Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

WEB-ДОДАТОК "ОРГАНАЙЗЕР ДЛЯ БІЗНЕСУ"

Виконав: студент 4 курсу, групи 3КН-21б

спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Авер'янов В.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)

«04» Червня 2025 р.

Рецензент: к.т.н., доц., каф САІТ
Козачко О.М.
(прізвище та ініціали)

«05» Червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» Червня 2025 р.

Вінниця – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А
«05» березня 2025 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Авер'янову Владиславу Валентиновичу

1. Тема роботи: WEB-додаток "Органайзер для бізнесу".
   Керівник роботи: д.т.н., професор Сілагін О.В. затверджені наказом вищого навчального закладу "20" березня 2025 року №97
2. Строк подання студентом роботи "30" травня 2025 р.
3. Вихідні дані до роботи : Клієнт-серверна архітектура WEB-додатку бізнес органайзера; Кількість сутностей в базі даних – 4 од; Мова програмування – об'єктно орієнтована; Відповідність стандарту ODBC; Користувацьких сторінок – 3 од.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): визначення об'єкта, предмета, мети та задачі подальшого дослідження; аналіз предметної області, існуючих систем для організації бізнесу; проектування програмних модулів WEB-додатку "Органайзер для бізнесу"; програмна реалізація модулів WEB-додатку "Органайзер для бізнесу".
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема алгоритму роботи WEB-додатку "Органайзер для бізнесу"; UML-діаграма компонентів WEB-додатку "Органайзер для бізнесу"; головна сторінка WEB-додатку "Органайзер для бізнесу"; сторінка авторизації WEB-додатку "Органайзер для бізнесу"; сторінка

подій WEB-додатку "Органайзер для бізнесу"; сторінка kanban-дошки WEB-додатку "Органайзер для бізнесу";

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-3	Сілагін О.В., д.т.н. каф. КН		

7. Дата видачі завдання "05" травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області, існуючих систем для організації бізнесу	05.05.25 – 12.05.25	
2	Проектування програмних модулів для організації бізнесу	13.05.25 – 21.05.25	
3	Програмна реалізація модулів WEB-додатку "Органайзер для бізнесу"	22.05.25 – 30.05.25	
4	Тестування модулів WEB-додатку "Органайзер для бізнесу"	31.05.25 – 04.06.25	

Студент _____ Авер'янов В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Сілагін О.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 83 сторінки формату А4, на яких є 16 рисунків, список використаних джерел містить 21 найменувань.

Бакалаврська дипломна робота є повним циклом розробки WEB-додатку “Органайзер для бізнесу”. У цій бакалаврській роботі розроблено систему, яка забезпечує приємний досвід використання інструментів для організації бізнесу індивідуальному користувачу з використанням сучасних технологій, таких як Django для клієнтської та серверної частини та PostgreSQL для зберігання даних. Система спрямована на підвищення ефективності процесу організації бізнесу шляхом покращення інтерфейсу та збільшення спеціалізованості.

Ключові слова: WEB-додаток, організація бізнесу, покращення інтерфейсу, Django, PostgreSQL.

ABSTRACT

The bachelor's thesis consists of 83 A4 pages, including figures, and the list of references contains 21 titles.

The bachelor's thesis is a complete cycle of developing a WEB application “Business Organizer.” In this thesis, a system was developed that provides a pleasant user experience with business organization tools for an individual user, utilizing modern technologies such as Django for the client and server sides and PostgreSQL for data storage. The system aims to enhance the efficiency of the business organization process by improving the interface and increasing specialization.

Keywords: WEB application, business organization, interface improvement, Django, PostgreSQL.

ЗМІСТ

ВСТУП

Актуальність дослідження полягає в тому, що в умовах швидкого технологічного прогресу зростає потреба в сучасних інструментах для організації бізнесу. Такі інструменти сприяють ефективному плануванню, управлінню часом і ресурсами, підвищують швидкість і якість роботи, а також зменшують ризик втрати інформації про ключові події. Ефективне планування та координація діяльності є вирішальними факторами успіху компанії. Проте наявні технології в сфері організації бізнесу є орієнтованими на великі та малі команди та не передбачають використання додатку індивідуальними користувачами.

Фізичні особи підприємці (ФОП) не мають доступного інструментарію для організації та планування бізнесу, який був би призначений спеціально для одноосібного використання, без непотрібних функцій та обмежених за підписками інструментів. Тому створення інтегрованого WEB-додатку для підтримки організації та управління бізнес-процесами, створений спеціально для індивідуальних користувачів, є актуальним і необхідним.

Метою роботи є покращення швидкості доступу до функцій інтерфейсу та збільшення спеціалізованості WEB-додатку для індивідуальних користувачів.

Об’єктом роботи є процес створення WEB-додатків

Предметом є алгоритми, програмні засоби та технічні рішення, які використані для розробки WEB-додатку,а також інтерфейс для технічної реалізації WEB-додатку.

Щоб досягнути мети, потрібно вирішити наступні задачі:

- Аналіз проблем при організації бізнесу
- Аналіз відомих програмних рішень для організації бізнесу
- Проектування модулів WEB-додатку “Органайзер для бізнесу”
- Програмна реалізація модулів WEB-додатку “Органайзер для бізнесу”
- Тестування модулів WEB-додатку “Органайзер для бізнесу”

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ WEB-ДОДАТКУ "БІЗНЕС ОРГАНАЙЗЕР"

1.1Аналіз проблем при організації та управлінні бізнесом

У сучасному бізнес-середовищі малі та середні підприємства стикаються з посиленням конкуренції, швидкою зміною ринкових умов і зростаючою складністю внутрішніх бізнес-процесів. Ефективне планування й координація діяльності стають ключовими чинниками, які визначають успіх компанії. Водночас багатозадачність, обмеженість ресурсів і відсутність централізованого інструментарію для управління призводять до втрати важливої інформації, затримок у виконанні завдань та зниження оперативності прийняття рішень. Тому розробка інтегрованого WEB-додатку для підтримки організації й управління бізнесом є надзвичайно ак Тому створення інтегрованого WEB-додатку для підтримки організації та управління бізнес-процесами є актуальним і необхідним.туальною і необхідною.

Малі та середні підприємства у різних галузях стикаються з подібними проблемами, які можна порівняти з навігацією в бурхливому морі без компаса. Наприклад, невеликий роздрібний магазин може пропустити графік поповнення запасів через відсутність чіткого планування, що призводить до втрати продажів під час пікового попиту. У сфері послуг фрілансер, який одночасно працює над кількома проєктами, ризикує пропустити дедлайн через розрізнені нотатки в різних додатках.

Неефективне планування впливає не лише на фінанси, а й на всіх зацікавлених сторін. Працівники відчувають стрес і демотивацію через необхідність гасити “пожежі” в останній момент. Через погане планування працівники можуть дійти до перенавантажень, що в свою чергу може привести до зниження якості роботи. Клієнти, зі свого боку, втрачають довіру до бізнесу, коли дедлайни зриваються: у випадку магазину покупці можуть перейти до

конкурентів через відсутність товарів. Навіть постачальники страждають, коли замовлення надходять з запізненням.

Традиційні способи планування — паперові щоденники, електронні таблиці або фрагментарні рішення у вигляді поодиноких мобільних додатків надають лиш часткове вирішення проблеми, проте все ще зберігає значний ризик упущення важливої події або прострочення дедлайну: щотижневі наради, регулярні аудити, обов'язкові звітні зустрічі чи дотримання контрактних зобов'язань можуть бути «забуті» через недостатню структурованість планування. З огляду на це, підприємства витрачають зайвий час на пошук розрізненої інформації та коригування планів у ході роботи, що прямо впливає на їхню продуктивність і конкурентоспроможність[1].

Перехід до цифрових інструментів також стикається з бар'єрами. Багато підприємців, особливо старшого віку або з обмеженим технічним досвідом, відчувають опір до змін через страх складного навчання чи високих витрат. Наприклад, власник магазину може віддати перевагу паперовому щоденнику, вважаючи цифрові інструменти надто складними. Електронні таблиці, хоча й популярні, не підтримують автоматичні нагадування чи синхронізацію між пристроями, що ускладнює віддалену роботу. Мобільні додатки часто обмежені у функціональності, змушуючи користувачів комбінувати кілька рішень, що лише погіршує хаос[2][3].

Розробка WEB-додатку "Органайзер для бізнесу" дозволить централізувати функції планування, відстеження й нагадування про завдання та події. Такий інструмент матиме календарну систему з можливістю встановлення одноразових і циклічних нагадувань, а також виставляти тривалість кожної події, що дозволяє планувати підготовку з урахуванням необхідних ресурсів і часу.

В епоху технологій та мережі Інтернет WEB-додаток є гарним інструментом, надаючи доступ до функцій майже в будь-який час і в будь-якому місці, зберігаючи пам'ять пристрою. Також можливість користуватись WEB-додатком з будь-якого пристрою дозволяє мати доступ до інформації незалежно від рівня заряду батареї.

WEB-додаток “Органайзер для бізнесу” повинен включати разові та повторювані події, з використанням часових рамок. Це дозволяє контролювати прогрес виконання чи проведення певних завдань чи подій, а також перевіряти, чи подія проходить згідно часових рамок. Забезпечення безпеки даних є критичним аспектом при розробці WEB-додатку незалежно від цілей. Потрібні сучасні методи захисту та шифрування даних для гарантії конфіденційності та цілісності інформації. Також WEB-додаток повинен мати можливість підлаштовуватись під різні пристрої, змінюючи маштабування інтерфейсу.

З поширенням віддаленої роботи, доступність з будь-якого пристрою є обов’язковою. Адаптивний дизайн і хмарна архітектура забезпечують зручність і захист даних. Органайзер також готовий до майбутнього зростання, підтримуючи масштабованість для обробки більшої кількості користувачів і даних.

1.2 Аналіз відомих програмних рішень для організації бізнесу

Головною ціллю WEB-додатку “Органайзер для бізнесу” є надання користувачам можливості створення події та завдання та їх контролю. Це дозволяє покращити планування та спростити виконання великих проектів. На сьогодні існує велика кількість додатків, які тією чи іншою мірою допомагають організувати та управляти бізнесом. Найпопулярнішими є Trello, Asana та Google Calendar. Також будуть згадані менш популярні Monday.com та ClickUp.

Trello є найпопулярнішним застосунком для управління проєктами, що використовує методологію Kanban. Він допомагає організувати роботу команди над проектом чи завданням за допомогою елементів дошки, списків та карток, призначаючи відповідальних за завдання[4].

Дошка – це робочий простір, який символізує певне завдання чи проект, картки – це певні задачі, які треба виконати для закінчення проекту, їх статус визначається за розташуванням у певному списку. Зазвичай є три списки: “Зробити” (To do), “В процесі виконання”(In progress) та “Зроблено”(Done), проте за потреби можна змінити наявні або додати нові списки.

Trello підтримує інтеграцію зі Slack та Google Drive, та має інструмент Butler, що дозволяє створити правила та команди для спрощення роботи. Приклад інтерфейсу веб-застосунку Trello зображено на рисунку 1.1.

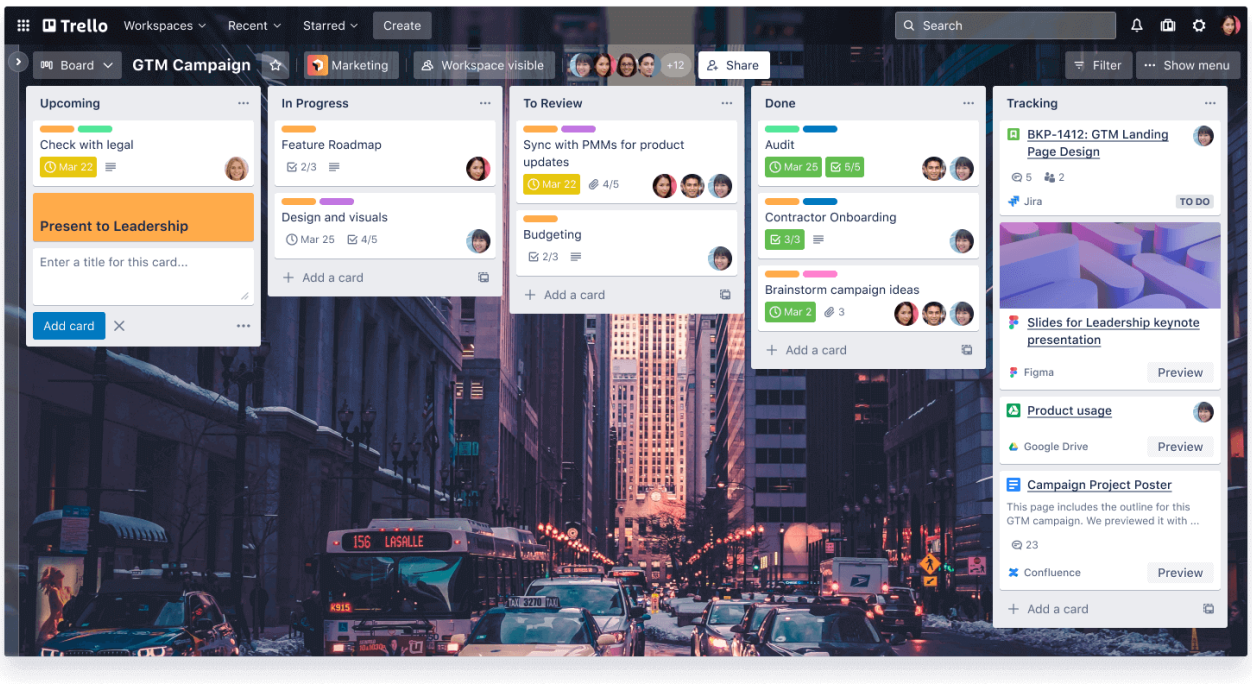

Рисунок 1.1 – Приклад інтерфейсу веб-застосунку Trello

Asana також є досить популярним вибором. Особливістю цього застосунку є його гнучкість: Всі завдання можна позначати не тільки на Kanban дошці, а й у вигляді списку, в календарі чи на часових шкалах[5].

Основними елементами тут слугують Проєкти та Завдання. Проєктом є обраний робочий простір, який в залежності від потреб зображається як список, дошка чи інше. В середину простору є можливість створити окремі завдання, а також призначити відповідальних за ці завдання та визначити часові межі виконання.

Застосунок також підтримує інтеграцію зі Slack, Google Drive та Microsoft Teams та має вбудовані інструменти для оцінки завантаженості команди, планування ресурсів та автоматизації частини дій. Приклад інтерфейсу веб-застосунку Asana зображено на рисунку 1.2

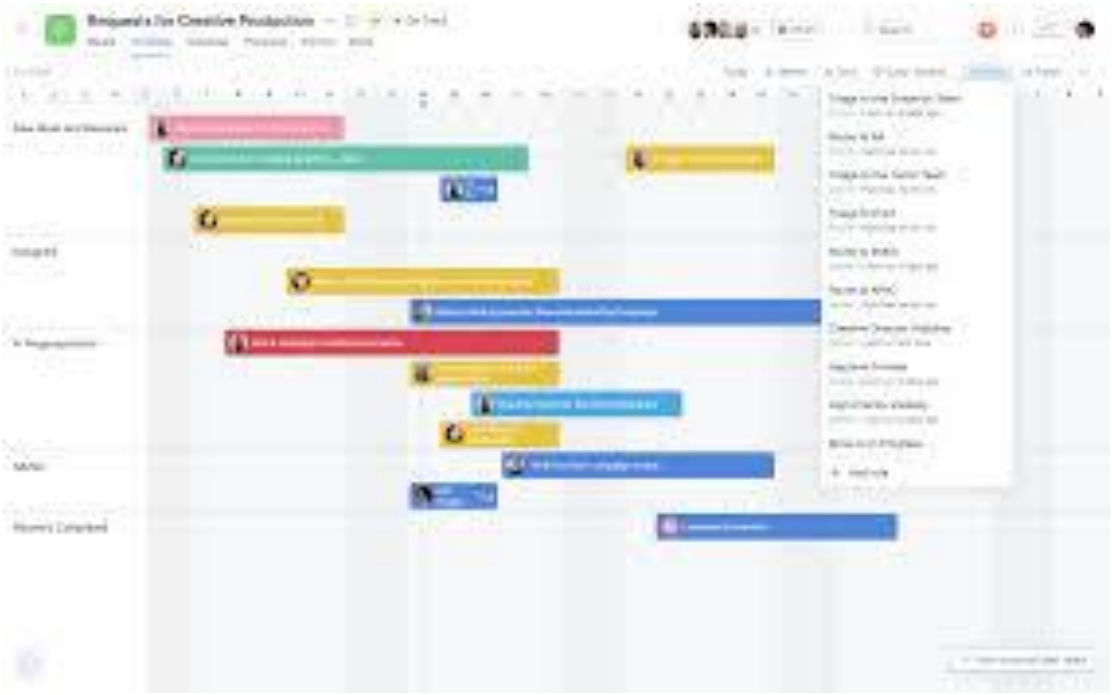

Рисунок 1.2 – Приклад інтерфейсу веб-застосунку Asana

Google Calendar є продуктом від компанії Google для планування справ чи зустрічей. Цей веб-застосунок має інтеграцію з іншими застосунками від Goggle, наприклад, Google Forms та Google Meet для організації онлайн зустрічей або Google Disk для перегляду або додавання додаткової інформації про подію[6].

Головною перевагою Google Calendar є його простота в інтерфейсіта інтеграція з популярними інструментами, які використовуються для зустрічей та планування. Завдяки цій інтеграції користувачі можуть легко організувати та змінювати зустрічі чи завдання, а також отримувати повідомлення на пошту за обліковим записом Google. Приклад інтерфейсу Google Calendar зображено на рисунку 1.3.

Рисунок 1.3 – Приклад інтерфейсу веб-застосунку Google Calendar

Monday.com вирізняється своєю високою кастомізацією, дозволяючи створювати робочі процеси для будь-яких типів проєктів, від маркетингу до розробки програмного забезпечення. Проте його складний інтерфейс і висока вартість (базовий платний план починається від $8 за користувача на місяць) роблять його менш доступним для індивідуальних користувачів або малих команд. Приклад інтерфейсу Monday.com зображено на рисунку 1.4[7].

ClickUp, з іншого боку, пропонує широкий набір функцій, включаючи управління завданнями, відстеження часу та навіть вбудовану документацію. Безкоштовна версія ClickUp є однією з найщедріших на ринку, але користувачі часто скаржаться, оскільки в спробі створити універсальний інтерфейс з різними варіантами відображення, привело до створення перенавантаженого інтерфейсу, ускладнюючи розуміння, використання, а також швидке вивчення додатку для нових користувачів. Приклад інтерфейсу ClickUp зображено на рисунку 1.5 [8].

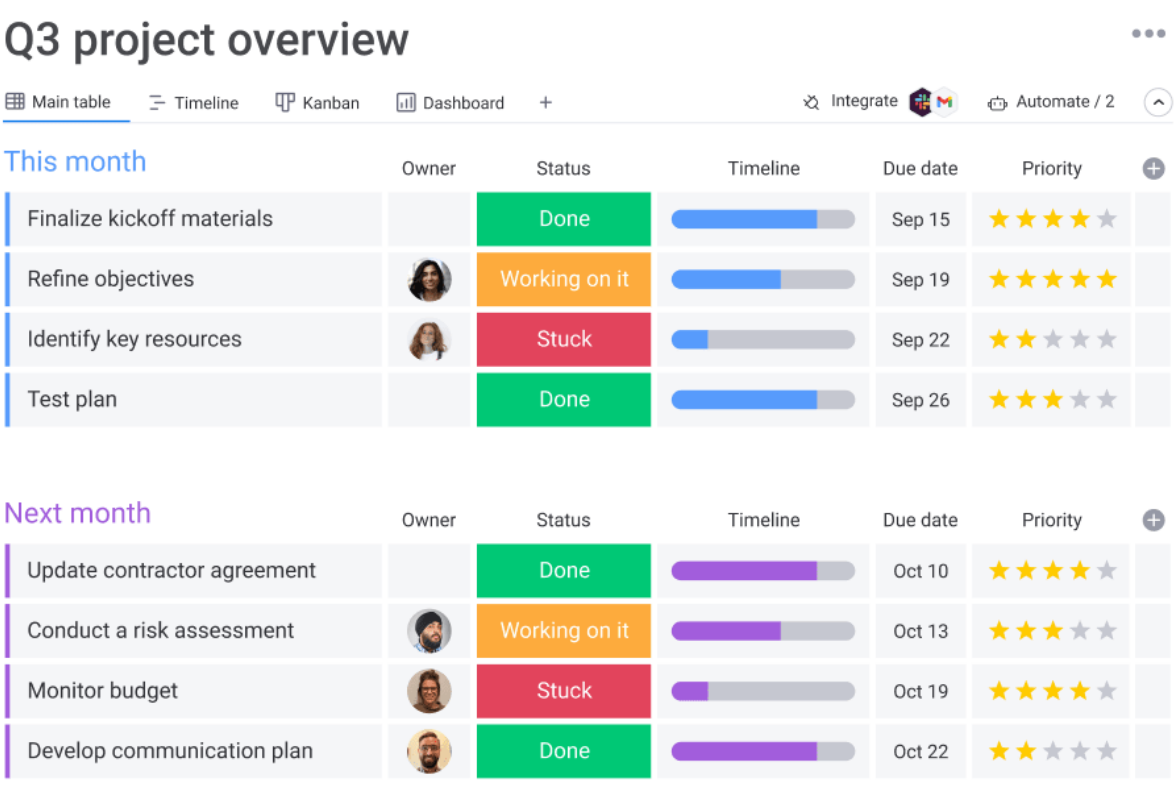

Рисунок 1.4 – Приклад інтерфейсу веб-застосунку Monday.com

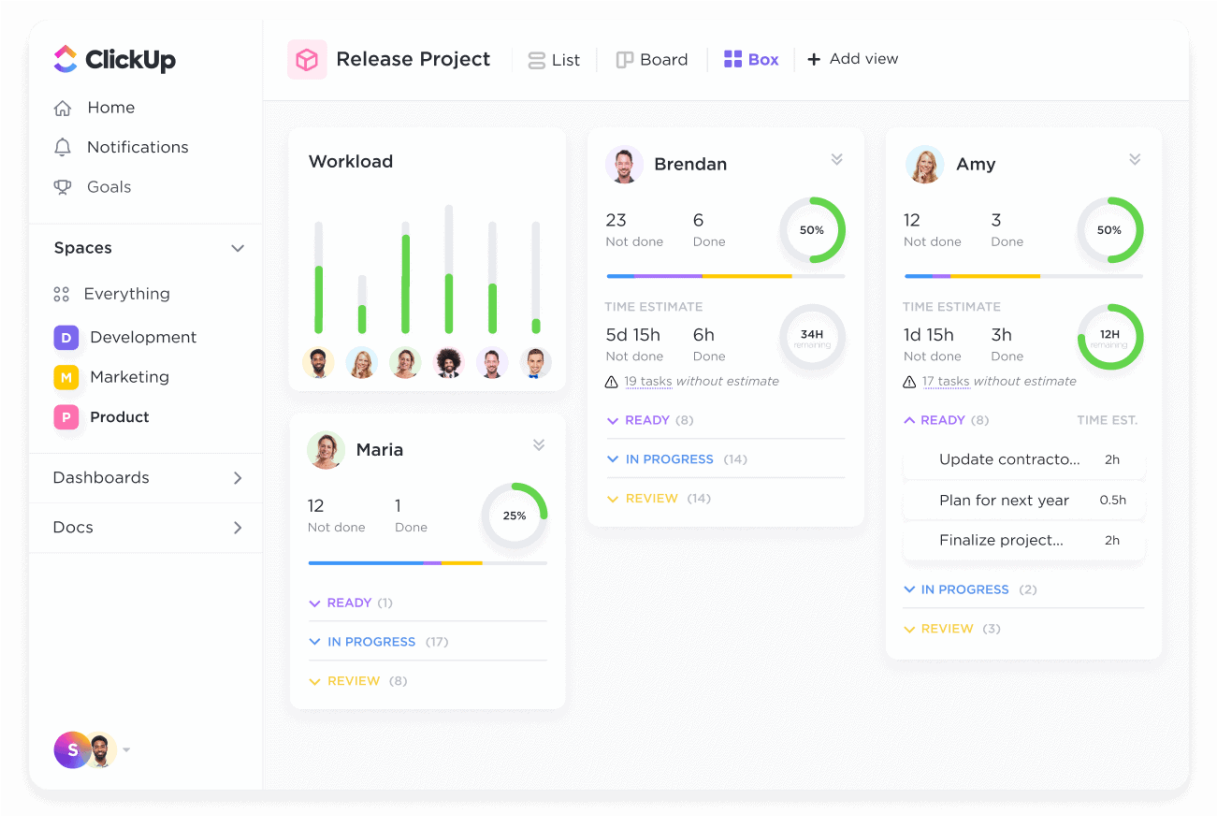

Рисунок 1.5 – Приклад інтерфейсу веб-застосунку ClickUp

Більшість існуючих рішень, таких як Trello чи Asana, розроблені з урахуванням потреб команд, а не окремих підприємців. Наприклад, Trello не пропонує вбудованих часових рамок для завдань у безкоштовній версії, що ускладнює планування подій з чіткими дедлайнами. Asana, хоча і має функцію часових шкал, вимагає значного часу на налаштування проєктів, що може бути незручним для користувачів, які шукають швидке та інтуїтивне рішення. Google Calendar, навпаки, ідеально підходить для планування подій, але не має інструментів для управління складними завданнями чи відстеження прогресу. На відміну від Google Calendar, який орієнтований на просте планування подій, Monday.com і ClickUp більше підходять для складних проєктів, але їхня орієнтація на командну роботу робить їх менш оптимальними для індивідуального управління бізнесом. Ці обмеження підкреслюють потребу в новому веб-ресурсі, який поєднує простоту Google Calendar з можливостями управління завданнями, орієнтованими на індивідуальних користувачів.

Таблиця 1.1 – Порівняння наявних вирішень в сфері організації бізнесу

	Trello	Asana	Google Calendar
Інтерфейс	Інтерактивний	Складний через велику кількість функцій	Простий
Часові межі	-	+	+
Орієнтованість	Малі команди	Великі команди	Планування подій
Відстеження прогресу	Переміщення карток	Часові шкали, статуси завдань	Відсутнє
Управління завданнями	Картки з чеклістами	Завдання з пріоритетами	Події з нагадуваннями
Ціна	Безкоштовний, Є платна підписка	Безкоштовний, Є платна підписка	Безкоштовний
Функціонал	Обмежений в безкоштовній версії	Обмежений в безкоштовній версії	Повний

Продовження таблиці 1.1 – Порівняння наявних вирішень в сфері організації бізнесу

	Monday.com	ClickUp
Інтерфейс	Перевантажений	Перевантажений
Часові межі	+	+
Орієнтованість	Великі команди	Великі команди
Відстеження прогресу	Часові шкали, статуси завдань	Часові шкали, статуси завдань
Управління завданнями	Завдання з пріоритетами	Завдання з пріоритетами
Ціна	Безкоштовний, Є платна підписка	Безкоштовний, Є платна підписка
Функціонал	Обмежений в безкоштовній версії	Досить широкий і в безкоштовній версії

1.3 Постановка задачі та розробка вимог до WEB-додатку “Органайзер для бізнесу”.

WEB-додаток повинен реалізувати певні вимоги, зокрема забезпечити зручний та інтуїтивний інтерфейс для створення завдань. Основні компоненти включають клієнтську та серверну частину на Django та базу даних PostgreSQL.

PostgreSQL – це потужна реляційна база даних із відкритим вихідним кодом, призначена для зберігання, управління та обробки структурованих даних. Вона підтримує розширені функції, такі як JSONB для роботи з неструктурованими даними, і забезпечує високу продуктивність завдяки ефективним механізмам індексації та транзакцій. PostgreSQL є гнучкою і надійною, що робить її ідеальним вибором для веб-додатків, які потребують складних запитів і масштабування.

Django був обраний для клієнтської та серверної частини через те, що він ідеально підходить для швидкої розробки безпечних і масштабованих веб-додатків. Його модульний підхід дозволяє створювати складні системи, використовуючи готові компоненти, такі як автентифікація користувачів і механізми управління контентом. Для веб-додатків це критично важливо, оскільки система потребує надійної автентифікації (реєстрація, логін, управління ролями), управління задачами (CRUD-операції) та аналітичних модулів. Django пропонує готові рішення, такі як вбудована ORM (Object-Relational Mapping), яка спрощує роботу з базою даних. Наприклад, створення моделей для задач чи користувачів займає кілька рядків коду, а їхня інтеграція з PostgreSQL відбувається автоматично. Це дозволяє зосередитися на бізнес-логіці, а не на низькорівневих операціях.

Django має вбудовані механізми захисту від поширених веб-атак, таких як SQL-ін’єкції, XSS (Cross-Site Scripting) і CSRF (Cross-Site Request Forgery). Для бізнес органайзера, де зберігаються персональні дані користувачів і конфіденційна інформація про задачі, це забезпечує високий рівень безпеки без додаткових зусиль. Django підтримує модульну архітектуру, що дозволяє легко додавати нові функції, наприклад, інтеграцію з зовнішніми API для синхронізації з календарями чи аналітичними інструментами.

Серед баз даних було обрано Postgre SQL, потужну реляційну базу даних із відкритим вихідним кодом, яка ідеально підходить для застосунків, що потребують складних запитів, надійності та масштабованості.

Для нашого додатку PostgreSQL забеспечить ефективне зберігання структурованих даних, таких як інформація про користувачів, задачі та дедлайни. PostgreSQL підтримує розширені типи даних, такі як JSONB, що дозволяє зберігати неструктуровані дані, наприклад, додаткові метадані задач чи налаштування користувачів. Це особливо корисно для гнучкого управління різними типами даних у бізнес органайзері. Завдяки ефективним механізмам індексації (наприклад, GIN і GiST) і підтримці паралельних транзакцій, PostgreSQL забезпечує швидке виконання складних запитів, таких як агрегація

даних для аналітичних звітів чи фільтрація задач за кількома критеріями. PostgreSQL має репутацію однієї з найнадійніших баз даних, із підтримкою ACID-транзакцій, що гарантує цілісність даних навіть у разі збоїв. Для бізнес органайзера, де втрата даних про задачі чи користувачів є неприпустимою, це ключова перевага.

Система повинна реалізувати наступні функції: реєстрація та аутентифікація користувачів для забезпечення безпеки та конфіденційності даних користувачів, управління завданнями з CRUD-операціями для створення, читання, оновлення та видалення завдань, візуалізація завдань та зустрічей за допомогою списків та карток, автоматичне повторення завдань чи зустрічей, що відбуваються регулярно зі збереженням можливість редагувати кожну з них індивідуально.

1.4 Висновок до розділу 1

У цьому розділі було доведено актуальність створення WEB-додатку “Органайзер для бізнесу”, оскільки він поєднує в собі інструменти для організації бізнесу, які спроектовані для ефективного індивідуального використання та є простими у використанні навіть для недосвідчених користувачів.

Проведено аналіз існуючих WEB-додатків для організації бізнесу. Дослідження показало, що існуючі рішення створені з орієнтацією на використання командою, через що здебільшого не передбачають індивідуального використання, мають складний та перенавантажений інтерфейс або мають обмежений функціонал для ефективної роботи в індивідуальному режимі.

Було сформульовано вимоги та поставлена задача дослідження, яка передбачатиме створення системи авторизації, опрацювання даних та взаємодій з базою даних.

2. ПРОЕКТУВАННЯ WEB-ДОДАТКУ “ОРГАНАЙЗЕР ДЛЯ БІЗНЕСУ”

2.1 Вибір архітектури WEB-додатку “Органайзер для бізнесу”.

Розробка WEB-додатку “Органайзер для бізнесу” потребує ретельного вибору архітектури, щоб забезпечити оптимальну продуктивність, масштабованість, безпеку та зручність для користувачів. Існує кілька популярних архітектурних підходів, які використовуються для створення подібних систем, зокрема монолітна, мікросервісна та клієнт-серверна архітектури. У цьому розділі буде проаналізовано особливості, переваги та недоліки кожної з них, щоб визначити оптимальний варіант для створення WEB-додатку.

Монолітна архітектура є традиційним підходом до створення WEB-додатків, де всі складові системи — інтерфейс користувача, серверна логіка та взаємодія з базою даних — об’єднані в єдину кодову базу. Такий підхід вирізняється своєю простотою, що є ключовою перевагою. Розробники можуть швидко розпочати роботу над проєктом, оскільки всі компоненти знаходяться в одному місці, що спрощує процеси розробки, тестування та налагодження. Розгортання монолітного додатку також є відносно простим, адже всі залежності зібрані разом, що зменшує складність конфігурації серверного середовища. Однак монолітна архітектура має значні обмеження. Одним із головних недоліків є складність масштабування. У міру зростання додатку кодова база може стати громіздкою, що ускладнює її підтримку та внесення змін. Навіть невелика модифікація одного компонента може вимагати повторного тестування та перерозгортання всієї системи, що підвищує ризик помилок і витрати часу. Крім того, впровадження нових технологій у монолітній архітектурі часто є проблематичним, оскільки всі компоненти мають бути сумісними з основною структурою[9].

Мікросервісна архітектура, на відміну від монолітної, пропонує модульний підхід, при якому додаток розбивається на окремі, незалежні сервіси. Кожен мікросервіс відповідає за певну функцію, наприклад, управління користувачами чи обробку платежів, і може розроблятися, розгортатися та масштабуватися окремо. Ця гнучкість є ключовою перевагою, оскільки дозволяє використовувати різні технології для різних частин системи, оптимізуючи їх залежно від потреб. Серед інших переваг мікросервісів — легкість оновлення та підтримки. Зміни в одному сервісі не впливають на роботу інших, що знижує ризики та спрощує процес внесення змін. Крім того, мікросервіси дозволяють ефективно розподіляти ресурси, масштабуючи лише ті компоненти, які зазнають високого навантаження. Це особливо корисно для складних систем із різноманітними функціями. Проте мікросервісна архітектура не позбавлена викликів. Управління великою кількістю сервісів потребує складних систем автоматизації, таких як оркестрація (наприклад, Kubernetes) та моніторинг. Обмін даними між сервісами через мережу може створювати затримки, що знижують продуктивність, якщо не оптимізувати мережеві взаємодії. Крім того, розробка мікросервісів вимагає від команди вищого рівня експертизи та координації[10].
Клієнт-серверна архітектура є однією з найпоширеніших для WEB-додатків. Вона чітко розмежовує клієнтську частину, яка відповідає за інтерфейс користувача, та серверну частину, яка обробляє бізнес-логіку та взаємодіє з базою даних. Такий поділ забезпечує модульність і полегшує розробку, дозволяючи командам працювати над клієнтською та серверною частинами паралельно. Клієнтська частина в цій архітектурі зосереджена на створенні інтерактивного та зручного інтерфейсу. Використання сучасних фреймворків, таких як Django, дозволяє створювати динамічні веб-додатки з високим рівнем взаємодії. Серверна частина, своєю чергою, відповідає за обробку запитів, управління даними та забезпечення безпеки. Наприклад, у Django для обробки запитів використовується вбудований URL Dispatcher, який ефективно маршрутизує запити до відповідних представлень. Однією з ключових переваг клієнт-серверної архітектури є її масштабованість. Клієнтська та серверна

частини можуть масштабуватися окремо залежно від потреб системи. Наприклад, серверну частину можна оптимізувати для обробки великої кількості запитів, використовуючи сервери типу Gunicorn або uWSGI, тоді як клієнтська частина може бути покращена за допомогою сучасних JavaScript-фреймворків[11].

Безпека є ще одним важливим аспектом клієнт-серверної архітектури. Оскільки серверна частина відповідає за обробку всіх запитів і доступ до бази даних, можна централізовано реалізувати механізми захисту, такі як HTTPS, автентифікація та авторизація. Це забезпечує надійний захист даних користувачів і системи в цілому. Проте клієнт-серверна архітектура також має свої складнощі. Вона вимагає ретельного проектування API для ефективного обміну даними між клієнтом і сервером. Ненадійне мережеве з’єднання може стати вузьким місцем, особливо при високих навантаженнях, що потребує додаткових зусиль для оптимізації[12].

Після аналізу різних архітектур клієнт-серверна архітектура видається оптимальним вибором для розробки WEB-додатку “Органайзер для бізнесу”. Вона забезпечує чіткий розподіл між клієнтською та серверною частинами, що сприяє створенню зручного інтерфейсу та надійної серверної логіки. Завдяки підтримці таких інструментів, як Django, ця архітектура забезпечує високу продуктивність, безпеку та гнучкість. Можливість окремого масштабування клієнтської та серверної частин дозволяє ефективно використовувати ресурси та адаптувати систему до зростаючих потреб. Використання Django для обробки запитів через URL Dispatcher і представлення забезпечує швидке реагування на запити користувачів, а також полегшує розширення функціональності додатку в майбутньому без потреби використання сторонніх бібліотек, зберігаючи високий рівень безпеки.

2.2 Розробка UML-діаграм та проектування клієнтської частини WEB-додатку “Органайзер для бізнесу”.

При розробці WEB-додатку “Органайзер для бізнесу” головна увага повинна приділятись створенню інтуїтивного, зручного та ефективного інтерфейсу, який забезпечить комфортну роботу для користувачів. У цьому підрозділі буде розглянуто та обґрунтовано проектування інтерфейсу та вибір технологій, які використовувалися для його реалізації, зокрема Django для клієнтської та серверної частини, Bootstrap для стилізації та Postgre SQL як базу даних.

Проектування інтерфейсу складається з кількох етапів. Перший – створення прототипу системи у вигляді блок-схем та діаграм. Основна ціль – визначити структуру інтерфейсу, логіку взаємодії користувачів із системою та організувати послідовність реалізації функціональності. Прототипи дозволяють візуалізувати розташування елементів на сторінці, їхню взаємодію та послідовність виконання завдань, що дозволить виправити частину помилок ще на етапі планування[13].

WEB-додаток має перевагу над традиційними програмами, оскільки не потребує інсталяції клієнтів на комп’ютер і додатково спрощує управління версіями програмного забезпечення. Вся робота відбувається в межах WEB-браузера, що зручно як для користувачів, так і адміністраторів системи. Окрім того, всі оновлення програми відбуваються на сервері, що дозволяє впроваджувати нові функції та виправляти помилки безперервно, без необхідності користувачів самостійно встановлювати оновлення.

Щодо питань безпеки, то всі дані користувачів зберігаються на сервері, що дозволяє забезпечувати централізоване резервне копіювання і захист інформації. Протоколи HTTPS і JSON дозволяють безпечний обмін даними між клієнтською та серверною частинами, захищаючи інформацію від несанкціонованого доступу і створюючи наскрізне шифрування даних.

Основні принципи проектування інтерфейсу:

1. Простота та мінімалізм. WEB-інтерфейс має бути інтуїтивно зрозумілим та простим для користувачів. Дизайн повинен бути зрозумілий для користувачів незалежно від досвіду їхньої роботи з комп'ютерами. Також це передбачає, що для досягнення бажаного результаті потрібні мінімальна кількість кроків та виборів. Також потрібно забезпечити оптимізовану будову для елементів управління, зберігаючи лише потрібні функції в потрібній кількості, зберігаючи інтерфейс від перевантаження.
2. Контекстність. Інформація та елементи управління повинні бути видимі лише при відповідному контексті, тобто коли вони дійсно потрібні користувачеві, що дозволяє зменшити навантаження інформацією та покращити користувацький досвід.
3. Інтерактивність та асинхронність. Інтерфейс повинен бути динамічним та реагувати на дії користувачів, забезпечуючи зручне використання, з оновленням контенту без необхідності перезавантаження сторінок. Це можна забезпечити використанням, наприклад, AJAX, що дозволить додати динамічний інтерфейс, який оновлюється незалежно від сторінки.
4. Повноекранний дизайн. Він є важливим елементом сучасного WEB-інтерфейсу. Він передбачає максимальне використання доступного екранного простору, щоб забезпечити зручність і ефективність взаємодії користувача з додатком. Адаптивний дизайн дозволяє елементам інтерфейсу динамічно змінювати розмір і розташування залежно від роздільної здатності екрану. Це гарантує комфортне використання додатку на різних пристроях — від великих моніторів до компактних смартфонів. Такий підхід сприяє створенню гнучкого інтерфейсу, який автоматично підлаштовується під розміри екрану, забезпечуючи оптимальне розташування елементів і зручність навігації.

5. Оптимізація для мобільних пристроїв. Оптимізація є критично важливою в умовах зростання популярності смартфонів і планшетів. Мобільний інтерфейс повинен бути спрощеним і адаптованим до особливостей сенсорного управління та менших розмірів екранів. Адаптивний дизайн забезпечує коректне відображення контенту, враховуючи обмеження мобільних пристроїв, такі як менша роздільна здатність і необхідність інтуїтивного управління дотиком. Наприклад, великі кнопки, спрощені меню та оптимізовані форми введення даних полегшують взаємодію користувачів із системою на мобільних пристроях, підвищуючи загальну зручність.
6. Сумісність із сучасними. Використання технологій, таких як HTML5, CSS3 і JavaScript, дозволяє створювати сучасні, інтерактивні інтерфейси, які працюють у всіх популярних браузерах, таких як Chrome, Firefox, Safari та Edge. Водночас важливо забезпечити підтримку старіших версій браузерів, які все ще можуть використовуватися деякими користувачами. Для цього застосовуються так звані фелбеки (fallbacks) — альтернативні стилі або скрипти, які гарантують базову функціональність навіть у застарілих середовищах. Такий підхід забезпечує максимальну доступність додатку для всіх користувачів, незалежно від їхньої технічної платформи.

Для стилізації інтерфейсу використовується Bootstrap, один із найпопулярніших CSS-фреймворків, який значно спрощує розробку адаптивних і візуально привабливих WEB-сторінок. Bootstrap пропонує широкий набір готових компонентів, таких як кнопки, форми, навігаційні панелі, модальні вікна, каруселі та багато іншого. Це дозволяє розробникам економити час на створенні стилів і зосередитися на реалізації бізнес-логіки додатку[14].

Основною перевагою Bootstrap є його система сіток, яка забезпечує створення гнучких макетів, що автоматично адаптуються до різних розмірів екранів. Сітка дозволяє легко вибудовувати розташування елементів, забезпечуючи коректне відображення контенту на настільних комп'ютерах, планшетах і смартфонах. Завдяки цьому розробники можуть створювати універсальні дизайни без необхідності писати складні медіа-запити для кожного типу пристрою.

Крім того, Bootstrap підтримує модульний підхід до стилізації, дозволяючи використовувати лише необхідні компоненти, що зменшує розмір CSS-файлів і сприяє швидшому завантаженню сторінок. Фреймворк також сумісний із сучасними стандартами WEB-розробки, що гарантує стабільну роботу в різних браузерах і на різних платформах.

Використання Bootstrap у поєднанні з адаптивним дизайном і сучасними технологіями, такими як HTML5 і CSS3, дозволяє створювати зручні, функціональні та візуально привабливі інтерфейси для WEB-додатку "Органайзер для бізнесу". Повноекранний дизайн і оптимізований мобільний інтерфейс забезпечують комфортне використання на всіх пристроях, тоді як підтримка стандартів і фелбеків гарантує сумісність із широким спектром браузерів. Bootstrap, своєю чергою, прискорює розробку, надаючи готові інструменти для створення адаптивних макетів і компонентів, що відповідають сучасним вимогам WEB-розробки.

2.3 Розробка алгоритмів WEB-додатку "Органайзер для бізнесу"

Розробка WEB-додатку "Органайзер для бізнесу" передбачає створення оптимального алгоритму, який забезпечить взаємодію між користувачем та системою, починаючи з реєстрації і закінчуючи роботою з завданнями, подіями чи kanban дошкою. Граф-схема з алгоритмом WEB-додатку "Органайзер для бізнесу" зображена на рисунку 2.1.

Для цього потрібна реалізація кількох ключових елементів: реєстрації та авторизації користувача, отримання збережених даних про події, завдання чи kanban картки, створення, редагування та видалення подій, завдань чи kanban карток, а також переміщення kanban карток між таблицями. UML-діаграму компонентів WEB-додатку “Органайзер для бізнесу” зображено на рисунку 2.2.

Для реалізації цього проєкту буде використано клієнт-серверну архітектуру, де серверна та клієнтська частина побудована на Django, забезпечуючи як інтерфейс, так і обробку запитів та взаємодію з базою даних PostgreSQL, яка зберігає всі необхідні дані.

Взаємодія користувача з WEB-застосунком розпочинається з його реєстрації та/чи авторизації. Коли користувач відкриває головну сторінку, він перенаправляється на меню авторизації, звідки він може або ввійти в акаунт, заповнивши форму авторизації, або перейти на меню реєстрації, де створить акаунт, заповнивши форму реєстрації. Після введення необхідної інформації, такі як логін та пароль, клієнтська частина відправляє API-запит до серверної. Серверна частина обробляє запит, перевіряє дані на коректність та, якщо це авторизація, на відповідність, після чого зберігає дані про користувача, та, якщо є, відображає дані про події, завдання чи картки з бази даних PostgreSQL. Після входу в систему користувач потрапляє на головну сторінку, де він може створити нові завдання чи події або переглянути ті, що відбуватимуться найближчим часом. При створенні завдання чи події користувач спочатку обирає тип, після чого вводить деталі щодо події чи завдання, такі як ім’я та дата початку або додаткові, такі як тривалість чи дата кінця, а також наявність та частота повторень (якщо подія чи завдання має відбуватись кілька разів).

Після цього користувач відправляє запит до серверної частини для створення відповідного запису в базі даних. URL dispatcher обробляє запит та додає в базу даних запис з відповідними деталями, після чого список подій на наступні 24 години оновлюється та (за умови, що подія відбудеться в наступні 24 години) відображається у списку.

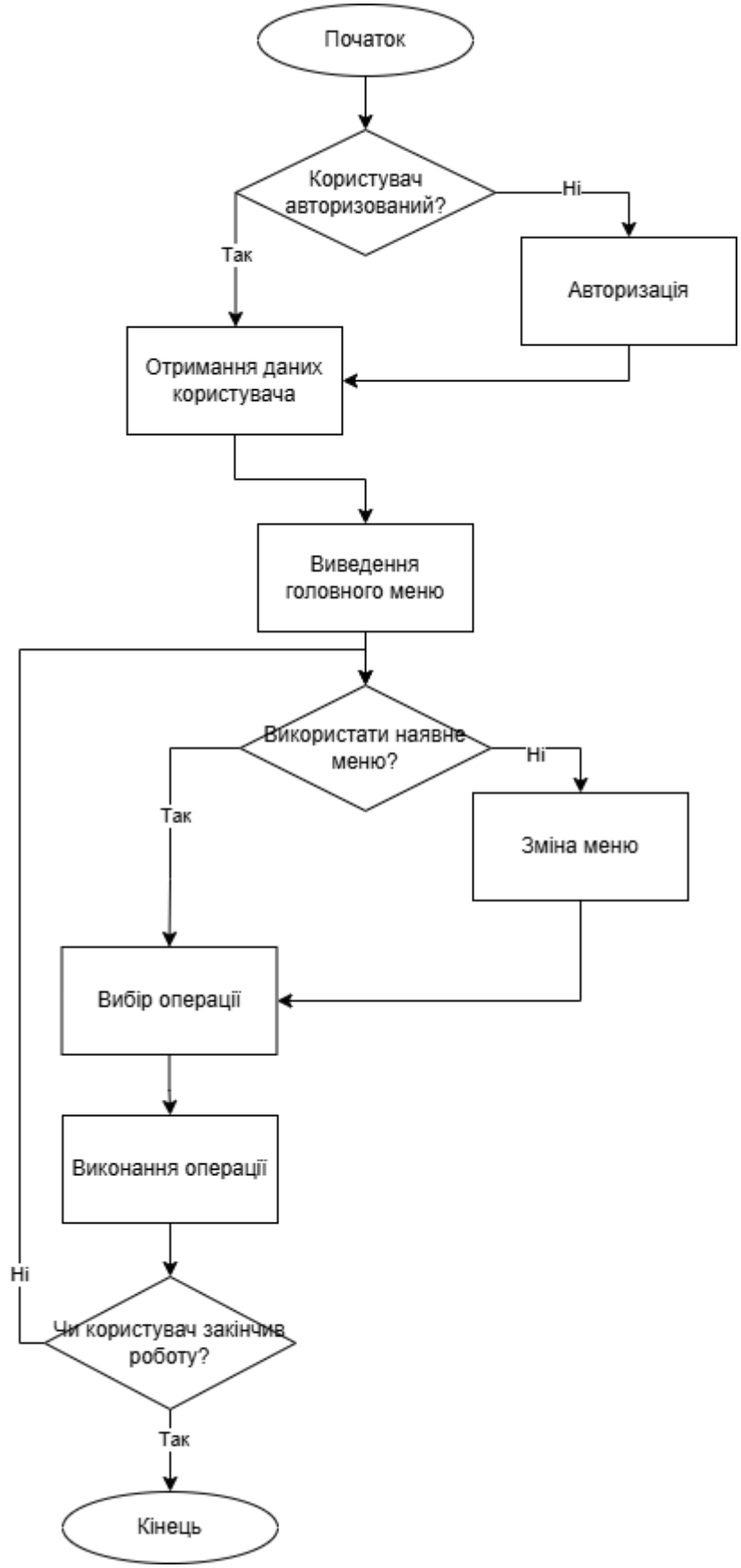

Рисунок 2.1 – Граф-схема алгоритму роботи WEB-додатку “Органайзер для бізнесу”

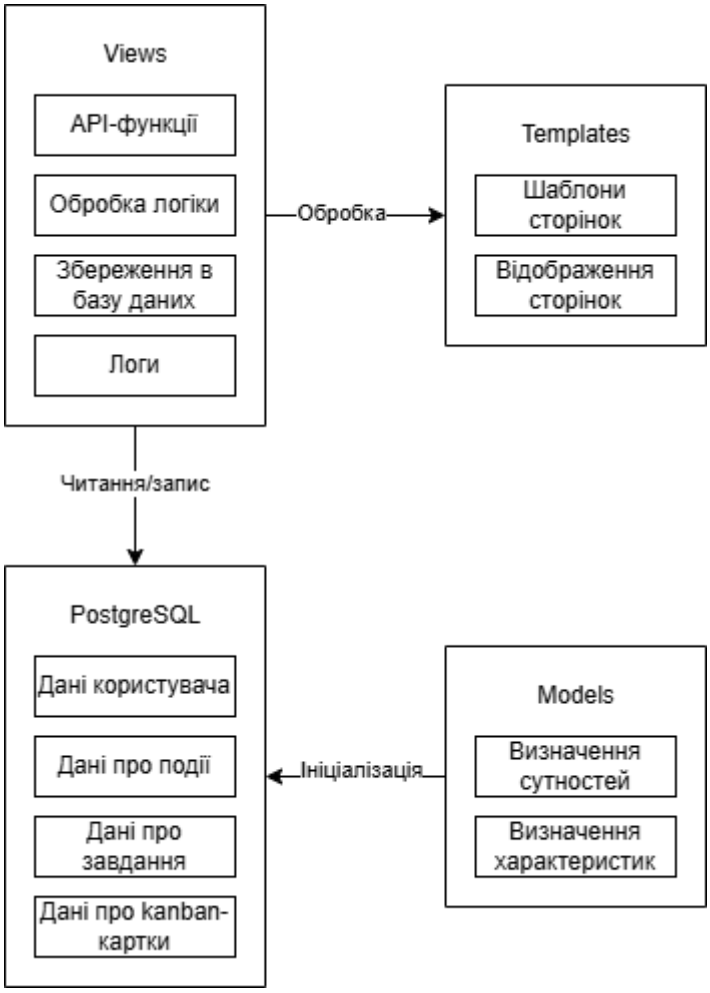

Рисунок 2.2 – UML-діаграма компонентів WEB-додатку “Органайзер для бізнесу”

Також користувач може відредагувати певну подію чи завдання, для цього він може або натиснути на кнопку “змінити” в модулі з найближчими подіями, або на таку ж кнопку в меню “Події”, де події відсортовано за датою та часом початку. Для того, щоб перейти в меню “Події”, користувач може скористатись навігаційним меню зверху сторінки, після чого створюється запит на відтворення сторінки з усіма подіями, відсортованими по часу та даті. В разі натискання на кнопку “змінити”, Клієнтська частина відправляє запит на серверну, URL dispatcher обробляє запит та витягує з бази даних необхідні дані. Після редагування даних на потрібні клієнтська частина відправляє запит на оновлення даних про певну подію чи завдання в базу даних на ті, що вказав користувач.

Також в меню “Події” та в модулі головного меню є можливість видалення події чи завдання за необхідності. В такому разі клієнтська частина відправляє запит на видалення певного запису з бази даних. Окрім того, користувач може перейти в меню Kanban, що відкриє меню для створення та управління Kanban дошкою та картками. При запиті на створення картки URL dispatcher обробляє запит та додає в базу даних запис з відповідними деталями, проте Kanban містить власну базу даних, що не взаємодіє з рештою програми, оскільки потрібна для інших завдань. Kanban-картки також можна редагувати та видаляти, але окрім цього їх також можна переміщувати між таблиця, змінюючи статус завдання, написаного на картці.

Забезпечення безпеки та цілісності даних є критично важливим аспектом функціонування WEB-додатку “Органайзер для бізнесу”. Для захисту інформації, що передається між клієнтською та серверною частинами, використовується протокол HTTPS, який шифрує дані, запобігаючи їх перехопленню чи модифікації. Крім того, система автентифікації та авторизації, побудована на основі токенів JWT (JSON Web Tokens), забезпечує контроль доступу, дозволяючи лише авторизованим користувачам взаємодіяти з ресурсами системи. Це гарантує захист конфіденційних даних, таких як особиста інформація користувачів чи деталі подій і завдань, від несанкціонованого доступу.

Розробка WEB-додатку “Органайзер для бізнесу” передбачає реалізацію кількох ключових етапів взаємодії користувача з системою: Реєстрація та автентифікація користувача; Меню навігації між вкладками додатку; Створення подій і завдань; Редагування та видалення подій і завдань; Робота з Kanban-дошкою, яка включає створення, редагування, видалення та переміщення карток для візуального управління завданнями.

Реалізація проекту через клієнт-серверну архітектуру з використанням Django для клієнтської та серверної частини разом з базою даних PostgreSQL, забезпечує високий рівень інтерактивності, зберігаючи продуктивність системи зі збереженням всіх потрібних даних. Таке поєднання технологій дозволяє створити ефективний, сучасний та безпечний WEB-додаток “Органайзер для бізнесу”, який добре підійде для індивідуального використання.

2.4 Висновок до розділу 2

Було розглянуто різноманітні види архітектури для WEB-додатку “Органайзер для бізнесу”, зокрема монолітну, мікросервісну та клієнт-серверну. Враховуючи переваги та недоліки кожної з них, було обрано клієнт-серверну архітектуру, оскільки вона забезпечує гнучкість, маштабованість та ефективність.

Було розроблено та описано UML-діаграму, що описує серверну взаємодію між компонентами Models, Views, Templates та зовнішнім компонентом бази даних PosgreSQL. Також була розроблена граф-схема алгоритму роботи WEB-додатку, визначаючи необхідну для реалізації логіку та ілюструючи типову взаємодію користувача з додатком.

Описані проектування клієнтської та серверної частини, що дозволяє створити зручний та ефективний інтерфейс, та розробити алгоритм, що включає процеси реєстрації, авторизації та автентифікації користувача, а також перегляду, створення, редагування та видалення завдань, подій чи kanban карток, створюючи функціональний WEB-додаток. За рахунок обраних технологій була досягнута ефективна розробка та підтримка додатку, забезпечуючи стабільність та швидкість роботи.

3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ "БІЗНЕС ОРГАНАЙЗЕР"

3.1 Обгрунтування та вибір мов та технологій для реалізації WEB-додатку “Органайзер для бізнесу”

Python — це високорівнева мова програмування загального призначення, створена для забезпечення високої продуктивності розробників і читабельності коду. Вона підтримує об’єктно-орієнтоване, процедурне та функціональне програмування, що робить її універсальним інструментом для різноманітних задач. Python широко застосовується у WEB-розробці, аналізі даних, автоматизації, наукових дослідженнях і машинному навчанні. Завдяки зрозумілому синтаксису та великій стандартній бібліотеці Python дозволяє швидко створювати надійні та масштабовані програми. Активна спільнота розробників забезпечує доступ до численних бібліотек, фреймворків і навчальних ресурсів, що полегшує вирішення складних задач[15].

Python вирізняється простим і зрозумілим синтаксисом, який полегшує написання та підтримку коду, що є важливим для великих проєктів із командною роботою. Читабельність коду зменшує кількість помилок і спрощує процес налагодження. Велика стандартна бібліотека надає готові модулі для роботи з файлами, мережами та веб-розробкою, що економить час розробників. Python є кросплатформенною мовою, що дозволяє розгортати програми на Windows, macOS і Linux без значних змін у коді. Потужні фреймворки, такі як Django і Flask, забезпечують створення безпечних і масштабованих веб-систем.

Мова є лідером у сфері аналізу даних і машинного навчання завдяки бібліотекам, таким як NumPy, pandas і TensorFlow, що дозволяє ефективно обробляти великі обсяги даних. Python легко інтегрується з C, C++ та іншими мовами, що забезпечує оптимізацію продуктивності для обчислювально складних задач. Активна спільнота пропонує сучасні інструменти та підтримку, що сприяє швидкому освоєнню мови та розвитку проєктів. Ці якості роблять Python ефективним вибором для створення серверних рішень.

JavaScript — це високорівнева мова програмування, розроблена для створення інтерактивних і динамічних веб-сторінок, що є однією з ключових технологій веб-розробки поряд з HTML і CSS. Завдяки Node.js JavaScript також використовується для серверної розробки, що розширює її функціонал. Мова дозволяє створювати інтерактивні інтерфейси, анімації та односторінкові додатки, що робить її популярною серед розробників. JavaScript має багату екосистему та активну спільноту, яка постійно вдосконалює інструменти та стандарти мови[16].

JavaScript підтримується всіма сучасними браузерами, що забезпечує легкий доступ до веб-програм без додаткових налаштувань для користувачів. Асинхронна природа, реалізована через механізми, такі як async/await, дозволяє оновлювати контент без перезавантаження сторінки, покращуючи користувацький досвід. Завдяки npm і фреймворкам, таким як React, Angular і Vue.js, JavaScript пропонує широкий вибір інструментів для створення складних і масштабованих систем. Мова підтримує об’єктно-орієнтоване програмування, що сприяє розробці структурованого коду для великих проєктів. Активна спільнота та відкритий вихідний код забезпечують швидке впровадження нових технологій і підтримку сучасних стандартів.

Хоча JavaScript є незамінною мовою для створення інтерактивних клієнтських інтерфейсів завдяки своїй асинхронній моделі та широкій підтримці в браузерах, Python є кращим вибором для серверної розробки WEB-додатку “Органайзер для бізнесу”. Його простий синтаксис і читабельність полегшують створення та підтримку коду. Велика стандартна бібліотека та фреймворки, такі як Django і Flask, дозволяють швидко розробляти безпечні та масштабовані рішення для управління задачами, календарями та бізнес-даними. Кросплатформенність і можливості інтеграції з іншими технологіями, а також підтримка аналізу даних, роблять Python більш універсальним і ефективним для серверної частини, забезпечуючи зручність у розробці та підтримці.

3.2 Обґрунтування вибору фреймворків та бібліотек

WEB-додаток “Органайзер для бізнесу”, який розробляється, буде використовувати сучасні фреймоворки для високої продуктивності та зручності у використанні. Для цього додаток має фреймворк Django, який використовується як для серверної, так і для клієнтської частини (через систему шаблонів Django). Для зберігання даних застосовується реляційна база даних PostgreSQL, що забезпечує високу надійність і продуктивність.

Django – це високорівневий WEB-фреймворк на Python, який сприяє швидкій розробці безпечних і масштабових WEB-додатків. У проєкті "Органайзер для бізнесу" Django використовується для реалізації серверної логіки, включаючи маршрутизацію, обробку даних і авторизацію, а також для створення динамічного клієнтського інтерфейсу через систему шаблонів[17].

Однією з ключових переваг Django є підхід "батарейки в комплекті", який надає вбудовані інструменти для авторизації, валідації даних і взаємодії з базою даних. Це дозволяє розробникам зосередитися на бізнес-логіці, мінімізуючи час на створення базової функціональності. Django автоматично захищає від поширених вразливостей, таких як CSRF-атаки, SQL-ін’єкції та XSS, що підвищує безпеку WEB-додатку.

Для клієнтської частини Django використовує систему шаблонів, яка дозволяє створювати модульні та повторно використовувані HTML-компоненти. Це спрощує розробку інтерфейсу, забезпечуючи його адаптивність і підтримку динамічних оновлень за допомогою інтеграції з JavaScript. Такий підхід сприяє створенню інтерактивного та зручного інтерфейсу для користувачів.

Гнучка система маршрутизації Django дозволяє легко налаштовувати URL-адреси для доступу до різних функціональних частин додатку, забезпечуючи чітку структуру проєкту. Це полегшує масштабування та підтримку коду.

Завдяки великій спільноті та широкому набору бібліотек, Django підтримує швидку інтеграцію додаткових інструментів, таких як Django REST Framework для створення API чи Celery для асинхронної обробки завдань. Це робить WEB-додаток гнучким і готовим до розширення функціональності.

Flask – це легковаговий Python-фреймворк для створення WEB-додатків, який використовується для розробки серверної логіки та клієнтських інтерфейсів. У проєкті “Органайзер для бізнесу” Flask може застосовуватися для обробки маршрутизації, даних і авторизації, а також для створення інтерфейсу через шаблони Jinja2. Фреймворк дозволяє гнучко обирати інструменти для реалізації функціональності, що робить його придатним для невеликих або специфічних проєктів[18].

Flask пропонує мінімальний набір інструментів, що дає розробникам свободу в налаштуванні проєкту. Наприклад, для авторизації чи роботи з базою даних потрібно підключати бібліотеки, такі як Flask-Login або Flask-SQLAlchemy. Система шаблонів Jinja2 дозволяє створювати модульні HTML-компоненти, які підтримують інтеграцію з JavaScript для динамічних оновлень. Маршрутизація у Flask є простою і дозволяє налаштовувати URL-адреси для доступу до функціональних частин додатку.

Завдяки активній спільноті Flask підтримує інтеграцію з бібліотеками, такими як Flask-RESTful для створення API чи Celery для асинхронних задач. Це дозволяє розширювати функціональність додатку, хоча потребує додаткових зусиль для налаштування порівняно з комплексними фреймворками.

Порівнявши недоліки та переваги перелічених фреймворків, для роботи було обрано саме Django. Flask, хоч і гнучкий, вимагає додаткових бібліотек і налаштувань для реалізації базових функцій, таких як авторизація чи захист від вразливостей, що збільшує час розробки та ризик помилок. Django пропонує вбудовані інструменти, включаючи адміністративну панель, захист від CSRF, SQL-ін’єкцій і XSS, а також готові рішення для роботи з базами даних. Це забезпечує швидшу розробку, вищу безпеку та легшу підтримку, що робить Django кращим вибором для проєкту.

PostgreSQL – це потужна реляційна база даних з відкритим вихідним кодом, яка використовується в проєкті для зберігання даних про користувачів, події та завдання. Вона відома своєю надійністю, підтримкою складних запитів і високою продуктивністю[19].

Однією з переваг PostgreSQL є її підтримка розширених типів даних, таких як JSONB, що дозволяє гнучко зберігати структуровані дані. Це особливо корисно для інструментів типу канбан-дошки, де можуть зберігатися додаткові метадані для кожної картки.

PostgreSQL забезпечує високу продуктивність завдяки індексам, оптимізації запитів і підтримці транзакцій. База даних інтегрується з Django через ORM (Object-Relational Mapping), що дозволяє розробникам працювати з даними через об’єкти Python, замість написання складних SQL-запитів. Це підвищує читабельність коду та знижує ймовірність помилок.

PostgreSQL також підтримує механізми безпеки, такі як управління доступом на основі ролей, що забезпечує захист даних користувачів. Це критично важливо для системи авторизації, яка використовує UUID для ідентифікації користувачів.

MariaDB – це реляційна база даних з відкритим вихідним кодом, яка використовується для зберігання даних у WEB-додатках, таких як “Органайзер для бізнесу”. Вона забезпечує швидке виконання запитів і підтримує стандартні функції, такі як транзакції та індекси. MariaDB сумісна з MySQL, що дозволяє використовувати наявні інструменти та бібліотеки для інтеграції, наприклад, з Django через ORM. Вона підтримує типи даних, такі як JSON, для зберігання структурованих даних, але можливості індексації JSON менш розвинені, ніж у деяких інших базах. MariaDB пропонує механізми безпеки, такі як управління доступом, і підтримує реплікацію для масштабування[20].

MySQL – це ще одна популярна реляційна база даних, яка використовується для зберігання даних користувачів, подій і завдань. Вона відома своєю швидкістю та простотою у налаштуванні. MySQL підтримує базові типи даних, включаючи JSON, і забезпечує продуктивність завдяки індексам і оптимізації запитів.

Інтеграція з Django через ORM дозволяє розробникам працювати з даними через Python-об’єкти, що спрощує код. MySQL має механізми безпеки, такі як управління користувачами, але потребує додаткових налаштувань для складних сценаріїв. Велика спільнота та наявність інструментів сприяють легкому підключенню бібліотек для розширення функціональності[21].

Порівнявши недоліки та переваги перелічених баз даних, було обрано використання саме PostgreSQL. MariaDB і MySQL пропонують високу швидкість, але мають обмеження у підтримці складних типів даних, таких як JSONB у PostgreSQL, який забезпечує ефективну індексацію та гнучкість для структурованих даних, наприклад, метаданих канбан-дошки, та обмежує у виборі типів даних при майбутньому покращенню WEB-додатку. PostgreSQL підтримує розширені запити, такі як віконні функції та рекурсивні запити, що корисно для аналітики. Крім того, PostgreSQL має кращі можливості для масштабування та реплікації, що забезпечує стабільність при високих навантаженнях. На відміну від MySQL, який частково обмежений комерційною моделлю Oracle, PostgreSQL є повністю відкритою та активно підтримується спільнотою. MariaDB, хоч і відкрита, менш ефективна для складних транзакцій. PostgreSQL забезпечує вищу продуктивність, безпеку та гнучкість, що робить її оптимальною для надійного зберігання даних у цьому проєкті.

3.3Реалізація програмних модулів WEB-додатку

Щоб розробити WEB-додаток “Органайзер для бізнесу”, потрібно розробити два окремих модулі: event_manage та kanban_manage. Кожен з цих модулів є незалежним один від одного, проте складається з блоків, кожен з яких відповідає за свої чітко визначені фукнції. Блоки є частиною MVT (Models, Views, Templates) системи Django. В цій системі моделі відображають сутності програмних блоків, тобто характеристики кожного, а також фільтрація та валідація їх значень, як, наприклад, обмеження по кількості символів. Для WEB-додатку підійде подібна структура моделей:

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

class Event(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    start_time = models.DateTimeField()
    end_time = models.DateTimeField(null=True, blank=True)
    description = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class Task(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    start_time = models.DateTimeField()
    description = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class KanbanCard(models.Model):
    STATUS_CHOICES = (
        ('todo', 'Заплановано'),
        ('in_progress', 'В процесі'),
        ('done', 'Завершено'),
```

```
    )
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    extra_fields = models.JSONField(default=dict, blank=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='todo')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

Моделі UserProfile, Event, Task на KanbanCard відображають головні об’єкти, на основі яких буде будуватись логіка. auto_now_add=True дозволяє автоматично створювати дані, не потребуючи їх від користувача, а null=True та blank=True – отримувати відсутні значення та пусті строки. Також є можливість самостійно описати множину доступних значень, як, наприклад, в характеристиці STATUS_CHOICES.

Види описують серверну логіку та взаємодії, відштовхуючись від існуючих моделей та їх значень, а також функціональність об’єктів на клієнтській частині. Програма потребує багато функцій, щоб коректно працювати, наприклад, відображення подій чи завдань, які скоро розпочнуться.

```
    @method_decorator(login_required, name='dispatch')
class MainView(TemplateView):
    template_name = 'main/main.html'

        #Отримання інформації
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Tasko'
```

```
        now = timezone.now() #Початок часової межі
        end_time = now + timedelta(hours=24) #Кінець часової межі
        user = self.request.user

        #Отримання з бази даних всіх подій
        events = Event.objects.filter(
            user=user,
            start_time__gte=now,
            start_time__lte=end_time
        ).order_by('start_time')

        #Отримання з бази даних всіх завдань
        tasks = Task.objects.filter(
            user=user,
            start_time__gte=now,
            start_time__lte=end_time
        ).order_by('start_time')

        items = []
        #Відображення всіх подій та завдань з даними про кожну
        for event in events:
            time_until = (event.start_time - now).total_seconds() / 60
            is_in_progress = now >= event.start_time and (event.end_time is None or
now <= event.end_time)
            time_str = f"{int(time_until)} хвилин" if time_until < 60 else
f"{int(time_until // 60)} годин"
            items.append({
                'item': event,
                'type': 'event',
                'time_until': time_str,
```

```
            'is_in_progress': is_in_progress,
            'start_time': event.start_time
        })
    for task in tasks:
        time_until = (task.start_time - now).total_seconds() / 60
        time_str = f"{int(time_until)} хвилин" if time_until < 60 else
f"{int(time_until // 60)} годин"
        items.append({
            'item': task,
            'type': 'task',
            'time_until': time_str,
            'is_in_progress': False,
            'start_time': task.start_time
        })

    items.sort(key=lambda x: x['start_time']) #Сортування по часу
    context['items'] = items
    return context
```

Для того, щоб відображення коректно працювало, потрібна логіка додавання подій чи завдань для їх відображення:

```
@method_decorator(login_required, name='dispatch')
class CreateItemView(View):
  #Надсилання post запиту до бази даних
  def post(self, request, *args, **kwargs):
    logger.info(f"Request body: {request.body.decode('utf-8')}")
    try:
      data = json.loads(request.body.decode('utf-8')) if request.body else {}
      logger.info(f"Parsed data: {data}")
```

```
    #У випадку, якщо дані введені невірно
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

    #Список значень, які приймає post запит
        item_type = data.get('item_type')
        title = data.get('title')
        start_time = data.get('start_time')
        end_time = data.get('end_time')
        description = data.get('description', '')
        is_recurring = data.get('is_recurring', False)
        repeat_interval = data.get('repeat_interval', 1)
        repeat_count = data.get('repeat_count', 1)

    #Перевірка обов'язкових даних на наявність
        logger.info(f"Item type: {item_type}, Title: {title}, Start time: {start_time}, End
time: {end_time}")
        if not item_type or not title or not start_time:
            logger.warning("Missing required fields")
            return JsonResponse({'status': 'error', 'message': 'Missing required fields'},
status=400)

    #Перенесення даних з полів введення і створення завдань чи подій
        try:
            if item_type == 'event':
                event = Event.objects.create(
                    title=title,
                    start_time=timezone.datetime.fromisoformat(start_time),
```

```
                end_time=timezone.datetime.fromisoformat(end_time) if end_time else
None,
                description=description,
                user=request.user
            )
            if is_recurring:
                for i in range(repeat_count):
                    Event.objects.create(
                        title=title,
                        start_time=event.start_time + timedelta(days=repeat_interval * (i +
1)),
                        end_time=event.end_time + timedelta(days=repeat_interval * (i +
1)) if event.end_time else None,
                        description=description,
                        user=request.user
                    )
        elif item_type == 'task':
            task = Task.objects.create(
                title=title,
                start_time=timezone.datetime.fromisoformat(start_time),
                description=description,
                user=request.user
            )
            if is_recurring:
                for i in range(repeat_count):
                    Task.objects.create(
                        title=title,
                        start_time=task.start_time + timedelta(days=repeat_interval * (i +
1)),
                        description=description,
```

```
                    user=request.user
                )
            else:
                return JsonResponse({'status': 'error', 'message': 'Invalid item type'},
status=400)
        except ValueError as e:
            logger.error(f"Value error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': str(e)}, status=400)
        return JsonResponse({'status': 'success'})
```

Також є потребна і в редагуванні та видаленні подій чи завдань, на випадок зміни умов зустрічі чи їх відміни:

```
@method_decorator(login_required, name='dispatch')
class EditItemView(View):
    def post(self, request, *args, **kwargs):
        try:
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

    #Отримання даних про подію чи завдання з БД
        item_type = data.get('item_type')
        item_id = data.get('id')
        title = data.get('title')
        start_time = data.get('start_time')
        end_time = data.get('end_time')
        description = data.get('description', '')
```

```
        user = request.user
    #Надсилання змінених даних в БД
        if item_type == 'event':
            item = get_object_or_404(Event, id=item_id, user=user)
            item.title = title
            item.start_time = timezone.datetime.fromisoformat(start_time)
            item.end_time = timezone.datetime.fromisoformat(end_time) if end_time else
None
            item.description = description
            item.save()
        elif item_type == 'task':
            item = get_object_or_404(Task, id=item_id, user=user)
            item.title = title
            item.start_time = timezone.datetime.fromisoformat(start_time)
            item.description = description
            item.save()


        return JsonResponse({'status': 'success'})

@method_decorator(login_required, name='dispatch')
class DeleteItemView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
            logger.info(f"Parsed data: {data}")
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)
```

```
        #Пошук події чи завдання за її id
        item_id = data.get('id')
        item_type = data.get('item_type')

        if not item_id or not item_type:
            logger.warning("Missing required fields: id or item_type")
            return JsonResponse({'status': 'error', 'message': 'Missing required fields: id or
item_type'}, status=400)

        user = request.user
        if item_type == 'event':
            item = get_object_or_404(Event, id=item_id, user=user)
        elif item_type == 'task':
            item = get_object_or_404(Task, id=item_id, user=user)
        else:
            return JsonResponse({'status': 'error', 'message': 'Invalid item type'},
status=400)

        item.delete()
        return JsonResponse({'status': 'success'})
```

Подібна, проте дещо інша, логіка також буде наявна і в kanban модулі:

```
    @method_decorator(login_required, name='dispatch')
class CreateKanbanCardView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
```

```
            logger.info(f"Parsed data: {data}")
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        title = data.get('title')
        description = data.get('description', '')
        extra_fields = data.get('extra_fields', {})
        status = data.get('status', 'todo')

        if not title:
            logger.warning("Missing required field: title")
            return JsonResponse({'status': 'error', 'message': 'Missing required field: title'},
status=400)

        user = request.user
        card = KanbanCard.objects.create(
            title=title,
            description=description,
            extra_fields=extra_fields,
            status=status,
            user=user
        )
        return JsonResponse({'status': 'success', 'card': {
            'id': card.id,
            'title': card.title,
            'description': card.description,
            'extra_fields': card.extra_fields,
            'status': card.status
```

```
        }})

@method_decorator(login_required, name='dispatch')
class EditKanbanCardView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        card_id = data.get('id')
        title = data.get('title')
        description = data.get('description')
        extra_fields = data.get('extra_fields', {})

        if not card_id or not title:
            return JsonResponse({'status': 'error', 'message': 'Missing required fields: id or
title'}, status=400)

        user = request.user
        card = get_object_or_404(KanbanCard, id=card_id, user=user)
        card.title = title
        card.description = description
        card.extra_fields = extra_fields
        card.save()
        return JsonResponse({'status': 'success', 'card': {
            'id': card.id,
            'title': card.title,
```

```
            'description': card.description,
            'extra_fields': card.extra_fields,
            'status': card.status
        }})

@method_decorator(login_required, name='dispatch')
class UpdateItemStatusView(View):
    def post(self, request, *args, **kwargs):
        try:
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        card_id = data.get('id')
        status = data.get('status')

        user = request.user
        card = get_object_or_404(KanbanCard, id=card_id, user=user)
        card.status = status
        card.save()
        return JsonResponse({'status': 'success', 'card': {
            'id': card.id,
            'title': card.title,
            'description': card.description,
            'extra_fields': card.extra_fields,
            'status': card.status
        }})

@method_decorator(login_required, name='dispatch')
```

```
class DeleteKanbanCardView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
            logger.info(f"Parsed data: {data}")
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        card_id = data.get('id')
        if not card_id:
            logger.warning("Missing required field: id")
            return JsonResponse({'status': 'error', 'message': 'Missing required field: id'},
status=400)

        user = request.user
        card = get_object_or_404(KanbanCard, id=card_id, user=user)
        card.delete()
        return JsonResponse({'status': 'success'})
```

3.4 Тестування програмної реалізації WEB-додатку

З самого початку перед користувачем виникає меню авторизації. Специфіка WEB-додатку вимагає заборонити користувачеві користуватись функціями без авторизації, тому першим чином йому потрібно ввійти у свій акаунт. Меню авторизації зображене на рисунку 3.1

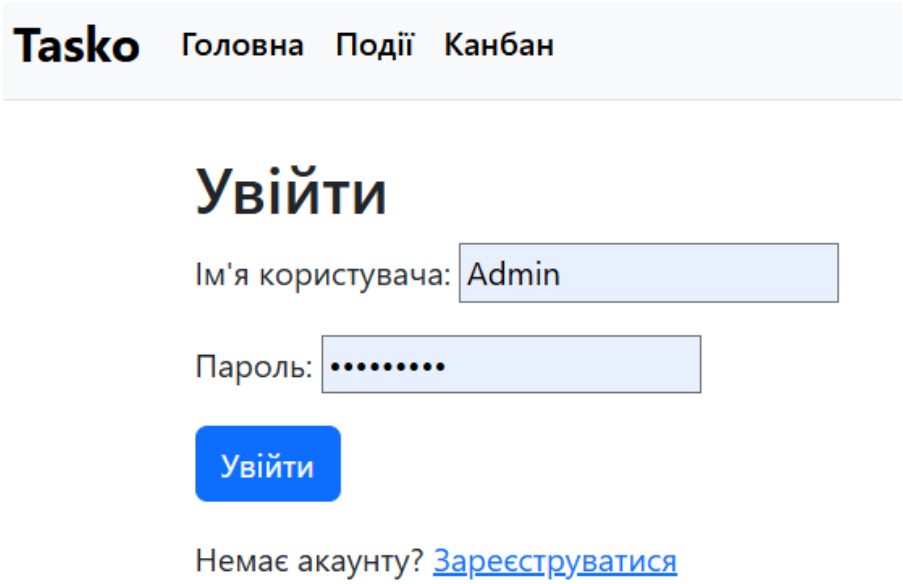

Рисунок 3.1 – Меню авторизації користувача в WEB-додатку “Органайзер для бізнесу”

Для авторизації потрібно ввести дані користувача, тобто логін та пароль. Проте, якщо користувач вперше користується додатком, йому потрібно перейти в меню реєстрації через посилання “Зареєструватися”, після чого він потрапить в меню реєстрації.

В цьому меню йому потрібно придумати логін та надійний пароль. Пароль повинен бути довжиною від 8 символів, а також включати в себе хоча б 2 з 3 перелічених вимог:

- Містити латинські літери (a-z, A-Z)
- Містити цифри (0-9)
- Містити спеціальні символи (! , - , _ , % , & та ін.)

Меню реєстрації зображено на рисунку 3.2. Спроба ввійти з паролем довжиною менше 8 символів зображена на рисунку 3.3. Спроба ввійти з паролем, який не відповідає вимогам надійності зображена на рисунку 3.4.

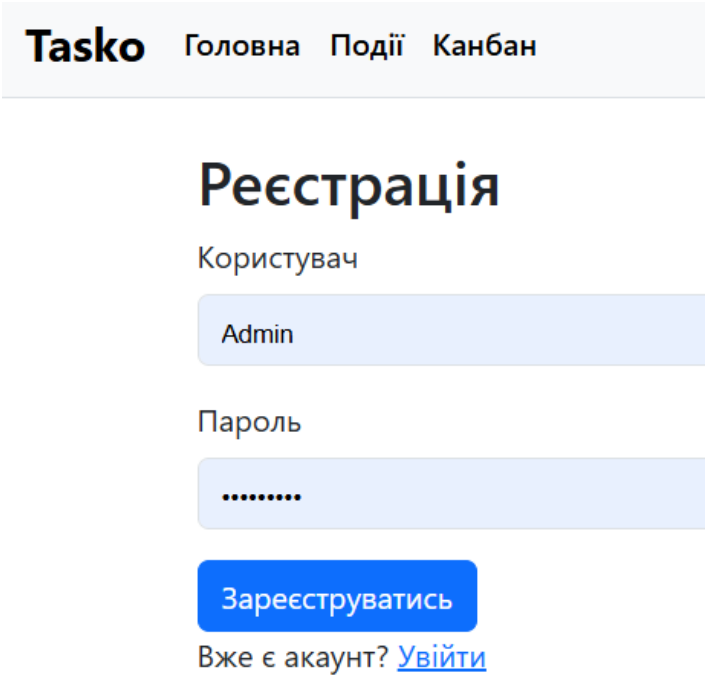

Рисунок 3.2 – Меню реєстрації користувача WEB-додатку “Органайзер для бізнесу”

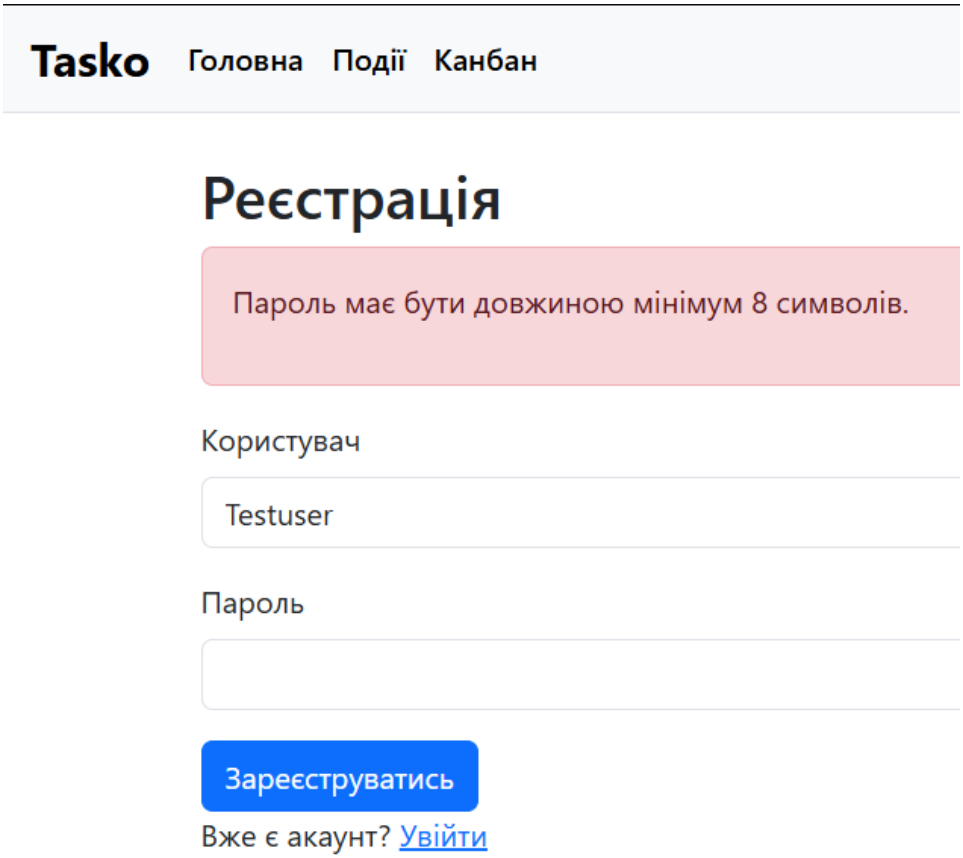

Рисунок 3.3 – Спроба зареєструватись з паролем довжиною менше 8 символів.

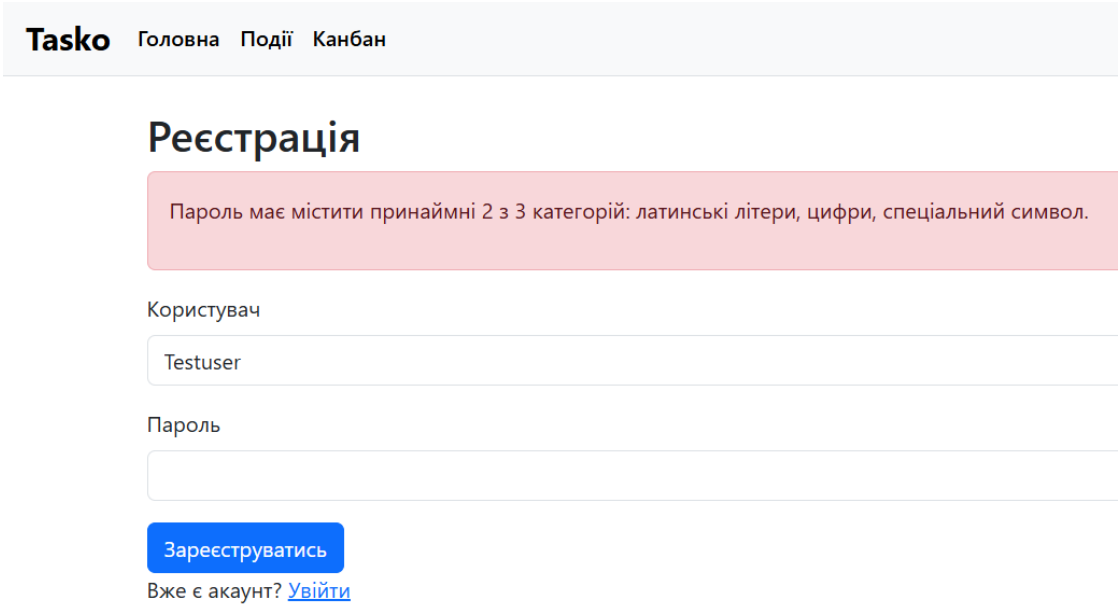

Рисунок 3.4 Спроба зареєструватись без відповідності вимог надійності пароля.

Після успішної авторизації користувач перенаправляється в головне меню, де він може переглянути події чи завдання, які розпочнуться в найближчі 24 години, а також створити нові. Окрім того, В верхній частині екрану знаходиться меню навігації для переміщення між частинами WEB-додатку та кнопка виходу з акаунту, якщо, наприклад, користувач має кілька акаунтів для різних цілей. Інтерфейс головного меню зображено на рисунку 3.5.

Після цього користувач може перейти у вкладку “Події”, щоб переглянути всі заплановані події та завдання в межах місяця. Подіх відображаються у вигляді таблиць, кожна з яких відповідає певній даті, у випадку відсутності подій на той день вона не відображається, зберігаючи практичність вкладки. Також тут користувач може відредагувати та видалити конкретну подію чи завдання у випадку зміни деталей, натискаючи на відповідні кнопки. Інтерфейс меню “Події” зображено на рискунку 3.6.

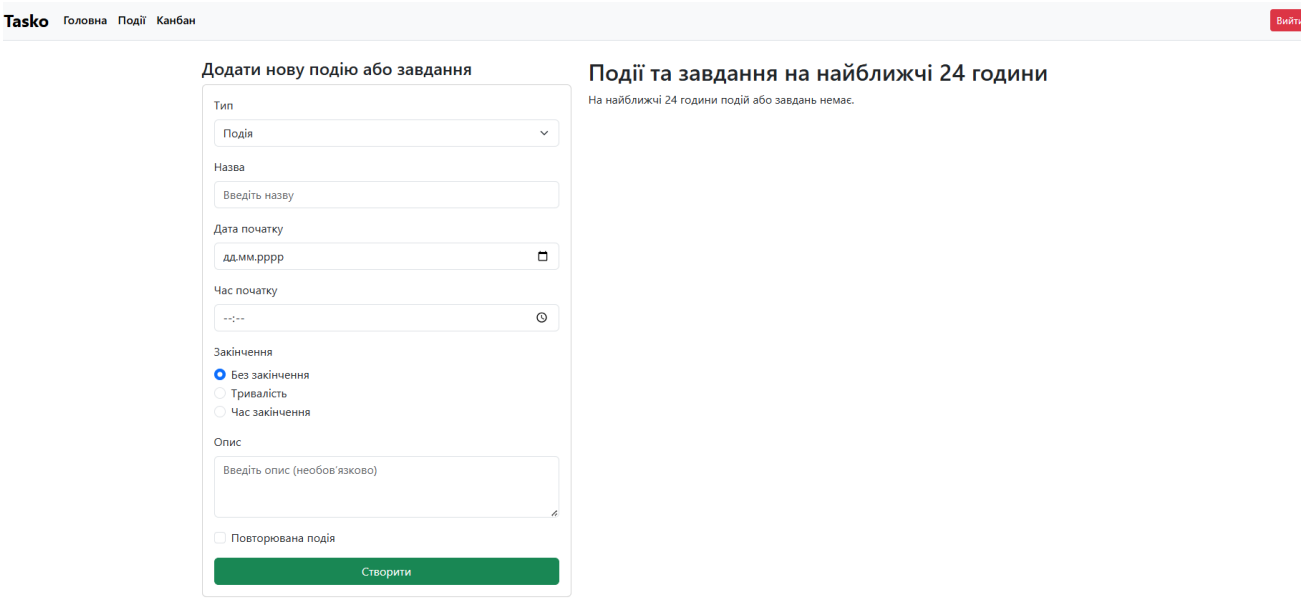

Рисунок 3.5 – Інтерфейс головного меню WEB-додатку “Органайзер для бізнесу”.

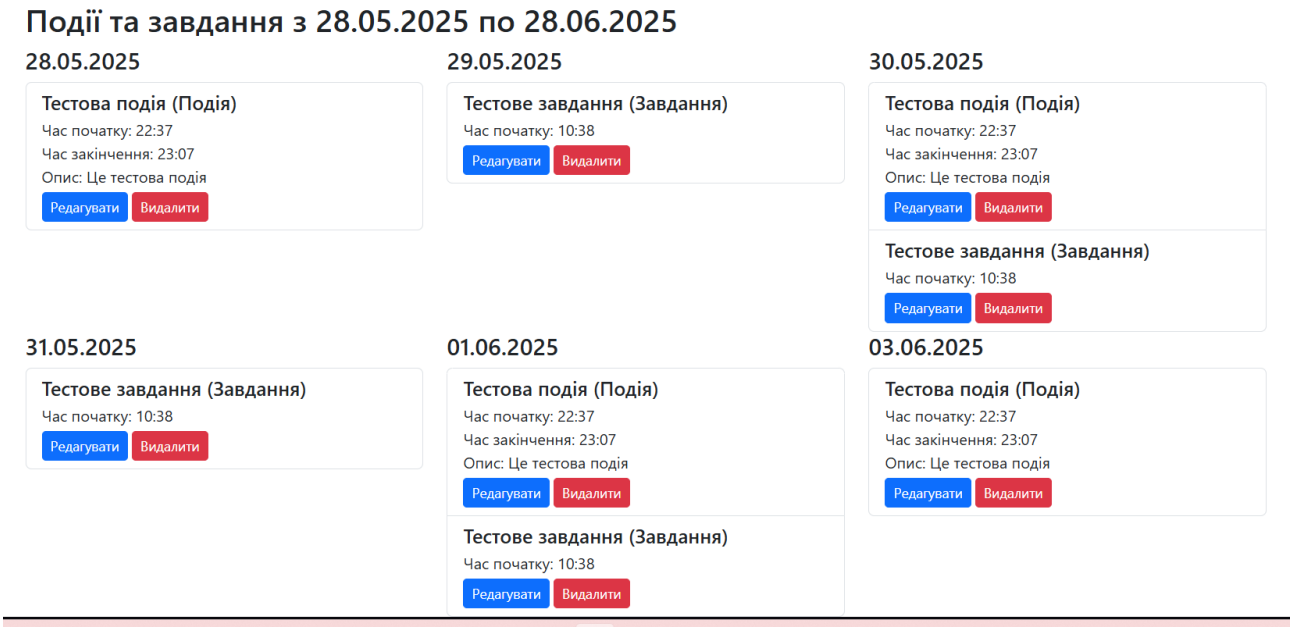

Рисунок 3.6 – Інтерфейс меню “Події” WEB-додатку “Органайзер для бізнесу”.

Після натискання на кнопку “Редагувати” користувач буде перенаправлений в меню редагування, де він може змінити дані, які вже є в події чи завданні, або додати нові. Після зміни даних він буде перенаправлений в меню, з якого перейшов до зміни. Інтерфейс меню редагування зображено на рисунку 3.7.

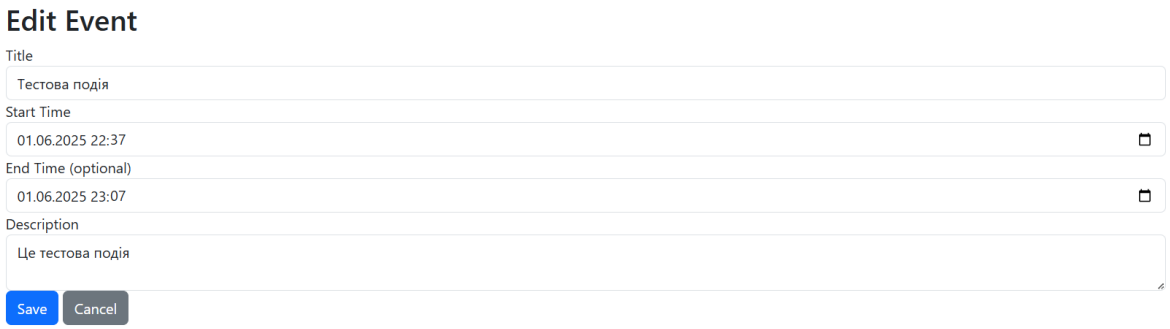

Рисунок 3.7 – Інтерфейс меню редагування WEB-додатку “Органайзер для бізнесу”.

Також користувач може перейти на вкладку kanban-дошки, натиснувши на кнопку “Канбан” в меню навігації. Після натискання він буде перенаправлений в меню управління kanban-дошкою, де відображаються вже створені картки та меню створення нових. Користувач може як створити нову картку, так і перемістити чи змінити наявну. Для переміщення користувач використовує мишу. Інтерфейс меню kanban-дошки зображено на рисунку 3.8.

За потреби редагування чи видалення kanban-картки користувач повинен натиснути на назву картки, після чого відкриється меню редагування картки, де він може змінити назву, опис, додаткові параметри чи видалити картку. Інтерфейс меню редагування kanban-карти зображено на рисунку 3.9.

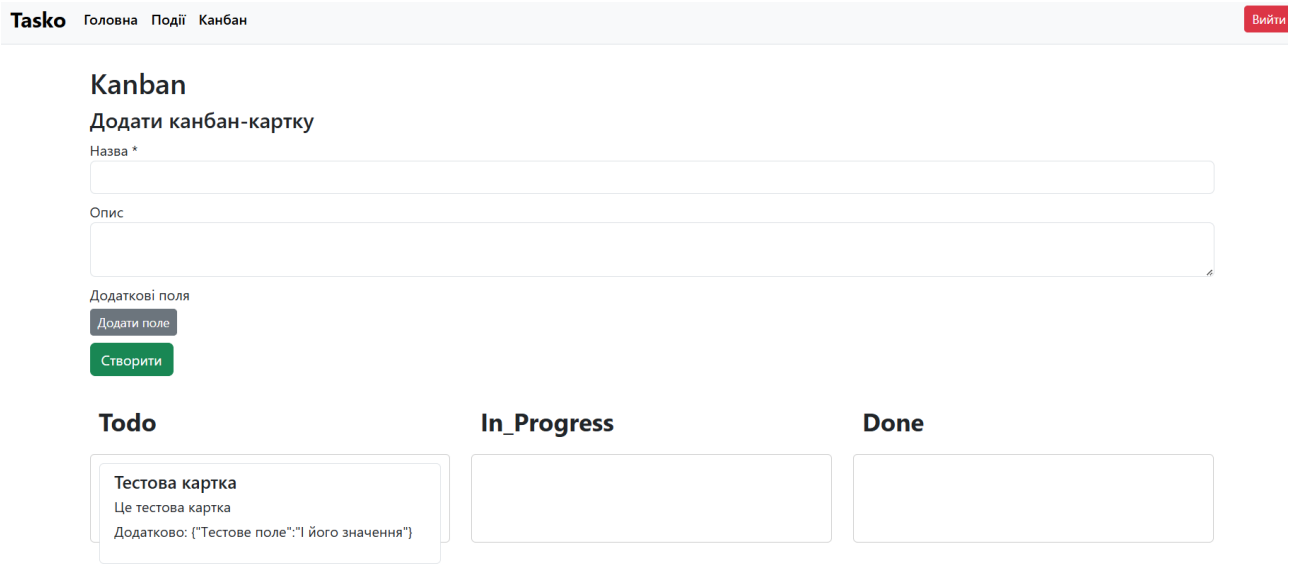

Рисунок 3.8 - Інтерфейс меню kanban-дошки WEB-додатку “Органайзер для бізнесу”.

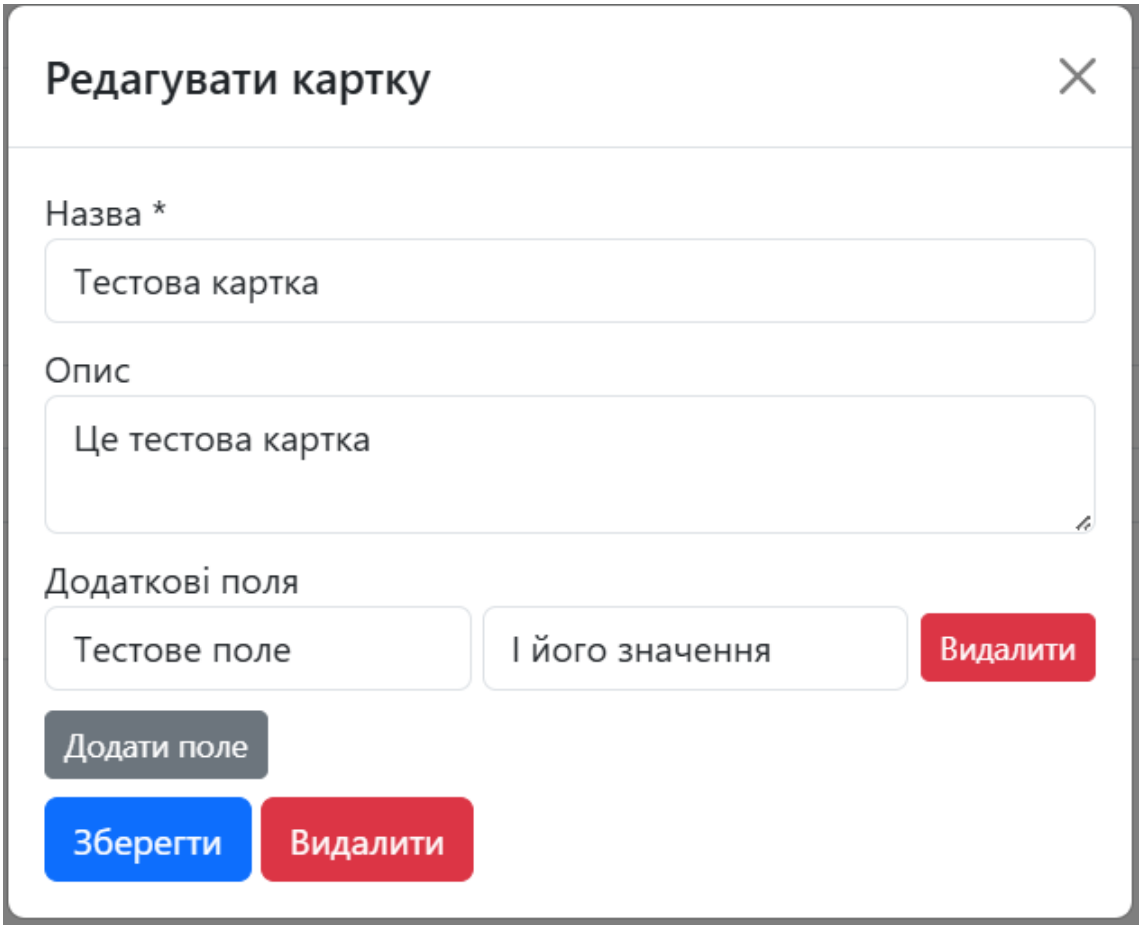

Рисунок 3.9 – інтерфейс меню редагування kanban-карти WEB-додатку “Органайзер для бізнесу”.

3.5 Висновок до розділу 3

У цьому розділі був розроблений WEB-додаток “Органайзер для бізнесу”. Обґрунтовано вибір інтрументів для розробки програмного модуля, а також проведено тестування. Було визначено, що Python є оптимальною мовою для серверної частини завдяки зручному синтаксису та великій кількості бібліотек, що покращує розробку та підтримку додатку. Для клієнтської та серверної частини було обрано вбудовану в Django Models-Views-Templates систему, що забезпечує інтерактивність та ефективне управління станом додатку. Окрім цього, було розглянуто переваги Django для швидкої та ефективної розробки API за рахунок великої кількості вбудованих функцій, які забезпечують не тільки безпечну взаємодію за рахунок відповідності до сучасних норм, а також PostgreSQL як надійну реляційну базу даних через її гнучку підтримку типів даних, відкритий вихідний код та активну спільноту.

В процесі розробки було створено блоки для відображення, створення, редагування та видалення подій, завдань та kanban-карток. На відміну від аналогічних, даний модуль зберіг лише необхідні функції, позбувшись атрибутів та функцій для командного керування, тим самим створивши менш навантажений дизайн та більш зручний для індивідуального використання дизайн.

Також було проведено тестування програмної реалізації. Під час тестування було перевірено усі сценарії взаємодії користувача з додатком, в тому числі, проте не обмежуючись, спробою авторизуватись без реєстрації, використання шкідливих даних з метою отримання несанкціонованого доступу, та стандартного використання додатку користувачем. Тестування показало стабільну, безпечну та ефетктивну роботу додатку.’

ВИСНОВКИ

У даній дипломній роботі було розглянуто задачу зі створення WEB-додатку “Органайзер для бізнесу”. В ході аналізу предметної області було визначено, що є необхідність у створенні додатку, який матиме спеціалізацію з індивідуального використання продуктом. Було розглянуто вже наявні інструменти для організації бізнесу та обрано оптимальні технології для реалізації проекту.

Усі поставленні перед БКР задачі було виконано в повному об'ємі, а саме:

- Обґрунтування доцільності розробки WEB-додатку “Органайзер для бізнесу”.
- Аналіз існуючих програмних рішень для організації бізнесу
- Проектування програмних модулів WEB-додатку “Органайзер для бізнесу”.
- Програмна реалізація програмних модулів WEB-додатку “Органайзер для бізнесу”.
- Тестування програмних модулів WEB-додатку “Органайзер для бізнесу”.

Було розроблено програмну реалізацію WEB-додатку, Включаючи створення та інтеграцію основних модулів. Також було проведення тестуваня WEB-додатку, яке підтвердило працездатність та ефективність розробленої програмної реалізації.

Мета роботи з покращення інтерфейсу та збільшення спеціалізованості для індивідуальних користувачів була досягнута за рахунок інтеграції популярних рішень з організації бізнесу зі зміною та видаленням функцій, які необхідні для командного використання, що спростило інтерфейс та покращило взаємодію саме для індивідуального використання[22].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Традиційне планування [Електронний ресурс]. – Режим доступу: https://riabova.io/blog/upravlenie-riskami-kak-zaschitit-svoj-biznes-ot-nepredskazuemyh-situatsij?utm_source=chatgpt.com, Назва з екрана
2. Проблеми використання цифрових технологій в освіті дорослих [Електронний ресурс]. – Режим доступу: https://dspace.hnpu.edu.ua/items/d9d44ca2-a712-4da5-9aa0-363fde6702f4, Назва з екрана.
3. Навчання медіаграмотності людей старшого віку [Електронний ресурс]. – Режим доступу: https://www.jta.com.ua/trends/navchannia-mediahramotnosti-liudey-starshoho-viku-ievropeyskyy-dosvid/, Назва з екрана.
4. Trello [Електронний ресурс]. – Режим доступу: https://trello.com, Назва з екрана.
5. Asana [Електронний ресурс]. – Режим доступу: https://asana.com, Назва з екрана.
6. Google Calendar [Електронний ресурс]. – Режим доступу: https://calendar.google.com, Назва з екрана.
7. Monday.com [Електронний ресурс]. – Режим доступу: https://monday.com, Назва з екрана.
8. ClickUp [Електронний ресурс]. – Режим доступу: https://clickup.com, Назва з екрана.
9. Монолітна архітектура [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/, Назва з екрана
10. Мікросервісна архітектура [Електронний ресурс]. – Режим доступу: https://wezom.com.ua/ua/blog/scho-take-mikroservisna-arhitektura-znachennya-skladovi-perevagi, Назва з екрана.
11. Клієнт-серверна архітектура [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/, Назва з екрана.

12. Розуміння авторизації, автентифікації та шифрування [Електронний ресурс]. – Режим доступу: https://www.bu.edu/tech/about/security-resources/bestpractice/auth/, Назва з екрана.
13. Повне керівництво процесу продукт-дизайну [Електронний ресурс]. – Режим доступу: https://prodesign.in.ua/2017/08/povne-kerivnytstvo-z-protsesu-produkt-dyzajnu/, Назва з екрана.
14. Bootstrap [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/, Назва з екрана.
15. Python [Електронний ресурс] Режим доступу: https://www.python.org, Назва з екрана.
16. JavaScript [Електронний ресурс] Режим доступу: https://cases.media/article/sho-take-javascript, Назва з екрана.
17. Django [Електронний ресурс]. – Режим доступу: https://www.djangoproject.com, Назва з екрана.
18. Flask [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/stable/, Назва з екрана.
19. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org, Назва з екрана.
20. MariaDB [Електронний ресурс]. – Режим доступу: https://mariadb.org, Назва з екрана.
21. MySQL [Електронний ресурс]. – Режим доступу: https://www.mysql.com, Назва з екрана.
22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)
Протокол перевірки кваліфікаційної роботи

Назва роботи: WEB-додаток "Органайзер для бізнесу"

Тип роботи: бакалаврська кваліфікаційна робота

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук .
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 10,35 % .

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Сілагін О.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Авер'янов В.В..
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)
Лістинг програми

Models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone

class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

class Event(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    start_time = models.DateTimeField()
    end_time = models.DateTimeField(null=True, blank=True)
    description = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class Task(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    start_time = models.DateTimeField()
    description = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
```

```
    def __str__(self):
        return self.title

class KanbanCard(models.Model):
    STATUS_CHOICES = (
        ('todo', 'Заплановано'),
        ('in_progress', 'В процесі'),
        ('done', 'Завершено'),
    )
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    extra_fields = models.JSONField(default=dict, blank=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='todo')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

Views.py

```
from django.views.generic import TemplateView, ListView
from django.views import View
from django.shortcuts import redirect, get_object_or_404, render
from django.http import JsonResponse
from .models import Event, Task, KanbanCard, UserProfile
from django.utils import timezone
from datetime import timedelta
```

```
import json
import logging
from django.contrib.auth.decorators import login_required
from django.utils.decorators import method_decorator
from django.contrib.auth.views import LoginView, LogoutView
from django.urls import reverse_lazy
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login
from .forms import RegisterForm
from calendar import isleap

# Налаштування логування
logger = logging.getLogger(__name__)

@method_decorator(login_required, name='dispatch')
class MainView(TemplateView):
    template_name = 'main/main.html'

        #Отримання інформації
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Tasko'
        now = timezone.now() #Початок часової межі
        end_time = now + timedelta(hours=24) #Кінець часової межі
        user = self.request.user

        #Отримання з бази даних всіх подій
        events = Event.objects.filter(
            user=user,
            start_time__gte=now,
```

```
            start_time__lte=end_time
        ).order_by('start_time')

        #Отримання з бази даних всіх завдань
        tasks = Task.objects.filter(
            user=user,
            start_time__gte=now,
            start_time__lte=end_time
        ).order_by('start_time')

        items = []
        #Відображення всіх подій та завдань з даними про кожну
        for event in events:
            time_until = (event.start_time - now).total_seconds() / 60
            is_in_progress = now >= event.start_time and (event.end_time is None or
now <= event.end_time)
            time_str = f"{int(time_until)} хвилин" if time_until < 60 else
f"{int(time_until // 60)} годин"
            items.append({
                'item': event,
                'type': 'event',
                'time_until': time_str,
                'is_in_progress': is_in_progress,
                'start_time': event.start_time
            })
        for task in tasks:
            time_until = (task.start_time - now).total_seconds() / 60
            time_str = f"{int(time_until)} хвилин" if time_until < 60 else
f"{int(time_until // 60)} годин"
            items.append({
```

```
                'item': task,
                'type': 'task',
                'time_until': time_str,
                'is_in_progress': False,
                'start_time': task.start_time
            })

        items.sort(key=lambda x: x['start_time']) #Сортування по часу
        context['items'] = items
        return context

@method_decorator(login_required, name='dispatch')
class EventListView(TemplateView):
    template_name = 'main/event_list.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        now = timezone.now()
        start_date = now.date()

        # Визначаємо кількість днів залежно від місяця
        month = start_date.month
        year = start_date.year
        if month in [4, 6, 9, 11]:
            days_in_month = 30
        elif month == 2:
            days_in_month = 29 if isleap(year) else 28
        else:
            days_in_month = 31
```

```
end_date = start_date + timedelta(days=days_in_month)

user = self.request.user

events = Event.objects.filter(
    user=user,
    start_time__date__range=(start_date, end_date),
).order_by('start_time')

tasks = Task.objects.filter(
    user=user,
    start_time__date__range=(start_date, end_date),
    start_time__gte=now
).order_by('start_time')

days_with_events = {}

for event in events:
    if event.start_time >= now or (event.end_time and event.end_time >= now):
        date = event.start_time.date()
        if date not in days_with_events:
            days_with_events[date] = {'events': [], 'tasks': []}
        days_with_events[date]['events'].append(event)

for task in tasks:
    date = task.start_time.date()
    if date not in days_with_events:
        days_with_events[date] = {'events': [], 'tasks': []}
    days_with_events[date]['tasks'].append(task)
```

```
        context['days_with_events'] = dict(sorted(days_with_events.items()))
        context['start_date'] = start_date
        context['end_date'] = end_date
        return context

@method_decorator(login_required, name='dispatch')
class KanbanView(TemplateView):
    template_name = 'main/kanban.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        user = self.request.user
        cards = KanbanCard.objects.filter(user=user)
        columns = {
            'todo': [],
            'in_progress': [],
            'done': []
        }
        for card in cards:
            columns[card.status].append(card)
        context['columns'] = columns
        return context

@method_decorator(login_required, name='dispatch')
class CreateItemView(View):
    #Надсилання post запиту до бази даних
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
```

```
            logger.info(f"Parsed data: {data}")
    #У випадку, якщо дані введені невірно
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

    #Список значень, які приймає post запит
        item_type = data.get('item_type')
        title = data.get('title')
        start_time = data.get('start_time')
        end_time = data.get('end_time')
        description = data.get('description', '')
        is_recurring = data.get('is_recurring', False)
        repeat_interval = data.get('repeat_interval', 1)
        repeat_count = data.get('repeat_count', 1)

    #Перевірка обов'язкових даних на наявність
        logger.info(f"Item type: {item_type}, Title: {title}, Start time: {start_time}, End
time: {end_time}")
        if not item_type or not title or not start_time:
            logger.warning("Missing required fields")
            return JsonResponse({'status': 'error', 'message': 'Missing required fields'},
status=400)

    #Перенесення даних з полів введення і створення завдань чи подій
        try:
            if item_type == 'event':
                event = Event.objects.create(
                    title=title,
```

```
                start_time=timezone.datetime.fromisoformat(start_time),
                end_time=timezone.datetime.fromisoformat(end_time) if end_time else
None,
                description=description,
                user=request.user
            )
            if is_recurring:
                for i in range(repeat_count):
                    Event.objects.create(
                        title=title,
                        start_time=event.start_time + timedelta(days=repeat_interval * (i +
1)),
                        end_time=event.end_time + timedelta(days=repeat_interval * (i +
1)) if event.end_time else None,
                        description=description,
                        user=request.user
                    )
        elif item_type == 'task':
            task = Task.objects.create(
                title=title,
                start_time=timezone.datetime.fromisoformat(start_time),
                description=description,
                user=request.user
            )
            if is_recurring:
                for i in range(repeat_count):
                    Task.objects.create(
                        title=title,
                        start_time=task.start_time + timedelta(days=repeat_interval * (i +
1)),
```

```
                    description=description,
                    user=request.user
                )
            else:
                return JsonResponse({'status': 'error', 'message': 'Invalid item type'},
status=400)
        except ValueError as e:
            logger.error(f"Value error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': str(e)}, status=400)
        return JsonResponse({'status': 'success'})

@method_decorator(login_required, name='dispatch')
class EditItemView(View):
    def post(self, request, *args, **kwargs):
        try:
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

    #Отримання даних про подію чи завдання з БД
        item_type = data.get('item_type')
        item_id = data.get('id')
        title = data.get('title')
        start_time = data.get('start_time')
        end_time = data.get('end_time')
        description = data.get('description', '')

        user = request.user
    #Надсилання змінених даних в БД
```

```
        if item_type == 'event':
            item = get_object_or_404(Event, id=item_id, user=user)
            item.title = title
            item.start_time = timezone.datetime.fromisoformat(start_time)
            item.end_time = timezone.datetime.fromisoformat(end_time) if end_time else
None
            item.description = description
            item.save()
        elif item_type == 'task':
            item = get_object_or_404(Task, id=item_id, user=user)
            item.title = title
            item.start_time = timezone.datetime.fromisoformat(start_time)
            item.description = description
            item.save()

        return JsonResponse({'status': 'success'})

@method_decorator(login_required, name='dispatch')
class CreateKanbanCardView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
            logger.info(f"Parsed data: {data}")
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        title = data.get('title')
```

```
        description = data.get('description', '')
        extra_fields = data.get('extra_fields', {})
        status = data.get('status', 'todo')

        if not title:
            logger.warning("Missing required field: title")
            return JsonResponse({'status': 'error', 'message': 'Missing required field: title'},
status=400)

        user = request.user
        card = KanbanCard.objects.create(
            title=title,
            description=description,
            extra_fields=extra_fields,
            status=status,
            user=user
        )
        return JsonResponse({'status': 'success', 'card': {
            'id': card.id,
            'title': card.title,
            'description': card.description,
            'extra_fields': card.extra_fields,
            'status': card.status
        }})

@method_decorator(login_required, name='dispatch')
class EditKanbanCardView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
```

```
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        card_id = data.get('id')
        title = data.get('title')
        description = data.get('description')
        extra_fields = data.get('extra_fields', {})

        if not card_id or not title:
            return JsonResponse({'status': 'error', 'message': 'Missing required fields: id or
title'}, status=400)

        user = request.user
        card = get_object_or_404(KanbanCard, id=card_id, user=user)
        card.title = title
        card.description = description
        card.extra_fields = extra_fields
        card.save()
        return JsonResponse({'status': 'success', 'card': {
            'id': card.id,
            'title': card.title,
            'description': card.description,
            'extra_fields': card.extra_fields,
            'status': card.status
        }})

@method_decorator(login_required, name='dispatch')
class UpdateItemStatusView(View):
```

```
    def post(self, request, *args, **kwargs):
        try:
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        card_id = data.get('id')
        status = data.get('status')

        user = request.user
        card = get_object_or_404(KanbanCard, id=card_id, user=user)
        card.status = status
        card.save()
        return JsonResponse({'status': 'success', 'card': {
            'id': card.id,
            'title': card.title,
            'description': card.description,
            'extra_fields': card.extra_fields,
            'status': card.status
        }})

@method_decorator(login_required, name='dispatch')
class DeleteKanbanCardView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
            logger.info(f"Parsed data: {data}")
        except json.JSONDecodeError as e:
```

```
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        card_id = data.get('id')
        if not card_id:
            logger.warning("Missing required field: id")
            return JsonResponse({'status': 'error', 'message': 'Missing required field: id'},
status=400)

        user = request.user
        card = get_object_or_404(KanbanCard, id=card_id, user=user)
        card.delete()
        return JsonResponse({'status': 'success'})

@method_decorator(login_required, name='dispatch')
class DeleteItemView(View):
    def post(self, request, *args, **kwargs):
        logger.info(f"Request body: {request.body.decode('utf-8')}")
        try:
            data = json.loads(request.body.decode('utf-8')) if request.body else {}
            logger.info(f"Parsed data: {data}")
        except json.JSONDecodeError as e:
            logger.error(f"JSON decode error: {str(e)}")
            return JsonResponse({'status': 'error', 'message': 'Invalid JSON data'},
status=400)

        #Пошук події чи завдання за її id
        item_id = data.get('id')
        item_type = data.get('item_type')
```

```
        if not item_id or not item_type:
            logger.warning("Missing required fields: id or item_type")
            return JsonResponse({'status': 'error', 'message': 'Missing required fields: id or
item_type'}, status=400)

        user = request.user
        if item_type == 'event':
            item = get_object_or_404(Event, id=item_id, user=user)
        elif item_type == 'task':
            item = get_object_or_404(Task, id=item_id, user=user)
        else:
            return JsonResponse({'status': 'error', 'message': 'Invalid item type'},
status=400)

        item.delete()
        return JsonResponse({'status': 'success'})

@method_decorator(login_required, name='dispatch')
class EditItemPageView(TemplateView):
    template_name = 'main/edit_event.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        item_type = self.kwargs['item_type']
        item_id = self.kwargs['item_id']
        user = self.request.user

        if item_type == 'event':
            item = get_object_or_404(Event, id=item_id, user=user)
```

```
        elif item_type == 'task':
            item = get_object_or_404(Task, id=item_id, user=user)
        else:
            raise ValueError("Invalid item type")

        context['item'] = item
        context['item_type'] = item_type
        return context

@method_decorator(login_required, name='dispatch')
class GetItemView(View):
    def get(self, request, *args, **kwargs):
        item_type = self.kwargs['item_type']
        item_id = self.kwargs['item_id']
        user = request.user

        if item_type == 'event':
            item = get_object_or_404(Event, id=item_id, user=user)
            data = {
                'id': item.id,
                'title': item.title,
                'start_time': item.start_time.isoformat(),
                'end_time': item.end_time.isoformat() if item.end_time else None,
                'description': item.description if item.description else ''
            }
        elif item_type == 'task':
            item = get_object_or_404(Task, id=item_id, user=user)
            data = {
                'id': item.id,
                'title': item.title,
```

```
            'start_time': item.start_time.isoformat(),
            'description': item.description if item.description else ''
        }
    elif item_type == 'kanban':
        item = get_object_or_404(KanbanCard, id=item_id, user=user)
        data = {
            'id': item.id,
            'title': item.title,
            'description': item.description if item.description else '',
            'extra_fields': item.extra_fields if item.extra_fields else {},
            'status': item.status  # Додано: повертаємо статус
        }
    else:
        return JsonResponse({'status': 'error', 'message': 'Invalid item type'})

    return JsonResponse(data)

@method_decorator(login_required, name='dispatch')
class GetItemsView(View):
    def get(self, request, *args, **kwargs):
        date_str = request.GET.get('date')
        try:
            date = timezone.datetime.strptime(date_str, '%Y-%m-%d').date()
        except (ValueError, TypeError):
            date = timezone.now().date()

        user = request.user
        events = Event.objects.filter(start_time__date=date, user=user)
        tasks = Task.objects.filter(start_time__date=date, user=user)
```

```
        items = []
        for event in events:
            items.append({
                'id': event.id,
                'type': 'event',
                'title': event.title,
                'start_time': event.start_time.isoformat(),
                'end_time': event.end_time.isoformat() if event.end_time else None,
                'description': event.description if event.description else ''
            })
        for task in tasks:
            items.append({
                'id': task.id,
                'type': 'task',
                'title': task.title,
                'start_time': task.start_time.isoformat(),
                'description': task.description if task.description else ''
            })

        return JsonResponse({'items': items})

class CustomLoginView(LoginView):
    template_name = 'main/login.html'

    def get_success_url(self):
        next_url = self.request.GET.get('next')
        if next_url:
            return next_url
        return reverse_lazy('main')
```

```
class CustomLogoutView(LogoutView):
    next_page = reverse_lazy('login')

class RegisterView(View):
    template_name = 'main/register.html'

    def get(self, request):
        form = RegisterForm()
        return render(request, self.template_name, {'form': form})

    def post(self, request):
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.set_password(form.cleaned_data['password'])
            user.save()
            UserProfile.objects.create(user=user)
            login(request, user)
            return redirect('main')
        return render(request, self.template_name, {'form': form})
```

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.MainView.as_view(), name='main'),
```

```
    path('events/', views.EventListView.as_view(), name='events'),
    path('kanban/', views.KanbanView.as_view(), name='kanban'),
    path('create_item/', views.CreateItemView.as_view(), name='create_item'),
    path('edit_item/', views.EditItemView.as_view(), name='edit_item'),
    path('create_kanban_card/', views.CreateKanbanCardView.as_view(),
name='create_kanban_card'),
    path('edit_kanban_card/', views.EditKanbanCardView.as_view(),
name='edit_kanban_card'),
    path('update_item_status/', views.UpdateItemStatusView.as_view(),
name='update_item_status'),
    path('delete_kanban_card/', views.DeleteKanbanCardView.as_view(),
name='delete_kanban_card'),
    path('delete_item/', views.DeleteItemView.as_view(), name='delete_item'),
    path('edit/<str:item_type>/<int:item_id>/', views.EditItemPageView.as_view(),
name='edit_item_page'),
    path('get_item/<str:item_type>/<int:item_id>/', views.GetItemView.as_view(),
name='get_item'),
    path('get_items/', views.GetItemsView.as_view(), name='get_items'),
    path('login/', views.CustomLoginView.as_view(), name='login'),
    path('logout/', views.CustomLogoutView.as_view(), name='logout'),
    path('register/', views.RegisterView.as_view(), name='register'),
]
```

ДОДАТОК В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

"WEB-ДОДАТОК "ОРГАНАЙЗЕР ДЛЯ БІЗНЕСУ""

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Авер'янов В.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)
«3» Червня 2025 р.

Вінниця ВНТУ - 2025 рік

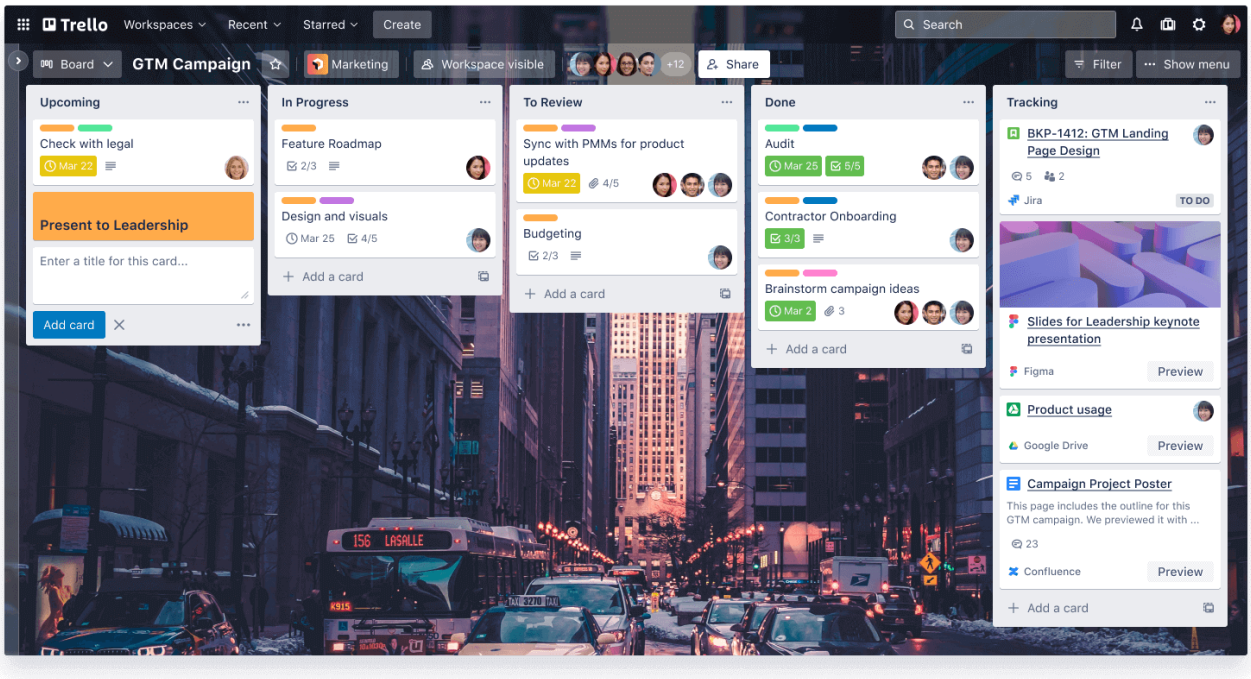

Рисунок В.1 – Приклад інтерфейсу веб-застосунку Trello

Рисунок В.2 – Приклад інтерфейсу веб-застосунку Google Calendar

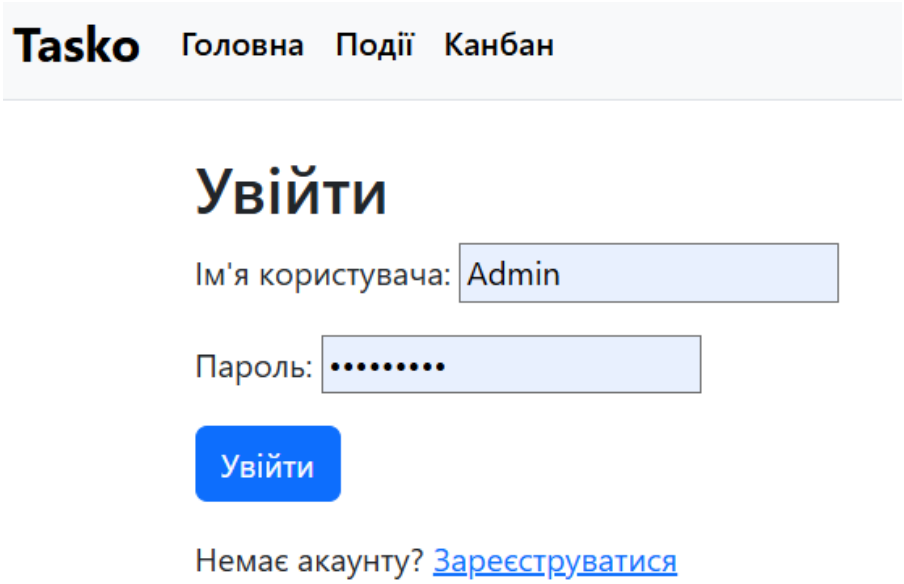

Рисунок В.3 – Меню авторизації користувача в WEB-додатку “Органайзер для бізнесу”

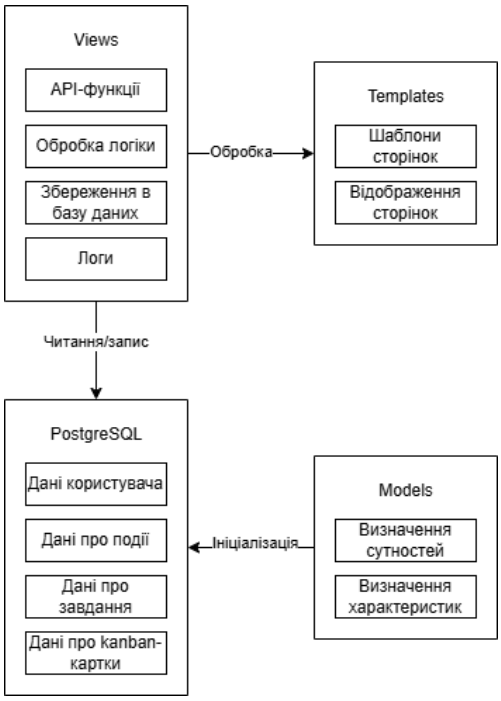

Рисунок В.4 – UML-діаграма компонентів WEB-додатку “Органайзер для бізнесу”

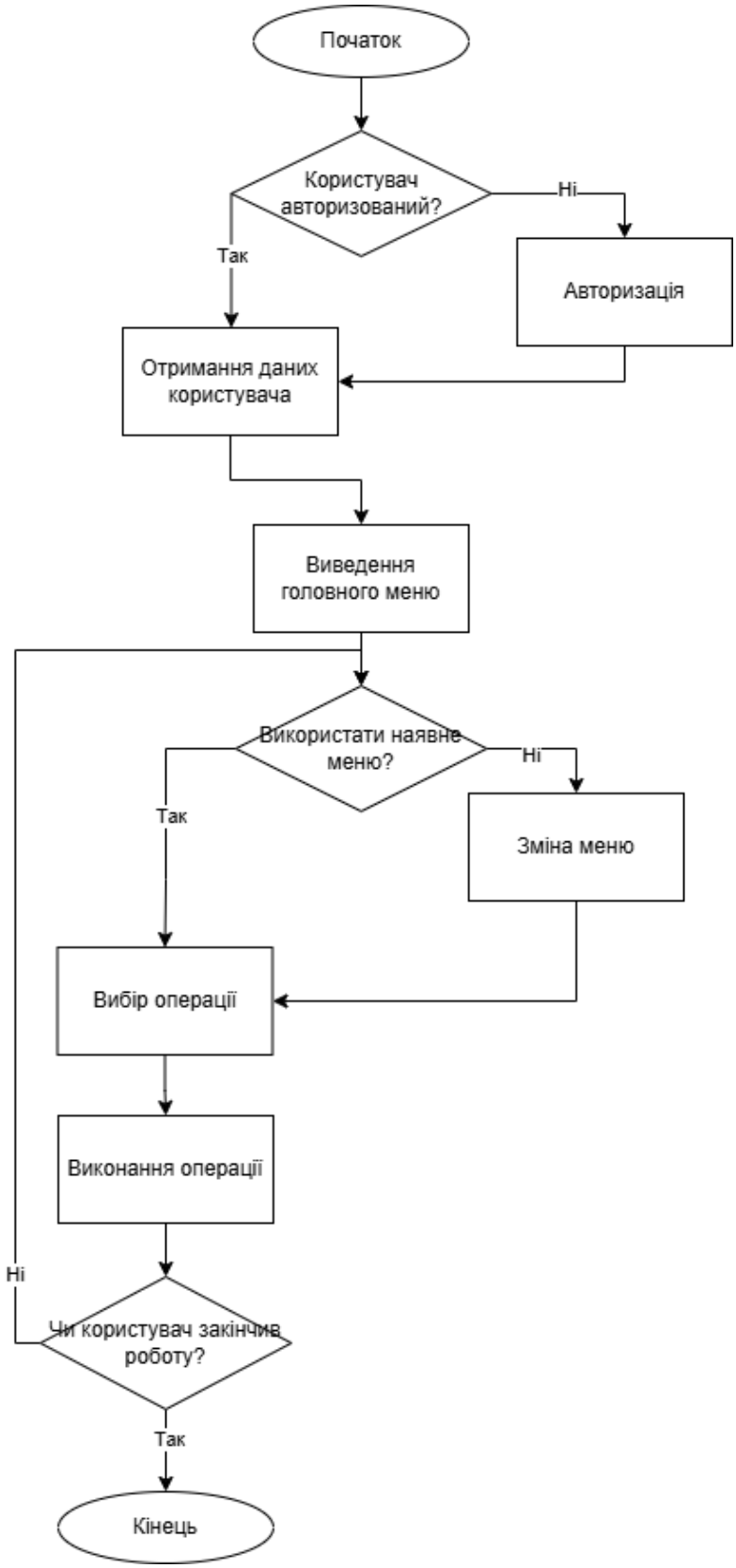

Рисунок В.5 – Граф-схема алгоритму роботи WEB-додатку “Органайзер для бізнесу”

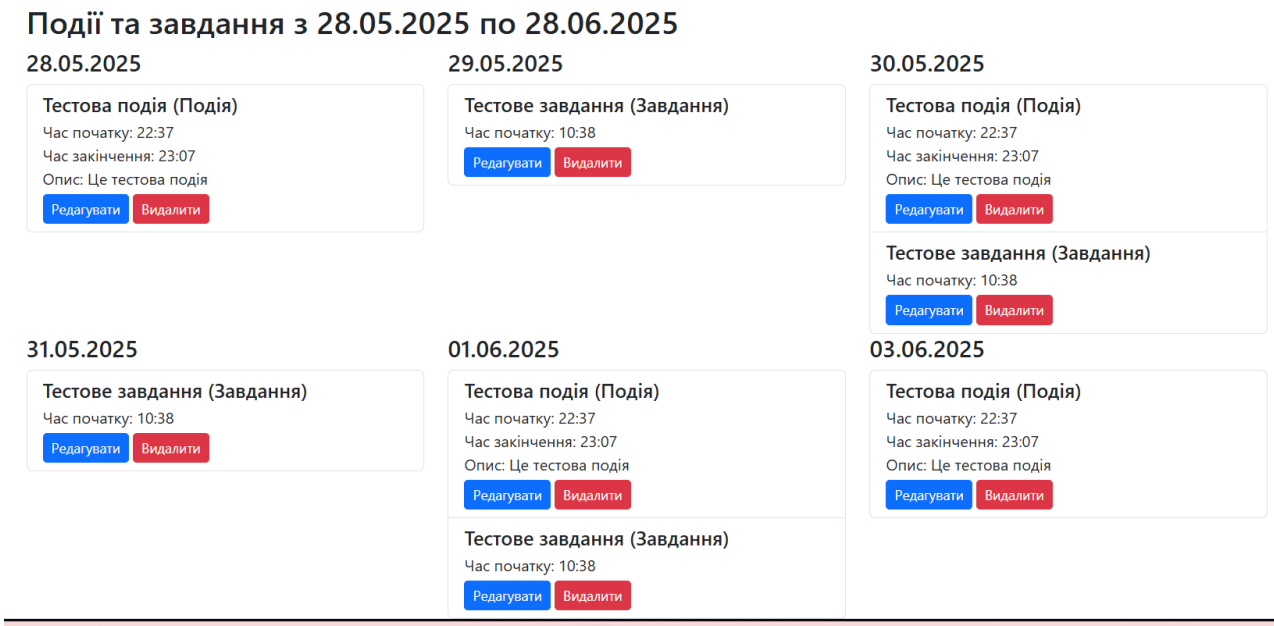

Рисунок В.6 – Інтерфейс меню “Події” WEB-додатку “Органайзер для бізнесу”.