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

ФАКУЛЬТЕТ ПРОГРАМНИХ СИСТЕМ ТА ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ В. О. РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
“___” червня 2026 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою**
**«Системне програмування та спеціалізовані комп’ютерні системи»**

по спеціальності **123 «Комп'ютерна інженерія»**
на тему: Комп’ютерна система планування та моніторингу виконання особистих завдань

Виконала:

студентка IV курсу, групи КВ-22

Гречишкіна Катерина Дмитрівна _____
(підпис)

Керівник: ас. каф. СПСКС ФПСПМ Кучмій О.О. _____
(підпис)

Консультант з нормоконтролю:
доц., к.т.н., доц. каф. СПСКС Клятченко Я. М. _____
(підпис)

Рецензент: _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2026 року

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

ФАКУЛЬТЕТ ПРОГРАМНИХ СИСТЕМ ТА ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

Освітньо-професійна програма

«Системне програмування та спеціалізовані комп’ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

«12» грудня 2025р.

# ЗАВДАННЯ

**на дипломний проєкт студентки**
Гречишкіної Катерини Дмитрівни

1. Тема проєкту «Комп’ютерна система планування та моніторингу виконання особистих завдань», керівник проєкту ас. каф. СПСКС ФПМ Кучмій О. О., затверджені наказом по університету №1984-С від « 28» травня 2026 р.

2. Термін подання студенткою проєкту: «10 » червня 2026 р.

3. Вихідні дані до проєкту: див. Технічне завдання

4. Зміст пояснювальної записки

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту*

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | доц.каф.СПСКС, к.т.н. Клятченко Я.М. | | |

7. Дата видачі завдання _____

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1. | Вивчення літератури за тематикою проєкту | 12.12.2025 | |
| 2. | Розроблення технічного завдання | 20.12.2025 | |
| 3. | Аналіз існуючих рішень | 11.01.2026 | |
| 4. | Підготовка матеріалів першого розділу дипломного проєкту | 16.02.2026 | |
| 5. | Підготовка матеріалів другого розділу дипломного проєкту | 01.03.2026 | |
| 6. | Підготовка матеріалів третього розділу дипломного проєкту | 17.03.2026 | |
| 7. | Підготовка матеріалів четвертого розділу дипломного проєкту | 10.04.2026 | |
| 8. | Підготовка графічної частини дипломного проєкту | 15.05.2026 | |
| 9. | Оформлення документації дипломного проєкту | 25.05.2026 | |
| 10. | Попередній огляд матеріалів дипломного проєкту на кафедрі | 05.06.2026 | |

Студент _____ Катерина ГРЕЧИШКІНА
(підпис) (І'мя та ПРІЗВИЩЕ)

Керівник проєкту _____ Оксана КУЧМІЙ
(підпис) (І'мя та ПРІЗВИЩЕ)

* Консультантом не може бути зазначено керівника дипломного проекту.

# АНОТАЦІЯ

Об’єкт розробки – створення комп’ютерної системи планування та моніторингу виконання особистих завдань.

Мета розробки – створення програмного забезпечення для ефективного планування часу та відстеження щоденних звичок.

Комп’ютерна система забезпечує можливість планувати щоденні завдання з урахуванням пріоритетів і дедлайнів, вести системний облік звичок з автоматичним лічильником прогресу, проводити детальний аналіз особистої продуктивності за різні періоди, забезпечувати зручне управління даними через сучасний інтерфейс. Система передбачає обов’язкову реєстрацію та автентифікацію користувачів, завдяки чому забезпечується конфіденційність персональних даних та можливість індивідуального налаштування профілю.

В ході розробки було:

- Проведено порівняльний аналіз існуючих програмних рішень;
- Сформульовані технічні вимоги до системи планування та моніторингу виконання особистих завдань;
- Спроєктовано архітектуру програмного забезпечення та структуру бази даних;
- Реалізовано клієнтську та серверну частину;
- Створено програмне забезпечення що повністю відповідає встановленим технічним вимогам.

Упровадження комп’ютерної системи дозволить користувачам значно підвищити рівень особистої продуктивності, покращити самодисципліну та ефективніше керувати власним часом.

Ключові слова: ВЕБЗАСТОСУНОК, ТРЕКЕР ЗВИЧОК, ПЛАНУВАННЯ ЧАСУ, ПЕРСОНАЛЬНА ПРОДУКТИВНІСТЬ, NESTJS, VUE 3, TYPESCRIPT, POSTGRESQL.

# ANNOTATION

The object of development is the creation of a computer system for planning and tracking personal tasks.

The main goal of the project is to create software for effective time management and tracking daily habits.

The computer system enables users to plan daily tasks taking into account priorities and deadlines, keep systematic track of habits with an automatic progress tracker, conduct detailed analysis of personal productivity over various periods, and manage data conveniently via a modern interface. The system requires mandatory user registration and authentication, thereby ensuring the confidentiality of personal data and the ability to customise profiles.

During the development process:

- A comparative analysis of existing software solutions was carried out;
- Technical requirements for the system for planning and tracking personal tasks were formulated;
- The software architecture and database structure were designed;
- The client and server components were developed;
- Software was created that fully meets the established technical requirements.

The implementation of this computer system will enable users to significantly increase their personal productivity, improve self-discipline and manage their time more effectively.

Keywords: WEB APPLICATION, HABIT TRACKER, TIME MANAGEMENT, PERSONAL PRODUCTIVITY, NESTJS, VUE 3, TYPESCRIPT, POSTGRESQL.

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількісь аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | А4 | ІАЛЦ.045440.002 ТЗ | Комп’ютерна система | 4 | | |
| | | | планування та моніторингу | | | |
| | | | виконання особистих | | | |
| | | | завдань | | | |
| | | | Технічне завдання | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045440.003 ТП | Комп’ютерна система | 2 | | |
| | | | планування та моніторингу | | | |
| | | | виконання особистих | | | |
| | | | завдань | | | |
| | | | Відомість технічного | | | |
| | | | проекту | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.004 ПЗ | Комп’ютерна система | 73 | | |
| | | | планування та моніторингу | | | |
| | | | виконання особистих | | | |
| | | | завдань | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.005 Д1 | Схема загальної технічної | 1 | | |
| | | | архітектури системи | | | |
| | | | Схема структурна | | | |

| Змін. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.045440.001 ОА | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Гречишкіна К. Д. | | | Комп’ютерна система планування та моніторингу виконання особистих завдань<br>***Опис альбому*** | Літ. | Аркуш 1 / Аркушів 2 |
| Перевірив | | Кучмій О.О. | | | | | |
| Консульт. | | | | | | КПІ ім. Ігоря Сікорського, ФПСПМ КВ-22 | |
| Н. контроль | | Клятченко Я.М. | | | | | |
| Зав. каф. | | Романкевич В.О. | | | | | |

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількість аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | А4 | ІАЛЦ. 045440.006 Д2 | Схема екранів користувача | 1 | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.007 Д3 | Схема взаємозв’язків бази | 1 | | |
| | | | даних. ER-діаграма | | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.008 Д4 | Блок-схема алгоритму | 1 | | |
| | | | дій користувача в системі | | | |
| | | | Схема алгоритму | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

| | | | | | ІАЛЦ.045440.001 ОА | Арк. |
|---|---|---|---|---|---|---|
| Змін. | Арк. | № докум. | Підпис | Дата | | 2 |

**ЗМІСТ**



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ. 045440.002 ТЗ | | |
| Зм | Арк. | № докум. | Підп. | Дата | | | |
| Розробив. | | Гречишкіна К.Д. | | | Комп'ютерна система планування та моніторингу виконання особистих завдань | Літ. | Аркуш 1 / Аркушів 4 |
| Перевірив. | | Кучмій О.О. | | | | | |
| Н. контр. | | Клятченко Я.М. | | | ***Технічне завдання*** | КПІ ім. Ігоря Сікорського ФПСПМ КВ-22 | |
| Затвердив. | | Романкевич В.О. | | | | | |

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: Комп’ютерна система планування та моніторингу виконання особистих завдань.

Галузь застосування: призначена для індивідуального використання з метою підвищення особистої продуктивності шляхом оптимізації процесів тайм-менеджменту та системного моніторингу звичок.

## 2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

## 3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даної роботи є створення програмного забезпечення для ефективного планування часу та відстеження щоденних звичок.

## 4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

# 5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється:

- реалізація у вигляді односторінкового вебзастосунку;
- підтримка повної авторизації користувачів (реєстрація, вхід, управління профілем);
- створення, редагування, виконання, видалення та сортування завдань за пріоритетом і дедлайном;
- створення, редагування, виконання та видалення звичок з автоматичною візуалізацією прогресу через лічильники;
- наявність інтерактивного календаря з відображенням завдань;
- система аналітики прогресу за тиждень, місяць та рік з графічним представленням;
- зручний, сучасний інтерфейс.

5.2 Вимоги до апаратного забезпечення:

- Процесор: 2-ядерний або більше;
- Оперативна пам’ять: не менше 4 Гб;
- Наявність доступу до мережі інтернет.

5.3 Вимоги до програмного та апаратного забезпечення користувача:

- Операційна система Windows 10/11, macOS, Linux, iOS або Android;

## 6. ЕТАПИ РОЗРОБКИ

| № з/п | Назва етапів виконання дипломного проекту | Термін виконання етапів |
|---|---|---|
| 1. | Вивчення літератури за тематикою проєкту | 12.12.2025 |
| 2. | Розроблення технічного завдання | 20.12.2025 |
| 3. | Аналіз існуючих рішень | 11.01.2026 |
| 4. | Підготовка матеріалів першого розділу дипломного проєкту | 16.02.2026 |
| 5. | Підготовка матеріалів другого розділу дипломного проєкту | 01.03.2026 |
| 6. | Підготовка матеріалів третього розділу дипломного проєкту | 17.03.2026 |
| 7. | Підготовка матеріалів четвертого розділу дипломного проєкту | 10.04.2026 |
| 8. | Підготовка графічної частини дипломного проєкту | 15.05.2026 |
| 9. | Оформлення документації дипломного проєкту | 25.05.2026 |
| 10. | Попередній огляд матеріалів дипломного проєкту на кафедрі | 05.06.2026 |

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількісь аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | А4 | ІАЛЦ.045440.002 ТЗ | Комп’ютерна система | 4 | | |
| | | | планування та моніторингу | | | |
| | | | виконання особистих | | | |
| | | | завдань | | | |
| | | | Технічне завдання | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045440.003 ТП | Комп’ютерна система | 2 | | |
| | | | планування та моніторингу | | | |
| | | | виконання особистих | | | |
| | | | завдань | | | |
| | | | Відомість технічного | | | |
| | | | проекту | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.004 ПЗ | Комп’ютерна система | 73 | | |
| | | | планування та моніторингу | | | |
| | | | виконання особистих | | | |
| | | | завдань | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.005 Д1 | Схема загальної технічної | 1 | | |
| | | | архітектури системи | | | |
| | | | Схема структурна | | | |

| Змін. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.045440.003 ТП | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Гречишкіна К. Д. | | | Комп’ютерна система планування та моніторингу виконання особистих завдань<br>***Відомість технічного проєкту*** | Літ. | Аркуш 1 / Аркушів 2 |
| Перевірив | | Кучмій О.О. | | | | | |
| Консульт. | | | | | | КПІ ім. Ігоря Сікорського, ФПСПМ КВ-22 | |
| Н. контроль | | Клятченко Я.М. | | | | | |
| Зав. каф. | | Романкевич В.О. | | | | | |

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількість аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | А4 | ІАЛЦ. 045440.006 Д2 | Схема екранів користувача | 1 | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.007 Д3 | Схема взаємозв’язків бази | 1 | | |
| | | | даних. ER-діаграма | | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ. 045440.008 Д4 | Блок-схема алгоритму | 1 | | |
| | | | дій користувача в системі | | | |
| | | | Схема алгоритму | | | |

| Змін. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.045440.003 ТП | Арк. 2 |
|---|---|---|---|---|---|---|

# ПОЯСНЮВАЛЬНА ЗАПИСКА
# ДО ДИПЛОМНОГО ПРОЄКТУ

На тему: Комп’ютерна система планування та моніторингу виконання особистих завдань

Київ – 2026

**ЗМІСТ**

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API (Application Programming Interface) – прикладний програмний інтерфейс взаємодії між клієнтською та серверною частинами системи.

HTML (HyperText Markup Language) – стандартизована мова розмітки для створення структури та архітектури вебсторінок у браузері.

MPA (Multi-Page Application) – багатосторінковий вебзастосунок із повним перезавантаженням сторінок при кожній дії користувача.

SPA (Single Page Application) – односторінковий вебзастосунок, у якому контент оновлюється динамічно без перезавантаження сторінки у браузері.

UX (User Experience) – досвід користувача.

# ВСТУП

У сучасному суспільстві темп життя постійно зростає, люди стикаються з двома проблемами: дефіцитом часу та втратою концентрації уваги. Постійні сповіщення, інформаційний шум, відсутність системного підходу – все це робить увагу поверхневою та неефективною.

Велика кількість завдань, дедлайни, професійні та особисті обов'язки, необхідність постійно опановувати нові навички, адаптуватися до швидких змін – все це вимагає високого рівня самоорганізації. Саме відсутність системного підходу до планування часто призводить до зниження продуктивності, накопичення хронічного стресу, вигорання та неможливості досягнення довгострокових цілей. Тому систематичне відстеження щоденних звичок та впровадження системного планування здатне суттєво підвищити рівень самодисципліни та загальну результативність діяльності людини.

Незважаючи на значну кількість існуючих вебзастосунків для управління завданнями та звичками, більшість з них мають суттєві недоліки: вони або надто складні та перевантажені функціями, або не поєднують у собі інструменти планування часу та відстеження звичок в єдиній системі. Це зумовлює необхідність розробки рішення яке б поєднувало все це в межах одного зручного інтерфейсу.

Таким чином, розробка комп'ютерної системи планування та моніторингу виконання особистих завдань дозволить користувачам ефективно планувати та організовувати свій час, системно відстежувати прогрес у формуванні звичок та аналізувати особисту продуктивність за різні часові періоди, що в підсумку сприяє підвищенню рівня самодисципліни та успішнішому досягненню поставлених цілей.

# 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

## 1.1. Опис предметної області

Людина щодня стикається з величезною кількістю інформації: завдання, дедлайни, професійні та особисті обов’язки. Зростання обсягу роботи, віддалена форма зайнятості, постійний доступ до соціальних мереж та інформаційних ресурсів призводять до когнітивного перевантаження, розсіювання уваги та труднощів у самоорганізації. За даними сучасних психологічних досліджень, здатність до самоконтролю є обмеженим ресурсом, а не вродженою рисою характеру, тому більшість людей періодично стикаються з труднощами під час виконання складних завдань. Відсутність структури в повсякденному житті часто призводить до прокрастинації, хронічного стресу, емоційного вигорання та неможливості досягнення довгострокових цілей.

Предметна область дипломного проєкту охоплює сферу персональної продуктивності сучасної людини. Вона включає сукупність процесів пов’язаних з плануванням, організацією, виконанням, контролем та аналізом особистих завдань, а також формування, підтримку і моніторинг корисних звичок.

До ключових компонентів предметної області належать:

- *Планування завдань* – процес визначення цілей, розбиття великих завдань на менші, встановлення пріоритетів, дедлайнів. Це стосується як короткострокового планування (на день чи тиждень), так і довгострокового.

- *Виконання та контроль* – передбачає фіксацію факту виконання завдань, відстеження прогресу в реальному часі, внесення коректив у разі зміни обставин.

- *Формування та відстеження звичок* – систематичне повторення бажаних дій, створення стійких патернів поведінки, подолання шкідливих звичок. Важливим елементом є регулярне відстеження виконання (так званий стрік – кількість днів поспіль).

- *Моніторинг і аналіз продуктивності* – збір даних про виконані завдання та звички, розрахунок ключових показників (відсоток виконання, кількість днів поспіль, прогрес за періодами), візуалізація результатів та виявлення закономірностей.

- *Психологічні аспекти самоорганізації* – мотивація, самоконтроль, боротьба з прокрастинацією, формування довгострокової дисципліни.

Отже, предметна область дипломного проєкту охоплює взаємопов’язані процеси самоорганізації сучасної людини в умовах інформаційного перевантаження та дефіциту часу.

Таким чином, розробка єдиної комп’ютерної системи планування та моніторингу виконання особистих завдань є актуальним рішенням, оскільки воно спрямоване на вирішення реальних проблем сучасної людини та сприяє підвищенню якості її життя.

## 1.2. Огляд існуючих програмних рішень

На сьогоднішній день користувачі мають доступ до великої кількості застосунків для управління завданнями та звичками. Однак більшість з них не повністю задовольняє потреби сучасної людини. У цьому підрозділі розглядаються три найбільш популярні системи – Todoist, Habitica, Notion.

### 1.2.1. Todoist

Todoist – один з найпопулярніших у світі менеджерів завдань, який активно використовується мільйонами людей з 2007 року. Система позиціонується як універсальний інструмент для особистого та командного управління завданнями.

Серед сильних сторін Todoist варто виділити зручний, інтуїтивний інтерфейс (Рисунок 1.1), потужну систему фільтрів і пошуку, а також широкі можливості інтеграції з іншими сервісами (Google Calendar, Gmail, Outlook тощо).

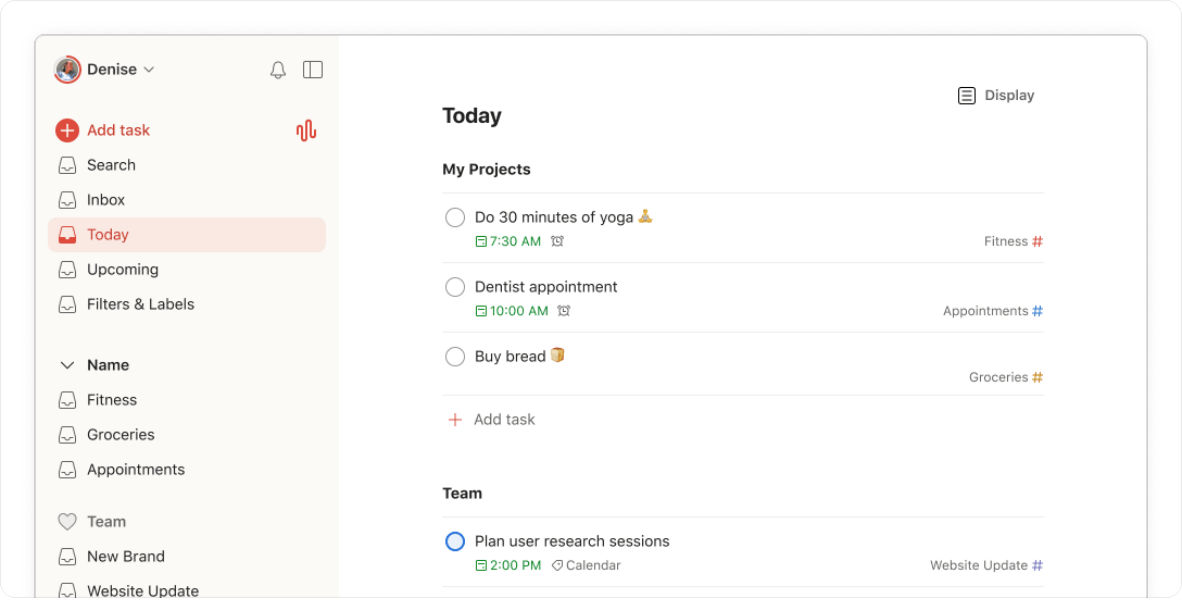



*Рисунок 1.1 – Головна сторінка застосунку Todoist*

В Todoist добре реалізований механізм нагадувань, коментарів до завдань, прикріплення файлів та відстеження активності. Крім того, система пропонує вбудовану статистику яка відображає кількість виконаних завдань за день, тиждень та місяць.

Завдяки гнучкості налаштувань і розвиненій екосистемі інтеграцій Todoist став одним з лідерів ринку персональних та командних менеджерів

завдань. Він особливо популярний серед студентів, фрілансерів, менеджерів проєктів та людей, які працюють з великою кількістю завдань.

Разом з тим, при всій своїй функціональності Todoist залишається переважно інструментом управління завданнями. Він не має вбудованого повноцінного модуля для відстеження звичок, графіку прогресу та детальної довгострокової аналітики формування навичок. Через це система не забезпечує комплексного підходу до розвитку особистої продуктивності, що є суттєвим обмеженням для користувачів які прагнуть не лише виконувати завдання, а й системно змінювати свої звички.

### 1.2.2. Habitica

Habitica – гейміфікований трекер завдань і звичок який позиціонує себе як «рольова гра реального життя». Проєкт був запущений у 2013 році і на сьогодні має мільйони користувачів по всьому світу. Основна концепція Habitica полягає в тому щоб перетворити виконання реальних завдань і звичок на елементи рольової гри: користувач створює персонажа, який прокачується за виконання щоденних справ, втрачає «здоров'я» при їх пропуску та отримує нагороди.

Сильними сторонами Habitica є висока мотивація завдяки гейміфікації, наявність соціальної складової (гільдії, партії, спільні квести), а також можливість відстеження регулярності виконання звичок. Завдяки ігровим механікам багато користувачів відзначають значне підвищення мотивації на початкових етапах використання.

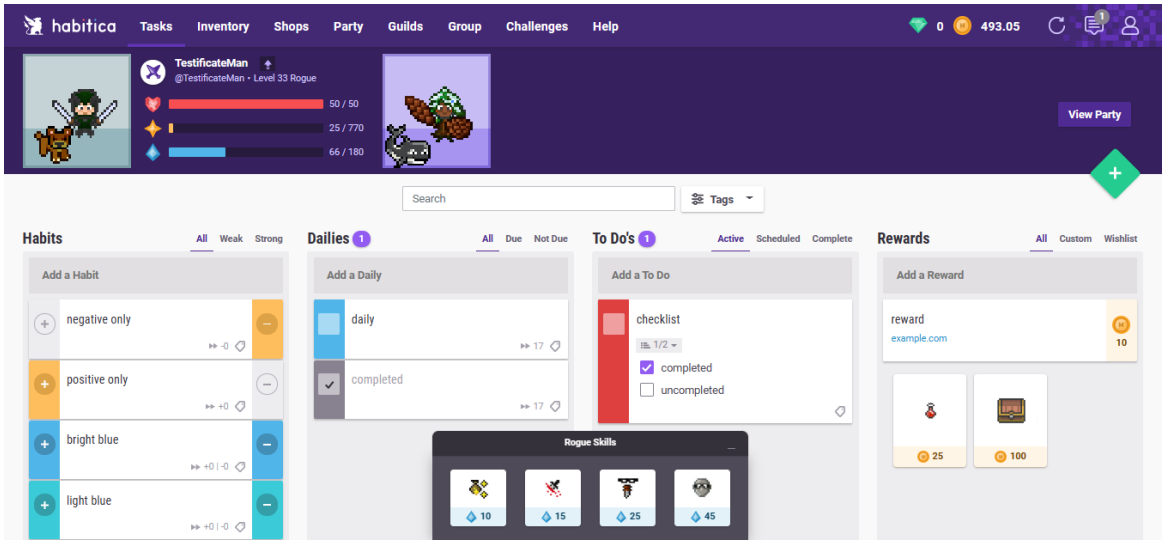



*Рисунок 1.2 – Головна сторінка застосунку Habitica*

Проте Habitica має ряд суттєвих недоліків. Система має застарілий і неінтуїтивний інтерфейс (Рисунок 1.2). Це значно поступається сучасним застосункам. Інструменти планування завдань у Habitica досить слабкі: відсутня зручна система пріоритетів, гнучких дедлайнів та підзавдань. Аналітика продуктивності в системі поверхнева і не дозволяє проводити глибокий аналіз прогресу. Крім того, надмірна гейміфікація для частини користувачів викликає додатковий стрес замість мотивації.

Таким чином, Habitica добре підходить для людей які потребують зовнішньої мотивації через гру, але не є оптимальним рішенням для тих, хто шукає комплексний і сучасний інструмент управління завданнями та звичками.

### 1.2.3. Notion

Notion – це універсальна платформа для створення нотаток, баз знань, баз даних і систем управління завданнями. Завдяки високій гнучкості користувач може самостійно будувати практично будь-які структури – від простих списків завдань до складних персональних систем продуктивності.

Серед сильних сторін Notion варто виділити величезні можливості кастомізації. Вбудовані шаблони, можливість поєднувати нотатки, завдання, календарі в одному робочому просторі. Notion також підтримує спільну роботу в реальному часі та має розширену систему інтеграцій. Інтерфейс застосунку представлено на рисунку 1.3.

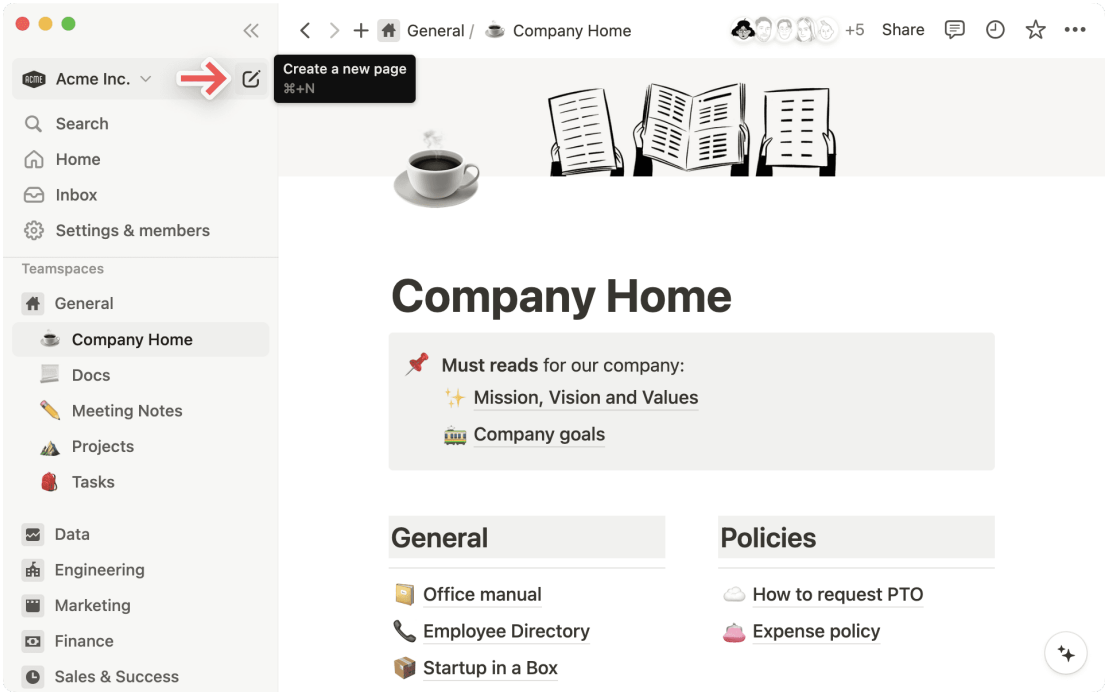



*Рисунок 1.3 – Головна сторінка застосунку Notion*

Проте при всій своїй універсальності Notion має суттєві недоліки для використання як основного інструменту персональної продуктивності. Через високу гнучкість система вимагає значних зусиль на початковому етапі налаштування, що вимагає значного часу на освоєння. Крім того, вбудована аналітика продуктивності в Notion досить обмежена і не дозволяє проводити глибокий аналіз ефективності звичок і завдань.

Таким чином, Notion є потужним універсальним інструментом, але не оптимальним рішенням для користувачів, які шукають спеціалізовану, швидку та інтуїтивно зрозумілу систему управління завданнями і звичками.

## 1.3. Порівняльний аналіз існуючих систем

Для обґрунтування необхідності розробки власної комп'ютерної системи було проведено порівняльний аналіз трьох найбільш популярних рішень у сфері персональної продуктивності – Todoist, Habitica та Notion. Порівняння здійснювалося за ключовими критеріями важливими для користувачів: функціональні можливості управління завданнями, відстеження звичок, аналітика, зручність інтерфейсу, мотиваційні механізми.

Як видно з таблиці 1.1, жодна з розглянутих систем не забезпечує повноцінної інтеграції всіх необхідних функцій.

*Таблиця 1.1 – Порівняльний аналіз існуючих систем*

| ***Критерій порівняння*** | ***Todoist*** | ***Habitica*** | ***Notion*** |
|---|---|---|---|
| *1. Управління завданнями* | + | + | + |
| *2. Відстеження звичок* | - | + | - |
| *3. Графік відстеження прогресу* | - | ± | - |
| *4. Детальна аналітика продуктивності* | - | - | - |
| *5. Сучасний та інтуїтивний інтерфейс* | + | - | + |
| *6. Мотиваційні механізми* | - | + | - |

***Todoist*** добре справляється з класичним управлінням завданнями завдяки системі пріоритетів, фільтрів, дедлайнів і інтеграцій. Однак він практично не має інструментів для відстеження звичок і аналітики продуктивності. Система орієнтована на виконання окремих завдань, а не на формування довгострокових навичок.

***Habitica*** пропонує оригінальний гейміфікований підхід до відстеження звичок і планування завдань. Ігрові механіки добре мотивують на початковому етапі, але система має застарілий інтерфейс, обмежену аналітику та може викликати додатковий стрес через механіку втрати «здоров'я» персонажа.

***Notion*** вирізняється надзвичайною гнучкістю та можливістю створення власних структур даних. Проте саме ця гнучкість стає недоліком, бо користувач змушений самостійно проєктувати всю систему управління завданнями та звичками, а це вимагає значного часу і технічних навичок. Notion не має вбудованих спеціалізованих інструментів відстеження звичок та детальної аналітики продуктивності.

### 1.4. Обґрунтування необхідності розробки власної системи

Загальний висновок порівняльного аналізу показує що жодна з розглянутих систем не пропонує повноцінного комплексного рішення.

Todoist сильний у завданнях, Habitica – у мотивації через гру, Notion – у гнучкості, але жодна з них не поєднує ефективне управління завданнями, відстеження звичок, інтерактивний календар та глибоку аналітику продуктивності в єдиному зручному інтерфейсі.

У результаті користувачі змушені використовувати кілька різних програм одночасно, що призводить до втрати мотивації та зниження загальної ефективності.

Саме тому актуальною є розробка власної системи, яка усуває виявлені недоліки та пропонує цілісне сучасне рішення для управління особистою продуктивністю.

# 2. ВИБІР ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ СИСТЕМИ

Вибір технологій та засобів розробки є одним з найбільш відповідальних етапів створення будь-якої сучасної комп’ютерної системи. Правильно підібрані технології визначають не тільки швидкість і якість розробки, але й майбутні характеристики системи, такі як продуктивність, масштабованість, надійність, безпеку та зручність довгострокової підтримки.

Метою даного розділу є обґрунтування вибору технологій та засобів розробки для комп’ютерної системи планування та моніторингу виконання особистих завдань. У розділі проводиться порівняльний аналіз альтернативних рішень та пояснюється чому саме обрані технології є найбільш оптимальними для досягнення поставлених цілей проєкту.

## 2.1. Вибір архітектури для системи

Вибір архітектури є одним з найбільш важливих етапів проєктування. Архітектура визначає фундаментальні принципи побудови системи, способи взаємодії між її компонентами, розподіл відповідальностей, механізми масштабування та зручність подальшої підтримки й розвитку. Неправильно обрана архітектура може призвести до значних труднощів на етапі реалізації, ускладнення розширення функціоналу та підвищення вартості підтримки системи в майбутньому.

Саме тому розробка комп’ютерної системи планування та моніторингу виконання особистих завдань розпочалася саме з визначення її архітектури. Це вибір, який визначає наскільки ефективно система зможе вирішувати поставлені завдання, наскільки зручно буде працювати з нею користувачам і наскільки легко буде розвивати її в майбутньому.

### 2.1.1. Монолітна архітектура

Монолітна архітектура – це класичний і найбільш традиційний підхід до проєктування програмного забезпечення. Основна ідея цього підходу полягає в тому що вся комп'ютерна система проєктується, розробляється та запускається як один єдиний блок коду (процес), де всі частини взаємодіють прямо одна з одною.

Будь-яку систему в такій архітектурі можна розділити на три взаємопов'язані шари кожен з яких відповідає за певну функціональність:

- *Презентаційний шар (інтерфейс користувача)* – зовнішня графічна оболонка програми з якою взаємодіє користувач.
- *Логічний шар (бізнес-логіка)* – набір правил, алгоритмів та функцій, які обробляють запити й виконують обчислення.
- *Шар доступу до даних (робота з базою даних)* – код, який відповідає за з'єднання з базою даних та збереження інформації.

Усі запити користувача обробляються цим єдиним додатком. Взаємодія між його внутрішніми модулями відбувається миттєво всередині оперативної пам'яті сервера без залучення мережевих протоколів.

***Переваги монолітної архітектури:***

- *Простота розробки.* На початкових етапах значно легше писати код, тому що всі модулі знаходяться в одному місці, мають спільні бібліотеки.
- *Висока продуктивність взаємодії.* Завдяки виклику функцій безпосередньо в пам'яті повністю відсутні мережеві затримки.

- *Легкість відлагодження.* Процес пошуку помилок є прямолінійним, оскільки можна покроково простежити весь шлях запиту від клікання на екрані до запису в базу даних в одному середовищі.

- *Легше тестувати програму, оскільки вона одна ціла.* Тести запускаються для всього додатка одночасно.

***Недоліки монолітної архітектури:***

- *Складність підтримки та масштабування коду при зростанні системи*. При додаванні нових модулів та розширення функціоналу, обсяг коду стрімко збільшується. Чіткі межі між окремими логічними компонентами стираються, а зв'язки між ними стають заплутаними. Таке ускладнення структури суттєво уповільнює процес написання нового функціоналу.

- *Крихкість системи.* Через те що всі функціональні шари та модулі виконуються всередині одного системного процесу, будь-яка критична помилка неминуче призводить до аварійного завершення роботи всього додатка. В результаті локальний збій повністю зупиняє функціонування всієї комп'ютерної системи для всіх користувачів.

- *Складність масштабування окремих функцій системи.* Монолітна архітектура не дозволяє здійснювати гнучке масштабування окремих компонентів системи. Зміна однієї частини коду може вплинути на інші частини коду, це може призвести до помилок.

- Обмежена гнучкість. Увесь монолітний проєкт від самого початку жорстко прив'язаний до однієї обраної мови програмування, конкретної версії базового фреймворку та сторонніх бібліотек. Інтегрування іншого технологічного стеку всередину діючого моноліту буде технічно неможливим, це змушує розробників йти на компроміси або повністю переписувати систему з нуля.

Монолітна архітектура є максимально ефективним рішенням на початкових етапах життєвого циклу програмного забезпечення. Вона ідеально підходить для невеликих проєктів завдяки простоті розробки, швидкому тестуванню та високій продуктивності всередині однієї системи.

Проте з розширенням монолітна архітектура перетворюється на стримуючий фактор. Проблеми з масштабуванням, жорстка прив'язка до одного технологічного стеку та ризик повного падіння системи через локальну помилку змушують великі проєкти з часом трансформувати моноліт у гнучкішу архітектуру.

Під час проєктування комп’ютерної системи планування та моніторингу виконання особистих завдань класична закрита монолітна архітектура (де інтерфейс та логіка виконуються в одному процесі) була відхилена. Система створюється з розрахунком на подальший розвиток. У моноліті обчислення пов'язані з розрахунком прогресу звичок та генерацією графіків за тривалі періоди, створювали б високе навантаження на процесор. Оскільки весь додаток працює як один процес, ці аналітичні операції блокували б роботу всього сервера, викликаючи затримки навіть при звичайному створенні завдань. Для забезпечення стабільності, ізоляції модулів та плавності інтерфейсу було прийнято рішення відмовитися від монолітної архітектури.

### 2.1.2. Мікросервісна архітектура

Мікросервісна архітектура – це архітектурний підхід при якому додаток розбивається на невеликі, дрібні, незалежні мікросервіси. Кожен мікросервіс відповідає за виконання конкретної бізнес-функції (наприклад сервіс авторизації, сервіс управління завданнями, сервіс аналітики тощо) і може бути розроблений, розгорнутий та масштабований незалежно від інших.

***Переваги мікросервісної архітектури:***

- *Технологічна гнучкість.* Відсутність прив'язки до технологічного стеку. Різні сервіси можуть бути написані на різних мовах і використовувати різні технології, що дозволяє обирати найкращий інструмент для конкретного мікросервісу.
- *Незалежна розробка та розгортання.* Кожен мікросервіс має власний життєвий цикл. Зміна коду в одному сервісі вимагає тестування та розгортання лише цього конкретного компонента. Команди можуть працювати над різними сервісами паралельно, не заважаючи один одному. Оновлення одного сервісу не вимагає перезбірки всього застосунку.
- *Стійкість до відмов.* При виході з ладу одного сервісу інші продовжують працювати, це підвищує загальну доступність системи.
- *Легкість впровадження нових функцій*. Розширення функціоналу системи відбувається шляхом створення та інтеграції нових мікросервісів. Це дозволяє масштабувати систему без ризику порушити цілісність.

***Недоліки мікросервісної архітектури:***

- *Значна складність налаштування взаємодії.* Необхідно вирішувати проблеми міжсервісної комунікації, узгодження даних.
- *Складність налагодження та тестування.* Локальне тестування ускладнюється через велику кількість міжсервісних залежностей. Оскільки програма складається з багатьох частин, важливо добре продумати її тестування та безпеку.

Мікросервісна архітектура є підходом який забезпечує максимальну масштабованість, технологічну свободу та високу стійкість системи до відмов. Вона ідеально підходить для великих, високонавантажених проєктів із численними командами розробників, оскільки дозволяє паралельно розвивати окремі функції без ризику зламати весь додаток.

Для комп'ютерної системи планування та моніторингу виконання особистих завдань мікросервісна архітектура була відхилена. На етапі дипломного проєкту вона є надмірно складною та ресурсоємною. Крім того, поточні функціональні вимоги системи не виправдовують такої високої складності.

### 2.1.3. Багатосторінкова архітектура

Багатосторінкова архітектура (Multi-Page Application, MPA) – це традиційний підхід до побудови вебзастосунків, при якому кожна дія користувача (перехід на іншу сторінку, відправка форми, оновлення даних) призводить до повного завантаження нової сторінки з сервера. Кожна сторінка є окремою HTML-сторінкою, яка генерується на сервері та передається клієнту.

Принцип роботи MPA простий: браузер надсилає HTTP-запит на сервер, сервер формує повну HTML-сторінку і повертає її клієнту. Браузер повністю «перемальовує» вікно. Цей цикл повторюється при кожній навігації.

***Переваги багатосторінкової архітектури:***

- *Простота реалізації на сервері*. Сервер повністю відповідає за генерацію HTML-сторінок, що суттєво спрощує логіку на клієнтській стороні. Розробнику не потрібно будувати окремий API (Application Programming Interface), управляти станом на фронтенді чи синхронізувати дані між клієнтом і сервером. Весь потік даних однонаправлений і передбачуваний: запит – обробка – відповідь.

- *Потужна пошукова оптимізація (*Search Engine Optimization *, SEO).* Кожна сторінка має власний, унікальний URL і повністю завантажується з сервера у вигляді готового HTML, саме так як його очікують бачити пошукові системи. Пошукові системи (Google, Bing та інші) без зусиль індексують весь контент, оскільки їм не потрібно виконувати JavaScript чи чекати на асинхронні запити. Для контентних проєктів, блогів, інтернет-магазинів, новинних порталів це є перевагою.

- *Менший обсяг клієнтського JavaScript.* Клієнтська частина зазвичай мінімальна або взагалі відсутня: бізнес-логіка живе на сервері. Це безпосередньо впливає на час завантаження сторінки, особливо на слабких або старих пристроях із обмеженими обчислювальними ресурсами та повільним інтернетом. Менше JavaScript – менше часу компіляцію та виконання коду в браузері.

***Недоліки багатосторінкової архітектури****:*

- *Низька інтерактивність інтерфейсу.* Кожна дія користувача – відмітка завдання як виконаного, перемикання між вкладками, фільтрація списку призводить до повного перезавантаження сторінки. Браузер заново завантажує HTML, CSS, шрифти, зображення. Користувач бачить миготіння екрану і відчуває затримку навіть при швидкому інтернеті. У сучасному контексті, де люди звикли до моментальної реакції інтерфейсу, це суттєво знижує сприйняття якості продукту.

- *Велике навантаження на сервер*. При кожному переході сервер змушений генерувати повну HTML-сторінку з нуля. При великій кількості одночасних користувачів це швидко перетворюється на вузьке місце.

- *Повільніше оновлення даних.* Користувач не отримує миттєве оновлення інформації (наприклад прогресу звичок або нових завдань) без перезавантаження сторінки.

- *Складність створення сучасного* UX (User Experience). Важко реалізувати плавні анімації переходів між сторінками, динамічні елементи які очікують сучасні користувачі.

Для комп'ютерної системи планування та моніторингу виконання особистих завдань багатосторінкова архітектура була відхилена. Система потребує високої інтерактивності: швидкого оновлення списків завдань, миттєвої відмітки виконання, оновлення прогресу звичок. Багатосторінкова модель не здатна забезпечити такий рівень користувацького досвіду, тому вона не відповідає сучасним вимогам до вебзастосунків продуктивності.

### 2.1.4. Односторінкова архітектура

Односторінкова архітектура (Single Page Application, SPA) – це підхід при якому після початкового завантаження однієї HTML-сторінки весь подальший контент оновлюється динамічно за допомогою JavaScript безпосередньо в браузері користувача без повного перезавантаження сторінки.

У SPA-моделі сервер в основному виконує роль API-провайдера, надаючи дані у форматі JSON, тоді як вся логіка відображення, маршрутизація та взаємодія з користувачем відбувається на клієнтській стороні.

***Переваги односторінкової архітектури:***

- *Висока швидкість та плавність інтерфейсу*. Користувач отримує миттєву реакцію на свої дії (відмітка завдання, перемикання вкладок, оновлення календаря) без неприємного «блимання» при перезавантаженні сторінки.

- *Комфорт для користувача (Сучасний UX).* Можливість створення плавних анімацій, динамічних елементів інтерфейсу.

- *Зменшення навантаження на сервер*. Після початкового завантаження більшість запитів обробляється на клієнті, а сервер займається лише наданням даних через API.

- *Зручність розробки складних інтерфейсів.* Легко реалізовувати складні динамічні елементи які оновлюються в реальному часі, інтерактивні дашборди тощо.

***Недоліки односторінкової архітектури:***

- *Довге перше завантаження.* Коли користувач уперше відкриває сайт, йому доводиться чекати трохи довше, ніж зазвичай. Браузер повинен завантажити пакет JavaScript-коду, в якому зашитий увесь функціонал сайту. На пристроях з обмеженими апаратними ресурсами перші кілька секунд можна бачити просто білий екран або значок завантаження.

- *Збільшення складності клієнтської частини.* Більше логіки переноситься на клієнт, що вимагає ретельної організації коду, управління станом і обробки помилок.

Для комп’ютерної системи планування та моніторингу виконання особистих завдань односторінкова архітектура є найбільш доцільним вибором. Система повинна забезпечувати швидке та комфортне щоденне використання: миттєве оновлення списків завдань, прогресу звичок, календаря та аналітики. Саме SPA дозволяє створити сучасний, приємний інтерфейс, який відповідає очікуванням сучасних користувачів.

### 2.1.5. Обґрунтування вибору

Після проведення аналізу основних архітектурних підходів (монолітної, мікросервісної, багатосторінкової та односторінкової) для комп’ютерної системи планування та моніторингу виконання особистих завдань було прийнято рішення використовувати односторінкову архітектуру SPA із чітким розділенням системи на клієнтську та серверну частину. Цей вибір пов'язаний із тим, як саме має працювати система.

По-перше, система повинна забезпечувати високу інтерактивність користувацького досвіду через інтерфейс. Користувач не повинен чекати по кілька секунд і дивитися на білий екран при кожній відмітці виконаного завдання чи перемиканні дня в календарі. SPA забезпечує відчуття роботи зі звичайним мобільним чи комп'ютерним застосунком, де все реагує на кліки миттєво.

По-друге, система потребує динамічного оновлення даних у реальному часі. Прогрес виконання завдань, оновлення найближчих дедлайнів – все це має відображатися без затримок. Односторінкова архітектура дозволяє оновлювати тільки необхідні частини інтерфейсу, що суттєво підвищує швидкодію та комфорт використання.

По-третє, важливим є зручність щоденного використання. Користувач повинен мати можливість швидко перемикатися між списком завдань, календарем, звичками та аналітикою. SPA-архітектура забезпечує плавну навігацію без «стрибків» сторінок, що характерно для багатосторінкових додатків.

Монолітна архітектура, хоча й проста на початкових етапах, була відхилена через майбутні проблеми з підтримкою та масштабуванням системи. Мікросервісна архітектура визнана надмірно складною для даного проєкту.

Багатосторінкова архітектура не відповідає сучасним вимогам до інтерактивності та користувацького досвіду.

Таким чином, вибір односторінкової архітектури SPA є найбільш оптимальним рішенням. Він найкраще відповідає функціональним вимогам системи, забезпечує сучасний рівень користувацького досвіду, високу швидкодію роботи та створює зручну основу для подальшого розвитку комп'ютерної системи планування та моніторингу виконання особистих завдань.

## 2.2. Вибір технологій для розробки клієнтської частини

Клієнтська частина є основним інструментом взаємодії користувача з системою. Від її якості залежить зручність щоденного використання, швидкодія інтерфейсу, інтерактивність та загальне враження від продукту. Тому вибір технологій для фронтенду мав критично важливе значення. Для реалізації клієнтської частини було розглянуто три найпопулярніші сучасні фронтенд-технології: React [1], Angular [2] та Vue.js [3].

### 2.2.1. React

React – це відкрита JavaScript-бібліотека для створення користувацьких інтерфейсів, розроблена компанією Meta (колишній Facebook) у 2013 році. За більш ніж десятиліття існування вона перетворилася на один із найпоширеніших інструментів у світі фронтенд-розробки та фактично встановила стандарти компонентного підходу до побудови вебінтерфейсів.

Основна ідея React полягає у компонентному підході, де інтерфейс збирається як конструктор із маленьких незалежних частин (компонентів), а спеціальний механізм дозволяє ефективно оновлювати тільки ті елементи екрана, які реально змінилися, без повного оновлення всієї сторінки.

- *Компонентний підхід та повторне використання коду.* Весь інтерфейс сайту розбивається на окремі незалежні блоки (компоненти): кнопка, картка завдання, меню, аналітичний графік. Написавши логіку та дизайн кнопки один раз, розробник може використовувати її у десятках різних місць проєкту. Це значно прискорює написання коду, робить його охайним і полегшує його підтримку.

- *Висока швидкість роботи завдяки Віртуальному DOM (VDOM*). Коли в застосунку змінюються дані, класичний браузер змушений повністю перерахувати і перемалювати сторінку. React натомість підтримує у пам'яті легку віртуальну копію DOM-дерева. При кожній зміні він швидко порівнює нову версію VDOM зі старою і точково оновлює лише ті реальні елементи сторінки, які дійсно змінилися. Завдяки цьому інтерфейс реагує миттєво навіть при частих оновленнях даних.

- *Однонаправлений потік даних.* Дані в React передаються строго зверху вниз. Це робить поведінку інтерфейсу повністю передбачуваною: розробник завжди точно знає, звідки прийшла помилка і який саме шматок коду змінив стан системи.

- *React – це лише бібліотека, а не повноцінний фреймворк.* На відміну від Angular, який одразу має в собі все необхідне, React відповідає тільки за відображення інтерфейсу. Щоб побудувати повноцінний додаток, розробнику доводиться самостійно вибирати та підключати сторонні бібліотеки для налаштування маршрутизації (переходу між сторінками) або управління глобальними даними. Неправильний вибір цих інструментів на старті може ускладнити проєкт у майбутньому.

React добре зарекомендував себе у проєктах, де інтерфейс є складним, динамічним і потребує частого оновлення даних без перезавантаження сторінки. Насамперед це односторінкові застосунки SPA, де користувач

постійно взаємодіє з інтерфейсом і очікує миттєвої реакції. Компонентна модель добре масштабується і дозволяє великим командам розробляти різні частини продукту паралельно та незалежно, тому React широко використовується у великих комерційних продуктах. Водночас для простих проєктів із мінімальною інтерактивністю, таких як лендинг, інформаційні сайти, статичні сторінки, бібліотека може виявитися надлишковим інструментом.

У контексті дипломного проєкту React є потужним і перевіреним рішенням, здатним забезпечити високу інтерактивність і масштабованість. Однак для проєкту середньої складності, така гнучкість часто перетворюється на зайву складність і потребує додаткових зусиль на організацію архітектури.

### 2.2.2. Angular

Angular – це повноцінний фреймворк для розробки вебзастосунків із відкритим вихідним кодом, створений і підтримуваний компанією Google. Перша версія, відома як AngularJS, з'явилася у 2010 році, однак у 2016 році фреймворк був повністю переписаний і перейменований на Angular. Сьогодні Angular активно розвивається і широко використовується у корпоративному секторі.

На відміну від React і Vue.js, Angular є комплексним рішенням, бо він одразу включає всі необхідні інструменти для побудови повноцінного застосунку. Вбудований роутер, HTTP-клієнт, систему форм, інструменти для тестування та власну систему впровадження залежностей. Розробнику не потрібно витрачати час на вибір і налаштування сторонніх бібліотек, бо архітектура проєкту чітко визначена фреймворком із самого початку.

- *Суворо типізована розробка на TypeScript.* Angular використовує TypeScript як основну мову розробки, що не є опціональним вибором, а

обов'язковою умовою. Статична типізація дозволяє виявляти помилки ще на етапі написання коду, покращує автодоповнення у редакторах і робить великі кодові бази значно безпечнішими у підтримці.

- *Двонаправлене зв'язування даних.* Angular підтримує two-way data binding: зміни у моделі даних автоматично відображаються в інтерфейсі, і навпаки дії користувача в інтерфейсі одразу оновлюють модель. Це зручно для форм і інтерактивних елементів, де потрібна постійна синхронізація між станом і відображенням.

- *Чітка структура.* Angular нав'язує жорстку структуру проєкту: модулі, компоненти, сервіси, директиви кожна сутність має своє чітко визначене місце і призначення. Для великих команд це перевага: будь-який розробник, знайомий із Angular, одразу розуміє структуру незнайомого проєкту. Водночас для невеликих проєктів ця структура може здаватися надлишково громіздкою.

Angular найбільше підходить для масштабних корпоративних застосунків із великими командами розробників, де важлива стандартизація підходів, строга типізація і довгострокова підтримка коду. Типові приклади – корпоративні портали, CRM-системи, фінансові платформи і великі SPA із складною бізнес-логікою.

У контексті дипломного проєкту Angular є надлишковим рішенням. Його архітектурна складність і великий обсяг шаблонного коду вимагають значних витрат часу на налаштування та вивчення ресурсів, які в умовах індивідуальної розробки краще спрямувати на реалізацію функціональності продукту.

### 2.2.3. Vue.js

Vue.js – це прогресивний JavaScript-фреймворк створений Еваном Ю у 2014 році. На відміну від Angular, який розроблявся великою корпорацією для корпоративних потреб, Vue з'явився як проєкт одного розробника з метою поєднати найкращі ідеї React і Angular, позбавившись їхніх недоліків. Сьогодні Vue.js підтримується активною спільнотою та є одним із трьох найпопулярніших фронтенд-фреймворків у світі.

Фреймворк можна впроваджувати поступово: починаючи від додавання інтерактивності на окремій частині поточної сторінки і до побудови повноцінного SPA-застосунку з маршрутизацією та управлінням станом. Vue не нав'язує жорсткої архітектури і однаково добре підходить як для невеликих проєктів, так і для великих застосунків.

- *Однофайлові компоненти.* Vue використовує формат .vue-файлів, де розмітка, логіка і стилі одного компонента зберігаються разом в одному файлі. Це робить код самодостатнім і організованим: усе, що стосується конкретного компонента, знаходиться в одному місці, без необхідності стрибати між різними файлами.

- *Впровадження Composition API та повна інтеграція з TypeScript.* Починаючи з версії Vue 3, основним стандартом розробки стало Composition API. Цей підхід дозволяє групувати логіку за функціональними ознаками, а не за типами опцій, що забезпечує відмінну статичну типізацію за допомогою мови TypeScript. Це спрощує проведення рефакторингу та виявлення помилок на етапі компіляції застосунка.

- *Офіційна екосистема суміжних інструментів.* На відміну від React, де вибір бібліотек для маршрутизації та стану повністю покладається на

розробника, Vue пропонує офіційно підтримувані рішення: Vue Router для маршрутизації і Pinia для управління глобальним станом. Це знімає проблему вибору і гарантує сумісність інструментів між собою.

Vue.js добре підходить для проєктів будь-якого масштабу, де важливий баланс між простотою розробки та функціональністю. Фреймворк широко використовується для SPA-застосунків які розробляються невеликими командами або окремими розробниками.

Для комп'ютерної системи планування та моніторингу виконання особистих завдань Vue є оптимальним вибором. Він забезпечує швидку розробку сучасного інтерфейсу, відмінну типізацію TypeScript та зручність підтримки.

### 2.2.4. Обґрунтування вибору

Усі три розглянуті технології є широко використовуваними інструментами з великою спільнотою та активною підтримкою. Однак вони суттєво відрізняються за філософією, складністю і сферою застосування.

Angular є найпотужнішим і найструктурованішим рішенням, однак його складність і громіздкість виправдовують себе лише у великих корпоративних проєктах із командами досвідчених розробників. Для індивідуальної розробки в межах дипломного проєкту він є надлишковим як за архітектурною складністю, так і за часом необхідним для освоєння.

React пропонує гнучкість і потужну екосистему, однак ця гнучкість має зворотний бік: розробнику доводиться самостійно приймати численні архітектурні рішення, вибирати бібліотеки, визначати структуру проєкту, організовувати управління станом. Для одного розробника це означає додаткові витрати часу і ризик прийняти невдале рішення на старті.

Vue.js займає оптимальну позицію для задач даного проєкту. Він поєднує низький поріг входу з достатньою функціональністю для побудови повноцінного SPA-застосунку, пропонує офіційно підтримувану екосистему (Vue Router, Pinia), а Composition API у Vue 3 забезпечує зручну організацію логіки без надлишкової складності. Формат однофайлових компонентів робить код самодостатнім і добре структурованим, що особливо важливо при індивідуальній розробці.

З урахуванням цих факторів для реалізації клієнтської частини дипломного проєкту було обрано Vue 3. Цей вибір забезпечує оптимальний баланс між швидкістю розробки, якістю коду і достатньою функціональністю для реалізації всіх запланованих можливостей застосунку.

## 2.3. Вибір технологій для розробки серверної частини

Серверна частина комп'ютерної системи планування та моніторингу виконання особистих завдань відповідає за бізнес-логіку, автентифікацію користувачів, роботу з базою даних, обробку запитів, безпеку даних та інтеграцію всіх модулів системи. Від вибору серверних технологій залежить швидкодія API, масштабованість, безпека даних, зручність підтримки та можливість розширення функціоналу в майбутньому.

Для реалізації серверної частини було розглянуто три найбільш популярні та сучасні технологічні стеки: Python [4], Java [5] та JavaScript [6].

### 2.3.1. Python

Python – мова програмування загального призначення, що з'явилася у 1991 році і з часом стала однією з найпоширеніших у світі.

Мова вирізняється лаконічним синтаксисом і низьким порогом входу, що робить її популярним вибором. Велика стандартна бібліотека і розвинена

екосистема закривають більшість типових задач серверної розробки. Окремою перевагою є природна інтеграція з інструментами машинного навчання та аналітики даних, це напрями де Python фактично є галузевим стандартом.

Проте для серверної розробки у поєднанні з TypeScript-фронтендом Python має помітні обмеження. Динамічна типізація поступається статичній за надійністю у великих проєктах. Підтримка двох різних підходів до типізації на фронтенді і бекенді ускладнює обмін моделями і типами між частинами застосунку. Крім того, стандартна реалізація мови має обмеження продуктивності при високому паралельному навантаженні.

Для дипломного проєкту Python не є оптимальним вибором насамперед через відсутність єдиного стеку з клієнтською частиною на Vue 3 + TypeScript.

### 2.3.2. Java

Java – строго типізована об'єктно-орієнтована мова програмування, що існує з 1995 року і залишається одним із основних інструментів корпоративної розробки.

Сильні сторони надійність і передбачуваність. Статична типізація дозволяє виявляти помилки ще на етапі компіляції, а справжня багатопоточність забезпечує стабільну продуктивність при інтенсивному навантаженні. Технологія перевірена роками у банківських системах, корпоративних платформах і високонавантажених сервісах, де ціна помилки висока.

Для дипломного проєкту Java є надлишковим інструментом. Її складність і ресурсоємність виправдані у великих командах над довгостроковими корпоративними системами, але не в умовах індивідуальної розробки обмеженого масштабу.

### 2.3.3. JavaScript (Node.js)

JavaScript – мова програмування, яка спочатку створювалася виключно для додавання інтерактивності на стороні клієнта у браузері. Проте поява у 2009 році програмної платформи Node.js [7] повністю змінила екосистему розробки, перетворивши JavaScript на повноцінну та високоефективну серверну мову промислового рівня.

Головна архітектурна особливість Node.js – це відкрите середовище виконання, яке дозволяє запускати JavaScript-код поза межами браузера безпосередньо на стороні сервера. На відміну від традиційних серверних платформ, які виділяють під кожен новий запит окремий системний потік, Node.js обробляє тисячі паралельних з'єднань в одному потоці за допомогою циклу подій. Це забезпечує мінімальні витрати оперативної пам'яті.

У сучасній практиці розробка серверних рішень на базі Node.js майже завжди реалізується із залученням TypeScript. Ця мова є суворо типізованою надмножиною JavaScript, яка додає в екосистему переваги статичної типізації. Впровадження TypeScript у поєднанні з Node.js дозволяє мінімізувати появу архітектурних помилок ще на етапі написання коду, підвищує загальну безпеку серверних модулів та робить систему максимально масштабованою і зручною для тривалого супроводу.

Особливе місце серед серверних фреймворків Node.js займає NestJS – прогресивний фреймворк, архітектура якого натхненна Angular. Він надає потужні інструменти для створення масштабованих застосунків: модульну структуру та вбудовану підтримку TypeScript.

### 2.3.4. Обґрунтування вибору

На основі проведеного аналізу серверних технологій для розробки серверної частини комп’ютерної системи планування та моніторингу виконання особистих завдань було обрано Node.js із використанням фреймворку NestJS та мови програмування TypeScript.

Цей вибір зумовлений такими ключовими факторами:

1. *Єдність технологічного стеку.* Використання TypeScript як єдиної мови розробки для Vue 3 та NestJS забезпечує цілісність кодової бази та спрощує обмін структурами даних.

2. *Висока швидкість обробки запитів.* Node.js гарантує моментальний відгук API при щоденних операціях користувача із завданнями, звичками та календарем.

3. *Надійна архітектура додатка*. Модульний підхід NestJS дозволяє чітко розмежувати зони відповідальності сервера, забезпечуючи легкість відлагодження, високу стійкість до збоїв та гнучкий фундамент для подальшого масштабування комп'ютерної системи.

Таким чином, вибір NestJS + TypeScript на базі Node.js є найбільш раціональним і перспективним рішенням, яке повністю відповідає технічним вимогам проєкту та забезпечує оптимальний баланс між продуктивністю, надійністю та зручністю розробки.

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Другий розділ дипломного проєкту був присвячений обґрунтуванню вибору архітектури, технологій та засобів розробки. У ньому було проаналізовано альтернативні рішення та обґрунтовано вибір односторінкової архітектури SPA, фреймворку Vue 3 з TypeScript для клієнтської частини, фреймворку NestJS з TypeScript для серверної частини.

Третій розділ присвячений безпосередній програмній реалізації комп’ютерної системи планування та моніторингу виконання особистих завдань. Тут розглядаються конкретні технічні рішення, що були реалізовані в процесі розробки.

Метою цього розділу є детальний опис процесу створення програмного забезпечення системи, демонстрація практичного застосування обраних технологій, а також висвітлення основних технічних рішень, які забезпечують функціональність, продуктивність, безпеку та зручність використання системи.

## 3.1. Логічна та технічна архітектура системи

Логічна архітектура системи визначає її основні функціональні блоки, принципи їх взаємодії та роль у загальній роботі системи. У розробленій комп’ютерній системі передбачено такі ключові логічні блоки:

- *Блок автентифікації та авторизації користувачів.*
  Відповідає за реєстрацію нових користувачів, авторизацію, вихід з системи та управління персональними налаштуваннями. Цей блок забезпечує безпеку доступу до всіх даних.

- *Блок управління завданнями*.
  Реалізує повний життєвий цикл завдання: створення, редагування, встановлення пріоритету, дедлайну, позначення виконання, фільтрацію та видалення.

- *Блок календаря*.

  Забезпечує візуальне планування часу, відображення завдань за датами, інтерактивний вибір дня.

- *Блок управління звичками.*

  Відповідає за створення, відстеження та аналіз регулярних дій користувача, підрахунок поточного стріку та фіксацію щоденного виконання.

- *Блок аналітики продуктивності.*

  Збирає, обробляє та візуалізує статистику виконання завдань і звичок за різні періоди (тиждень, місяць, рік), формує ключові показники та інтерактивні графіки.

Система побудована на розподіленій клієнт-сервер архітектурі та складається з трьох основних частин:

1. Клієнтська частина (Frontend) – відповідає за інтерфейс користувача, візуалізацію даних та інтерактивну взаємодію. Реалізована як односторінковий застосунок SPA на базі Vue 3 з використанням TypeScript.

2. Серверна частина (Backend) – забезпечує бізнес-логіку, роботу з базою даних, автентифікацію користувачів та безпеку. Побудована в середовищі Node.js на фреймворку NestJS із використанням TypeScript.

3. База даних (Database) – відповідає за надійне зберігання та організацію всіх даних системи. Для цього інтегровано реляційну СУБД PostgreSQL [8], взаємодія з якою відбувається через бібліотеку TypeORM, що дозволяє описувати сутності у вигляді TypeScript-класів.

На рисунку 3.1 зображено схему загальної технічної архітектуру системи.

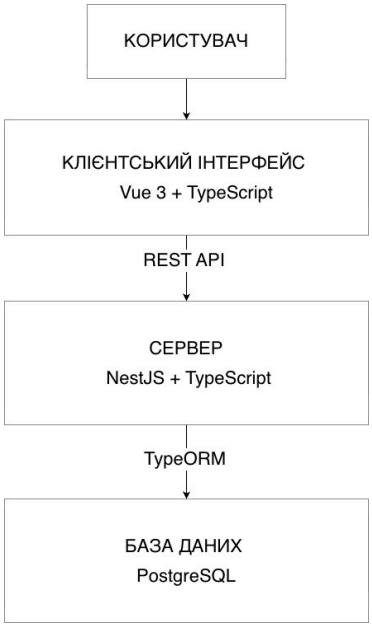



*Рисунок 3.1 – Схема загальної технічної архітектури системи*

Між клієнтською та серверною частинами організовано обмін даними через REST API з використанням JSON-формату. Взаємодія між компонентами відбувається наступним чином:

Користувач через браузер взаємодіє з клієнтською частиною, яка реалізована на фреймворку Vue 3 з використанням TypeScript. При виконанні дій (створення завдання, реєстрація, перегляд аналітики, тощо) фронтенд через бібліотеку Axios надсилає HTTP-запити до серверної частини через REST API. Серверна частина, побудована на фреймворку NestJS з використанням TypeScript, обробляє запити, виконує бізнес-логіку, перевіряє авторизацію за допомогою JWT-токенів та взаємодіє з базою даних через ORM-бібліотеку TypeORM. Результат виконання запиту у форматі JSON повертається на клієнтську частину, де завдяки реактивності Pinia інтерфейс оновлюється без перезавантаження сторінки.

Такий підхід до архітектури дозволив створити сучасну, швидку, безпечну та легко масштабовану систему, яка повністю відповідає поставленим технічним вимогам.

## 3.2. Розробка клієнтської частини системи

Клієнтська частина комп'ютерної системи планування та моніторингу виконання особистих завдань є основним інтерфейсом взаємодії користувача з системою. Вона реалізує всі ключові сценарії: перегляд та управління завданнями, відстеження звичок, роботу з календарем, аналіз продуктивності та управління профілем. Від якості реалізації клієнтської частини безпосередньо залежить зручність, інтуїтивність та загальна ефективність системи.

Система реалізована як односторінковий додаток SPA з використанням фреймворку Vue 3 у поєднанні з мовою TypeScript. Для збирання проєкту застосовано інструмент Vite, який забезпечує високу швидкість обробки модулів.

Для реалізації основних функціональних можливостей клієнтської частини використовувалися такі ключові бібліотеки та інструменти:

- Vue Router 4 – бібліотека для маршрутизації, яка забезпечує навігацію між сторінками без перезавантаження, підтримує динамічні маршрути, та реалізацію захисту маршрутів для авторизованих користувачів;
- Pinia – використовується для централізованого управління даними завдань та звичок, забезпечує реактивність, простоту синхронізації стану між компонентами та повну підтримку TypeScript;
- Axios – HTTP-клієнт для взаємодії з REST API бекенду;
- Chart.js – бібліотека для візуалізації даних, яка використовується в модулі «Аналітика» для побудови інтерактивних графіків (лінійний графік виконання завдань та кругова діаграма пріоритетів);

- Bootstrap 5 + Bootstrap Icons – CSS-фреймворк та набір іконок, що забезпечують адаптивний дизайн, сучасний вигляд інтерфейсу та швидку розробку компонентів;
- TypeScript – основна мова програмування проєкту, яка забезпечує статичну типізацію, покращує читабельність коду, дозволяє виявляти помилки на етапі розробки та значно полегшує рефакторинг і підтримку проєкту.

Таке поєднання технологій дозволяє створювати швидкий, масштабовний, типізований та зручний у підтримці фронтенд, що відповідає сучасним стандартам веб-розробки.

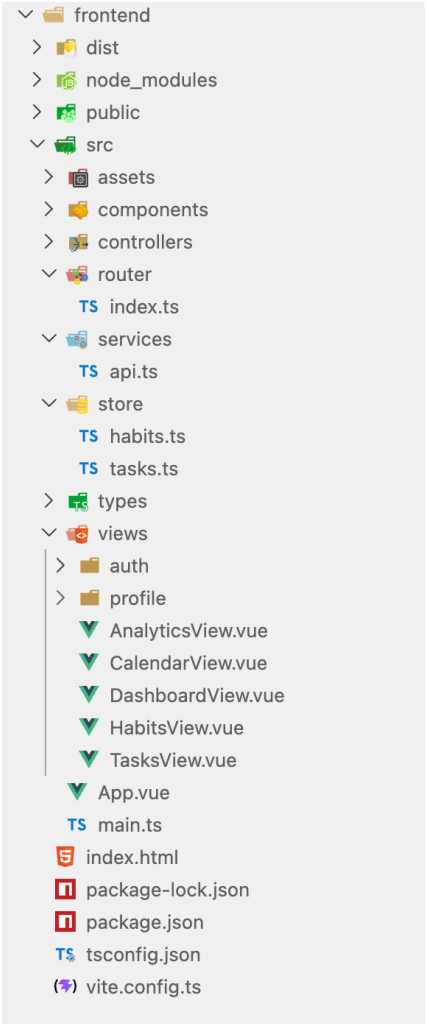



*Рисунок 3.2 – Структура клієнтської частини*

Основний код застосунку зосереджений у директорії src/, яка є робочим простором розробки. Ця директорія містить усі вихідні файли, які після обробки Vite перетворюються у фінальну оптимізовану збірку для виконання в браузері.

Основні каталоги мають таке призначення:

- *assets/* – каталог для статичних ресурсів (зображення, іконки, шрифти).
- *components/* – каталог багаторазових компонентів інтерфейсу. Тут зберігаються універсальні елементи, такі як картки завдань, модальні вікна, елементи навігації.
- *controllers/* – каталог що містить логіку обробки дій користувача та взаємодії між компонентами.
- *router/* – конфігурація маршрутизації за допомогою Vue Router 4. Містить файл index.ts, де визначені всі маршрути, захищені сторінки та логіка перенаправлення.
- *services/* – сервіси для роботи з REST API бекенду.
- *store/* – централізоване управління станом за допомогою Pinia.

  У цій папці розташовані два основних файли:

  - *tasks.ts* – Pinia-сховище для управління завданнями (завантаження, створення, оновлення, видалення, сортування завдань).
  - *habits.ts* – Pinia-сховище для управління звичками (завантаження, створення, оновлення стріку, відстеження виконання за датами).
- *types*/ – централізоване зберігання TypeScript-інтерфейсів та типів даних, що використовуються по всьому проєкту.

- *views/* – каталог основних сторінок застосунку. Кожна сторінка є окремою Vue-компонентою і відповідає за певний розділ системи:
    - DashboardView.vue – головна сторінка;
    - TasksView.vue – управління завданнями;
    - CalendarView.vue – інтерактивний календар;
    - HabitsView.vue – управління звичками;
    - AnalyticsView.vue – модуль аналітики продуктивності;
    - SettingsView.vue – налаштування профілю (у підпапці profile/);
    - ProfileView.vue – відображення інформації профілю (у підпапці profile/);
    - LoginView.vue, RegisterView.vue – сторінки авторизації (у підпапці auth/).

Окрім основних директорій, важливу роль у структурі проєкту відіграють кореневі файли, розташовані безпосередньо в папці frontend/src:

- *App.vue* – кореневий компонент, який містить основну розмітку для відображення сторінок.
- *main.ts* – точка входу в застосунок (створення Vue-додатка, підключення роутера, Pinia, глобальних стилів).

Файли розташовані безпосередньо в папці frontend/ забезпечують точку входу в застосунок, конфігурацію збірки, типізацію та управління залежностями.

- *index.html* – єдиний HTML-файл застосунку.
- *package.json* – файл з метаданими проєкту (назва, версії, списки залежностей тощо).

- *package-lock.json* – автоматично генерований файл, який фіксує точні версії всіх встановлених пакетів та їх залежностей, забезпечуючи стабільність та відтворюваність проєкту.
- *tsconfig.json* – конфігураційні файли TypeScript, що визначають правила компіляції та типізації.
- *vite.config.ts* – конфігураційний файл Vite.

Така структура забезпечує логічне розділення коду за функціональністю, зручність навігації по проєкту та можливість швидкого розширення системи новими модулями.

Розглянемо роутінг клієнтської частини. Маршрутизація реалізована за допомогою бібліотеки Vue Router 4. Основні маршрути системи наведені в таблиці 3.1.

*Таблиця 3.1 – Роутінг клієнтської частини*

| ***Шлях*** | ***Призначення*** |
|---|---|
| / | Головна сторінка |
| /tasks | Сторінка управління завданнями |
| /calendar | Сторінка з інтерактивним календарем |
| /habits | Сторінка управління звичками |
| /analytics | Сторінка аналітики продуктивності |
| /profile | Сторінка профілю користувача |
| /settings | Сторінка налаштування профілю |
| /login | Сторінка авторизації |
| /register | Сторінка реєстрації |

Логіка навігації між екранами та взаємозв’язки основних сторінок системи представлені на Рисунку 3.3.

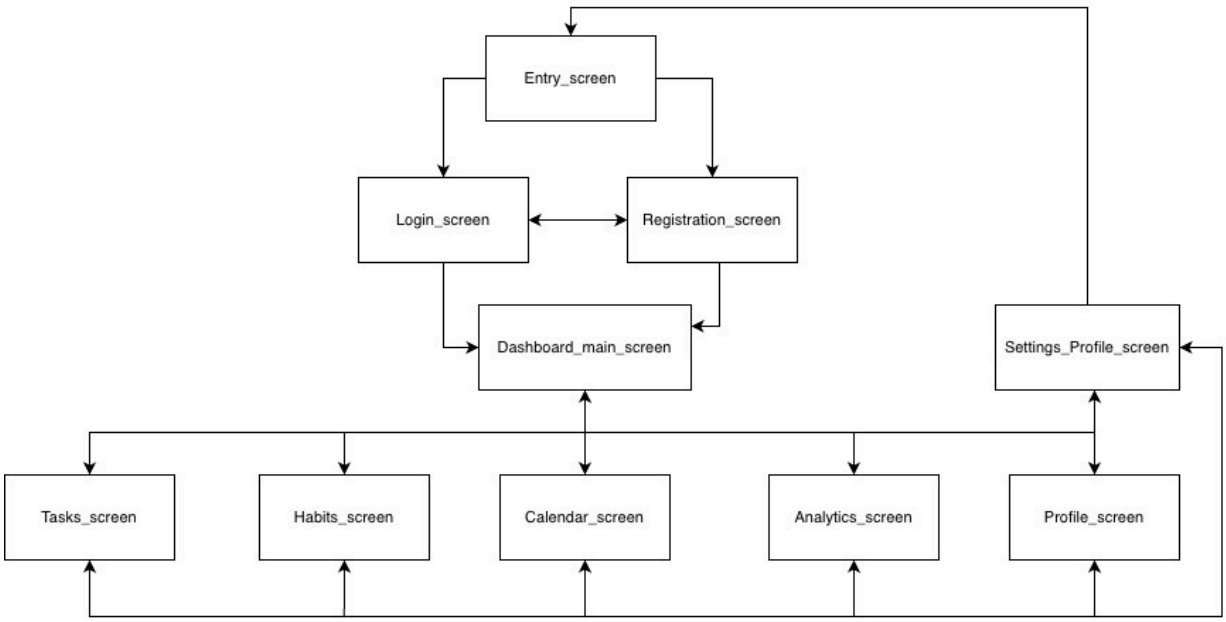



*Рисунок 3.3 – Схема екранів користувача*

На схемі чітко видно, що після успішної авторизації користувач потрапляє на головну сторінку Dashboard, яка є центральним елементом навігації. Звідти через бокове меню здійснюється перехід до основних функціональних модулів. Усі захищені екрани доступні лише авторизованим користувачам, а перехід між ними відбувається без перезавантаження сторінки завдяки Vue Router.

## 3.3. Розробка серверної частини системи

Серверна частина комп’ютерної системи планування та моніторингу виконання особистих завдань забезпечує бізнес-логіку, роботу з базою даних, автентифікацію користувачів та безпеку.

Вона побудована за допомогою середовища Node.js, прогресивного фреймворку NestJS та мови TypeScript. Для надійного збереження даних інтегровано реляційну СУБД PostgreSQL, взаємодія з якою відбувається через

бібліотеку TypeORM. Такий підхід дозволяє описувати структуру бази даних у вигляді типізованих TypeScript-класів (сутностей) та забезпечує зручну роботу з даними.

На рисунку 3.4 показано структуру каталогів серверної частини.

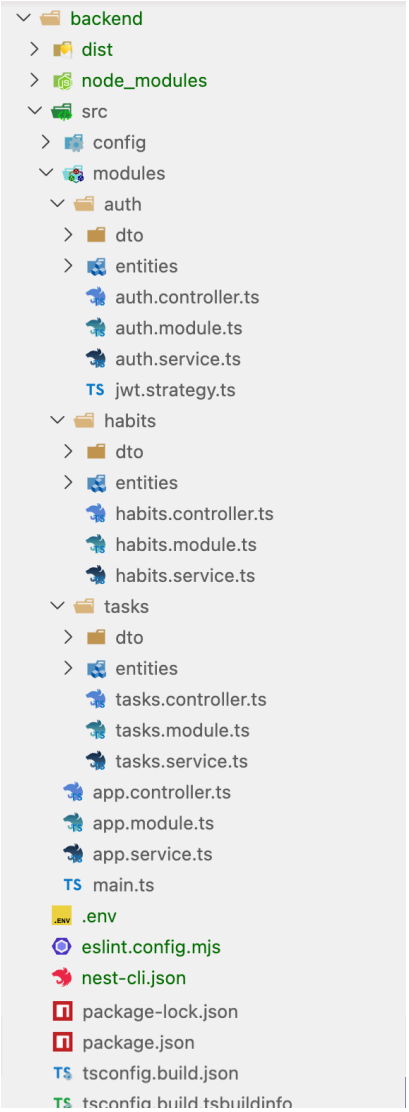



*Рисунок 3.4 – Структура серверної частини*

Серверна частина системи як і клієнтська має чітку модульну структуру, що забезпечує зручність підтримки, масштабованість та логічне розділення відповідальностей. Основний код зосереджений у директорії src/, яка є робочим простором розробки. Це дозволяє чітко відокремити вихідний код від

конфігураційних файлів і скомпільованої версії dist/ та залежностей node_modules/.

Основні каталоги серверної частини мають таке призначення:

- *src/* – головна робоча директорія що містить весь вихідний код.
    - *config*/ – директорія для конфігураційних файлів. Містить в собі файл *typeorm.config.ts* з налаштуваннями підключення до бази даних PostgreSQL
    - *auth*/ – модуль автентифікації та авторизації. Реалізує реєстрацію, вхід користувача, роботу з JWT-токенами, оновлення профілю та завантаження аватара.
    - *tasks*/ – модуль управління завданнями. Забезпечує створення, редагування, видалення завдань, зміну статусу виконання, фільтрацію (включаючи завдання на сьогодні) та сортування.
    - *habits*/ – модуль управління звичками. Реалізує створення, редагування та видалення звичок, фіксацію відміток їх виконання, розрахунок стріку, зберігання історії виконання у масив дат.

При цьому кожен окремий функціональний модуль системи (auth/, habits/, tasks/) містить у собі однакову внутрішню підструктуру для чіткого розподілу відповідальності:

- *dto/* – об’єкти передачі даних (Data Transfer Object), які визначають структуру вхідних даних та використовуються для валідації запитів.
- *entities/* – сутності TypeORM, які безпосередньо відображають структуру та зв'язки таблиць бази даних у вигляді типізованих TypeScript-класів.
- **.controller.ts* – обробники HTTP-запитів.

- **.service.ts* – бізнес-логіка модуля.
- **.module.ts* – конфігурація модуля та підключення залежностей.
    - *main.ts* – точка входу в застосунок. Тут відбувається створення NestJS-додатка, підключення до бази даних та запуск HTTP-сервера.
    - *app.module.ts* – кореневий модуль, який об’єднує всі функціональні модулі системи.

Окрім робочої директорії, важливу роль у структурі проєкту відіграють кореневі файли конфігурації, розташовані безпосередньо в каталозі backend/:

- *tsconfig.json* та *tsconfig.build.json* – конфігураційні файли TypeScript, що визначають правила компіляції та типізації.
- *nest-cli.json* – конфігурація NestJS CLI.
- *package.json* – файл з метаданими проєкту, залежностями та скриптами.

Така структура серверної частини має схожість зі структурою клієнтської частини. Обидві побудовані за принципами модульності, використовують TypeScript, мають чітке розділення. Це забезпечує єдиний стиль розробки по всьому проєкту, а також створює фундамент для подальшого масштабування системи.

### 3.4. Розробка бази даних системи

База даних комп’ютерної системи планування та моніторингу виконання особистих завдань відповідає за надійне зберігання, структурування та організацію всіх даних користувачів. Для реалізації було інтегровано реляційну систему управління базами даних PostgreSQL, яка забезпечує високу швидкість обробки вибірок даних та надійність збереження інформації. Взаємодія між

сервером NestJS та базою даних реалізована за допомогою об’єктно-реляційного відображення (ORM) через бібліотеку TypeORM, що дозволяє описувати схему та структуру таблиць у вигляді типізованих TypeScript-класів (сутностей).

База даних побудована за принципом нормалізації і складається з трьох основних таблиць:

1. users – користувачі системи.
2. tasks – особисті завдання користувача.
3. habits – регулярні звички користувача з історією їх виконання.

Нижче наведено детальний опис кожної таблиці, призначення полів, типи даних та обґрунтування їх використання.

Таблиця users є центральною таблицею системи та зберігає основну інформацію про зареєстрованих користувачів. У таблиці 3.2 представлено її детальний опис.

*Таблиця 3.2 – Опис структури таблиці users*

| ***Поле*** | ***Тип даних*** | ***Обмеження*** | ***Призначення*** |
|---|---|---|---|
| *id* | SERIAL | PRIMARY KEY | Унікальний ідентифікатор користувача |
| *name* | VARCHAR | NOT NULL | Ім’я користувача |
| *email* | VARCHAR | UNIQUE, NOT NULL | Електронна пошта (використовується для входу) |
| *password* | VARCHAR | NOT NULL | Хешований пароль |
| *avatarUrl* | TEXT | NULL | Посилання на аватар профілю |
| *createdAt* | TIMESTAMP | NOT NULL, DEFAULT now() | Дата і час реєстрації |

Таблиця забезпечує ідентифікацію, авторизацію та персоналізацію профілю кожного користувача. Всі інші таблиці системи tasks та habits мають зовнішній ключ *userId*, що встановлює зв’язок «один до багатьох» з таблицею користувачів.

Обґрунтування структури таблиці users:

- Поле *password* зберігає не відкритий пароль, а його криптографічний хеш, що відповідає сучасним стандартам безпеки.
- Обмеження UNIQUE на полі *email* запобігає дублюванню акаунтів.
- Поля *createdAt* автоматично фіксує дату реєстрації користувача.
- Поле *avatarUrl* забезпечує можливість персоналізації профілю зберігаючи посилання на аватар без необхідності зберігати файли безпосередньо в базі даних.

Переходимо до огляду наступної таблиці.

Таблиця tasks призначена для зберігання та управління особистими завданнями користувачів. Вона є одним з основних джерел даних системи та активно використовується в модулях «Завдання», «Календар» та «Аналітика».

Таблиця підтримує повний життєвий цикл завдання, від створення до виконання, та містить усі необхідні атрибути для ефективного планування, контролю та аналізу продуктивності.

Структура таблиці tasks розроблена з урахуванням ключових технічних вимог системи. У таблиці 3.3 представлено її детальний опис.

*Таблиця 3.3 – Опис структури таблиці tasks*

| ***Поле*** | ***Тип даних*** | ***Обмеження*** | ***Призначення*** |
|---|---|---|---|
| *id* | SERIAL | PRIMARY KEY | Унікальний ідентифікатор завдання |
| *userId* | INTEGER | NOT NULL, FOREIGN KEY (ON DELETE CASCADE) | Зовнішній ключ, посилання на конкретного користувача |
| *title* | VARCHAR | NOT NULL | Назва завдання |
| *description* | TEXT | NULL | Детальний опис завдання |
| *dueDate* | DATE | NOT NULL | Дата, до якої потрібно виконати завдання (дедлайн) |
| *time* | TIME | NULL | Час виконання |
| *priority* | VARCHAR | DEFAULT 'medium' | Пріоритет завдання, використовується для сортування (high/medium/low) |
| *completed* | BOOLEAN | NOT NULL, DEFAULT false | Позначка статусу виконання завдання |
| *createdAt* | TIMESTAMP | NOT NULL, DEFAULT now() | Дата створення завдання |

Обґрунтування структури таблиці tasks:

- Поле *id* є первинним ключем і забезпечує унікальну ідентифікацію кожного завдання в системі.
- Поле userId з зовнішнім ключем та каскадним видаленням забезпечує, що при видаленні користувача автоматично видаляються всі його завдання, що підтримує цілісність бази даних.

- Поле *title* є обов’язковим, оскільки назва завдання це основний ідентифікатор для користувача.
- Поле *description* має тип TEXT, що дозволяє зберігати розгорнуті описи завдань без суттєвих обмежень по довжині. Поле є необов’язковим, оскільки не всі завдання потребують детального опису.
- Поля *dueDate* та *time* розділені навмисно. Це дає користувачу гнучкість, бо можна планувати завдання лише на день або з точним часом. Дані з цих полів активно використовуються в календарі та для фільтрації завдань.
- Поле *priority* зі значенням за замовчуванням medium. Таке рішення дозволяє сортувати та візуально виділяти завдання за пріоритетом у інтерфейсі.
- Поле *completed* типу BOOLEAN зі значенням за замовчуванням FALSE є ключовим для відстеження статусу виконання. Воно використовується в аналітиці для розрахунку відсотка виконання завдань, прогресу за період та формування статистики.

Переходимо до огляду наступної таблиці.

Таблиця habits є ключовою для модуля «Звички» та використовується в модулі «Аналітика». Зберігає інформацію про регулярні звички користувача. Основна відмінність від завдань – звички повторюються і мають механізм лічильника який рахує кількість днів поспіль.

Структура таблиці habits розроблена з урахуванням ключових технічних вимог системи. У таблиці 3.4 представлено її детальний опис.

*Таблиця 3.4 – Опис структури таблиці habits*

| ***Поле*** | ***Тип даних*** | ***Обмеження*** | ***Призначення*** |
|---|---|---|---|
| *id* | SERIAL | PRIMARY KEY | Унікальний ідентифікатор звички |
| *userId* | INTEGER | NOT NULL, FOREIGN KEY (ON DELETE CASCADE) | Зовнішній ключ, посилання на конкретного користувача |
| *title* | VARCHAR | NOT NULL | Назва звички |
| *description* | TEXT | NULL | Короткий опис |
| *icon* | VARCHAR | NOT NULL | Емодзі-іконка для візуального розрізнення |
| *streak* | INTEGER | NOT NULL, DEFAULT 0 | Поточний стрік (кількість днів поспіль) |
| *bestStreak* | INTEGER | NOT NULL, DEFAULT 0 | Найкращий стрік (максимальна кількість днів поспіль) |
| *completedDates* | TEXT | NOT NULL, DEFAULT '[]' | Масив дат виконання звички (зберігається як текст) |
| *createdAt* | TIMESTAMP | NOT NULL, DEFAULT now() | Дата створення звички |

Обґрунтування структури таблиці habits:

- Поле *userId* з зовнішнім ключем та каскадним видаленням забезпечує, що при видаленні користувача автоматично видаляються всі його звички, що підтримує цілісність бази даних.
- Поле *title* є обов’язковим, оскільки назва це основний спосіб ідентифікації звички для користувача.
- Поле *description* дозволяє користувачу додавати розгорнутий опис мети або правил виконання звички, що підвищує мотивацію та усвідомлення.

- Поле *icon* (емодзі) призначене для візуального оформлення інтерфейсу. Воно значно покращує користувацький досвід, роблячи список звичок більш привабливим і зручним для сприйняття.

- Поле *streak* зберігає поточну серію (кількість днів поспіль виконання звички).

- Поле *completedDates* зберігає історію виконання у вигляді текстового масиву дат.

На рисунку 3.5 представлено детальну схему взаємозв’язків бази даних, ER-діаграму «сутність-зв’язок», що відображає основні сутності, їх атрибути, типи даних, обмеження та взаємозв’язки між таблицями.

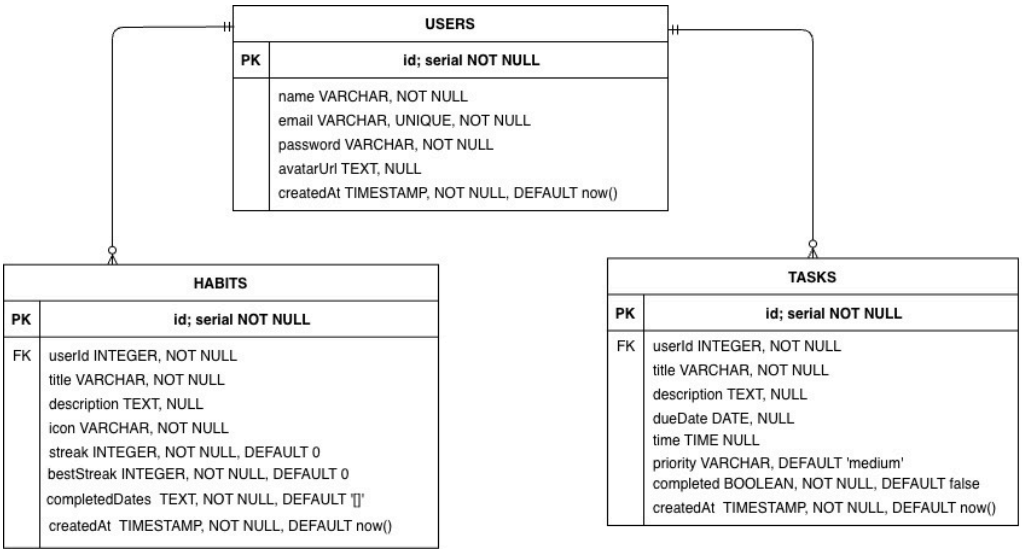



*Рисунок 3.5 – Схема взаємозв’язків бази даних. ER-діаграма*

Структура бази даних складається з трьох взаємопов’язаних таблиць, які в повному обсязі забезпечують виконання всіх функціональних вимог системи планування та моніторингу особистих завдань. Архітектура бази даних побудована з урахуванням принципів модульності, завдяки чому вона легко піддається масштабуванню та розширенню у разі додавання нового функціоналу до комп'ютерної системи.

# 4. ОГЛЯД ІНТЕРФЕЙСУ ТА ТЕСТУВАННЯ СИСТЕМИ

У цьому розділі представлено огляд розробленої комп’ютерної системи планування та моніторингу виконання особистих завдань. Якщо попередні розділи були зосереджені на теоретичному обґрунтуванні, виборі технологій, архітектурі та програмній реалізації, то цей розділ відображає результат у вигляді готового програмного продукту, що відповідає вимогам технічного завдання.

Метою розділу є огляд інтерфейсу системи, демонстрація основних функціональних можливостей з точки зору кінцевого користувача.

## 4.1. Загальний огляд інтерфейсу користувача

Інтерфейс розробленої системи побудовано за принципами SPA з максимальною увагою до зручності, інтуїтивності. Дизайн виконано в сучасному мінімалістичному стилі з використанням фіолетової кольорової гами як основного акцентного кольору, що асоціюється з продуктивністю, креативністю та спокоєм.

Система складається з п’яти основних розділів, доступ до яких здійснюється через бокове меню.

1. Головна сторінка;
2. Сторінка управління завданнями;
3. Інтерактивний календар;
4. Сторінка управління звичками;
5. Аналітика продуктивності.

Для кращого розуміння логіки взаємодії користувача з цими розділами було створено структурну блок-схему користувацьких сценаріїв. На рисунку 4.1 представлено блок-схему дій користувача в комп'ютерній системі.

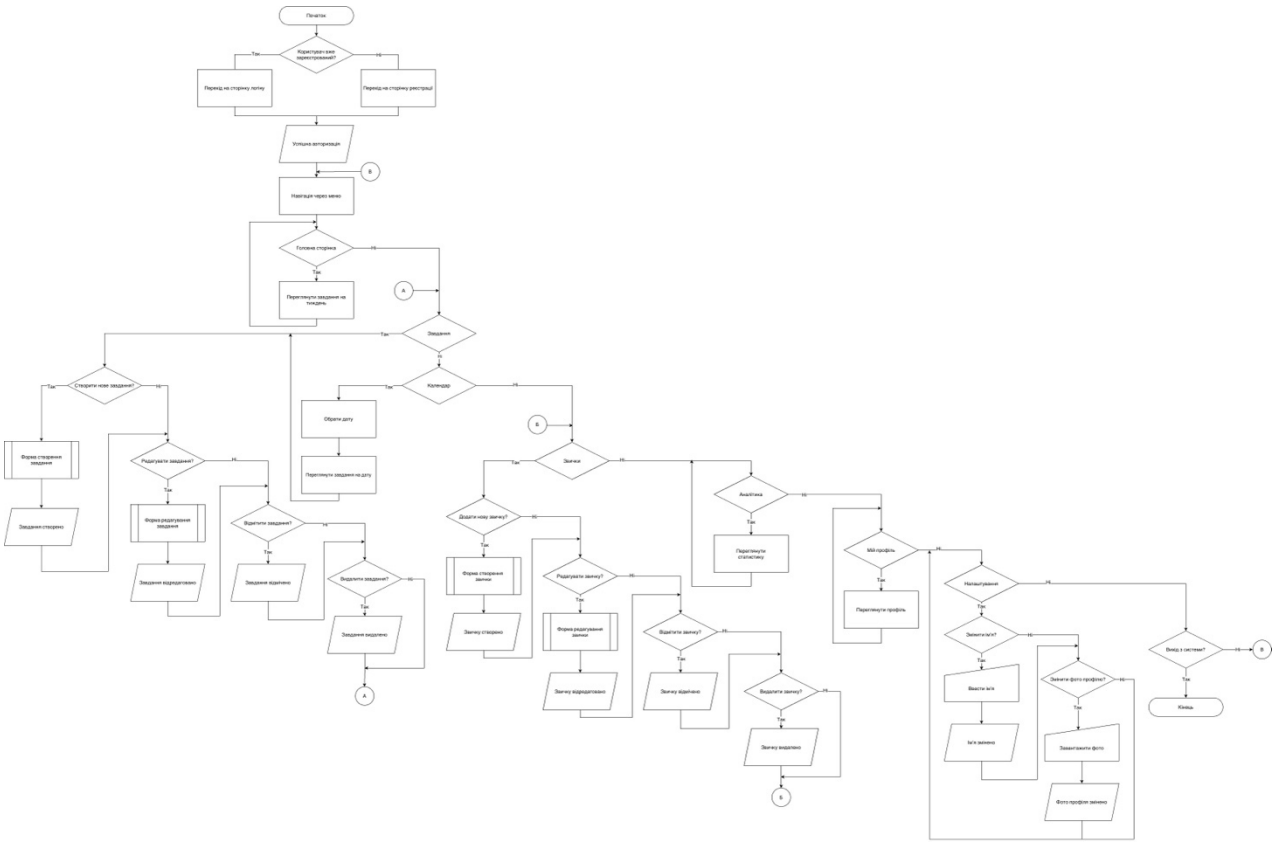



*Рисунок 4.1 – Блок-схема алгоритму дій користувача в системі*

Усі розділи доступні лише авторизованим користувачам, тому неавторизований користувач потрапляє спочатку на сторінку авторизації.

### 4.1.1. Сторінка автентифікації та реєстрації

Оскільки всі основні модулі комп'ютерної системи планування та моніторингу виконання особистих завдань доступні виключно авторизованим користувачам для забезпечення конфіденційності та персоналізації даних, неавторизований користувач при первинному відвідуванні вебзастосунка автоматично потрапляє на сторінку входу.

На рисунку 4.2 зображено графічний інтерфейс сторінки авторизації. Користувач бачить навігаційну панель яка залишається неактивною до моменту підтвердження особи.

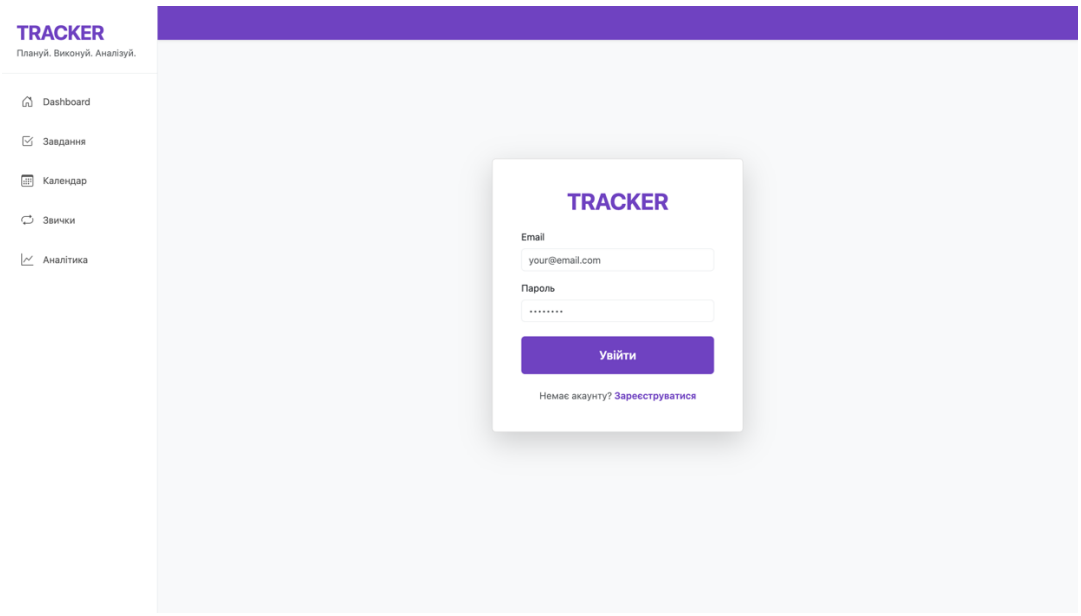



*Рисунок 4.2 – Сторінка авторизації*

Якщо користувач вже має обліковий запис, то він просто заповнює поля «Email» та «Пароль» і натискає кнопку «Увійти», після чого система відразу пропускає його до головної сторінки додатка.

Якщо користувач ще не зареєстрований у системі, йому потрібно натиснути на «Зареєструватися». Після кліку додаток миттєво перенаправляє його на сторінку реєстрації.

На рисунку 4.3 зображено графічний інтерфейс сторінки реєстрації.

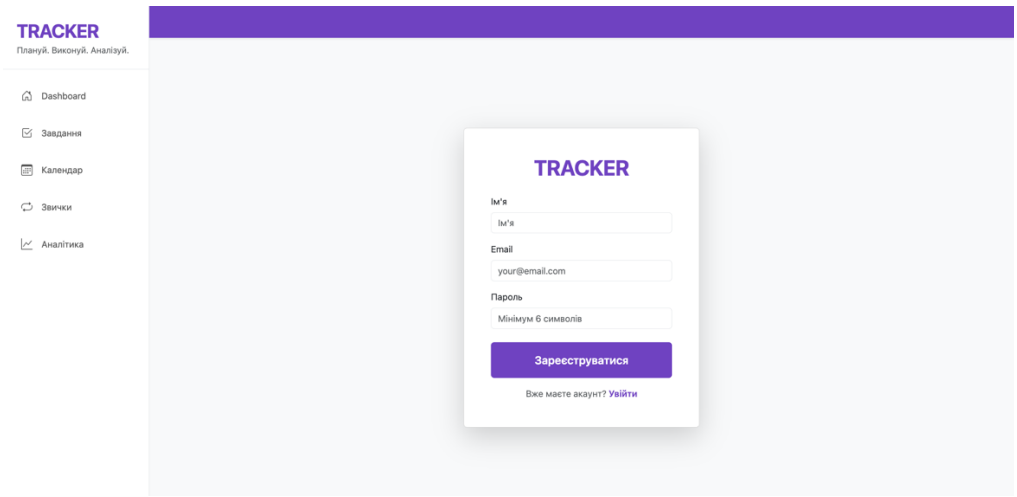



*Рисунок 4.3 – Сторінка реєстрації*

У першому полі форми користувач вказує своє ім'я. Це необхідно для подальшої персоналізації системи. У наступному полі вводиться пошта, яка надалі буде використовуватися як унікальний логін для входу в систему. Користувач придумує та вводить надійний пароль, символи автоматично маскуються крапками задля конфіденційності. Після заповнення всіх полів користувач натискає кнопку «Зареєструватися».

Система уважно стежить за коректністю введення даних. Якщо користувач спробує надіслати порожню форму, введе пошту в неправильному форматі або спробує зареєструватися на ту саму пошту повторно, то додаток не відправить дані, спрацює валідація, а сторінка відразу виведе підказку з описом помилки.

### 4.1.2. Головна сторінка та керування профілем

У разі успішної авторизації або створення нового облікового запису користувач автоматично перенаправляється на головну сторінку системи – Dashboard, а бокове навігаційне меню стає повністю інтерактивним.

На рисунку 4.4 зображено графічний інтерфейс головної сторінки комп'ютерної системи.

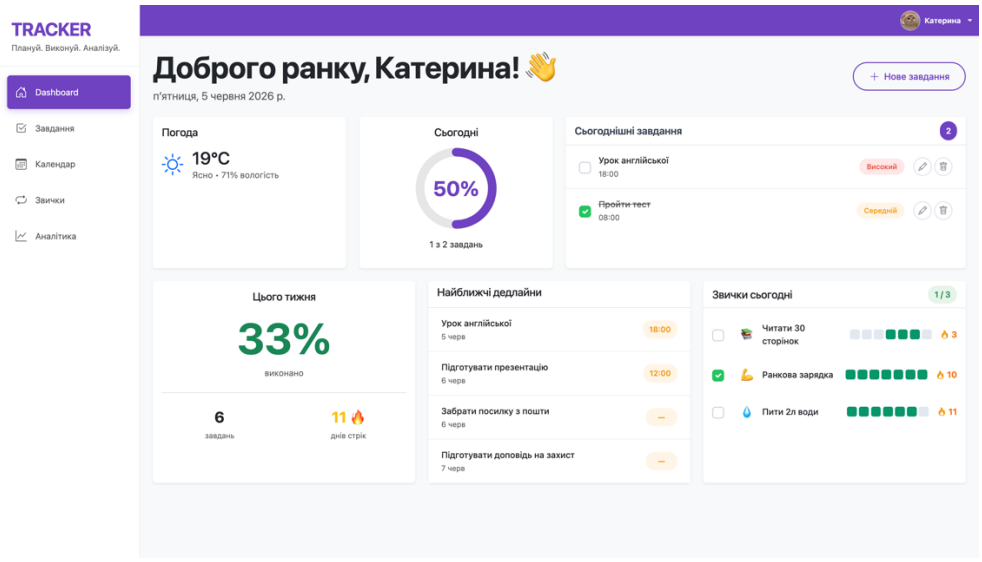



*Рисунок 4.4 – Головна сторінка*

Простір головної сторінки організований у вигляді зручних віджетів, які дозволяють користувачу швидко зорієнтуватися у своїх справах.

У самому верхньому лівому кутку система персоналізовано вітає користувача, відображає поточний день тижня та точну календарну дату. Поруч розташована помітна кнопка «Нове завдання», яка викликає модальне вікно створення завдання, та дозволяє миттєво додати його до списку, не переходячи в інші розділи додатка.

Інформаційний віджет «Погода» показує актуальну температуру повітря, рівень вологості. Це допомагає користувачу комфортно планувати свої завдання поза домом.

У центрі екрана розташована кругова діаграма, вона у відсотках показує яку частину завдань на сьогодні вже виконано. Коли користувач завершує справу, шкала індикатора заповнюється фіолетовим кольором.

Великий віджет праворуч «Сьогоднішні завдання» містить перелік завдань на поточну добу. Користувач може в один клік позначити завдання як виконане (при цьому назва завдання закреслюється). Також є можливість відредагувати завдання або видалити його за допомогою відповідних іконок.

Нижній лівий віджет «Цього тижня» показує у % кількість виконаних завдань за тиждень, нижче кількість всього завдань на тиждень, та лічильник стріку – кількості днів безперервної дисципліни.

Віджет «Найближчі дедлайни» відображає хронологічний список завдань на найближчий тиждень із зазначенням точного часу та дати, що допомагає заздалегідь оцінити майбутнє навантаження.

Правий нижній віджет  містить список щоденних регулярних дій, звичок. Кожна звичка має свою яскраву емодзі-іконку та індикатор, який показує

скільки днів поспіль користувач дотримується цієї корисної звички. Також тут можна і відмітити звічку як виконану.

Для зручного керування особистими даними користувач може взаємодіяти з елементом профілю у правому верхньому кутку. При натисканні на ім’я користувача або його аватар відкривається випадаюче меню швидкого доступу, як зображено на рисунку 4.5.

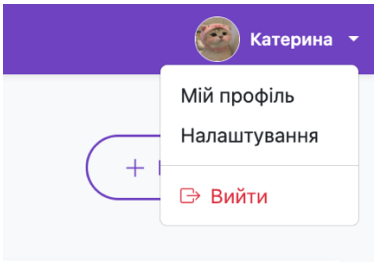



*Рисунок 4.5 – Випадаюче меню керування профілем*

Це меню дозволяє користувачу в один клік обрати один із трьох сценаріїв взаємодії:

- Перехід на сторінку перегляду інформації про поточний обліковий запис «Мій профіль»;
- Перехід на сторінку редагування персональних даних «Налаштування»;
- Вихід із системи «Вийти», користувача буде повернено на початковий екран авторизації.

При виборі пункту «Мій профіль» додаток відкриває сторінку з поточною інформацією про користувача, графічний інтерфейс сторінки представлено на рисунку 4.6.

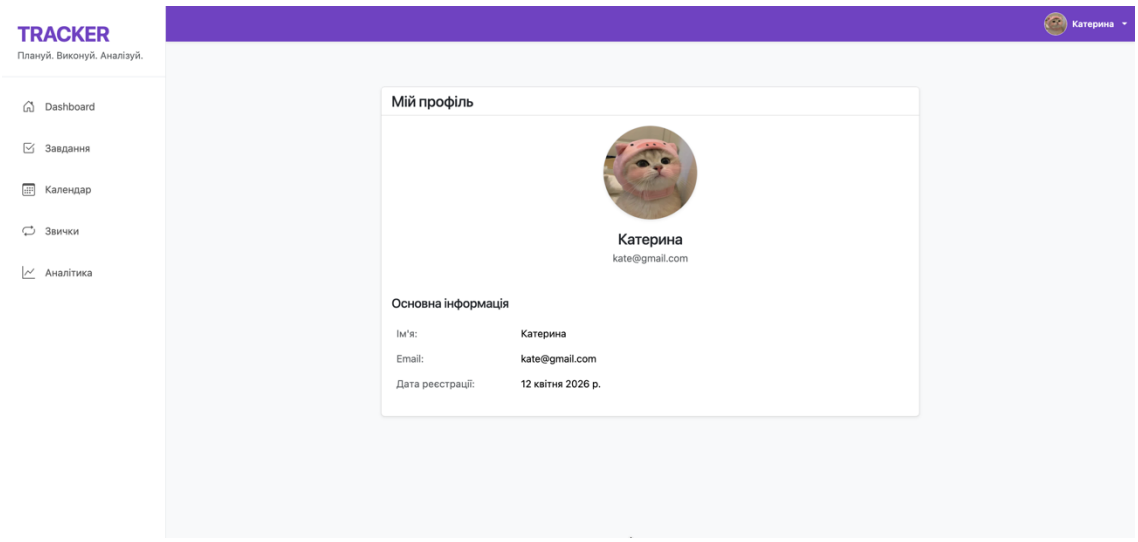



*Рисунок 4.6 – Сторінка перегляду інформації про профіль користувача*

На цій сторінці користувач бачить свій аватар, під яким дублюються основна інформація – ім'я та електронна пошта. У блоці «Основна інформація» виводяться детальні дані облікового запису користувача з бази даних: його повне ім'я, адреса електронної пошти та точна дата реєстрації в системі.

Якщо користувачу необхідно оновити свої дані, він переходить до розділу налаштувань. На рисунку 4.7 зображено графічний інтерфейс сторінки налаштування профілю.

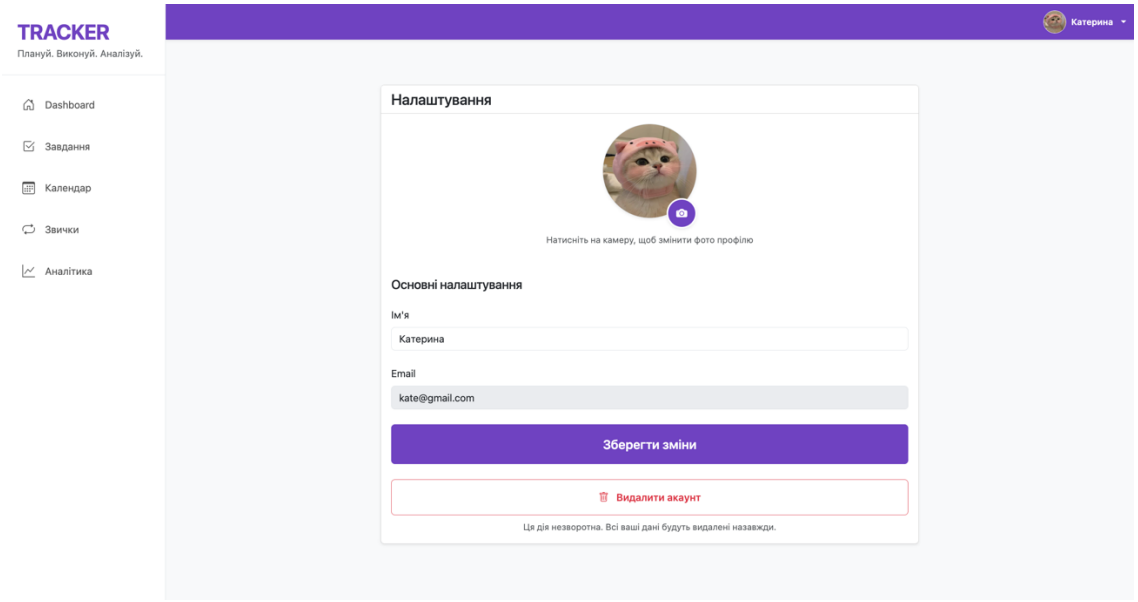



*Рисунок 4.7 – Сторінка налаштування та редагування профілю*

Функціональність цього вікна надає користувачу такі можливості для керування акаунтом:

- *Зміна фото профілю.* Користувач може натиснути на круглу іконку камери, щоб завантажити нове зображення з пристрою та оновити свій візуальний профіль у системі.

- *Редагування персональних даних.* Сторінка містить поле для зміни поточного імені користувача та заблоковане (інформаційне) поле з адресою електронної пошти. Воно заблоковано так як слугує унікальним ідентифікатором користувача. Після внесення змін користувач натискає яскраву фіолетову кнопку «Зберегти зміни», і система миттєво оновлює дані в базі.

- *Видалення облікового запису*. У самому нижньому рядку розташована кнопка «Видалити акаунт», виділена червоним кольором. Під кнопкою є попередження про те, що ця дія є незворотною, а всі особисті завдання, звички та накопичена статистика користувача будуть видалені з комп'ютерної системи назавжди.

### 4.1.3. Сторінка управління завданнями

Для комплексного моніторингу, розподілу та коригування поточних справ у системі передбачено спеціалізований розділ «Завдання».

На рисунку 4.8 зображено графічний інтерфейс сторінки управління завданнями системи.

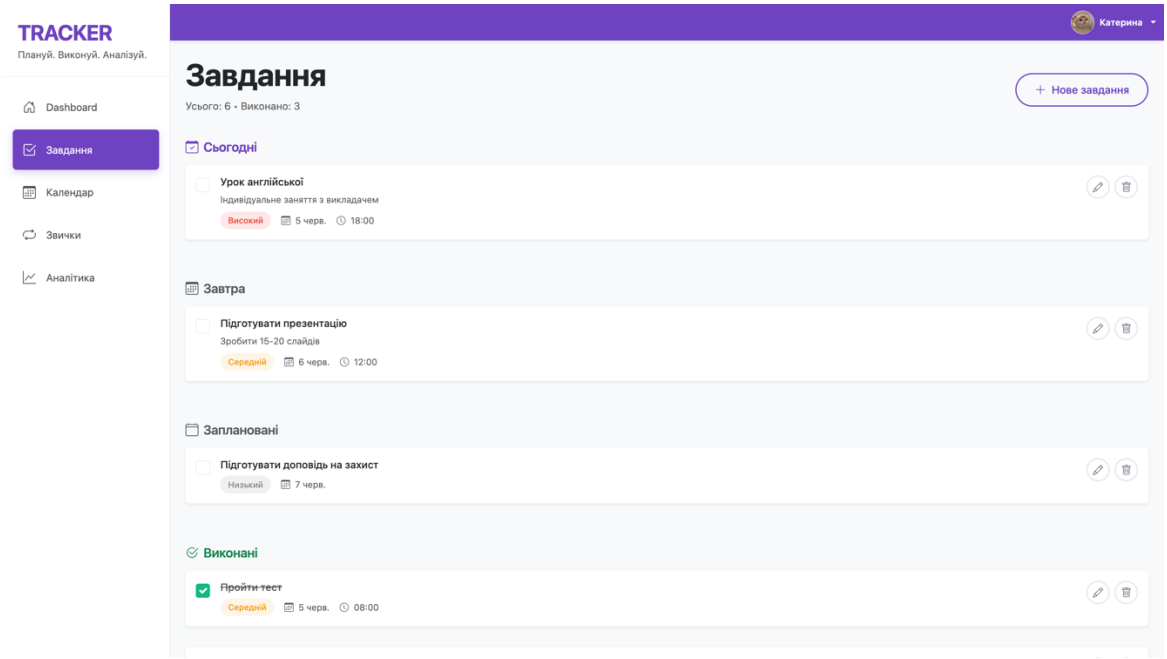



*Рисунок 4.8 – Сторінка управління завданнями*

При переході до цього розділу завдань користувач бачить повний структурований список своїх запланованих справ. Функціональність сторінки дозволяє користувачу виконувати такі дії:

- *Моніторинг загальної кількості завдань*. У верхній частині сторінки, під заголовком «Завдання», система виводить коротку підсумкову статистику. Це дозволяє швидко оцінити загальний обсяг запланованої роботи та кількість уже закритих завдань.

- *Зміна статусу виконання.* Навпроти кожного завдання розташовано інтерактивний прапорець (чекбокс). Користувач може в один клік позначити справу як виконану. При цьому система автоматично зафарбовує прапорець зеленим кольором, візуально закреслює назву завдання та миттєво оновлює загальний лічильник виконаних справ у верхній частині екрана.

- *Кольорове маркування пріоритетів.* Користувач може легко орієнтуватися у важливості та терміновості справ завдяки пріоритетності:

- *Високий пріоритет* – маркується червоним кольором та написом «Високий» (для критично важливих завдань);
- *Середній пріоритет* – маркується жовтим кольором та написом «Середній» (для поточних повсякденних справ);
- *Низький пріоритет* – маркується сірим кольором та написом «Низький» (для нетермінових або побутових задач).

- *Контроль дедлайнів та часу.* Користувач одразу бачить точну дату та встановлений час виконання, що унеможливлює пропуск важливого дедлайну. Якщо завдання не має точного часу, відображається лише дата.
- *Перегляд детальних нотаток.* Під назвою кожного активного завдання виводиться короткий розгорнутий опис, який користувач додав при створенні завдання.
- *Створення, редагування, видалення завдань.* За допомогою кнопки «Нове завдання» у правому верхньому кутку користувач може додати нове завдання. На рисунку 4.9 зображено зміст модального вікна що з’являється при натисканні цієї кнопки.

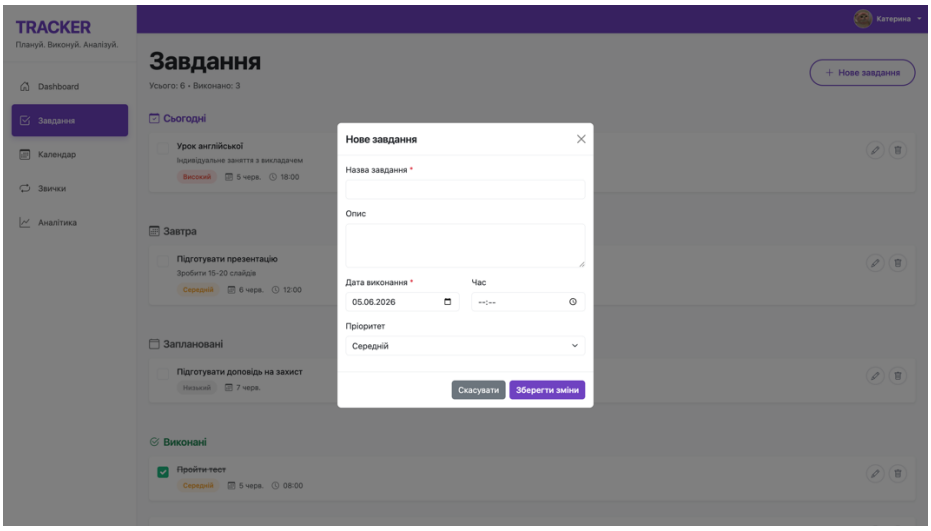



*Рисунок 4.9 – Модальне вікно створення нового завдання*

Процес створення є інтуїтивно зрозумілим і передбачає заповнення таких полів форми:

1. *Назва завдання.* Обов'язкове текстове поле (позначене червоною зірочкою), куди вводиться короткий зміст задачі.
2. *Опис завдання.* Необов'язкове поле для деталізації кроків або нюансів виконання.
3. *Дата виконання.* Обов'язкове поле для вибору конкретного дня.
4. *Час виконання.* Необов'язкове поле для встановлення точного часу дедлайну.
5. *Пріоритет.* Випадаючий список, де користувач обирає один із трьох рівнів важливості (Низький, Середній, Високий).

Після заповнення форми користувач натискає кнопку «Зберегти зміни», дані відправляються на сервер через API-запит, і нове завдання миттєво з'являється у загальному списку.

У разі зміни планів користувач має можливість скоригувати параметри будь-якого вже існуючого завдання. Для цього у правій частині картки завдання передбачено дві інтерактивні іконки. Натискання на іконку червоного кошика призводить до безповоротного видалення завдання з бази даних. Натискання на іконку олівця активує модальне вікно редагування, графічний інтерфейс якого представлено на рисунку 4.10.

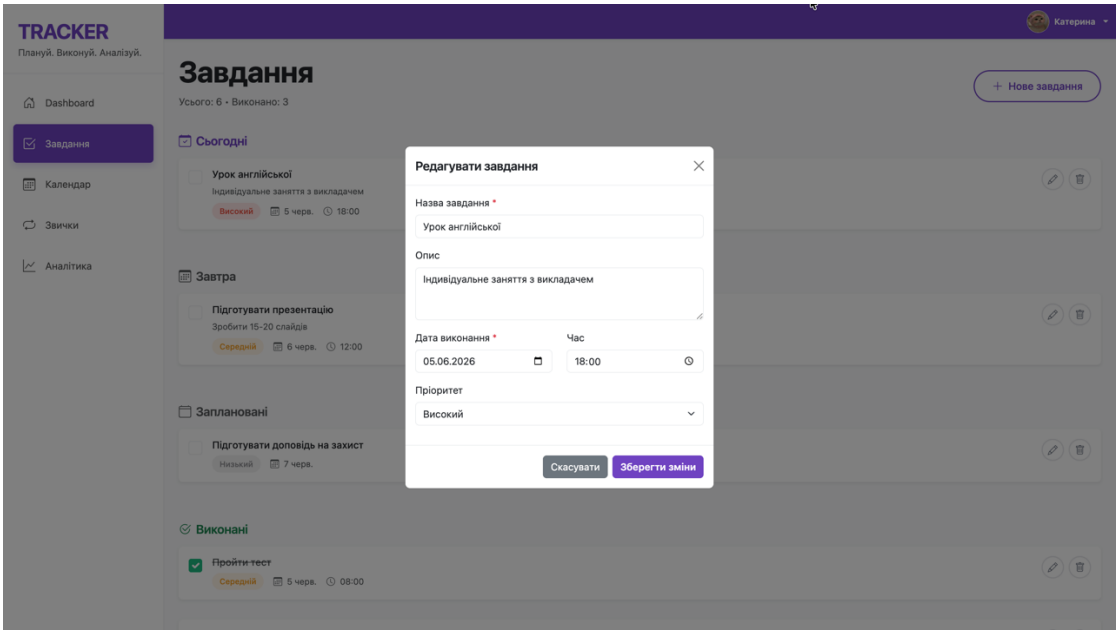



*Рисунок 4.10 – Модальне вікно редагування завдання*

Це вікно автоматично підтягує всі актуальні дані вибраної задачі з бази даних. Користувач може змінити будь-яке поле (назву, опис, пріоритет чи терміни), після чого натискає кнопку «Зберегти зміни». Завдяки принципам SPA-архітектури, інтерфейс головної сторінки оновлюється динамічно, відображаючи актуальний стан завдання без повного перезавантаження сторінки у браузері.

### 4.1.4. Інтерактивний календар

Для глобального планування та візуалізації завантаженості протягом місяця користувач використовує розділ «Календар». Якщо сторінка завдань подає справи у вигляді загального списку, то цей модуль дозволяє побачити розподіл завдань у календарній сітці, що допомагає краще розуміти часові рамки та вільні дні.

На рисунку 4.11 зображено графічний інтерфейс сторінки інтерактивного календаря.

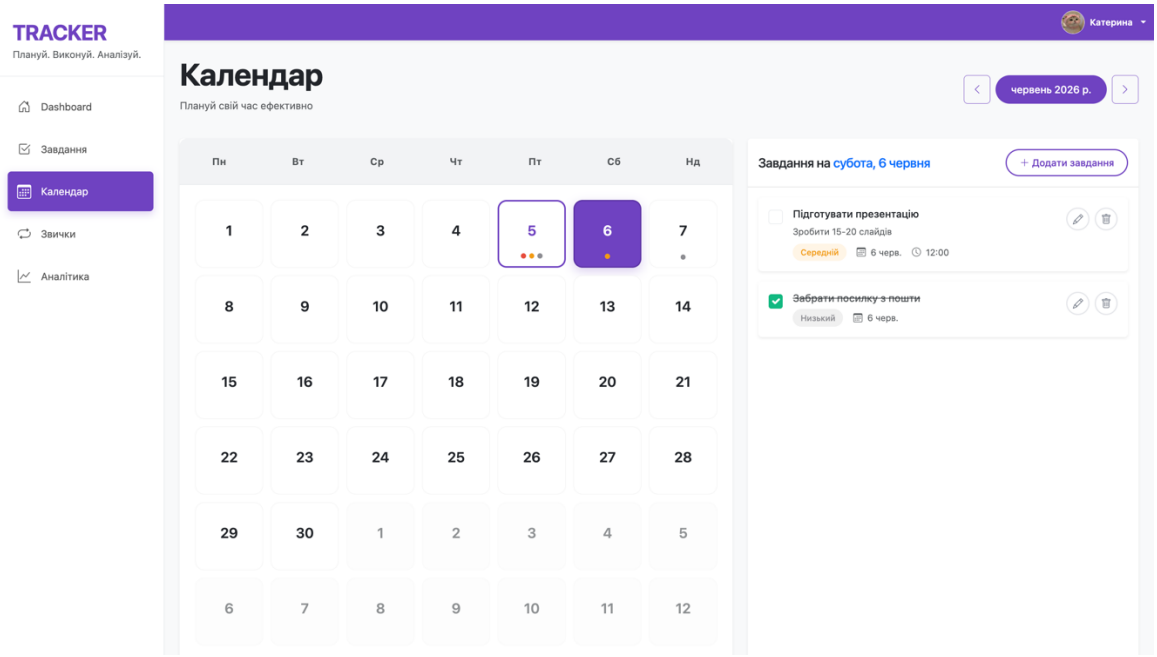



*Рисунок 4.11 – Інтерактивний календар*

Інтерфейс сторінки логічно розділений на дві основні робочі зони, які тісно взаємодіють між собою:

- *Календарна сітка (ліва частина).* Користувач бачить повноцінний місячний календар. Справа у верхній частині розташовані кнопки навігації (стрілки), які дозволяють легко перемикатися між місяцями. Під числами, на які заплановані справи, система автоматично відображає невеликі крапки-маркери що відповідають пріорітетам. Це дозволяє користувачу з першого погляду оцінити, наскільки насиченим є той чи інший день. Поточний день підсвічується фіолетовим контуром, а обраний день підсвічується фіолетовим для зручності орієнтації.

- *Панель денних завдань (права частина).* Коли користувач натискає на будь-яке число в календарі, у правій частині екрана миттєво з'являється блок «Завдання на...» із зазначенням обраної дати та дня тижня. Тут виводиться детальний перелік усіх справ, запланованих саме на цей день.

Користувачу достатньо клікнути на будь-яку дату в сітці, щоб побачити список справ праворуч. Картки завдань у календарі ідентичні за виглядом та логікою тим, що знаходяться у розділі «Завдання»: вони містять назву, опис, точний час та пріоритет.

Безпосередньо з вікна календаря користувач може додати нове завдання на обраний день за допомогою кнопки «додати завдання», також користувач може змінити деталі вже існуючого або видалити його за допомогою знайомих іконок олівця та кошика. Будь-які зміни, внесені тут, автоматично синхронізуються із загальним списком завдань та головною сторінкою.

### 4.1.5. Сторінка управління звичками

Для формування регулярних корисних дій та відстеження щоденної дисципліни користувач переходить до розділу «Звички». На відміну від одноразових завдань, цей модуль орієнтований на тривалий моніторинг повторюваних процесів і розрахунок безперервних серій виконання.

На рисунку 4.12 зображено графічний інтерфейс головної сторінки управління звичками.

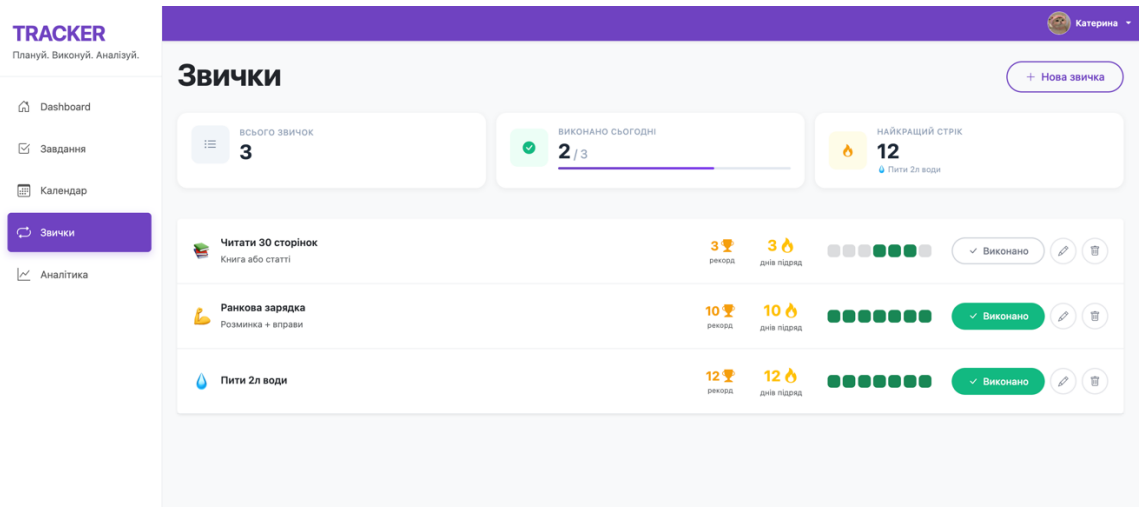



*Рисунок 4.12 – Сторінка управління звичками*

При переході до розділу, користувач бачить статистику по звичкам у вигляді зручних віджетів та список звичок, де кожна звичка представлена окремим рядком зі своїми унікальними інтерактивними елементами:

- *Візуальна ідентифікація*. Кожна звичка має власну яскраву емодзі-іконку, назву та короткий опис-уточнення.
- *Моніторинг серії (стріку).* Для кожного рядка система відображає золотистий лічильник днів поспіль та рекорд по стріку. Це головний мотивуючий елемент, який показує тривалість утримання корисної звички.
- *Відстеження тижневого прогресу.* Праворуч від лічильника розташовано сім квадратних індикаторів, що відповідають дням. Зафарбований зеленим кольором кубик означає, що звичку в цей день було успішно виконано, а сірі кубики вказують на майбутні дні або пропуски.
- *Кнопка відмітки.* Натисканням на кнопку «Виконано» користувач фіксує дію за поточну добу. При цьому система автоматично збільшує лічильник стріку на одиницю та зафарбовує кубик у прогресі. Якщо поточний стрік став більшим за попередній рекорд, автоматично оновлюється значення рекорду. Бо рекорд це постійне значення, яке не скидається і служить мотиваційним показником прогресу.

Для додавання нової звички користувач натискає велику кнопку «Нова звичка» у правому верхньому кутку сторінки. На рисунку 4.13 зображено вміст модального вікна, що з'являється після натискання.

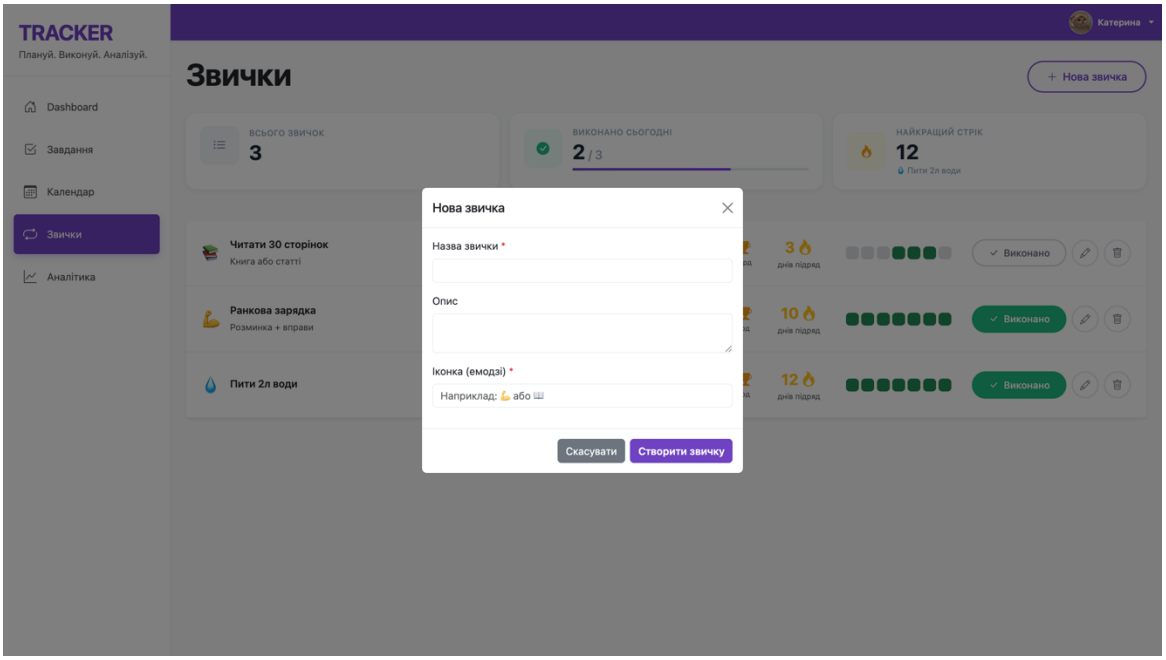



*Рисунок 4.13 – Модальне вікно створення нової звички*

У цьому вікні користувачу доступні такі поля для заповнення:

1. *Назва звички.* Обов'язкове для заповнення текстове поле, де вказується суть дії. Поле позначене червоною зірочкою, що вказує на обов'язковість введення.

2. *Опис.* Додаткове поле, де користувач може деталізувати умови виконання звички.

3. *Іконка (емодзі).* Обов'язкове для заповнення поле, де користувач вписує тематичну іконку для красивого візуального відображення звички у загальних списках звичок та на головній сторінці.

Якщо користувачу необхідно внести зміни в уже існуючу регулярну дію, він натискає на іконку олівця у правій частині рядка звички. На рисунку 4.14 зображено вміст вікна редагування.

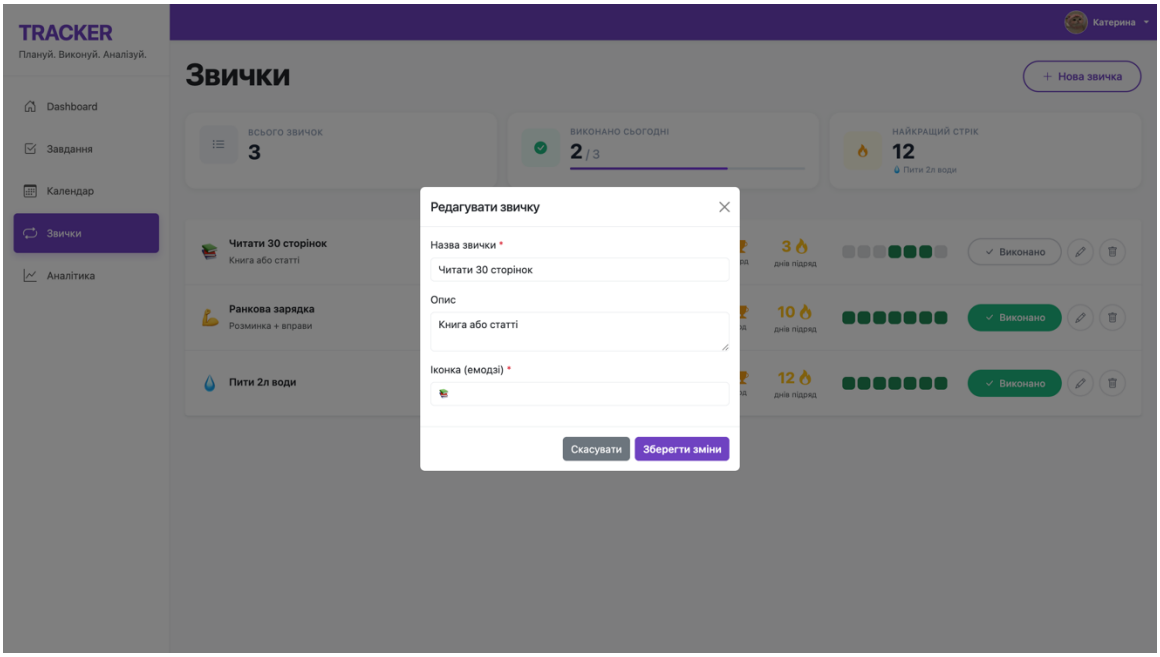



*Рисунок 4.14 – Модальне вікно редагування звички*

У цьому вікні користувач може скоригувати назву, опис або змінити іконку, після чого натискає кнопку «Зберегти зміни» для миттєвого оновлення даних у системі. Також, якщо звичка втратила актуальність, користувач може повністю видалити її разом з усім прогресом, натиснувши на іконку кошика на головній сторінці управління звичками.

### 4.1.6. Сторінка аналітики

Для підбиття підсумків, оцінки індивідуального прогресу та аналізу ефективності виконання завдань і звичок у системі реалізовано розділ «Аналітика». Цей модуль автоматично збирає дані про активність користувача та візуалізує їх у вигляді інформаційних віджетів, інтерактивних графіків і статистичних метрик, що дозволяє виявляти закономірності у продуктивності та коригувати плани на майбутнє.

На рисунку 4.15 зображено графічний інтерфейс сторінки аналітики продуктивності.

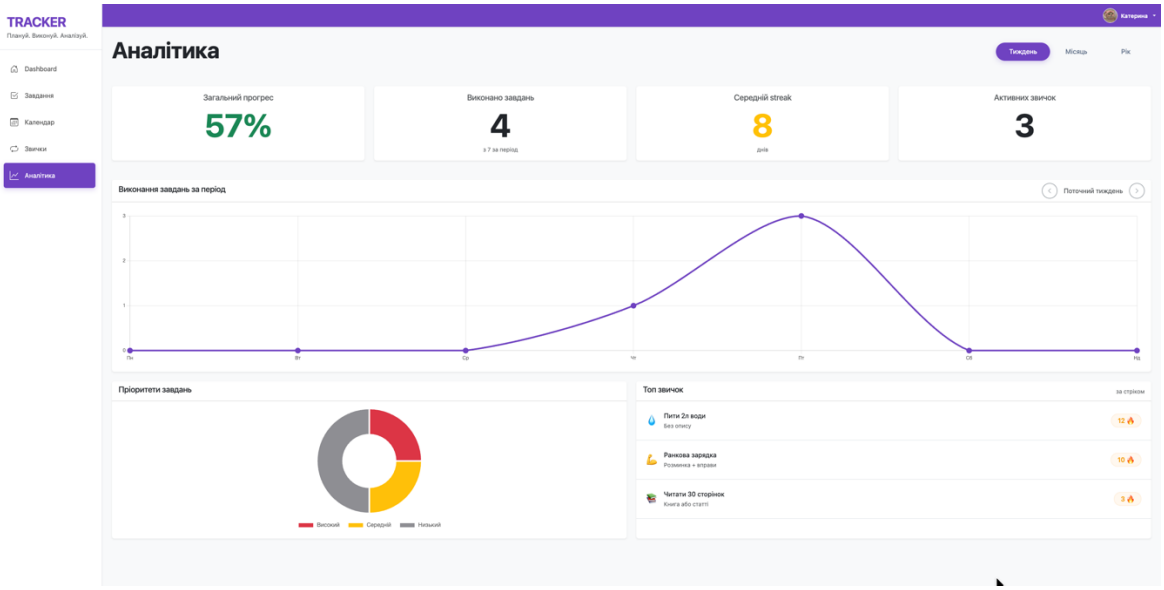



*Рисунок 4.15 – Сторінка аналітики*

Інтерфейс розділу розроблений із фокусом на максимальну наочність великих масивів даних. У правому верхньому кутку користувачеві доступний перемикач часових інтервалів («Тиждень», «Місяць», «Рік»), який дозволяє гнучко змінювати масштаб фільтрації та відображення інформації.

Простір сторінки логічно розділений на кілька ключових аналітичних блоків спостереження:

- *Панель зведених метрик (верхній ряд).* Блок складається з чотирьох інформаційних карток, що відображають головні показники ефективності за обраний період:
    - *«Загальний прогрес»* – інтегральний відсотковий показник успішності виконання запланованого обсягу завдань;
    - *«Виконано завдань»* – кількісне співвідношення завершених завдань до їх загальної кількості за вказаний проміжок часу;
    - *«Середній streak»* – усереднений показник регулярності утримання звичок, виражений у днях безперервного дотримання;

- *«Активних звичок»* – загальна кількість звичок, які користувач наразі відстежує у системі.

- *Графік динаміки виконання завдань (центральна частина).* Представлений у вигляді лінійної діаграми, яка відображає коливання щоденної продуктивності користувача протягом обраного періоду. Кожна точка на графіку дозволяє користувачеві чітко відстежувати піки своєї активності та виявляти дні спаду продуктивності.

- *Діаграма розподілу пріоритетів (ліва нижня частина).* Створена у вигляді кругової діаграми, яка наочно демонструє відсоткове співвідношення між завданнями різного рівня важливості, що були успішно закриті за аналізований період. Кожен сектор діаграми зафарбований у колір відповідного пріоритету. Це допомагає користувачеві оцінити, на задачі якого типу він витрачає найбільше своїх ресурсів.

- *Блок «Топ звичок» (права нижня частина).* Відображає інтерактивний список регулярних дій. Це дозволяє тримати у фокусі найбільш успішні та стабільні звички.

Завдяки інтеграції цього модуля користувач комп'ютерної системи отримує не просто інструмент для сухої фіксації щоденних справ, а повноцінну систему аналітичного аналізу власної дисципліни, що є ключовим елементом вебзастосунку для моніторингу та виконання особистих завдань.

# ВИСНОВКИ

У межах виконання дипломного проєкту було повністю спроєктовано та розроблено комп’ютерну систему планування та моніторингу виконання особистих завдань. Створений вебзастосунок забезпечує комплексний підхід до організації щоденних завдань, ведення обліку звичок та автоматизованого розрахунку метрик для відстеження індивідуального прогресу. Розроблене програмне забезпечення повністю відповідає всім технічним вимогам проєкту.

Попередній аналіз наявних програмних аналогів дозволив виявити їхні типові недоліки, серед яких головними є надмірна архітектурна складність та перевантаженість зайвими функціями. Це підтвердило доцільність розробки інтуїтивно зрозумілого вебзастосунку, який поєднує гнучке короткострокове планування, довгостроковий моніторинг особистої дисципліни та аналітику в межах єдиного робочого простору. Обрана односторінкова архітектура додатка SPA забезпечила високу плавність роботи користувацького інтерфейсу, мінімальний час завантаження екранів та миттєвий відгук системи без постійного перезавантаження вебсторінок у браузері.

Програмна реалізація системи базується на сучасному технологічному стеку. Клієнтську частину побудовано за допомогою фреймворку Vue 3, а серверну бізнес-логіку розгорнуто на базі фреймворку NestJS у середовищі Node.js. Використання мови TypeScript для обох частин додатка забезпечило сувору типізацію та чистоту коду клієнта та сервера, а обмін даними організовано через REST API у форматі JSON. Для збереження інформації використано реляційну СУБД PostgreSQL, взаємодія з якою реалізована за допомогою бібліотеки TypeORM.

Перспективи подальшого розвитку та масштабування комп’ютерної системи полягають у розширенні її функціональних можливостей. На базі вже створеного програмного інтерфейсу є можливість реалізувати мобільний

застосунок для забезпечення постійного доступу користувачів до своїх планів з будь-яких пристроїв. Також гнучка архітектура дозволяє впровадити алгоритми машинного навчання для глибокого аналізу поведінкових патернів користувача, що забезпечить автоматичну генерацію персоналізованих підказок щодо оптимізації щоденного графіку.

Таким чином, розроблена комп’ютерна система повністю відповідає всім технічним вимогам та є перспективним вебзастосунком, який успішно вирішує поставлені завдання. Створене рішення має високу практичну цінність у сфері підвищення особистої продуктивності та самоорганізації.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. React – The library for web and native user interfaces [Електронний ресурс]. – Режим доступу: https://react.dev/
2. Angular – The web development platform for building apps [Електронний ресурс]. – Режим доступу: https://angular.dev/
3. Vue.js – The Progressive JavaScript Framework [Електронний ресурс]. – Режим доступу: https://vuejs.org/
4. Welcome to Python.org [Електронний ресурс]. – Режим доступу: https://www.python.org/
5. Java | Oracle [Електронний ресурс]. – Режим доступу: https://www.java.com/
6. JavaScript.com [Електронний ресурс]. – Режим доступу: https://www.javascript.com/
7. Node.js – Run JavaScript Everywhere [Електронний ресурс]. – Режим доступу: https://nodejs.org/
8. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/

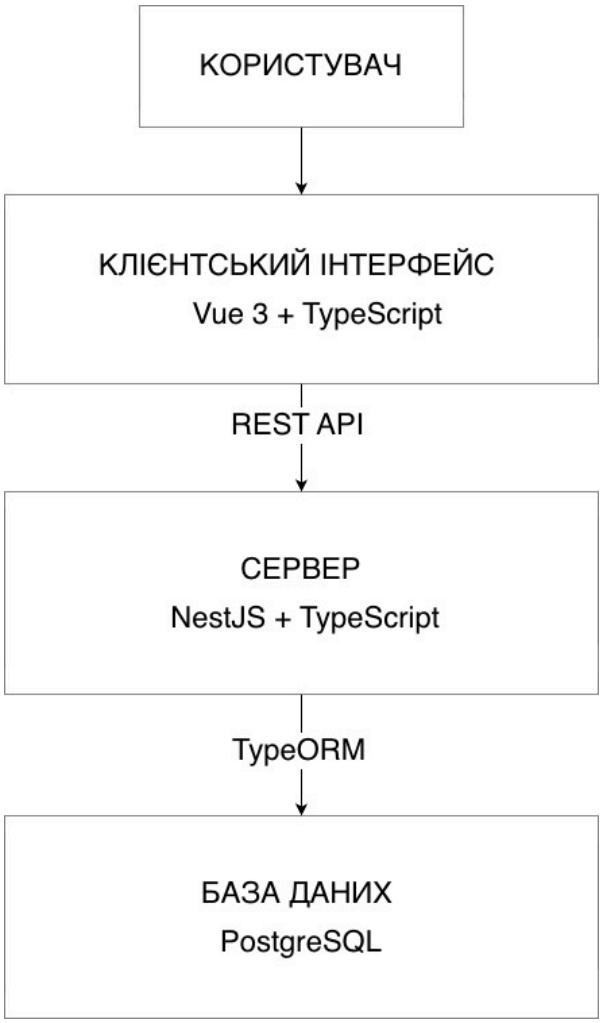
**ДОДАТОК 1**

**Схема загальної технічної архітектури системи. Схема структурна**.

ІАЛЦ. 045440.005 Д1

Аркушів 1

Київ – 2026




| Зм | Лист | № докум. | Підп. | Дата | ІАЛЦ.045440.005 Д1 | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Гречишкіна К.Д. | | | Комп'ютерна система планування та моніторингу виконання особистих завдань | ***Літ.*** | ***Аркуш*** | ***Аркушів*** |
| Перевірив | | Кучмій О.О. | | | | | 1 | 1 |
| Н. контр. | | Клятченко Я.М. | | | ***Схема загальної технічної архітектури системи. Схема структурна.*** | КПІ ім. Ігоря Сікорського ФПСПМ , КВ-22 | | |
| Затвердив | | Романкевич В.О. | | | | | | |

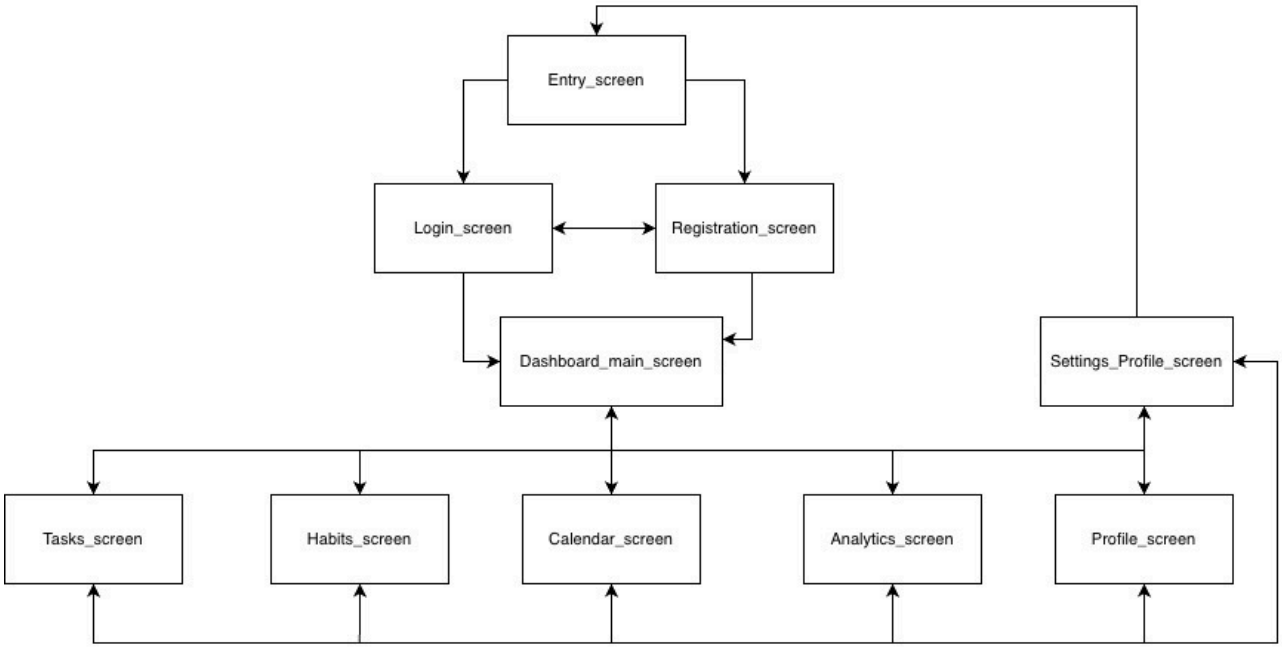
**ДОДАТОК 2**

**Схема екранів користувача. Схема структурна**.

ІАЛЦ. 045440.006 Д2

Аркушів 1

Київ – 2026




| Зм | Лист | № докум. | Підп. | Дата | ІАЛЦ.045440.006 Д2 | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Гречишкіна К.Д. | | | Комп'ютерна система планування та моніторингу виконання особистих завдань | *Літ.* | *Аркуш* 1 / *Аркушів* 1 |
| Перевірив | | Кучмій О.О. | | | | | |
| Н. контр. | | Клятченко Я.М. | | | ***Схема екранів користувача.*** ***Схема структурна.*** | КПІ ім. Ігоря Сікорського ФПСПМ , КВ-22 | |
| Затвердив | | Романкевич В.О. | | | | | |

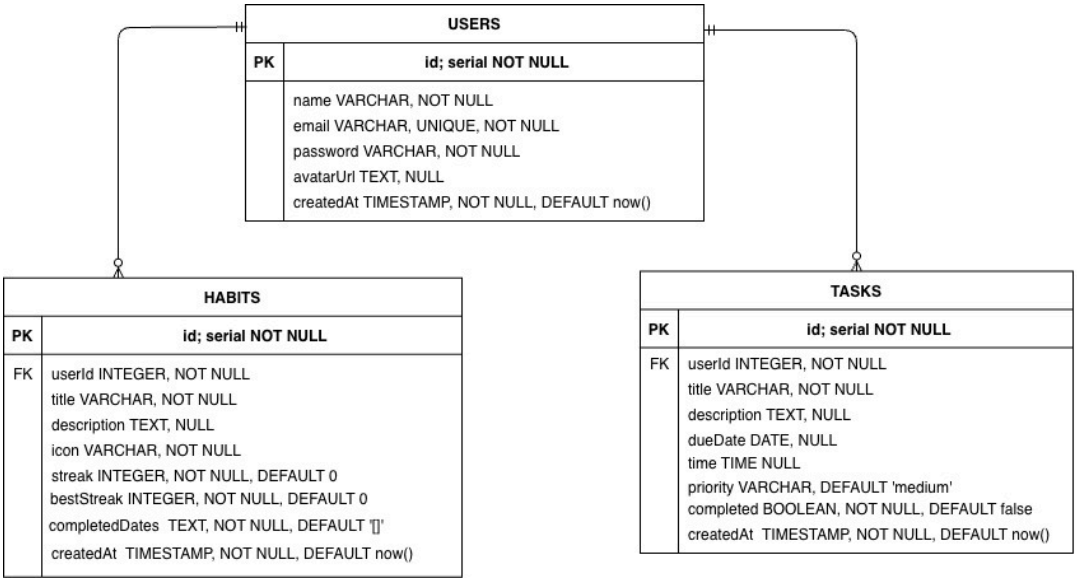
**ДОДАТОК 3**

**Схема взаємозв’язків бази даних. ER-діаграма. Схема структурна**.

ІАЛЦ. 045440.007 Д3

Аркушів 1

Київ – 2026




| Зм | Лист | № докум. | Підп. | Дата | ІАЛЦ.045440.007 ДЗ | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Гречишкіна К.Д. | | | Комп'ютерна система планування та моніторингу виконання особистих завдань | *Літ.* | *Аркуш* 1 / *Аркушів* 1 |
| Перевірив | | Кучмій О.О. | | | | | |
| Н. контр. | | Клятченко Я.М. | | | ***Схема взаємозв'язків бази даних. ER-діаграма. Схема структурна.*** | КПІ ім. Ігоря Сікорського ФПСПМ , КВ-22 | |
| Затвердив | | Романкевич В.О. | | | | | |

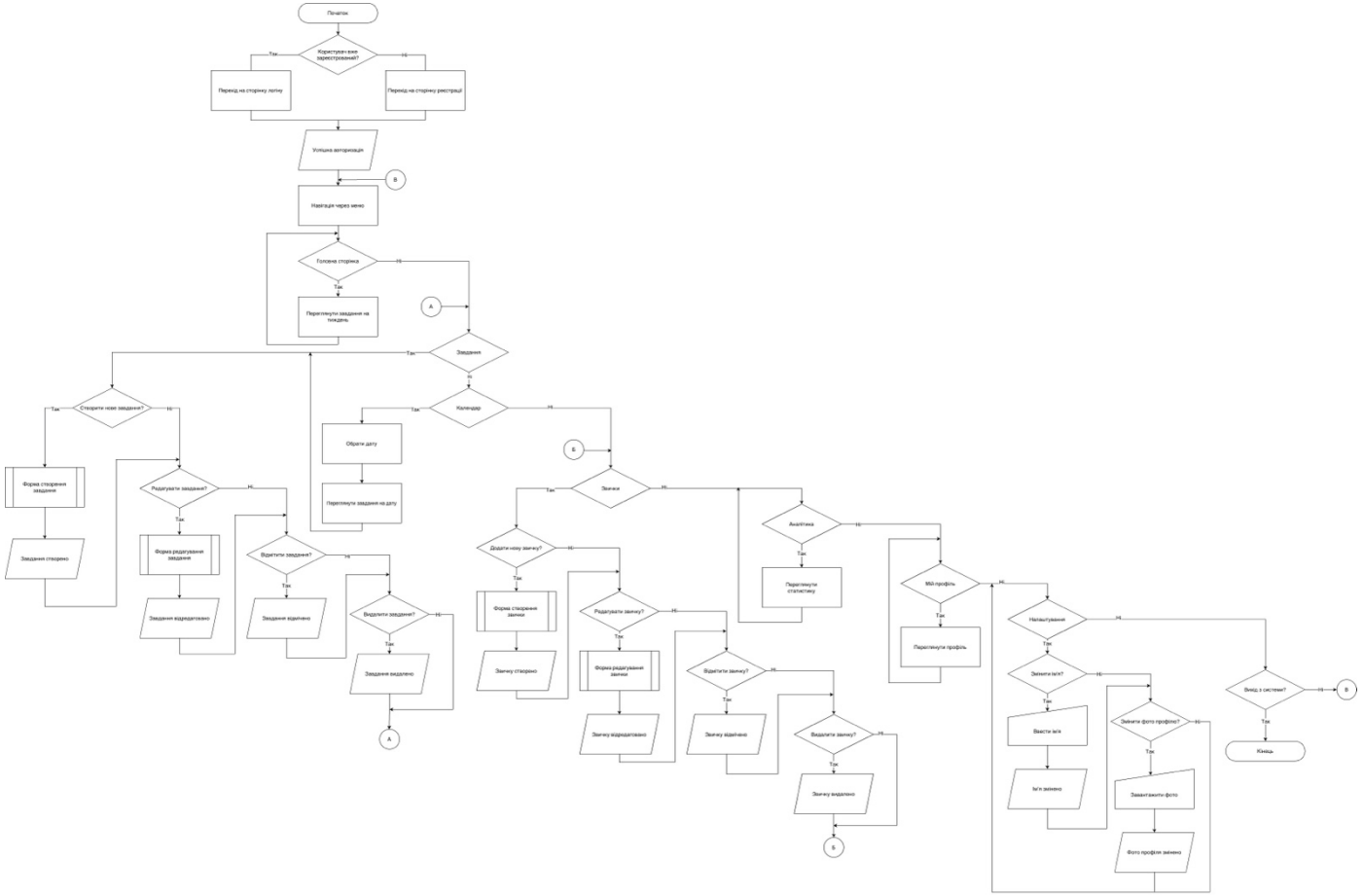
**ДОДАТОК 4**

**Блок-схема алгоритму дій користувача в системі. Схема алгоритму**.

ІАЛЦ. 045440.008 Д4

Аркушів 1

Київ – 2026




| Зм | Лист | № докум. | Підп. | Дата | ІАЛЦ.045440.008 Д4 | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Гречишкіна К.Д. | | | Комп’ютерна система планування та моніторингу виконання особистих завдань | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Кучмій О.О. | | | | | |
| Н. контр. | | Клятченко Я.М. | | | ***Блок-схема алгоритму дій користувача в системі. Схема алгоритму.*** | КПІ ім. Ігоря Сікорського ФПСПМ , КВ-22 | |
| Затвердив | | Романкевич В.О. | | | | | |