можна ознайомитися з ідеєю планувальника (рис. 4.17).

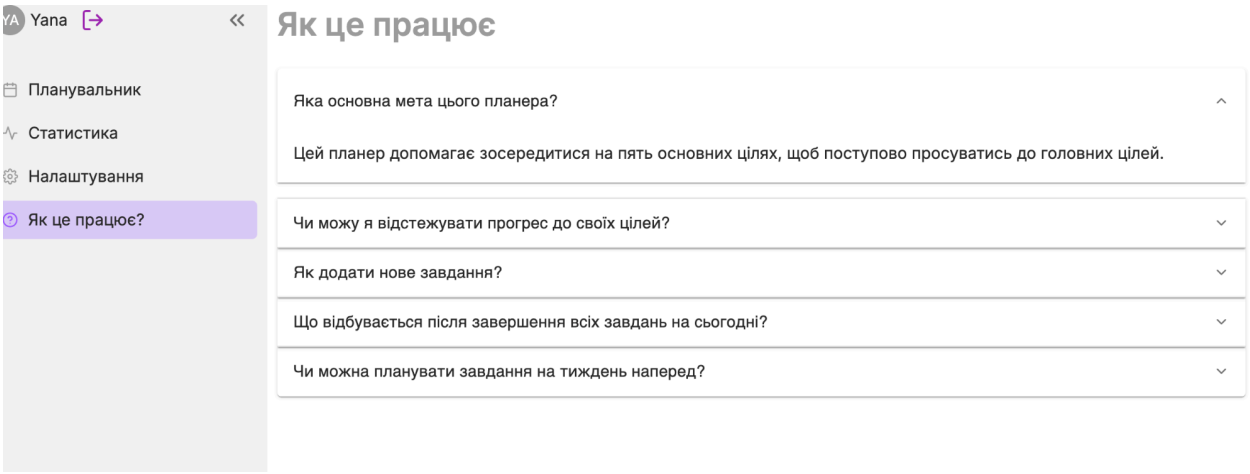

Рисунок 4.17 – Сторінка як це працює

При виході з планувальника за допомогою кнопки «Вихід» завершується поточна сесія, і користувач автоматично перенаправляється на головну сторінку (рис. 4.18).

Рисунок 4.18 – Кнопка вихід

Після виходу дані залишаються збереженими, і при наступному вході відновлюється поточний стан планувальника.

4.3 Аналіз роботи планувальника на основі тестового використання

У межах практичного тестування веб-додатку було проведено експериментальне використання планувальника протягом 10 тижнів для організації особистих завдань і досягнення конкретних цілей. Метою тестування було перевірити, наскільки обмеження кількості активних цілей і система підрахунку виконаних завдань сприяють зростанню продуктивності користувача.

Під час тестового періоду було встановлено 5 активних цілей:

- написати всі тести та контрольні з англійської мови;
- завершити дипломну роботу;
- виконувати завдання з програми навчання (інших навчальних дисциплін);
- покращити фізичну форму (тренування 3 рази на тиждень);
- прочитати 2 книги з самоорганізації.

До кожної з цілей додавались завдання у вигляді конкретних дій. Наприклад, до цілі «Дипломна робота» — «написати розділ 4», «зробити UML-діаграму», «оформити список джерел» тощо.

За час тестування фіксувалась кількість виконаних завдань щотижня. На графіку нижче видно поступове зростання активності, що свідчить про формування звички до щоденного планування (рис. 4.19).

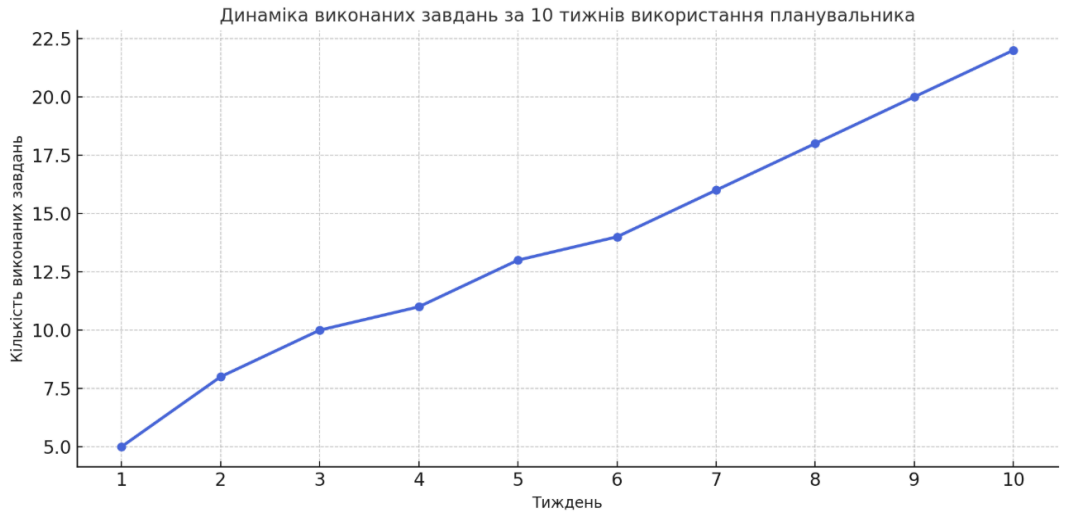

Рисунок 4.19 – Графік активності

Кількість завершених завдань зросла з 5 у першому тижні до 22 у десятому. Особливо помітним було зростання після впровадження додаткового підкріплення – візуалізації прогресу по цілях.

Висновки з тестування:

- обмеження до 5 активних цілей виявилось оптимальним: при більшій кількості концентрація розпорошується, а менша кількість не охоплює всі сфери життя;
- усі 5 цілей поступово наближались до завершення завдяки постійній роботі з ними, особливо завдяки прив’язці щоденних завдань до однієї з них;
- ведення статистики та акцент на вже виконаних завданнях підвищували мотивацію, знижували почуття провини за незроблене та сприяли регулярному поверненню до планувальника;

- цифрова система допомогла краще усвідомлювати власний темп роботи та уникати вигорання за рахунок фокусування на поступовому прогресі.

Ці результати підтверджують гіпотезу, що обмеження активних цілей та відображення досягнень користувача позитивно впливають на рівень продуктивності та емоційний стан.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно реалізовано головну мету – розробку програмного забезпечення на тему «Розробка сайту для управління своїм розкладом та завданнями». У процесі виконання було досягнуто наступних основних результатів:

- проведено аналіз переваг і недоліків сучасних цифрових планувальників;
- здійснено огляд існуючих аналогів, виявлено сильні сторони та основні труднощі їх використання;
- розроблено та описано структуру вебзастосунку, включно з архітектурою фронтенду та бекенду;
- побудовано алгоритм роботи сайту;
- створено UML-діаграму, що відображає логіку системи;
- обґрунтовано вибір технологій і мов програмування, зокрема Next.js, NestJS, TypeScript і MongoDB;
- детально описано процеси розробки, включаючи створення модулів, сервісів, контролерів, структуру DTO, валідацію, захист маршрутів;
- представлено функціональні можливості застосунку, а також особливості роботи з планувальником з точки зору користувача.

З наукової точки зору, проведене дослідження засобів особистого планування дозволило не тільки розробити практичний інструмент, але й виявити ефективні підходи до органічної інтеграції ключових когнітивних та мотиваційних аспектів людської діяльності у цифровому середовищі. Зокрема, впровадження обмеження на кількість активних цілей, що базується на принципах теорії когнітивного навантаження та необхідності фокусування уваги, прямо корелює з сучасними дослідженнями у галузі психології продуктивності та ефективного самоменеджменту.

З соціальної точки зору, створений планувальник може сприяти підвищенню загального рівня продуктивності серед користувачів, покращуючи їх здатність до саморегуляції та ефективного управління своїм часом. Це важливий фактор, особливо в умовах сучасного світу, де люди часто стикаються з переповненням завдань і інформації. Системи планування, які допомагають людям ефективно організовувати свій день, мають величезну соціальну цінність, адже вони можуть допомогти зменшити стрес і покращити якість життя, а також підвищити рівень досягнень у професійній і особистій сферах. Також, завдяки функціоналу планувальника, користувачі можуть розвивати свої довгострокові цілі, що є корисним інструментом для планування кар'єрного росту, покращення здоров'я або навіть організації особистих відносин. Це підвищує соціальну відповідальність користувачів за своє життя та дає можливість відстежувати прогрес.

Розроблений планувальник має широкі можливості застосування в різних сферах: в освіті – для організації навчального процесу, підготовки до сесій та захисту наукових робіт; у сфері особистого розвитку – для досягнення індивідуальних цілей, таких як вивчення мов, заняття спортом чи розвиток хобі; у фрілансі та віддаленій роботі – для структурованого управління завданнями, дотримання дедлайнів і підвищення продуктивності; у терапевтичній і коучинговій практиці – для постановки цілей.

Подальший розвиток планувальника бачиться у тісній взаємодії зі зворотним зв'язком від реальних користувачів. На основі їхніх відгуків та пропозицій планується розширення функціоналу додатка, зокрема, впровадження більш гнучких можливостей для налаштування та кастомізації завдань і цілей, додавання нових типів візуалізації прогресу, інтеграція з календарями та іншими сервісами, а також розробка функцій для спільного планування та роботи над проєктами. Це дозволить перетворити розроблений планувальник з ефективного інструменту особистої продуктивності на комплексне рішення для управління часом та досягнення цілей у найрізноманітніших сценаріях використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маслюківська , А. (2018). ЗНАЧЕННЯ ТАЙМ-МЕНЕДЖМЕНТУ ДЛЯ ПІДВИЩЕННЯ ПЕРСОНАЛЬНОЇ ЕФЕКТИВНОСТІ . *Молодий вчений*, 11 (63), 467-471. вилучено із URL: https://www.molodyivchenyi.ua/index.php/journal/article/view/3513 (дата звернення: 29.01.2025)

2. Жуковська , А. (2023). МЕТОДИ ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАВДАНЬ У ТАЙМ-МЕНЕДЖМЕНТІ. Socio-Economic Relations in the Digital Society, 1(47), 50-60. URL: https://doi.org/10.55643/ser.1.47.2023.478

3. Peter A. Heslin, Jay B. Carson, Don Vandewalle Practical Applications of Goal Setting Theory to Performance Management URL: https://www.researchgate.net/publication/228211042_Practical_Applications_of_Goal_Setting_Theory_to_Performance_Management (дата звернення: 30.01.2025).

4. Atlassian компанія. Trello – візуальний інструмент для управління завданнями та проєктами. URL: https://trello.com.

5. Doist компанія. Todoist – це популярний додаток для управління завданнями. URL: https://todoist.com.

6. Locke, E. A., & Latham, G. P. (2002). Building a practically useful theory of goal setting and task motivation: A 35-year odyssey. *American Psychologist, 57*(9), 705–717. URL: https://doi.org/10.1037/0003-066X.57.9.705

7. Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*(2), 257–285. URL: https://doi.org/10.1207/s15516709cog1202_4

8. Allen, D. (2002). *Getting Things Done: The Art of Stress-Free Productivity*. Penguin Books. Publisher: Penguin Books. 288 p

9. Norman, D. A. (2013). *The Design of Everyday Things*. MIT Press. Publisher: The MIT Press. 268 p.

10. Few, S. (2006). *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media. Publisher: O'Reilly. 224 p

11. Microsoft компанія. Typescript – мова програмування на основі JavaScript. URL: https://www.typescriptlang.org/docs/.

12. Ерік Фрімен, Елізабет Робсон Програмування на JavaScript. Head First. Легкий для сприйняття довідник. Харків: Фабула, 2022, 672с.

13. Vercel компанія. Next.js – відкритий JavaScript фреймворк. URL: https://nextjs.org/docs.

14. Kamil Myśliwiec. Nest JS – сучасний фреймворк для створення ефективних та масштабованих серверних програм на Node.js. URL: https://docs.nestjs.com/.

15. Ranjeet Jha, MongoDB Essentials: Build Faster, Perform Better, Store Smarter, Scale Bigger, and Master Document Database, 2025, 380 p.

16. Microsoft компанія. Visual Studio Code (VS Code) – відкритий редактор коду. URL: https://code.visualstudio.com.

17. Методичні рекомендації до виконання кваліфікаційної роботи студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава: ПУЕТ, 2025. – 58 с.

ДОДАТОК А

Код А.1 – auth.controller.ts

```
import {
  Body,
  Controller,
  Get,
  Post,
  Req,
  Res,
  UseGuards,
} from '@nestjs/common';
import { UnauthorizedException } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthDto, RegistrationDto } from 'src/typing/dto';
import { AuthGuard } from '@nestjs/passport';
import { Request, Response } from 'express';
import { clearAuthCookies, setAuthCookies } from './helpers/auth';
import { ConfigService } from '@nestjs/config';
import { Throttle } from '@nestjs/throttler';
@Controller('auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly configService: ConfigService,
  ) {}
  @Throttle({ default: { limit: 3, ttl: 60000 } })
  @Post('login')
  async login(@Body() dto: AuthDto, @Res({ passthrough: true }) res: Response) {
    const { accessToken, refreshToken, id, name, email, image } =
     await this.authService.login(dto);
    setAuthCookies(res, accessToken, refreshToken);
    return {
      message: 'Login successful',
      user: { id, name, email, image },
    };
  }
  @Throttle({ default: { limit: 5, ttl: 60000 } })
  @Post('register')
  async register(
    @Body() dto: RegistrationDto,
    @Res({ passthrough: true }) res: Response,
  ) {
```

```
  const { accessToken, refreshToken, id, name, email, image } =
    await this.authService.register(dto);
  setAuthCookies(res, accessToken, refreshToken);
  return {
    message: 'Registration successful',
    user: { id, name, email, image },
  };
}
@Post('logout')
logout(@Res({ passthrough: true }) res: Response) {
  clearAuthCookies(res);
  return { message: 'Logged out successfully' };
}
@Throttle({ default: { limit: 3, ttl: 60000 } })
@Post('access-token')
async refresh(
  @Req() req: Request,
  @Res({ passthrough: true }) res: Response,
) {
  try {
    const refreshToken = req.cookies?.refreshToken;
    if (!refreshToken) {
      clearAuthCookies(res);
      throw new UnauthorizedException('Missing refresh token');
    }
    const {
      accessToken,
      refreshToken: newRefreshToken,
      id,
      name,
      email,
      image,
    } = await this.authService.getNewTokens(refreshToken);
    setAuthCookies(res, accessToken, newRefreshToken);
    return {
      user: { id, name, email, image },
    };
  } catch (err) {
    clearAuthCookies(res);
    console.error('Token refresh failed:', err?.message || err);
    throw new UnauthorizedException('Refresh token invalid or expired');
  }
}
@Get('google')
@UseGuards(AuthGuard('google'))
```

```
  async googleAuth() {}
  @Get('google/callback')
  @UseGuards(AuthGuard('google'))
  async googleCallback(@Req() req: Request, @Res() res: Response) {
    const locale = req.cookies?.NEXT_LOCALE ?? 'en';
    try {
      const { accessToken, refreshToken } = await this.authService.googleLogin(
        req.user,
      );
      setAuthCookies(res, accessToken, refreshToken);
      return res.redirect(
        `${this.configService.get('FRONTEND_URL')}/${locale}/planner`,
      );
    } catch (err) {
      return res.redirect(

`${this.configService.get('FRONTEND_URL')}/${locale}/login?error=oauth_failed`,
      );
    }
  }
}
```

Код А.2 – auth.service.ts

```
import {
  BadRequestException,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { AuthDto, RegistrationDto } from 'src/typing/dto';
import { hash, genSalt, compare } from 'bcryptjs';
import { JwtService } from '@nestjs/jwt';
import { UserService } from '../user/user.service';
import { JwtPayload } from './types/jwt-payload.type';
import { ConfigService } from '@nestjs/config';
@Injectable()
export class AuthService {
  constructor(
    private readonly configService: ConfigService,
    private readonly jwtService: JwtService,
    private readonly userService: UserService,
  ) {}
  async login(loginDto: AuthDto) {
    const user = await this.userService.findByEmail(loginDto.email);
    if (!user) {
```

```
    throw new UnauthorizedException("User with this email doesn't exist");
  }
  const isPasswordValid = await compare(loginDto.password, user.password);
  if (!isPasswordValid) {
    throw new UnauthorizedException('Wrong password');
  }
  const tokens = await this.createTokenPair(user.id);
  return {
    id: user.id,
    name: user.name,
    email: user.email,
    image: user.image,
    accessToken: tokens.accessToken,
    refreshToken: tokens.refreshToken,
  };
}
async register(registrationDto: RegistrationDto) {
  const existingUser = await this.userService.findByEmail(
    registrationDto.email,
  );
  if (existingUser) {
    throw new BadRequestException('User with this email already exists');
  }

  const salt = await genSalt(10);
  const hashedPassword = await hash(registrationDto.password, salt);
  const newUser = await this.userService.create({
    ...registrationDto,
    password: hashedPassword,
  });
  const tokens = await this.createTokenPair(newUser.id);
  return {
    id: newUser.id,
    name: newUser.name,
    email: newUser.email,
    image: newUser.image,
    accessToken: tokens.accessToken,
    refreshToken: tokens.refreshToken,
  };
}
async getNewTokens(refreshToken: string) {
  if (!refreshToken) {
    throw new UnauthorizedException('Refresh token is missing');
  }
  let payload: JwtPayload;
```

```
  try {
    payload = await this.jwtService.verifyAsync<JwtPayload>(refreshToken, {
      secret: this.configService.get<string>('JWT_SECRET'),
    });
  } catch (err) {
    throw new UnauthorizedException('Invalid or expired refresh token');
  }
  const user = await this.userService.getByID(payload.id);
  if (!user) {
    throw new UnauthorizedException('User not found');
  }
  const tokens = await this.createTokenPair(user.id);
  if (!tokens.accessToken || !tokens.refreshToken) {
    throw new UnauthorizedException('Failed to create new tokens');
  }
  return {
    id: user.id,
    name: user.name,
    email: user.email,
    image: user.image,
    accessToken: tokens.accessToken,
    refreshToken: tokens.refreshToken,
  };
}
async googleLogin(userData: any) {
  let user = await this.userService.findByEmail(userData.email);
  if (!user) {
    user = await this.userService.create({
      name: userData.name || 'No Name',
      email: userData.email,
      password: userData.password,
      image: userData.picture,
      provider: 'google',
    });
  }
  const tokens = await this.createTokenPair(user.id);
  return {
    id: user.id,
    name: user.name,
    email: user.email,
    image: user.image,
    accessToken: tokens.accessToken,
    refreshToken: tokens.refreshToken,
  };
}
```

```
  private async createTokenPair(userId: string) {
    if (!userId) {
      throw new UnauthorizedException('User ID is required for token creation');
    }
    const payload = { id: userId };
    const [refreshToken, accessToken] = await Promise.all([
      this.jwtService.signAsync(payload, {
        expiresIn: '21d',
      }),
      this.jwtService.signAsync(payload, {
        expiresIn: '10h',
      }),
    ]);
    if (!refreshToken || !accessToken) {
      throw new UnauthorizedException('Failed to create tokens');
    }
    return { refreshToken, accessToken };
  }
}
```

Код А.3 – tasks.controller.ts

```
import { Controller, Body, Put, Delete, Post } from '@nestjs/common';
import { TaskService } from './tasks.service';
import { Auth } from 'src/decorators/auth.decorator';
import { User } from 'src/decorators/user.decorator';
import { CreateTaskDto, TaskDto } from 'src/typing/dto';
@Controller('tasks')
export class TaskController {
  constructor(private readonly taskService: TaskService) {}
  @Post()
  @Auth()
  async create(@User('id') userId: string, @Body() dto: CreateTaskDto) {
    return await this.taskService.create(userId, dto);
  }
  @Put()
  @Auth()
  async update(@Body() dto: TaskDto) {
    return await this.taskService.update(dto);
  }
  @Delete()
  @Auth()
  async delete(@Body() taskId: string) {
    return await this.taskService.delete(taskId);
  }
```

```
}
```

Код А.4 –tasks.service.ts

```
import {
  Injectable,
  NotFoundException,
  BadRequestException,
} from '@nestjs/common';
import { InjectModel } from 'nestjs-typegoose';
import { CreateTaskDto, TaskDto } from './dto/task.dto';
import { ModelType } from '@typegoose/typegoose/lib/types';
import { TaskModel } from 'src/models/tasks.model';
import { Types } from 'mongoose';
@Injectable()
export class TaskService {
  constructor(
    @InjectModel(TaskModel)
    private readonly taskModel: ModelType<TaskModel>,
  ) {}
  async getTasksByGoalIdsAndDateRange(
    goalIds: Types.ObjectId[],
    startDate: Date,
    endDate: Date,
  ) {
    return await this.taskModel
      .find({
        goalId: { $in: goalIds.map((id) => new Types.ObjectId(id)) },
        date: { $gte: startDate, $lte: endDate },
      })
      .select('goalId title date isCompleted')
      .exec();
  }
  async getUserTasksForStatistic(
    userId: string,
    startDate: Date,
    endDate: Date,
  ) {
    const tasks = await this.taskModel
      .find({
        userId: userId,
        isCompleted: true,
        date: { $gte: startDate, $lte: endDate },
      })
      .select('_id title goalId')
      .populate({
        path: 'goalId',
```

```
      select: '_id emoji',
    })
    .exec();

  return tasks;
 }
 async create(userId: string, dto: CreateTaskDto) {
  if (!userId || !dto.goalId || !dto.title || !dto.date) {
   throw new BadRequestException('Missing required fields');
  }
  return await this.taskModel.create({
   ...dto,
   userId: new Types.ObjectId(userId),
   goalId: new Types.ObjectId(dto.goalId),
  });
 }
 async update(dto: TaskDto) {
  const task = await this.taskModel.findByIdAndUpdate(dto._id, dto, {
   new: true,
   runValidators: true,
  });
  if (!task) {
   throw new NotFoundException(`Task with id ${dto._id} not found`);
  }
  return task;
 }

 async delete(taskId: string) {
  const task = await this.taskModel.findByIdAndDelete(taskId);
  if (!task) {
   throw new NotFoundException(`Task with id ${taskId} not found`);
  }
  return task;
 }
}
```

Код А.5 – goals.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectModel } from 'nestjs-typegoose';
import { ModelType } from '@typegoose/typegoose/lib/types';
import { Types } from 'mongoose';
import { GoalModel } from 'src/models/goal.model';
import { CreateGoalDto, UpdateGoalDto } from './dto/ManageGoalDto';
import { DateService } from '../date/date.service';
```

```
import { TaskService } from '../tasks/tasks.service';
@Injectable()
export class GoalsService {
  constructor(
    @InjectModel(GoalModel)
    private readonly goalModel: ModelType<GoalModel>,
    private readonly dateService: DateService,
    private readonly taskService: TaskService,
  ) {}

  async getActiveGoals(userId: string) {
    const { currentWeek, nextWeek } = this.dateService.getWeekData();
    const weekStart = new Date(currentWeek.weekStart);
    const weekEnd = new Date(nextWeek ? nextWeek.weekEnd :
currentWeek.weekEnd);
    const goals = await this.goalModel.find({ userId, isActive: true }).exec();
    const goalIds = goals.map((goal) => goal._id);
    const tasks = await this.taskService.getTasksByGoalIdsAndDateRange(
      goalIds,
      weekStart,
      weekEnd,
    );
    const activeGoals = goals.map((goal) => {
      const futureUncompletedTasks = tasks.filter(
        (task) =>
          task.goalId.toString() === goal._id.toString() &&
          !task.isCompleted &&
          new Date(task.date) > new Date(currentWeek.weekStart),
      ).length;
      return {
        _id: goal._id,
        title: goal.title,
        emoji: goal.emoji,
        isActive: goal.isActive,
        pendingTasks: futureUncompletedTasks,
      };
    });
    const tasksByDate = tasks.reduce(
      (acc, task) => {
        const dateStr = task.date.toISOString().split('T')[0];
        (acc[dateStr] ??= []).push(task);
        return acc;
      },
      {} as Record<string, (typeof tasks)[number][]>,
    );
```

```
  const allDates = currentWeek.weekDates;
  if (nextWeek) {
    allDates.push(...nextWeek.weekDates);
  }
  const weeklyTasks = allDates.map((date) => ({
    date,
    tasks: tasksByDate[date] || [],
  }));
  const completedTasks = tasks.filter((task) => task.isCompleted).length;
  const notCompletedTasks = tasks.filter((task) => !task.isCompleted).length;
  return {
    activeGoals,
    weeklyTasks,
    weeklyStatistics: {
      completedTasks,
      notCompletedTasks,
    },
  };
}
async create(dto: CreateGoalDto, userId: string) {
  const goal = new this.goalModel({
    ...dto,
    userId,
    isActive: dto.isActive ?? true,
  });
  return goal.save();
}
async update(dto: UpdateGoalDto) {
  const taskId = new Types.ObjectId(dto._id);
  const goal = await this.goalModel.findById(taskId);
  if (!goal) {
    throw new NotFoundException('Goal not found');
  }
  goal.title = dto.title;
  goal.emoji = dto.emoji;
  goal.isActive = dto.isActive;
  return goal.save();
}
async delete(dto: UpdateGoalDto) {
  const taskId = new Types.ObjectId(dto._id);
  const goal = await this.goalModel.findById(taskId);
  if (!goal) {
    throw new NotFoundException('Goal not found');
  }
  goal.isActive = false;
```

```
    return goal.save();
  }
}
```

Код А.6 – statistics.service.ts

```
import { Injectable, Logger } from '@nestjs/common';
import { InjectModel } from 'nestjs-typegoose';
import { ModelType } from '@typegoose/typegoose/lib/types';
import { NotFoundException } from '@nestjs/common';
import { UserService } from '../user/user.service';
import { StatisticsModel } from 'src/models/statistics.model';
import { TaskService } from '../tasks/tasks.service';
import { DateService } from '../date/date.service';
import { Cron, CronExpression } from '@nestjs/schedule';
import { TaskType } from './types/task.type';
@Injectable()
export class StatisticsService {
  private readonly logger = new Logger(StatisticsService.name);
  constructor(
    @InjectModel(StatisticsModel)
    private readonly statisticsModel: ModelType<StatisticsModel>,
    private readonly userService: UserService,
    private readonly dateService: DateService,
    private readonly taskService: TaskService,
  ) {}
  async getYearlyStatistics(userId: string, year: string) {
    const staistics = await this.statisticsModel
      .findOne({ userId, year })
      .select('-__v -_id -createdAt -updatedAt')
      .lean();
    if (!staistics)
      return new NotFoundException(`Staristics for year ${year} not found`);
    return staistics;
  }
  @Cron(CronExpression.EVERY_WEEK)
  async updateWeeklyStatistics() {
    this.logger.log('Updating weekly statistics...');
    const { weekStart, weekEnd } = this.dateService.getLastWeekData();
    const users = await this.userService.getAllUsers();
    for (const user of users) {
      const userId = user._id.toString();
      const tasks = await this.taskService.getUserTasksForStatistic(
        userId,
        new Date(weekStart),
```

```
      new Date(weekEnd),
    );
    if (!tasks.length) {
      this.logger.warn(`No completed tasks for user ${userId}`);
      continue;
    }
    await this.saveStatistics(userId, tasks);
  }
}
async saveStatistics(userId: string, tasks: TaskType[]) {
  const currentYear = new Date().getFullYear();
  const currentMonth = new Date().getMonth() + 1;
  const goalMap = new Map<
    string,
    {
      completedTasks: number;
      taskIds: Set<string>;
      title: string;
      emoji: string;
    }
  >();
  for (const task of tasks) {
    const goalId = task.goalId._id.toString();
    const emoji = task.goalId.emoji;
    const taskId = task._id.toString();
    if (!goalMap.has(goalId)) {
      goalMap.set(goalId, {
        completedTasks: 0,
        taskIds: new Set(),
        title: task.title,
        emoji: emoji,
      });
    }
    const goalStats = goalMap.get(goalId);
    goalStats.completedTasks += 1;
    goalStats.taskIds.add(taskId);
  }
  const goalStatsArray = Array.from(goalMap.entries()).map(
    ([goalId, data]) => ({
      goalId,
      completedTasks: data.completedTasks,
      taskIds: Array.from(data.taskIds),
      title: data.title,
      emoji: data.emoji,
    }),
```

```
);
let userStats = await this.statisticsModel.findOne({ userId });
const createUserStatistics = async () => {
  return await this.statisticsModel.create({
    userId,
    year: currentYear,
    totalCompleted: tasks.length,
    totalGoals: goalStatsArray.length,
    availableYears: [currentYear],
    monthlyStats: [
      {
        month: currentMonth,
        totalCompleted: tasks.length,
        totalGoals: goalStatsArray.length,
        goals: goalStatsArray,
      },
    ],
  });
};
const addNewMonthStats = () => {
  userStats.monthlyStats.push({
    month: currentMonth,
    totalCompleted: tasks.length,
    totalGoals: goalStatsArray.length,
    goals: goalStatsArray,
  });
};
const updateUserStatistics = async () => {
  const monthStats = userStats.monthlyStats.find(
    (m) => m.month === currentMonth,
  );

  if (monthStats) {
    for (const newGoal of goalStatsArray) {
      const existingGoal = monthStats.goals.find(
        (g) => g.goalId === newGoal.goalId,
      );
      if (existingGoal) {
        const existingTaskIds = new Set(existingGoal.taskIds);
        for (const id of newGoal.taskIds) {
          existingTaskIds.add(id);
        }
        existingGoal.taskIds = Array.from(existingTaskIds);
        existingGoal.completedTasks = existingGoal.taskIds.length;
      } else {
```

```
          monthStats.goals.push(newGoal);
        }
      }
      monthStats.totalCompleted = tasks.length;
      monthStats.totalGoals = monthStats.goals.length;
    } else {
      addNewMonthStats();
    }
    userStats.totalCompleted = userStats.monthlyStats.reduce(
      (acc, m) => acc + m.totalCompleted,
      0,
    );
    userStats.totalGoals = userStats.monthlyStats.reduce(
      (acc, m) => acc + m.totalGoals,
      0,
    );
    if (!userStats.availableYears.includes(currentYear)) {
      userStats.availableYears.push(currentYear);
    }
    await userStats.save();
  };
  if (!userStats) {
    await createUserStatistics();
  } else {
    await updateUserStatistics();
  }
 }
}
```

Код А.7 – Сервіс date (date.service.ts)

```
import { Injectable } from '@nestjs/common';
import {
 startOfWeek,
 endOfWeek,
 addWeeks,
 subWeeks,
 format,
 eachDayOfInterval,
} from 'date-fns';
interface WeekInterval {
 weekStart: string;
 weekEnd: string;
}
interface WeekData extends WeekInterval {
```

```
  weekDates: string[];
}
@Injectable()
export class DateService {
  private dateFormat: 'yyyy-MM-dd' = 'yyyy-MM-dd';
  private createDateArray(weekStart: Date, weekEnd: Date): string[] {
    return eachDayOfInterval({ start: weekStart, end: weekEnd }).map((date) =>
      format(date, this.dateFormat),
    );
  }
  public getWeekData(): { currentWeek: WeekData; nextWeek?: WeekData } {
    const today = new Date();
    const isSunday = today.getDay() === 0;
    const currentWeekStart = startOfWeek(today, { weekStartsOn: 1 });
    const currentWeekEnd = endOfWeek(today, { weekStartsOn: 1 });
    const currentWeek: WeekData = {
      weekStart: format(currentWeekStart, this.dateFormat),
      weekEnd: format(currentWeekEnd, this.dateFormat),
      weekDates: this.createDateArray(currentWeekStart, currentWeekEnd),
    };

    if (isSunday) {
      const nextWeekStart = addWeeks(currentWeekStart, 1);
      const nextWeekEnd = endOfWeek(nextWeekStart, { weekStartsOn: 1 });
      const nextWeek: WeekData = {
        weekStart: format(nextWeekStart, this.dateFormat),
        weekEnd: format(nextWeekEnd, this.dateFormat),
        weekDates: this.createDateArray(nextWeekStart, nextWeekEnd),
      };
      return { currentWeek, nextWeek };
    }
    return { currentWeek };
  }
  public getLastWeekData(): WeekInterval {
    const today = new Date();
    const lastWeekStart = startOfWeek(subWeeks(today, 1), { weekStartsOn: 1 });
    const lastWeekEnd = endOfWeek(subWeeks(today, 1), { weekStartsOn: 1 });
    return {
      weekStart: format(lastWeekStart, this.dateFormat),
      weekEnd: format(lastWeekEnd, this.dateFormat),
    };
  }
}
```

Код А.8 – interceptors.ts

```
import axios from 'axios';
import { services } from '@/shared/constants/api-services';
import { AuthService } from '@/features/auth/services/auth.service';
const baseURL = process.env.NEXT_PUBLIC_APP_URL;
export const axiosClassic = axios.create({
  baseURL: `${baseURL}/api`,
  headers: { 'Content-Type': 'application/json' },
  withCredentials: true,
});
const axiosAuth = axios.create({
  baseURL: `${baseURL}/api`,
  headers: { 'Content-Type': 'application/json' },
  withCredentials: true,
});
axiosAuth.interceptors.response.use(
  (res) => res,
  async (err) => {
    const originalRequest = err.config;
    const status = err.response?.status || err?.status;
    if (
      status === 401 &&
      !originalRequest._retry &&
      !originalRequest.url.endsWith(services.token)
    ) {
      originalRequest._retry = true;
      try {
        await AuthService.getNewTokens();
        return axiosAuth(originalRequest);
      } catch (error) {
        console.error('Token refresh failed:', error);
      }
    }
    return Promise.reject(err);
  }
);
export { axiosAuth };
```

Код А.9 – middleware.ts

```
import createMiddleware from 'next-intl/middleware';
import { NextRequest, NextResponse } from 'next/server';
import { routing } from './i18n/routing';
```

```
import { availableLocales } from './i18n/routing';
export function middleware(request: NextRequest) {
  const url = request.nextUrl;
  const pathname = url.pathname;
  const pathLocale = pathname.split('/')[1];
  if (!availableLocales.includes(pathLocale)) {
    const acceptLanguage = request.headers.get('accept-language') || '';
    const preferredLang =
      availableLocales.find((locale) => acceptLanguage.startsWith(locale)) ||
      'en';
    return NextResponse.redirect(
      new URL(`/${preferredLang}${pathname}`, request.url)
    );
  }
  const isAuth =
    request.cookies.has('accessToken') || request.cookies.has('refreshToken');
  const isPlannerPage = pathname.startsWith(`/${pathLocale}/planner`);
  const isAuthPage =
    pathname.startsWith(`/${pathLocale}/login`) ||
    pathname.startsWith(`/${pathLocale}/register`);
  if (!isAuth && isPlannerPage) {
    const redirectUrl = new URL(`/${pathLocale}/login`, request.url);
    return NextResponse.redirect(redirectUrl);
  }
  if (isAuth && isAuthPage) {
    return NextResponse.redirect(
      new URL(`/${pathLocale}/planner`, request.url)
    );
  }
  return createMiddleware(routing)(request);
}
export const config = {
  matcher: [
    '/(uk|en)/planner/:path*',
    '/(uk|en)/login',
    '/(uk|en)/register',
    '/((?!api|_next/|images/|.*\\.(?:png|jpg|jpeg|webp|svg|gif|avif|ico)$).*)',
  ],
};
```

Код А.10 – RegistrationForm.tsx

```
'use client';
import { useEffect } from 'react';
import { Controller, useForm } from 'react-hook-form';
```

```
import { useTranslations } from 'next-intl';
import { yupResolver } from '@hookform/resolvers/yup';
import { ButtonCustom } from '@/shared/components/ui/buttons/Button';
import Input from '@/shared/components/ui/input/Input';
import { routes } from '@/shared/constants/routes';
import icons from '@/shared/icons/icons';
import { User } from '@/shared/types/user.type';
import styles from '@/features/auth/components/AuthForm.module.scss';
import { createRegistrationSchema } from
'@/features/auth/helpers/registration-shema';
type RegisterFormValues = {
  email: string;
  password: string;
  confirmPassword: string;
  name: string;
};
type RegisterFormValuesProps = {
  user: User | null | undefined;
  isPending: boolean;
  error: Error | null;
  register: (data: RegisterFormValues) => void;
};
export const RegisterForm = ({
  isPending,
  error,
  register,
}: RegisterFormValuesProps) => {
  const t = useTranslations('Common');
  const registrationSchema = createRegistrationSchema(t);
  const {
    control,
    handleSubmit,
    clearErrors,
    trigger,
    formState: { errors },
    watch,
  } = useForm<RegisterFormValues>({
    resolver: yupResolver(registrationSchema),
    mode: 'onChange',
  });
  const password = watch('password');
  useEffect(() => {
    if (password) {
      trigger('confirmPassword');
    }
```

```
}, [password, trigger]);
const getError = (name: keyof RegisterFormValues) => {
  return errors?.[name]?.message || undefined;
};
const onSubmit = (values: RegisterFormValues) => {
  register(values);
};
const handleOnFocus = (field: keyof RegisterFormValues) => {
  trigger(field);
  clearErrors('root.serverError');
};
return (
  <form onSubmit={handleSubmit(onSubmit)} className={styles.Form}>
    <fieldset disabled={isPending}>
      <Controller
        name="name"
        control={control}
        defaultValue=""
        render={({ field }) => (
          <Input
            type="text"
            label={t('form.labels.name')}
            placeholder={t('form.placeholders.name')}
            {...field}
            icon={<icons.User />}
            error={getError('name')}
            serverError={!!error}
            onFocus={() => handleOnFocus('name')}
            onBlur={() => trigger('name')}
          />
        )}
      />
      <Controller
        name="email"
        control={control}
        defaultValue=""
        render={({ field }) => (
          <Input
            type="email"
            label={t('form.labels.email')}
            placeholder={t('form.placeholders.email')}
            {...field}
            icon={<icons.Mail />}
            error={getError('email')}
            serverError={!!error}
```

```
      onFocus={() => handleOnFocus('email')}
      onBlur={() => trigger('email')}
    />
  )}
/>
<Controller
  name="password"
  control={control}
  defaultValue=""
  render={({ field }) => (
    <Input
      type="password"
      label={t('form.labels.password')}
      placeholder={t('form.placeholders.password')}
      {...field}
      icon={<icons.Password />}
      hasAbilityHideValue
      error={getError('password')}
      serverError={!!error}
      onFocus={() => handleOnFocus('password')}
      onBlur={() => trigger('password')}
    />
  )}
/>
<Controller
  name="confirmPassword"
  control={control}
  defaultValue=""
  render={({ field }) => (
    <Input
      type="password"
      label={t('form.labels.confirmPassword')}
      placeholder={t('form.placeholders.confirmPassword')}
      {...field}
      icon={<icons.Password />}
      hasAbilityHideValue
      error={getError('confirmPassword')}
      serverError={!!error}
      onFocus={() => handleOnFocus('confirmPassword')}
      onBlur={() => trigger('confirmPassword')}
    />
  )}
/>
<ButtonCustom
  disabled={isPending}
```

```
        name={t('buttons.register')}
        style="outlined"
        onClick={handleSubmit(onSubmit)}
       />
       <div>
        {t('form.messages.alreadyRegistered')}
        <ButtonCustom
         href={routes.login}
         name={t('buttons.signIn')}
         style="text"
        />
       </div>
     </fieldset>
   </form>
  );
};
export default RegisterForm;
```

Код А.11 – auth.service.ts

```
import { axiosAuth, axiosClassic } from '@/api/interceptors';
import { services } from '@/shared/constants/api-services';
import { AuthResponse, User } from '../types/auth.types';
export const AuthService = {
  async login(email: string, password: string): Promise<AuthResponse> {
    const { data } = await axiosClassic.post<AuthResponse>(services.login, {
      email,
      password,
    });
    return data;
  },
  async register(
    email: string,
    password: string,
    name: string
  ): Promise<AuthResponse> {
    const { data } = await axiosClassic.post<AuthResponse>(services.register, {
      email,
      password,
      name,
    });
    return data;
  },
  async logout(): Promise<{ message: string }> {
    const { data } = await axiosClassic.post<{ message: string }>(
```

```
      services.logout
    );
    return data;
  },
  async getNewTokens(): Promise<AuthResponse> {
    const { data } = await axiosClassic.post<AuthResponse>(services.token);
    return data;
  },
  async getProfile(): Promise<User> {
    const { data } = await axiosAuth.get<User>(services.userProfile);
    return data;
  },
};
```

Код А.12 – AuthProvider.tsx

```
'use client';
import { createContext, useContext } from 'react';
import {
  LoginFormValues,
  RegisterFormValues,
} from '@/shared/types/interfaces/loginFormValues';
import { User } from '@/shared/types/user.type';
import { useAuth } from '@/features/auth/hooks/useAuth';
interface AuthContextType {
  user: User | null | undefined;
  isPending: boolean;
  error: Error | null;
  login: (data: LoginFormValues) => void;
  register: (data: RegisterFormValues) => void;
  logout: () => void;
}
const AuthContext = createContext<AuthContextType | undefined>(undefined);
export const AuthProvider = ({ children }: { children: React.ReactNode }) => {
  const auth = useAuth();
  const contextValue: AuthContextType = {
    user: auth.user,
    isPending: auth.isPending,
    error: auth.error,
    login: auth.login,
    register: auth.register,
    logout: auth.logout,
  };
  return (
```

```
    <AuthContext.Provider
value={contextValue}>{children}</AuthContext.Provider>
  );
};
export const useAuthContext = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuthContext must be used within an AuthProvider');
  }
  return context;
};
```

Код А.13 – registration-schema.ts

```
import { Formats, TranslationValues } from 'next-intl';
import * as yup from 'yup';
import { emailRegx } from '@/shared/utils/email-regx';
export const createRegistrationSchema = (t: {
  <TargetKey extends any>(
    key: TargetKey,
    values?: TranslationValues,
    formats?: Formats
  ): string;
}) => {
  return yup.object().shape({
    name: yup
      .string()
      .matches(/^[a-zA-Za-яA-ЯёЁiIїЇєЄ]+$/, t('form.validation.name.letters'))
      .min(3, t('form.validation.name.minLength'))
      .required(t('form.validation.name.required')),
    email: yup
      .string()
      .matches(emailRegx, t('form.validation.email.invalid'))
      .required(t('form.validation.email.required')),
    password: yup
      .string()
      .min(6, t('form.validation.password.minLength'))
      .required(t('form.validation.password.required')),
    confirmPassword: yup
      .string()
      .required(t('form.validation.confirmPassword.required'))
      .when('password', {
        is: (password: string) => !!password,
        then: (schema) =>
          schema
```

```
          .oneOf(
            [yup.ref('password')],
            t('form.validation.confirmPassword.match')
          )
          .required(t('form.validation.confirmPassword.required')),
        otherwise: (schema) => schema.notRequired(),
      }),
  });
};
```

Код А.14 – Planner.tsx

```
'use client';
import { useTranslations } from 'next-intl';

import { ErrorMessage } from
'@/shared/components/ui/errorMessage/ErrorMessage';
import { Skeleton } from '@/features/planning/components/_parts/skeleton/Skeleton';
import { GoalsList } from '@/features/planning/components/goals-list/GoalsList';
import { WeeklyStatistic } from
'@/features/planning/components/weekly-statistic/WeeklyStatistic';
import { WeeklyTodo } from
'@/features/planning/components/weekly-todo/WeeklyTodo';
import { PlanningContext } from '@/features/planning/context/usePlanningContext';
import { usePlanning } from '@/features/planning/hooks/usePlanning';
import styles from './Planner.module.scss';
export const Planner = () => {
  const t = useTranslations('Common.planning');
  const {
    weeklyStatistics,
    activeGoals,
    currentWeek,
    nextWeek,
    createGoal,
    updateGoal,
    deleteGoal,
    createTask,
    updateTask,
    deleteTask,
    isError,
    isPending,
  } = usePlanning();
  if (isPending) return <Skeleton />;
  if (isError) return <ErrorMessage message={t('error')} />;
  return (
```

```
    <PlanningContext.Provider
      value={{ activeGoals, createTask, updateTask, deleteTask, isPending }}
    >
      <div className={styles.Planner}>
        <h1 className={styles.Title}>{t('title')}</h1>
        <div className={styles.Header}>
          <GoalsList
            activeGoals={activeGoals}
            createGoal={createGoal}
            updateGoal={updateGoal}
            deleteGoal={deleteGoal}
          />
          <WeeklyStatistic
            statistics={weeklyStatistics}
            isError={isError}
            isPending={isPending}
          />
        </div>
        <WeeklyTodo
          weeks={nextWeek ? [currentWeek, nextWeek] : [currentWeek]}
        />
      </div>
    </PlanningContext.Provider>
  );
};
```

Код А.15 – ManageGoal.tsx

```
'use client';
import { useRef, useState } from 'react';
import { useTranslations } from 'next-intl';
import { IconButtonCustom } from '@/shared/components/ui/buttons/IconButton';
import { Dropdown } from '@/shared/components/ui/dropdown-custom/Dropdown';
import { PlanningInput } from
'@/shared/components/ui/planning-input/PlanningInput';
import { isObjectTheSame } from '@/shared/helpers/is-object-the-same';
import { useClickOutside } from '@/shared/hooks/ useClickOutside';
import icons from '@/shared/icons/icons';
import { ActiveGoal, CreateGoal, Goal } from '../../../types/goals.type';
import EmojiPickerCompoment from '../emojy-picker/EmojiPicker';
import styles from './ManageGoal.module.scss';
type ManageGoalsProps = {
  goal: ActiveGoal | null;
  onSave: (goal: Goal | CreateGoal) => void;
  onCancel: () => void;
```

```
  onDelete?: (goal: Goal) => void;
};
export const defaultGoal: CreateGoal = {
  title: ' ',
  emoji: '@',
  isActive: true,
};
const ManageGoals = ({
  goal,
  onSave,
  onCancel,
  onDelete,
}: ManageGoalsProps) => {
  const t = useTranslations('Common');
  const formRef = useRef<HTMLFormElement>(null);
  const [localGoal, setLocalGoal] = useState<Goal | CreateGoal>(
    goal || defaultGoal
  );
  const [isEmojiPickerOpen, setIsEmojiPickerOpen] = useState(false);
  const handleGoal = () => {
    if (
      !isObjectTheSame(localGoal, defaultGoal) &&
      goal &&
      !isObjectTheSame(localGoal, goal)
    ) {
      onSave(localGoal);
    } else {
      onCancel();
    }
  };
  useClickOutside(formRef, handleGoal);

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    handleGoal();
  };
  return (
    <form
      onSubmit={handleSubmit}
      className={styles.ManageGoalForm}
      ref={formRef}
    >
      <div className={styles.EmojyPickerWrapper}>
        <button
          onClick={() => setIsEmojiPickerOpen(!isEmojiPickerOpen)}
```

```
    className={styles.EmojiButton}
    type="button"
   >
    {localGoal.emoji || '@'}
   </button>
   {isEmojiPickerOpen && (
    <EmojiPickerCompoment
     onEmojiClick={({ emoji }) => {
      setLocalGoal((prev) => ({ ...prev, emoji: emoji }));
      setIsEmojiPickerOpen(false);
     }}
     handleClickOuside={() => setIsEmojiPickerOpen(false)}
    />
   )}
  </div>
  <PlanningInput
   value={localGoal.title}
   onChange={(e) =>
    setLocalGoal((prev) => ({ ...prev, title: e.target.value }))
   }
   placeholder={t('placeholders.addGoal')}
   maxLength={50}
  />
  <div className={styles.Actions}>
   <IconButtonCustom
    icon={<icons.Save />}
    name={t('buttons.save')}
    type="submit"
    size="small"
   />
   <Dropdown
    trigger={
     <IconButtonCustom
      icon={<icons.MoreVertical />}
      name={t('buttons.menu')}
      size="small"
     />
    }
    menuItems={[
     {
      icon: <icons.Trash />,
      title: !!goal?.pendingTasks
       ? t(notifications.forbiddenDeleteTask')
       : t('buttons.delete'),
      action: () => {
```

```
            if ('_id' in localGoal && onDelete) onDelete(localGoal as Goal);
          },
          type: !!goal?.pendingTasks ? 'message' : 'button',
        },
       ]}
      />
    </div>
  </form>
 );
};
export default ManageGoals;
```

Код А.16 – WeeklyStatistics.tsx

```
'use client';
import { useTranslations } from 'next-intl';
import classNames from 'classnames';
import icons from '@/shared/icons/icons';
import { WeeklyStatistics } from '@/features/planning/types/goals.type';
import styles from './WeeklyStatistic.module.scss';
type WeeklyStatisticProps = {
 statistics: WeeklyStatistics | null;
 isPending: boolean;
 isError: boolean;
};
export const WeeklyStatistic = ({ statistics }: WeeklyStatisticProps) => {
 const t = useTranslations('Common.statistics');
 return (
  <div className={styles.Statistic}>
   <div className={styles.Wrapper}>
    <h2 className={styles.Title}>{t('weeklyStatistics')}</h2>
    <p className={classNames(styles.SubTitle, styles.Completed)}>
     <span>
      <icons.CheckCircle />
      {t('completed')}
     </span>
     <span>{statistics?.completedTasks ?? '-'}</span>
    </p>
    <p className={styles.SubTitle}>
     <span>
      <icons.Calendar /> {t('todo')}
     </span>
     <span>{statistics?.notCompletedTasks ?? '-'}</span>
    </p>
   </div>
```

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

Код А.17 – WeeklyToDo.tsx

```
'use client';
import { useOptimistic, useState, useTransition } from 'react';
import { DragDropContext } from '@hello-pangea/dnd';
import classNames from 'classnames';
import { DaySection } from
'@/features/planning/components/_parts/day-section/DaySection';
import { WeekHeader } from
'@/features/planning/components/_parts/week-header/WeekHeader';
import { usePlanningContext } from
'@/features/planning/context/usePlanningContext';
import { filterDays } from '@/features/planning/helpers/filter-days';
import { isDateToday } from '@/features/planning/helpers/is-today';
import { useDragDropHandler } from
'@/features/planning/hooks/useDragDrogHandler';
import { WeeklyTasks } from '@/features/planning/types/task.type';
import styles from './WeeklyToDo.module.scss';
type WeekProps = {
  weeks: WeeklyTasks[];
};
export const WeeklyTodo = ({ weeks }: WeekProps) => {
  const { activeGoals, updateTask } = usePlanningContext();
  const [isListView, setIsListView] = useState(false);
  const [prioritizeTaskDays, setPrioritizeTaskDays] = useState(false);
  const [isTodayView, setIsTodayView] = useState(false);
  const [isPending, startTransition] = useTransition();
  const [optimisticWeeks, addOptimisticWeeks] = useOptimistic(
    weeks,
    (_, newWeeks: WeeklyTasks[]) => newWeeks
  );
  const { handleDragEnd } = useDragDropHandler({
    weeksData: optimisticWeeks,
    addOptimistic: (weeksData: WeeklyTasks[]) => {
      startTransition(() => {
        addOptimisticWeeks(weeksData);
      });
    },
  const toggleTodayView = () => {
    if (isListView) setIsListView(false);
    if (prioritizeTaskDays) setPrioritizeTaskDays(false);
```

```
    setIsTodayView((prev) => !prev);
  };
  return (
    <section className={styles.WeeklyTodo}>
      <DragDropContext onDragEnd={handleDragEnd}>
        {optimisticWeeks.map((week) => {
          const tasks = prioritizeTaskDays
            ? filterDays(week)
            : isTodayView
              ? week.filter(({ date }) => isDateToday(date))
              : week;
          return (
            <div
              key={week[0]?.date || 'empty-week'}
              className={styles.WeekContainer}
            >
              <WeekHeader
                week={week}
                isListView={isListView}
                prioritizeTaskDays={prioritizeTaskDays}
                isTodayView={isTodayView}
                onToggleListView={() => setIsListView((prev) => !prev)}
                onTogglePrioritizeDays={() =>
                  setPrioritizeTaskDays((prev) => !prev)
                }
                onToggleIsTodayView={toggleTodayView}
              />
              <div
                className={classNames(styles.Day, {
                  [styles.listView]: isListView || isTodayView,
                })}
              >
                {tasks.map(({ date, tasks: dayTasks }) => (
                  <DaySection key={date} date={date} dayTasks={dayTasks} />
                ))}
              </div>
            </div>
          );
        })}
      </DragDropContext>
    </section>
  );
};
```

Код А.18 – TasksSection.tsx

```
'use client';
import { useState } from 'react';
import { useTranslations } from 'next-intl';
import { Draggable } from '@hello-pangea/dnd';
import { IconButtonCustom } from '@/shared/components/ui/buttons/IconButton';
import icons from '@/shared/icons/icons';
import { ActiveGoal } from '../../../types/goals.type';
import { CreateTask, Task } from '../../../types/task.type';
import ManageTask from '../manage-task/ManageTask';
import styles from './TasksSection.module.scss';
interface TasksSectionProps {
  goal: ActiveGoal;
  goalTasks: Task[];
  date: string;
  allTasks: Task[];
}
export const TasksSection = ({
  goal,
  goalTasks,
  date,
  allTasks,
}: TasksSectionProps) => {
  const t = useTranslations('Common');
  const [manageTask, setManageTask] = useState<CreateTask | null>(null);
  const onFinishManage = () => setManageTask(null);
  return (
    <div className={styles.Goal}>
      <div className={styles.GoalHeader}>
        <h4 className={styles.GoalTitle}>
          {goal.emoji} {goal.title}
        </h4>
        <IconButtonCustom
          icon={<icons.PlusCircle />}
          name={t('buttons.addTask')}
          onClick={() =>
            setManageTask({
              date,
              goalId: goal._id,
            } as CreateTask)
          }
          size="small"
        />
      </div>
      {manageTask?.goalId === goal._id && manageTask?.date === date && (
```

```
      <ManageTask
        task={{
          title: '',
          goalId: goal._id,
          isCompleted: false,
          date,
        }}
        finishManage={onFinishManage}
      />
    )}
    <div className={styles.GoalTasks}>
      {goalTasks.map((task) => {
        const globalIndex = allTasks.findIndex((t) => t._id === task._id);
        return (
          <Draggable
            draggableId={task._id}
            index={globalIndex}
            key={task._id}
          >
            {(provided) => (
              <div
                className={styles.TaskItem}
                ref={provided.innerRef}
                {...provided.draggableProps}
                {...provided.dragHandleProps}
              >
                <div className={styles.DraggableIcon} role-type="button">
                  <icons.Draggable />
                </div>
                <ManageTask task={task} finishManage={onFinishManage} />
              </div>
            )}
          </Draggable>
        );
      })}
    </div>
  </div>
);
};
```

Код А.19 – useDragDropHandler.ts

```
import { DropResult } from '@hello-pangea/dnd';
import { filterTasksByGoals } from '@/features/planning/helpers/filter-tasks-by-goal';
import { ActiveGoal } from '@/features/planning/types/goals.type';
```

```
import { Task, WeeklyTasks } from '@/features/planning/types/task.type';
interface DragDropHandlerProps {
  weeksData: WeeklyTasks[];
  updateTask: (task: Task) => void;
  addOptimistic: (weeksData: WeeklyTasks[]) => void;
  activeGoals: ActiveGoal[];
}
export const useDragDropHandler = ({
  weeksData,
  updateTask,
  activeGoals,
  addOptimistic,
}: DragDropHandlerProps) => {
  const findTaskById = (taskId: string) => {
    for (let wi = 0; wi < weeksData.length; wi++) {
      for (let di = 0; di < weeksData[wi].length; di++) {
        const dayData = weeksData[wi][di];
        const foundIndex = dayData.tasks.findIndex((t) => t._id === taskId);
        if (foundIndex !== -1) {
          return {
            task: JSON.parse(JSON.stringify(dayData.tasks[foundIndex])),
            weekIndex: wi,
            dayIndex: di,
            taskIndex: foundIndex,
          };
        }
      }
    }
    return null;
  };
  const findDayByDate = (date: string) => {
    for (let wi = 0; wi < weeksData.length; wi++) {
      for (let di = 0; di < weeksData[wi].length; di++) {
        if (weeksData[wi][di].date === date) {
          return {
            weekIndex: wi,
            dayIndex: di,
          };
        }
      }
    }
    return null;
  };
  const getFlattenedTasks = (dayTasks: Task[]) => {
    const relevantGoals = [
```

```
    ...filterTasksByGoals(dayTasks, activeGoals).goalsWithOrderedTasks,
    ...filterTasksByGoals(dayTasks, activeGoals).goalsWithoutTasks,
  ];
  return relevantGoals.flatMap((goal) =>
    dayTasks.filter((task) => task.goalId === goal._id)
  );
};
const removeTaskFromSource = (
  updatedWeeks: WeeklyTasks[],
  source: { weekIndex: number; dayIndex: number; taskIndex: number }
) => {
  updatedWeeks[source.weekIndex][source.dayIndex].tasks.splice(
    source.taskIndex,
    1
  );
};
const insertTaskAtDestination = (
  updatedWeeks: WeeklyTasks[],
  destination: { weekIndex: number; dayIndex: number },
  destinationIndex: number,
  taskToMove: Task
) => {
  const destDayTasks =
    updatedWeeks[destination.weekIndex][destination.dayIndex].tasks;
  const destFlattenedTasks = getFlattenedTasks(destDayTasks);
  if (destinationIndex < destFlattenedTasks.length) {
    const referenceTaskId = destFlattenedTasks[destinationIndex]._id;
    const refTaskIndex = destDayTasks.findIndex(
      (t) => t._id === referenceTaskId
    );
    destDayTasks.splice(refTaskIndex, 0, taskToMove);
  } else {
    destDayTasks.push(taskToMove);
  }
};
const cloneWeeksData = () => JSON.parse(JSON.stringify(weeksData));
const handleDragEnd = async (result: DropResult) => {
  const { source, destination, draggableId } = result;
  if (!destination) return;
  if (source.droppableId === destination.droppableId) {
    return;
  }
  const [destGoalId, destDate] = destination.droppableId.split('|');
  const taskData = findTaskById(draggableId);
  if (!taskData) return;
```

```
    const destDay = findDayByDate(destDate);
    if (!destDay) return;
    const updatedWeeks = cloneWeeksData();
    removeTaskFromSource(updatedWeeks, {
      weekIndex: taskData.weekIndex,
      dayIndex: taskData.dayIndex,
      taskIndex: taskData.taskIndex,
    });
    const taskToMove = {
      ...taskData.task,
      date: new Date(destDate),
      goalId: destGoalId,
    };
    insertTaskAtDestination(
      updatedWeeks,
      { weekIndex: destDay.weekIndex, dayIndex: destDay.dayIndex },
      destination.index,
      taskToMove
    );
    addOptimistic(updatedWeeks);
    updateTask(taskToMove);
  };
  return { handleDragEnd };
};
```

Код А.20 – use-statistics.ts

```
import { useQuery } from '@tanstack/react-query';
import { getStatistics } from '../services/statistics.service';
import { UserStatistics } from '../types/statistics.type';
export const useStatistics = (year: string) => {
  const { data, isPending, isError, refetch } = useQuery<UserStatistics>({
    queryKey: ['user-statistics', year],
    queryFn: () => getStatistics(year),
    refetchOnWindowFocus: false,
  });
  return {
    refetch,
    data,
    isPending,
    isError,
  };
};
```

Код А.21 – Statistics.tsx

```
'use client';
import { useState } from 'react';
import { useTranslations } from 'next-intl';
import { AccordionUsage } from '@/shared/components/ui/accordion/Accordion';
import icons from '@/shared/icons/icons';
import { useStatistics } from '../hooks/use-statistics';
import { ErrorMessage } from './_parts/error-message/Error';
import { GoalsList } from './_parts/goals-list/GoalsList';
import { MonthlyStatsHeader } from
'./_parts/montly-states-header/MonthlyStatsHeader';
import { SelectYear } from './_parts/SelectYear';
import { Skeleton } from './_parts/skeleton/Skeleton';
import { StatisticsData } from './_parts/statistics-data/StatisticsData';
import styles from './Statistics.module.scss';
const currentYear = new Date().getFullYear().toString();
export const Statistics = () => {
  const [selectedYear, setSelectedYear] = useState(currentYear);
  const { data: statistics, isPending, isError } = useStatistics(selectedYear);
  const t = useTranslations('Common');
  if (isPending) return <Skeleton />;
  return (
    <div className={styles.Statistics}>
      <h1 className={styles.Title}>{t('statistics.annualStatistics')}</h1>
      <p className={styles.Notification}>{t('statistics.notification')}</p>
      {isError || !statistics?.monthlyStats ? (
        <ErrorMessage error={t('statistics.error')} />
      ) : (
        <>
          <SelectYear
            availableYears={statistics.availableYears}
            currentYear={statistics.year}
            onChange={(year) => setSelectedYear(year)}
          />
          <div className={styles.Total}>
            <StatisticsData
              icon={<icons.Award />}
              title={t('statistics.totalGoals')}
              total={statistics.totalGoals}
            />
            <StatisticsData
              icon={<icons.CheckCircle />}
```

```
            title={t('statistics.completedTasks')}
            total={statistics.totalCompleted}
          />
        </div>
        <section className={styles.Monthly}>
          <div className={styles.MonthlyContent}>
            <h2 className={styles.SubTitle}>
              {t('statistics.monthlyStatistics')}
            </h2>
            <AccordionUsage
              items={statistics.monthlyStats.map((item) => ({
                title: (
                  <MonthlyStatsHeader
                    month={item.month}
                    totalGoals={item.totalGoals}
                    totalCompleted={item.totalCompleted}
                  />
                ),
                description: <GoalsList goals={item.goals} />,
              }))}
            />
          </div>
        </section>
      </>
    )}
  </div>
 );
};
export default Statistics;
```

Код А.22 – api-services.ts

```
export const services = {
  auth: '/auth/',
  login: '/auth/login',
  register: '/auth/register',
  logout: '/auth/logout',
  googleAuth: '/auth/google',
  token: 'auth/access-token',
  updateUser: '/user/profile',
  activeGoals: '/goals',
  tasks: '/tasks',
  statistics: '/statistics',
  userProfile: '/user',
};
```

Код А.23 – routes.ts

```
export const routes = {
  home: '/',
  planner: '/planner',
  login: '/login',
  register: '/register',
  statistics: '/planner/statistics',
  settings: '/planner/settings',
  help: '/planner/help',
};
```

Код А.24 – Button.tsx

```
'use client';
import React, { MouseEventHandler } from 'react';
import { useLocale } from 'next-intl';
import Button from '@mui/material/Button';
type ButtonProps = {
  onClick?: MouseEventHandler<HTMLButtonElement>;
  iconStart?: React.ReactNode;
  iconEnd?: React.ReactNode;
  name: string;
  disabled?: boolean;
  className?: string;
  style?: 'contained' | 'outlined' | 'text';
  color?: 'secondary' | 'success' | 'error';
  href?: string;
};
export const ButtonCustom = ({
  onClick,
  iconStart,
  iconEnd,
  name,
  disabled = false,
  style = 'contained',
  color = 'secondary',
  href,
}: ButtonProps) => {
  const locale = useLocale();
  return (
    <Button
      variant={style}
```

```
      color={color}
      disabled={disabled}
      onClick={href ? undefined : onClick}
      href={`/${locale}/${href}`}
      startIcon={iconStart}
      endIcon={iconEnd}
    >
      {name}
    </Button>
  );
};
```

Код А.25 – Input.tsx

```
'use client';
import { forwardRef, useState } from 'react';
import {
  FiAlertTriangle,
  FiCheckCircle,
  FiEye,
  FiEyeOff,
} from 'react-icons/fi';
import classNames from 'classnames';
import styles from './Input.module.scss';
interface InputProps {
  value: string;
  onChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
  onFocus?: (event: React.FocusEvent<HTMLInputElement>) => void;
  onBlur?: (event: React.FocusEvent<HTMLInputElement>) => void;
  icon?: React.ReactNode;
  type?: 'text' | 'password' | 'email';
  label?: string;
  hasMessages?: boolean;
  error?: string;
  hasAbilityHideValue?: boolean;
  isLabelVisibilityHidden?: boolean;
  disabled?: boolean;
  placeholder?: string;
  serverError?: boolean;
}
export const Input = forwardRef<HTMLInputElement, InputProps>(
  (
    {
      value,
      onChange,
```

```
    onFocus,
    onBlur,
    type = 'text',
    icon,
    error,
    placeholder,
    hasMessages = true,
    hasAbilityHideValue = false,
    disabled = false,
    serverError = false,
  },
  ref
) => {
  const [fieldType, setFieldType] = useState(type);

  const toggleVisibility = (e: React.MouseEvent) => {
    e.preventDefault();
    setFieldType((prevType) =>
      prevType === 'password' ? 'text' : 'password'
    );
  };
  const isSuccess = hasMessages && !error && value && !serverError;
  const inputClass = classNames(styles.InputWrapper, {
    [styles.error]: error,
    [styles.success]: isSuccess,
  });
  return (
    <div className={styles.Wrapper}>
      <div className={inputClass}>
        {isSuccess && <FiCheckCircle className={styles.SuccessValidation} />}
        {icon && <div className={styles.Icon}>{icon}</div>}
        <input
          type={fieldType}
          value={value}
          onChange={onChange}
          onFocus={(e) => {
            if (onFocus) onFocus(e);
          }}
          onBlur={(e) => {
            if (onBlur) onBlur(e);
          }}
          placeholder={placeholder}
          disabled={disabled}
          className={styles.Input}
          ref={ref}
```

```
        />
        <span className={styles.Label}>{placeholder}</span>
        {hasAbilityHideValue && (
          <div
            role="button"
            onClick={toggleVisibility}
            className={styles.EyeIcon}
          >
            {fieldType === 'password' ? <FiEyeOff /> : <FiEye />}
          </div>
        )}
      </div>
      {hasMessages && (
        <p className={styles.Error}>
          {error && <FiAlertTriangle />}
          {error}
        </p>
      )}
    </div>
  );
 }
);
export default Input;
```

Код А.26 – Dropdown.tsx

```
'use client';

import { ReactNode, useRef, useState } from 'react';
import { useClickOutside } from '@/shared/hooks/ useClickOutside';
import styles from './Dropdown.module.scss';
import { Divider, DividerItemType } from './parts/divider/Divider';
import { MenuItemComponent, MenuItemType } from
'./parts/menu-item/MenuItem';
import { Message, MessageItemType } from './parts/message/Message';
type MenuItem = MenuItemType | DividerItemType | MessageItemType;
type DropdownProps = {
  trigger: ReactNode;
  menuItems: MenuItem[];
};
export const Dropdown = ({ trigger, menuItems }: DropdownProps) => {
  const [isOpen, setIsOpen] = useState(false);
  const dropdownRef = useRef<HTMLDivElement>(null);
  useClickOutside(dropdownRef, () => setIsOpen(false));
  const handleItemClick = (item: MenuItem) => {
```

```
    if ('action' in item) item.action?.();
    if (!('href' in item)) setIsOpen(false);
  };
  return (
    <div className={styles.Dropdown} ref={dropdownRef}>
      <div
        onClick={() => setIsOpen((prev) => !prev)}
        className={styles.Trigger}
      >
        {trigger}
      </div>
      {isOpen && (
        <ul className={styles.Menu} role="menu">
          {menuItems.map((item, index) => {
            if (item.type === 'divider') return <Divider key={index} />;
            if (item.type === 'message')
              return <Message key={index} {...item} />;
            return (
              <MenuItemComponent
                key={index}
                item={item}
                onClick={() => handleItemClick(item)}
              />
            );
          })}
        </ul>
      )}
    </div>
  );
};
export default Dropdown;
```

Код А.27 – SelectCustom.tsx

```
import { useState } from 'react';
import FormControl from '@mui/material/FormControl';
import FormHelperText from '@mui/material/FormHelperText';
import MenuItem from '@mui/material/MenuItem';
import Select, { SelectChangeEvent } from '@mui/material/Select';
import styles from './SelectCustom.module.scss';
interface CustomSelectProps {
  value: string;
  onChange: (event: string) => void;
  options: { label: string; value: string | number }[];
  helper?: string;
```

```
  showValue?: boolean;
  format?: 'standard' | 'outlined';
  disabled?: boolean;
  minWidth?: number;
}
export const SelectCustom = ({
  value,
  onChange,
  options,
  helper,
  format = 'outlined',
  showValue = false,
  disabled = false,
  minWidth = 100,
}: CustomSelectProps) => {
  const [selectedValue, setSelectedValue] = useState(value);
  const handleChange = (event: SelectChangeEvent) => {
    setSelectedValue(event.target.value);
    onChange(event.target.value);
  };
  return (
    <FormControl
      sx={{ m: 1, minWidth: minWidth }}
      size="small"
      className={styles.FormControl}
    >
      <Select
        className={styles.Select}
        onChange={handleChange}
        value={selectedValue}
        variant={format}
        disabled={disabled}
      >
        {options.map((option) => (
          <MenuItem key={option.value} value={option.value}>
            {option.label} {showValue && option.value}
          </MenuItem>
        ))}
      </Select>
      {helper && <FormHelperText>{helper}</FormHelperText>}
    </FormControl>
  );
};
export default SelectCustom;
```

Код А.29 – Header.tsx

```
'use client';
import { useTranslations } from 'next-intl';
import { usePathname } from 'next/navigation';
import classNames from 'classnames';
import LogOut from '@/shared/components/LogOut';
import { routes } from '@/shared/constants/routes';
import useScrollThreshold from '@/shared/hooks/useScrollThreshold';
import icons from '@/shared/icons/icons';
import { isCurrentRoute } from '@/shared/utils/check-current-route';
import { useAuth } from '@/features/auth/hooks/useAuth';
import { ButtonCustom } from '../ui/buttons/Button';
import Logo from '../ui/logo/Logo';
import styles from './Header.module.scss';
export const Header = () => {
  const isSticky = useScrollThreshold(100);
  const t = useTranslations('Common.buttons');
  const { user } = useAuth();
  const pathname = usePathname();
  const isHomePage = isCurrentRoute(pathname, routes.home);
  return (
    <header
      className={classNames(styles.Header, { [styles.Sticky]: isSticky })}
    >
      <div className={styles.Wrapper}>
        <Logo />
        {user?.id ? (
          <div className={styles.Menu}>
            <ButtonCustom
              href={routes.planner}
              name={t('profile')}
              style="outlined"
              iconStart={<icons.User />}
            />
            <LogOut />
          </div>
        ) : (
          isHomePage && <ButtonCustom href={routes.login} name={t('login')} />
        )}
      </div>
    </header>
  );
```

```
};
export default Header;
```

Код А.29 – check-current-route.ts

```
import { availableLocales } from '@/i18n/routing';
export const isCurrentRoute = (
  pathname: string | null,
  targetRoute: string
): boolean => {
  if (!pathname) return false;
  const localePattern = `^\/(${availableLocales.join('|')})(\/|$)`;
  const normalizedPath = pathname.replace(new RegExp(localePattern), '/');
  return (
    normalizedPath === targetRoute ||
    normalizedPath === `/${targetRoute.replace(/^\/+/, '')}`
  );
};
```

Код А.30 – useClickOutside.ts

```
import { RefObject, useEffect } from 'react';
export const useClickOutside = (
  ref: RefObject<HTMLDivElement | HTMLFormElement | null>,
  callback: () => void
) => {
  if (!ref) return;
  useEffect(() => {
    const handleClickOutside = (event: MouseEvent) => {
      const target = event.target as Node;
      const isInside = ref.current?.contains(target);
      if (!isInside) {
        callback();
      }
    };
    document.addEventListener('mousedown', handleClickOutside);
    return () => document.removeEventListener('mousedown', handleClickOutside);
  }, [ref, callback]);
};
```