ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка веб-додатку для управління завданнями на основі Django та Python»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олег БРАТКОВСЬКИЙ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Олег БРАТКОВСЬКИЙ
група ІСДМ-61

Керівник: Аліна ТУШИЧ
науковий ступінь, вчене звання д.ф., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Братковському Олегу Валерійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка веб-додатку для управління завданнями на основі Django та Python

керівник кваліфікаційної роботи Аліна ТУШИЧ д.ф., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, протоколи передачі даних, AJAX, вимоги до веб-додатку, результати досліджень та аналізу.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Характеристика та аналіз предметної області;
Моделювання та проектування програмного продукту;
Розробка веб-додатку для управління завданнями.
5. Перелік графічного матеріалу: *презентація*

1. Мета і завдання;
2. Актуальність;
3. UML-діаграми (прецедентів, компонентів, діяльності);
4. Фізична модель даних системи;
5. Специфікація вимог системи;
6. Екранні форми інтерфейсу;
7. Результати роботи веб-додатку у вигляді знімків екрану;
8. Висновки.

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури.	19.10-27.10.23	виконав
2	Дослідження предметної області. Культура безпеки даних. Аналіз існуючих рішень.	27.10-05.11.23	виконав
3	Обґрунтування вибору підходів і технологій для створення веб-додатку.	05.11-10.11.23	виконав
4	Аналіз протоколів передачі даних та дослідження використання AJAX-запитів.	10.11-23.11.23	виконав
5	Визначення специфікації вимог. Обґрунтування архітектурних рішень.	23.11-28.11.23	виконав
6	Розроблення алгоритму роботи програмного продукту. Обґрунтування вибору інструментальних засобів розроблення.	28.11-03.12.23	виконав
8	Розробка веб-додатку на основі Django та Python. Опис компонентів програмного продукту. Безпека даних у веб-додатку.	03.12-20.12.23	виконав
9	Аналіз результатів. Опис роботи програмного продукту	20.12-23.12.23	виконав
10	Підготовка доповіді, презентації, роздаткового матеріалу, реферату	23.12-26.12.23	виконав

Здобувач вищої освіти _____ Олег БРАТКОВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Аліна ТУШИЧ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Магістерська атестаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел і загалом має 105 стор., 37 рис., 32 джерел.

Мета роботи – розробка веб-додатку для управління персональними і робочими завданнями, а також створення завдань для учасників проекту і відстеження статусу їх виконання.

Об'єктом дослідження є процес розробки веб-додатка для ефективного управління завданнями та проектами з використанням Django.

Предмет дослідження – веб-додаток для управління завданнями.

Короткий зміст роботи: Проаналізовано сферу управління завданнями. Було здійснено аналіз існуючих рішень на ринку. Проаналізовано основні правила культури безпеки та забезпечення безпеки веб-додатків. Проаналізовано основні моделі розробки. Досліджено та створено діаграми вимог та варіантів використання веб-додатку. Розроблено алгоритм використання додатка. Здійснено аналіз існуючих архітектурних рішень для розробки проекту. Досліджено основні технології веб-розробки для реалізації додатка. Розроблено веб-додаток для управління завданнями на основі Django та Python.

Інструменти розроблення дипломного проекту: Visual Studio Code, Python, Django, JS, HTML, CSS, CASE–засіб Enterprise Architect, SQLite.

Сфера застосування: веб-додаток є універсальним інструментом, що спрощує рутинні завдання для кожного користувача та стає затребуваним у розвитку і досягненні успіху в бізнесі, управлінні проектами чи в повсякденному керуванні особистими справами. Завдяки своїй гнучкості, з часом може бути легко модифікований відповідно до вимог бізнес-середовища.

ABSTRACT

The master's thesis consists of an introduction, three chapters, general conclusions, a list of used sources and has a total of 105 pages, 37 figures, 32 sources.

The purpose of the work is to develop a web application for managing personal and work tasks, as well as creating tasks for project participants and tracking the status of their completion.

The object of research is the process of developing a web application for effective task and project management using Django.

The subject of research is a web application for task management.

Summary of the work: The field of task management is analyzed. Analysis of existing solutions on the market was carried out. The main rules of security culture and security of web applications are analyzed; The main development models are analyzed. Researched and created requirements and use case diagrams for the web application. An algorithm for using the application has been developed. Analysis of existing architectural solutions for project development was carried out. The main technologies of web development for the implementation of the application have been studied. Developed a web application for task management based on Django and Python.

Tools for the development of the diploma project: Visual Studio Code, Python, Django, JS, HTML, CSS, CASE–the Enterprise Architect tool, SQLite.

Field of application: the web application is a universal tool that simplifies routine tasks for each user and is in demand in business development and success, project management or in the daily management of personal affairs. Due to its flexibility, it can be easily modified over time according to the requirements of the business environment.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SMTP	Simple Mail Transfer Protocol
ПП	Програмний продукт
ПЗ	Програмне забезпечення
КБ	Комп'ютерна безпека
JSON	JavaScript Object Notation
DOM	Document Object Model
CI/CD	Continuous Delivery
URL	Uniform Resource Locator
UML	Unified Modeling Language
TLS	Transport Layer Security
SSL	Secure Sockets Layer
SQL	Structured query language
SaaS	Software as a service
FaaS	Function-as-a-service
PaaS	Platform as a service
IaaS	Infrastructure as a Service
XHTML	Extensible Hypertext Markup Language
XML	EXtensible Markup Language
HTTPS	Hyper Text Transfer Protocol Secure
HTTP	Hyper Text Transfer Protocol
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
AWS	Amazon Web Services
AJAX	Asynchronous JavaScript and XML)
API	Application Programming Interface
ORM	Object-relational mapping
IP	Internet Protocol

ВСТУП

Актуальність. В умовах стрімкого розвитку нових технологій і широкого переходу різних установ на дистанційну форму праці, з'являються нові виклики і проблеми у сфері управління завданнями та проектами, їх грамотному розподілу і ефективному використанню часу. Рано чи пізно накопичується велика кількість справ, все запам'ятати не завжди виходить, тому і спостерігаються проблеми з ефективністю виконання поставлених завдань, з'являється прокрастинація через домашню атмосферу і відсутність контролю, віддалена координація робочих процесів та зниження рівня продуктивності працівників стають актуальними проблемами, що суттєво впливають на результативність діяльності окремих людей і компаній в цілому.

У ситуаціях, коли є велика кількість завдань, які потрібно ефективно виконувати та різноманітних проектів, що потребують уваги та ресурсів, з'являється потреба впровадження інструментів для управління завданнями. Сучасні реалії вимагають від компаній незалежно від їх розміру ефективних інструментів для управління проектами та контролю ресурсів з метою забезпечення успішності та конкурентоспроможності на ринку.

Актуальність розробки веб-додатку для управління завданнями на основі Django та Python полягає в його потенційній здатності вирішувати вказані вище проблеми та оптимізувати робочі процеси.

Об’єктом дослідження є процес розробки веб-додатка для ефективного управління завданнями та проектами з використанням Django та Python.

Предмет дослідження: веб-додаток для управління завданнями.

Мета дослідження: розробка веб-додатку для управління персональними і робочими завданнями, а також створення завдань для учасників проекту і відстеження статусу їх виконання.

На меті надати майбутнім користувачам максимальну легкість використання та інтуїтивно зрозумілий інтерфейс системи для забезпечення потреб компанії,

команди, групи людей і просто звичайних користувачів, які мають потребу в ефективному управлінні своїми завданнями. Додаток повинен надавати зручний набір інструментів створення та відстеження прогресу на завданнях для учасників проекту чи звичайних користувачів.

Для досягнення цієї мети виникла необхідність виконання наступних завдань:

- дослідження предметної область для якої створюється система;
- встановити вимоги до системи (бізнес-вимоги, функціональні та нефункціональні);
- обґрунтування вибору технологій для розробки системи;
- створення функціональної моделі системи;
- розробка структурного алгоритму роботи системи;
- проектування архітектури системи;
- створення бази даних для системи;
- розробка інтерфейсу користувача;
- демонстрація готового веб-додатку.

Практичне значення одержаних результатів полягатиме в можливості застосування розробленого веб-додатку для полегшення робочих процесів, моніторингу виконання поставлених завдань та підвищення продуктивності на рівні окремих фахівців, команд та організацій.

Використані інструментальні засоби розробки:

- мови програмування JavaScript, Python та framework Django;
- CASE-засіб Enterprise Architect;
- середовище розробки Microsoft Visual Studio Code;
- операційна система Windows 10;
- текстовий редактор Microsoft Word 2016 для підготовки та оформлення пояснювальної записки до дипломного проекту.

Структура роботи, Робота складається зі вступу, трьох розділів, висновків, додатків та переліку посилань.

1 ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження предметної області

Завдання – це конкретні дії або цілі встановлені певною особою, які необхідно виконати для отримання бажаного результату[1]. Вони можуть бути, як повсякденними, особистими чи робочими обов’язками, так і стратегічними цілями в бізнесі чи навчанні.

Для ефективного і злагодженого управління часом та досягнення поставлених цілей використовується планування завдань. Сюди відносяться визначення пріоритетів, встановлення конкретних завдань та визначення часу на їх виконання. Це дозволяє визначити кроки, які потрібно зробити для успішного завершення завдання, забезпечуючи управління від моменту створення до його виконання. Управління завданнями є необхідним для планування та досягнення поставлених цілей. Цей процес включає в себе створення, планування та контроль виконання завдань і є фундаментальною частиною для організації робочого та особистого життя чітко та структуровано.

Кожна компанія або звичайна людина рано чи пізно зіштовхується з проблемою планування та розподілу завдань, ефективному управлінні часом. Раніше, люди вели різного роду щоденники та записи на папері для планування своїх задач, розподілу між працівниками та відстеження прогресу, що не було зручним, адже записи могли легко втрачатись. Вони займали додатковий простір в робочій зоні і були мало-ефективними для управління великою кількістю людей, так як потребували значних затрат в часі щоб розписати завдання кожному працівнику, а ще потребували найму кількох менеджерів, які б займались їх написанням та розподілом.

В сучасних умовах повномасштабної війни на території України бізнес змушений шукати максимально швидко нові підходи, шляхи покращення і

адаптації своїх форм та методів роботи. Гнучко реагувати на нові виклики і не втрачати своїх виробничих потужностей, зберігати контроль за наданням якісних послуг своїх працівників та мати ефективну комунікацію з ними для розподілу та призначення завдань.

Веб-додатки для управління завданнями являють собою інноваційний інструмент, що значно полегшує процеси створення, організації та відстеження статусу завдань і їх прогресу, добре підійдуть для дистанційної форми співпраці, завдяки доступному веб-інтерфейсу, що значно підвищує продуктивність людей, впливає на якість виконання роботи, підвищує дисциплінованість в житті. Він є не лише зручним, але й обов'язковим. Ефективного управління ресурсами та завданнями є ключем до успіху як в бізнесі так і в особистому житті.

Є кілька видів веб-додатків для управління завданнями:

- персональні організатори: створення, планування та відстеження особистих завдань;
- системи для управління командами: спільне використання у командах та проектах, моніторинг статусів і деталей;
- списки та календарі: вони надають змогу планувань на великі проміжки часу, широку оглядність і легкість відстеження завдань.

Основні переваги використання веб-додатку для управління завданнями:

- розмежування особистих та робочих завдань, що унеможливлює можливість виникнення плутанини та перемішування;
- гнучкість, що надає можливість редагування та оновлення прогресу завдань і режимі реального часу;
- підвищений захист безпеки даних завдяки вбудованим і налаштованим технологіям Django;
- просте управління завданнями та можливість відстеження затраченого часу;

- можливість створення груп користувачів для розподілу працівників за структурними підрозділами і їх подальшого фільтрування для зручності призначення завдань;
- отримання сповіщень про нові завдання;
- відстеження дедлайну завершення завдань.

1.2 Культура безпеки даних

Культура безпеки даних є важливим елементом у взаємодії та розробці веб-додатку для управління завданнями. Кожного дня інформаційні системи та технології все більше проникають в життя сучасних людей[3]. Кожен бажаючий завдяки доступу в мережу інтернет може легко скористатись цифровими послугами з купівлі та продажу чогось, замовлень товарів, обміну повідомленнями. Усі ці дії в мережі здійснюють обмін особистою інформацію чи конфіденційними даними. В сучасному світі приватні дані стають все більшою ціллю для злочинців та шахраїв, вашу інформацію можуть використати в самих різних цілях і тим самим принести багато шкоди. Тому потрібно мати чітке розуміння основних правил кібергігієни всіх користувачам, а підприємствам вибудовувати культуру корпоративної безпеки для використання та дотримання працівниками всередині установи.

Корпоративна безпека – це певна система заходів та стратегій, які мають чітку структуру і поєднують в собі основні принципи та методи в області безпеки. Ці заходи впроваджені у бізнес-процеси враховуючи кращі практики корпоративного управління[2]. Сукупність цих заходів дозволяє забезпечити безперебійну та стійку роботу компанії, безперебійне отримання прибутку та дозволяє визначати стратегії розвитку на майбутнє.

Культура корпоративної безпеки – це набір правил та порад, завдяки яким співробітники чітко розуміють важливість заходів безпеки та активно приймають

участь в забезпеченні безпеки компанії від зовнішніх та внутрішніх кіберзагроз, сприяють зниженню кількості ризиків.

Приклади культури корпоративної безпеки:

- працівники знають різницю між слабкими і надійними паролями, згідно інформаційних політик компанії регулярно змінюють пароль;
- працівники вміють виявляти фішингові електронні листи і в разі їх отримання, терміново інформують менеджерів відділу КБ;
- співробітники повинні негайно інформувати керівництво в разі підозри на скомпрометовані дані облікового запису;
- співробітникам заборонено ділитись паролями, кодами доступу, електронними пропусками або іншими ресурсами;
- при потребі працювати на особистому пристрої, працівник має отримати дозвіл від менеджера КБ , після чого створити окремий корпоративний акаунт для роботи та використовувати його лише за призначенням;
- весь персонал компанії повинен бути проінструктований про місце, роль та зміст політик безпеки компанії, пройти відповідні тренінги зі стандартів безпеки, знати робочі інструкції;
- на корпоративних облікових записах забороняється встановлювати будь-яке не перевірене та не ліцензійне програмне забезпечення;
- керівники всіх відділів повинні співпрацювати з відділом безпеки відповідно до своїх повноважень.

Хоч бізнес в більшості випадків має розвинену інфраструктуру безпеки, досвідчених менеджерів і цілі відділи кібербезпеки, а також власні захищені сервери – цього все ще недостатньо, якщо працівники компанії не використовуватимуть її ресурси правильно, відповідально та відповідно до безпекових стандартів.

Як приклад, навіть використовуючи найкращий антивірус та вбудовані технології захисту інформації не вбережуть від її можливого витоку, якщо некомпетентний працівник попадеться на фішингове повідомлення і ненавмисно

передасть свої облікові дані злочинцю чи загубить блокнот з логінами та паролями[4]. Для бізнесу це може призвести до великих збитків та перебоїв в роботі, що негативно скажеться на стабільності виробничих процесів, нашкодить репутації чи може і взагалі їх зупинити шляхом виводу інформаційних систем з ладу.

Забезпечення цілісності даних та захисту конфіденційної інформації стає все більш затребуваним завданням найвищого пріоритету для будь-якого веб-додатку. Кібербезпека повинна займати перше місце в кожному бізнес-процесі і враховуватись в управлінських рішеннях. Компанії витрачають великі суми коштів на покращення захисту конфіденційної інформації, пробують вводити нові стандарти безпеки, розробляти додаткові програмні методи захисту, використовувати шифрування та посилювати методи автентифікації користувачів шляхом одноразових кодів підтвердження входу і тд. Проте, користувачам не треба покладатись лише на захист, який надають розробники додатку чи організації в яких вони працюють, потрібно також знати основні правила культури безпеки даних для зменшення ризиків несвідомої допомоги зловмисникам в отриманні доступу до своїх особистих персональних даних та закритої інформації з веб-системи. Компаніям незалежно від їх розмірів, які використовують веб-додатки для реалізації та налаштування бізнес-процесів, процесів взаємодії між собою структурних підрозділів потрібно розробити цілий комплекс заходів для працівників з дотримання правил корпоративної безпеки аби унеможливити витік конфіденційної інформації чи проникнення злочинців до внутрішніх систем.

Серед загальних правил безпеки для усіх користувачів можна виділити наступні:

- використання двоетапної автентифікації в додатках де це можливо;
- використання надійних паролів;
- використання безпечного підключення, уникайте публічних точок доступу для важливих операцій;
- регулярне оновлення систем захисту від вірусів;

- потрібно уважно переглядати посилання веб-сайтів, злочинці можуть робити копії оригіналів і користувач через неуважність самовільно передає свої авторизаційні дані кіберзлочинцям;
- уникайте відкриття електронних листів та переходу за посиланнями, які вони містять, якщо відправником є незнайома для вас людина чи організація, це може бути фішингова атака;
- за можливості, потрібно завжди виконувати резервне копіювання важливих даних;
- в жодному разі не передавати свої дані облікових записів стороннім людям.

Пароль від облікового запису є дуже важливим і чим він надійніший тим складніше злочинцю отримати доступ до конфіденційної інформації у поєднанні з стандартними налаштуваннями функцій захисту додатку, який використовується.

Для створення надійного паролю необхідно дотримуватись наступних правил:

- надійний пароль повинен мати довжину 8-12 символів;
- потрібно використовувати комбінації різних знаків та спеціальних символів, застосовувати великі та малі літери у поєднанні з цифрами та спецсимволами, уникати повторення символів більше двох разів;
- не використовувати особистої інформації в паролях по типу дати народження, імені чи номерів телефону, адже їх доволі легко дізнатись;
- періодично потрібно проводити зміну паролю та уникати його використання для всіх додатків якими користуєтесь, подбайте про унікальність;
- можна використовувати спеціальні менеджери, які надають можливості збереження та генерації надійних паролів.

1.3 Аналіз існуючих рішень веб-додатків для управління завданнями

Виконуючи аналіз існуючих рішень серед веб-додатків, які забезпечують можливість управління завданнями, було розглянуто ряд відомих програмних рішень. Серед найбільш популярних інструментів, які досліджувались в даній сфері, варто виділити Todoist, Trello, Asana та Microsoft To Do. Майже всі додатки надають подібний між собою функціонал, проте мають певні відмінності та ряд переваг і недоліків.

Першим було проаналізовано популярний додаток для управління завданнями Todoist. (див. рис. 1.1)

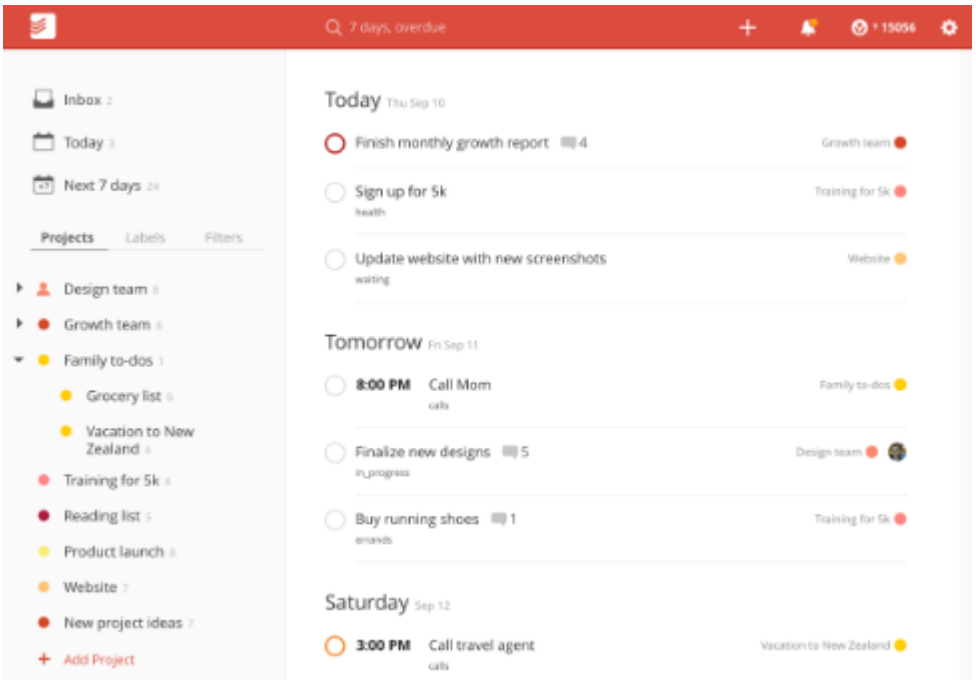


Рисунок 1.1 Інтерфейс веб-додатку для управління завданнями «Todoist»

Основними функціями додатка є:

– створення завдань: Todoist дозволяє користувачам легко створювати нові завдання. Вони можуть швидко вводити опис завдань та встановлювати пріоритети;

- встановлення дедлайнів: Користувачі можуть призначати терміни виконання завдань, встановлюючи дедлайни. Це допомагає ефективно планувати та керувати часом;
- відстеження прогресу: Todoist дозволяє користувачам відстежувати прогрес виконання завдань. Це може бути корисно для визначення того, які завдання вже виконані, а які ще потребують уваги;
- пріоритети та мітки: Завдання можна позначати пріоритетами або додавати до них мітки, що сприяє більш детальному класифікації та організації завдань;
- інтеграція з іншими сервісами: Todoist інтегрується з різними сервісами, такими як електронна пошта, календарі та інші, щоб забезпечити зручність користування та обмін даними.

Попри хороший функціонал веб-додатку, є і недоліки на які треба звернути увагу:

- безкоштовна версія обмежена у функціоналі порівняно із платною версією. Додаткові можливості, такі як збереження завдань для подальшого перегляду, статистика використання є доступними лише для користувачів, які придбали преміум-план;
- у безкоштовній версії, співпраця в команді обмежена. Більшість розширених можливостей для командної роботи доступні лише в рамках платних планів, що є непридатним для використання малим бізнесом, невеликим командами та звичайними групами користувачів;
- для деяких користувачів інтерфейс може здаватися не таким інтуїтивним, особливо для тих, хто тільки починає використовувати дане програмне рішення;
- для великих корпоративних команд чи організацій, може виявитися менш потужним порівняно з аналогами, які пропонують більш продвинуті функції для співпраці та управління завданнями.

Далі аналізувався не менш популярний додаток для управління завданнями Trello. (див. рис. 1.2) Він вирізняється гнучкістю та можливістю візуально керувати завданнями завдяки представленню завдань на спеціальній кард-дошці.

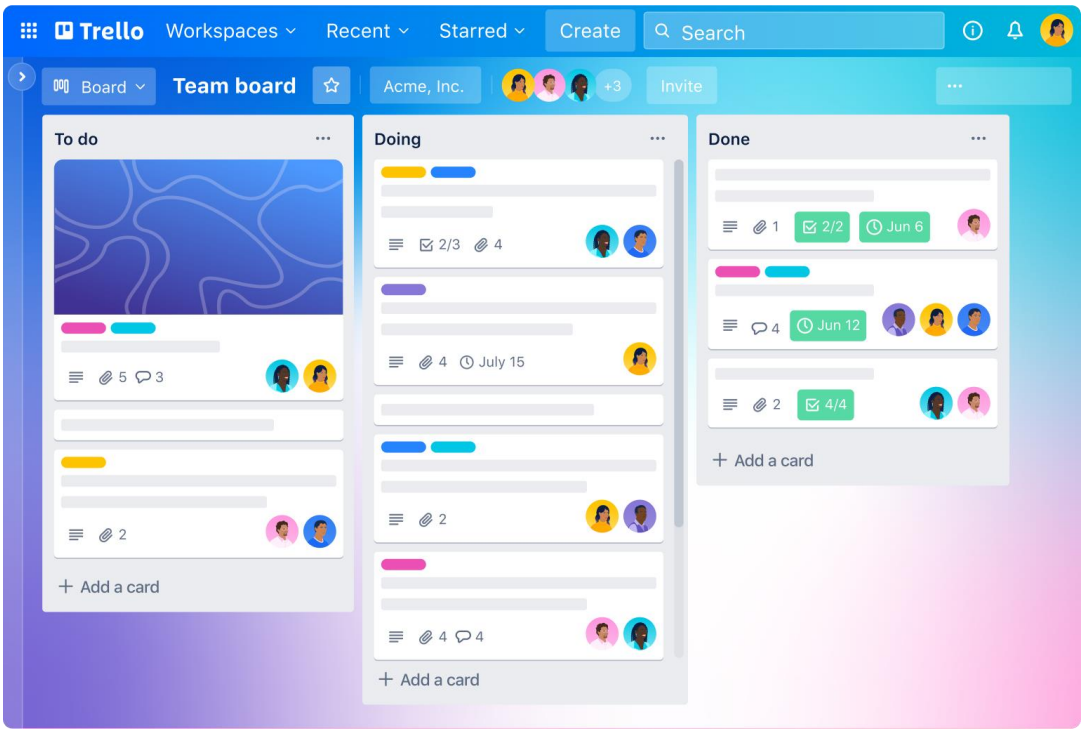


Рисунок 1.2 Інтерфейс веб-додатку для управління завданнями «Trello»

Основні переваги функціоналу, який надай веб-додаток:

- кард-дошка: завдання розміщуються у вигляді спеціальних карток на дошці, що дозволяє легко переглядати завдання в легкий спосіб та організовувати їх;
- завдання можна організовувати в списки та окремі категорії щоб легше встановлювати пріоритети та стадії виконання завдань;
- у випадку корпоративного чи групового використання додатку, до кожної картки можна прикріпити відповідальну особу, що допомагає більш чітко розподілити завдання;
- для обговорення кожного завдання доступна система коментарів, що полегшує рівень комунікації між учасниками проекту, а також можливість прикріплення файлів та додаткової інформації;
- можливість інтеграції з іншими сервісами.

До недоліків Trello можна віднести наступні:

- складність для особистого використання: попри свою гнучкість інтерфейс Trello, може здатися складним для тих, хто шукає простий інструмент для управління особистими завданнями. Велика кількість функцій та налаштувань може спричинити непотрібну складність. Оскільки цей веб-додаток більше спеціалізується для командного використання;
- для великих організацій або проектів, які вимагають високого рівня командної роботи та аналізу, Trello може виявитися менш ефективним у порівнянні з іншими інструментами, які пропонують більше продвинуті функції для більшого корпоративного використання;
- безкоштовна версія має також свох обмеження і для повноцінного використання деяких функцій може знадобитися перехід на платну підписку, яка може бути невигідною для малого бізнесу або груп користувачів, які тільки починають розробку свого стартапу з нуля.

Наступним розглядався додаток для управління завданнями Asana (див. рис. 1.3), який має досить великий функціонал для оптимізації робочих процесів в середині команди.

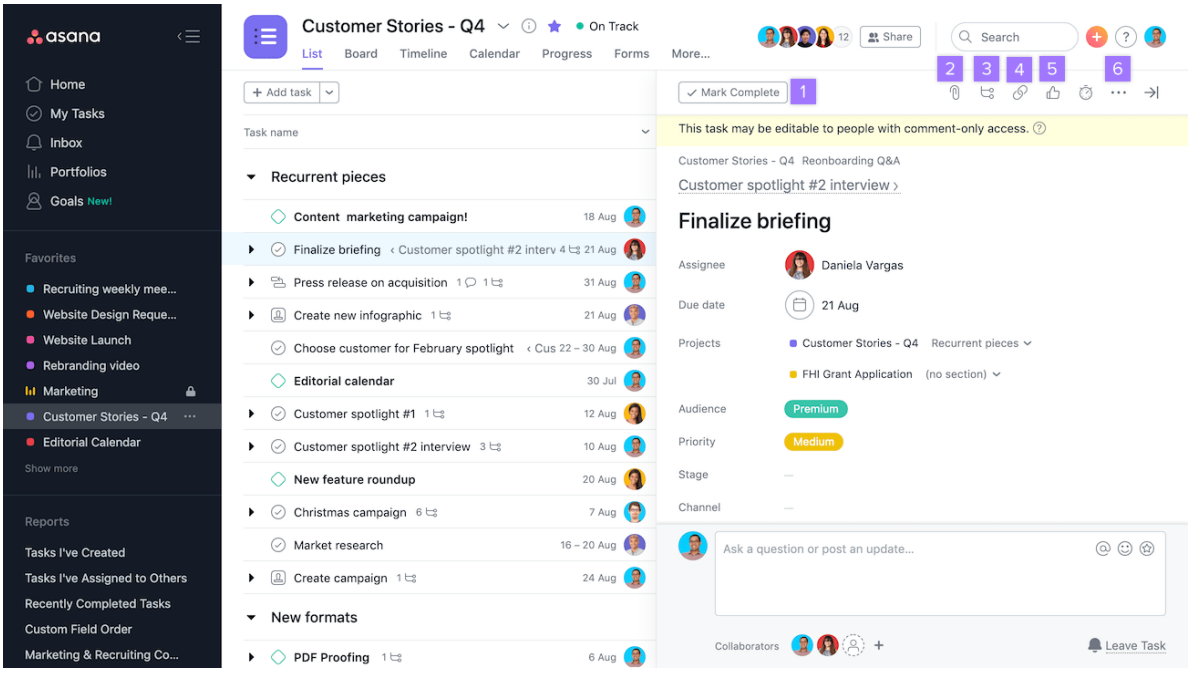


Рисунок 1.3 Інтерфейс веб-додатку для управління завданнями «Asana»

Основні функції включають:

- створення нових завдань, визначаючи опис та пріоритет;
- можна призначати дедлайн;
- можливість призначати конкретних користувачів до завдань;
- організація проектів, що дозволяє створювати ієрархію завдань та категоризувати їх для кращого управління;
- є можливість коментувати та обговорювати завдання;
- присутня можливість інтеграція з іншими сервісами.

Недоліки веб-додатку:

- інтерфейс може виявитися занадто складним для користувачів, які просто шукають ефективний інструмент для особистого управління завданнями;
- відсутність вбудованих таймерів для відстеження часу виконання завдань, що може бути недоліком для тих, хто приділяє увагу ефективному відстеженні та управлінні часом;
- через велику кількість функціональних можливостей користувачам потрібен час для повного освоєння веб-додатку;
- є обмеження в функціональності системи, якщо користувач не придбав преміум-підписку, для малого та середнього бізнесу з великою кількістю працівників може бути не вигідним.

На останок, було розглянуто не менш популярний та розповсюджений додаток для управління завданнями Microsoft To Do (див. рис. 1.4), який використовує велика кількість людей, хто бажає контролювати свій робочий час.

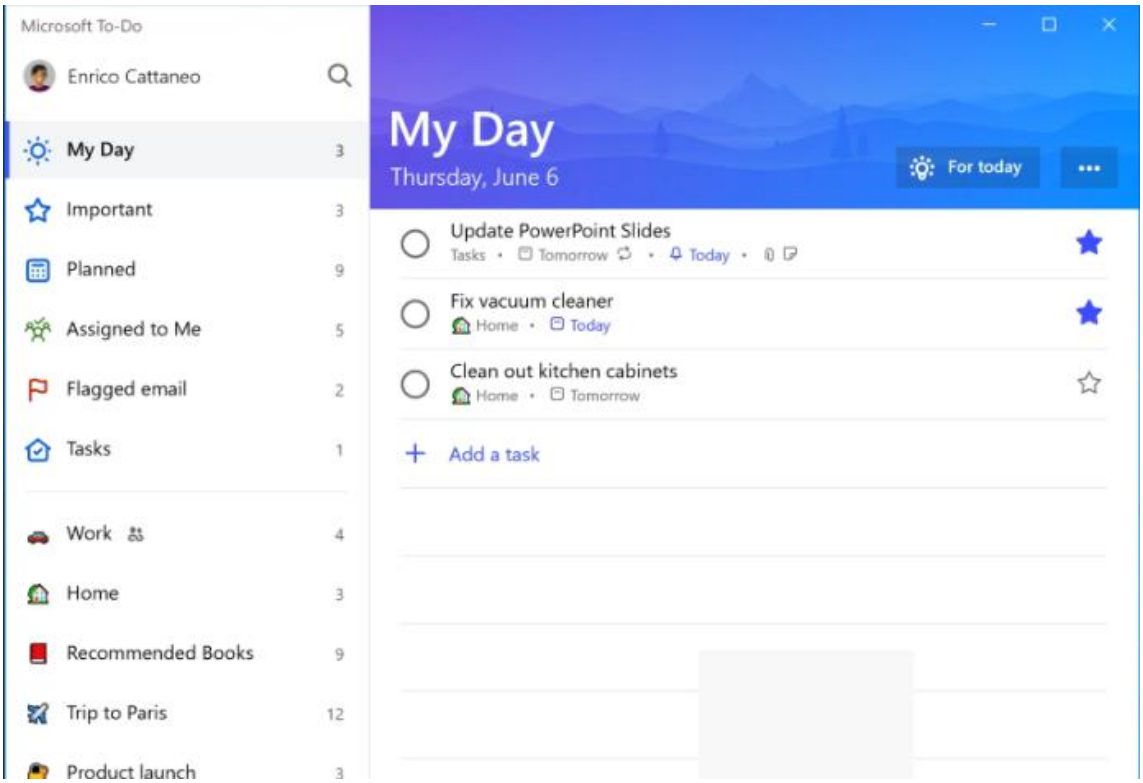


Рисунок 1.4 Інтерфейс додатку для управління завданнями «Microsoft To Do»

Microsoft To Do – це простий інструмент для створення списків завдань і встановлення дедлайнів них, основні функції функції додатку:

- можна створювати списки завдань та робити їх розбиття на окремі підзавдання;
- можливість встановлення дедлайну на виконання завдань;
- є зручна функція імпорту завдань з Outlook для їх подальшої організації та виконання;
- присутня інтеграція з усіма сервісами Microsoft 365;
- добре підходить для створення і контролю особистих завдань;
- можливість встановлення позначок до завдань, що забезпечує їх легку фільтрацію.

Недоліки Microsoft To Do:

- відсутність таймерів для відслідковування затраченого часу на виконання завдання;
- в додатку відсутня можливість створення та розподілу завдань між учасниками команди;

- у порівнянні з аналогами є менший функціонал.

Провівши аналіз чотирьох подібних між собою веб-додатків для управління завданнями, які мають схожі функціональні можливості, було прийнято рішення розробити окремий додаток, що дозволив би користувачам безкоштовно використовувати самий основний та затребуваний функціонал в сфері управління завданнями. Розроблюваний додаток забезпечить потреби більшості користувачів та компаній, він є легким в адмініструванні, безпечним та доступний для використання середнім та малим бізнесом, а також різними групами користувачів.

Наголос робиться на тому, що розробка спрямована на забезпечення високої ефективності та задоволення потреб користувачів. Додаток створює сприятливі умови для продуктивної роботи та співпраці між користувачами та групами, а доступність його основного функціоналу робить його доступним для широкого кола використання, у порівняння від існуючих аналогів де за доступ до подібного функціоналу потрібно сплачувати кошти.

1.4 Обґрунтування вибору моделей розробки для створення веб-додатку управління завданнями

Сучасні технології та стратегії процесу розробки програмного забезпечення визначають актуальність вибору моделі відповідно до конкретних вимог та особливостей проекту, що забезпечує його успішність та раціональну реалізацію, як можна ефективніше. Існує багато різних моделей, кожна з яких зробила свій внесок для розуміння та організації процесу створення додатків, незалежно від вибору моделі та підходу до розробки, мета залишається однією – успішна та ефективна реалізація ПЗ. Проте важливо вибрати саме той підхід, який найкраще відповідатиме конкретним вимогам та особливостям проекту.

Є кілька популярних моделей, які використовуються для ефективної розробки програмного забезпечення:

- каскадна модель: за принципом роботи проект розділяється і структурується на послідовні етапи, кожен етап розробки починається після завершення попереднього[15]. Можливість повернення до попереднього етапу відсутня. Основною відмінністю є жорстка послідовність виконання етапів, що робить її використання ефективним для проектів де є чітко встановлені вимоги. Проте на пізніших етапах важче вносити зміни до проекту;
- модель прототипування: замість встановлення всіх вимог на початкових етапах, спочатку розробляється прототип, що дає змогу з'ясувати протреби користувача. Відмінністю цієї моделі є більша взаємодія з користувачем для отримання фідбеку, але вимагає більшої затрати часу на розробку. Добре підходить до проектів де важко одразу визначити головні вимоги на початку розробки;
- ітераційна модель: метод, що передбачає розбиття процесу розробки на невеликі, послідовні цикли, що називаються ітераціями[16]. На кожній ітерації спочатку розробляється функціональність продукту, а потім її тестування. Метод дозволяє гнучко адаптуватись до змін, що можуть виникнути під час розробки та забезпечує отримання працюючої версії програмного продукту на ранніх етапах;
- спіральна модель розробки: цей метод поєднує в собі елементи каскадної та ітераційної моделей та відбувається у вигляді циклів або послідовних обертів спіралі. Кожен оберт спіралі представляє собою новий цикл розробки програмного забезпечення і включає в себе планування, розробку і тестування визначених функцій програмного продукту. Під час кожного циклу проводиться оцінка ризиків та вносяться зміни до проекту, коли з’являється така необхідність. Завдяки цьому, даний метод дозволяє гнучко адаптуватись до змін та знизити ризики;
- модель екстремального програмування: даний метод розробки програмного забезпечення зосереджується на простоті, комунікації між учасниками команди та здатності швидко адаптуватися до змін. Його

характерними особливостями є парне програмування, що забезпечує високу якість коду та ефективну комунікацію, ітераційна розробка де покроково реалізується певна функціональність продукту, постійне тестування та відстеження змін. Дана модель добре підходить до проектів де немає чітко визначених вимог, що дозволяє команді швидко адаптуватися до змін;

- модель спрямованого розвитку: метод розробки програмного забезпечення, що фокусується на реалізації конкретних функцій програмного продукту. Проект розбивається на набори окремих функціональностей, для кожної з яких виділяється окрема група розробників. В даній моделі використовується ітераційна розробка, але перед реалізацією потрібно створити тверду основу проекту, що включає в себе готові архітектурні рішення та дизайн. Дана модель добре підходить для великих проектів над якими працює валика кількість розробників, оскільки чітко розподіляє відповідальність між командами, що забезпечує ефективне управління проектом.

Методології розробки є одним із типів моделей розробки. Вибір підходу до розробки веб-додатку для управління завданнями є ключовим рішенням, яке визначатиме успішність всього процесу. Вірно обрана методологія сприяє ефективній комунікації та співпраці в команді, а також забезпечує контроль якості та дотримання термінів.

На ринку існує безліч методологій розробки веб-додатків, кожна з яких має свої переваги та недоліки. При виборі слід враховувати різноманітні фактори, такі як:

- розмір та складність проекту;
- очікувані навантаження на додаток;
- бюджет на розробку та підтримку додатку;
- професійні навички команди розробників.

Для веб-додатків управління завданнями часто використовуються наступні методології:

- гнучка методологія (Agile);
- водоспадна (Waterfall);
- гібридна (Hybrid).

Agile методологія ґрунтується на ітеративному та поетапному підході до розробки, де проект розбивається на компактні та адаптивні блоки, відомі як спринти[14]. Кожен спринт визначає конкретну мету, тривалість і результат, і піддається огляду та тестуванню як командою, так і клієнтом. Гнучка методологія надає можливість адаптуватися до змінних вимог, отримувати зворотний зв'язок і змінювати пріоритети, а також постачати цінність швидше і частіше. Проте для успішного впровадження гнучкої методології необхідна від команди велика співпраця, високий рівень комунікації і самоорганізації, а також чітке розуміння та підтримка від клієнта. Для управління гнучким проектом використовуються інструменти та методи, такі як Scrum, Kanban, історії користувачів та беклог.

Waterfall методологія ґрунтується на лінійному та послідовному розвитку проекту, де весь процес розділено на чітко визначені та жорсткі фази[17]. Кожна фаза має стандартизований порядок, вхідні та вихідні дані, а також критерії завершення перед переходом до наступної фази. Методологія водоспаду дозволяє створити детальний план, бюджет і графік для проекту, забезпечуючи високу якість та послідовність. Однак цей підхід вимагає більше попереднього аналізу, документування та затвердження від команди та клієнта, і має меншу гнучкість і можливість реагування на зміни, відгуки та ризики. Для управління проектом водоспаду використовують інструменти, такі як діаграми Ганта та структури розподілу робіт.

Hybrid методологія використовує комбінацію елементів гнучкої та каскадної методології, орієнтуючись на особливості та потреби конкретного проекту[18]. Наприклад, можна використовувати каскадний підхід на початкових етапах планування і проектування, переходячи до гнучкої методології для подальших етапів розробки та тестування. Іншим варіантом є використання гнучкої методології для основних функцій проекту, а каскадної – для периферійних чи нормативних аспектів. Гібридна методологія дозволяє використовувати переваги

обох підходів і врегульовувати їх недолі, а також адаптувати проект до конкретного контексту та обмежень. Проте успішна реалізація гібридної методології вимагає від команди та замовника більшої інтеграції, координації та узгодженості, і може призвести до плутанини та неузгодженості, якщо не визначити її чітко та не забезпечити ясної комунікації. Управління гібридним проектом вимагає використання інструментів та методів, таких як гібридні фреймворки, гібридні команди та гібридні артефакти.

Загальний вибір методології повинен враховувати конкретні умови та потреби проекту, а також враховувати змінні фактори, щоб забезпечити ефективність та успішність розробки веб-додатка для управління завданнями.

Оскільки для розробки системи управління завданнями плануються чітко встановлені вимоги, було обрано Waterfall-методологію та каскадну модель, так як передбачається наявність визначених основних вимог до системи на початку її проектування та дозволяє легко і послідовно реалізувати їх покроково виконуючи кожен процес. (див. рис. 1.4)

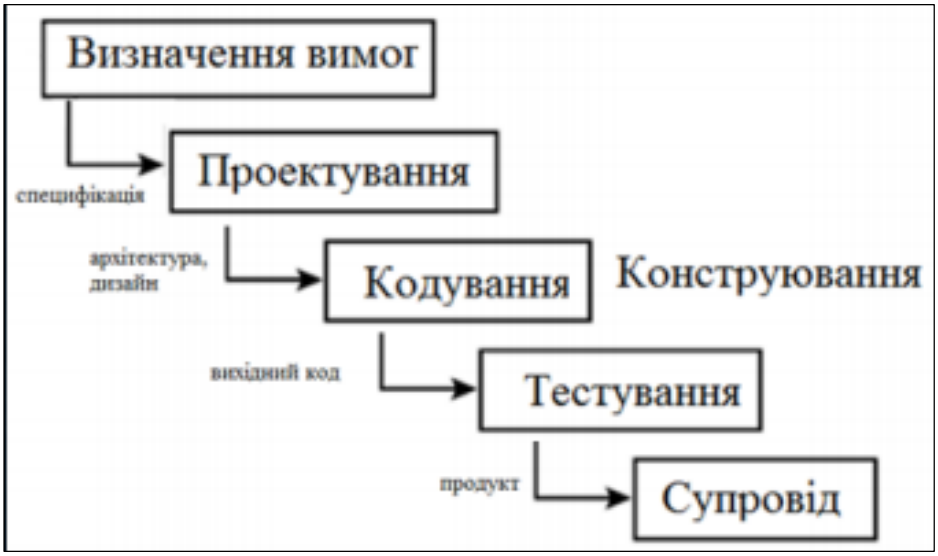


Рисунок 1.5 Каскадна модель

Процес розробки веб-додатку управління завданнями передбачає кілька основних стадій:

– аналіз вимог: документується здійснюється детальний аналіз вимог до ПЗ;

- проектування: даний етап передбачає створення дизайну та розробку архітектурних рішень для майбутнього додатку;
- кодування: на цьому етапі розпочинається реалізація раніше встановлених функціональних та нефункціональних вимог до системи з врахуванням архітектурних та дизайнерських рішень;
- впровадження та супровід: є кінцевим етапом розробки програмного продукту, надається доступ до системи кінцевим користувачам та забезпечується підтримка.

Вибрана модель для розробки системи управління завданнями забезпечує виконання етапів у строгому порядку, що дозволяє визначити вимоги на початку проектування та поетапно реалізовувати їх. Цей підхід передбачає зручність та мінімізацію ризиків, а також простоту та зрозумілість розробки. Здійснюється супровід на всіх етапах реалізації додатку, що мінімізує ризик виникнення помилок, послідовність реалізації дає змогу точно планувати терміни виконання та завершення робіт.

1.5 Опис і основні принципи роботи протоколів HTTP та HTTPS

HTTP (Hypertext Transfer Protocol) – це простий протокол, що використовується для забезпечення зв’язку між клієнтом та веб-сервером або локальною мережею, завдяки цьому з’являється можливість передачі та отримання даних між сервером та браузером[6]. Цей протокол використовує набір стандартних правил, які забезпечують ефективну взаємодію між клієнтами, серверами і проксі-серверами.

За принципом роботи HTTP має схожість з іншим протоколом, таким як SMTP (Simple Mail Transfer Protocol), вони обидва забезпечують передачу даних між клієнтом та сервером. Проте, дані протоколи відрізняються обміном повідомленнями між клієнтом та сервером і миттєвістю їх доставки. SMTP

спочатку зберігає, а потім пересилає повідомлення, тоді як HTTP надсилає дані від сервера до клієнта негайно, що забезпечує миттєве отримання даних.

HTTP протокол працює за принципом запит-відповідь, що дозволяє користувачам взаємодіяти з веб-ресурсами. Спочатку, клієнтом відправляється запит на сервер, який його обробляє і тільки тоді повертає відповідь. Для цього зазвичай використовується TCP (Transmission Control Protocol), який відповідає за зв’язок із сервером.

Для взаємодії між клієнтом та сервером HTTP протокол використовує спеціальні методи:

- GET: цей метод використовується для отримання клієнтом конкретних ресурсів із сервера;
- POST: використовується для передачі даних з клієнта на веб-сервер, зазвичай цей метод застосовується у формах для відправки;
- PUT: відповідає за оновлення та додавання нових даних до сервера, клієнт відправляє дані і сервер повинен оновлювати або створювати новий ресурс у разі його відсутності;
- DELETE: призначений для видалення даних;
- PATCH: даний метод застосовується для часткового оновлення даних не ресурсі, клієнт повинен передавати лише ті дані, що потребують оновлення;
- HEAD: метод повертає відповіді з сервера, але лише заголовки без основного вмісту ресурсу;
- OPTIONS: дозволяє клієнту отримувати дані про методи або параметри запиту для конкретного ресурсу, які підтримуються сервером;
- TRACE: використовується для тестування маршрутизації запитів до сервера і допомагає в діагностиці проблем, що можуть виникати з маршрутами.

Ці методи використовуються разом з HTTP-заголовками та тілами запиту для взаємодіє з сервером та його ресурсами. В Django зазвичай використовуються два найпоширеніших методи – це «GET» і «POST». Вони застосовуються для взаємодії

з формами на сторінці користувача, які слугують відповідно для виведення або отримання інформації. HTTP добре підходить для локального використання або під час розробки проекту і перегляду працездатності функцій, проте для виробничого середовища наполегливо рекомендується використовувати протокол HTTPS, так як він є більш надійним та захищеним особливо для безпечної передачі конфіденційної інформації.

HTTPS (Hypertext Transfer Protocol Secure) – це розширена та більш безпечніша версія протоколу HTTP, що широко використовується, завдяки забезпеченню шифрування та безпечної передачі даних через мережу[7]. Дані, що передаються проходять шифрування, яке забезпечує конфіденційність на цілісність даних, які запитує клієнт.

HTTPS запобігає можливості перехоплення конфіденційної інформація якою обмінюються клієнт та сервер. Якщо обмін інформацією здійснюється за допомогою звичайного HTTP-з’єднання, вона розбивається на окремі незахищені пакети даних, які можна легко перехоплювати використовуючи безкоштовне програмне забезпечення. Особливо легко такі дані перехоплюються на шляху від клієнта до сервера в незахищеному середовищі, наприклад відкритій Wi-Fi мережі, відсутність шифрування зв’язку робить дані легкодоступними для злочинців.

Щоб забезпечити шифрування даних HTTPS використовує протокол TLS(Transport Layer Security) раніше відомий як SSL (Secure Sockets Layer). Захист з’єднання забезпечується завдяки інфраструктурі асиметричного відкритого ключа.

Такий тип системи захисту використовує два різних ключі для шифрування між клієнтом та сервером:

- приватний ключ: використовується власником веб-системи та зберігається в закритому вигляді, Django генерує такий ключ самостійно при створенні проекту. Цей ключ знаходить на сервері і застосовується в автоматичному режимі для розшифрування інформації, що була зашифрована відкритим ключем;

- відкритий ключ: доступний кожному користувачу, який взаємодіє з системою, при передачі даних він зашифровує інформацію і вона розшифровується приватним ключем на сервері.

Коли користувач потрапляє до веб-сторінки, йому надходить SSL-сертифікат, що містить відкритий ключ для забезпечення безпечного з'єднання. Далі між клієнтом та сервером відбувається процес TLS/SSL рукостискання, коли здійснюється серія зворотніх зв’язків на відповідність ключів та встановлення довіри, після чого відбувається безпечне з’єднання, яке шифрує потенційні дані надаючи відповідний захист для сесії.

Таким чином використання ключів для шифрування захищає конфіденційну інформацію, що пересилається між клієнтом та сервером, забезпечуючи відповідну безпеку обміну даними.

В цілому, використання протоколів безпеки та методів захисту є досить затребуваним в наш час, це вбереже компанію та її репутацію від небажаного проникнення до системи злочинців чи втрати конфіденційної інформації користувачів, які взаємодіють з програмним продуктом.

1.6 AJAX-запити та особливості їх використання

Ajax– це веб-технологія, що дозволяє реалізувати взаємодію веб-сторінки з сервером і забезпечує динамічне оновлення вмісту без ручного перезавантаження і затримок[8]. AJAX використовує об’єкт XMLHttpRequest для обміну даними між клієнтом та сервером асинхронно, завдяки чому надається покращена продуктивність веб-системи та кращий користувацький досвід. Після створення об’єкта інформація додається у створений XML-файл, який XMLHttpRequest передає на сервер. Веб-сервер здійснює обробку отриманого запиту і надає відповідь з необхідними даними. Це забезпечує оновлення вмісту сторінки без її повного перезавантаження.

У стандартному сценарії, коли веб-додаток працює без використання AJAX, обмін даними здійснюється шляхом відправки браузером HTTP-запиту на сервер, коли користувач здійснює певну дію на веб-сторінці. Спочатку сервер обробляє вхідний запит та перевіряє отримані дані на відповідність, після чого здійснює відправку відповіді клієнту та оновлює вміст сторінки новими даними. В такому випадку викликається повне перезавантаження сторінки, навіть, якщо зміни були мінімальними та не суттєвими. Також часті запити від клієнта можуть призводити до завантаження додаткового програмного забезпечення на сервері та перевірки безпечності даних, що не є ефективним та створює для нього додаткове навантаження, яке з часом може сказатись на його працездатності та довговічності.

Зазвичай AJAX-запити використовуються для забезпечення наступного функціоналу:

- динамічне оновлення даних на веб-сторінці, що дозволяє уникнути ручного оновлення сторінки та підвищити продуктивність веб-додатку у випадках, коли вимагається часта синхронізація з сервером для оновлення окремих даних, а не всієї сторінки;
- взаємодія з користувачем для забезпечення інтерактиву при роботі з елементами системи, що полегшує використання функціоналу веб-сторінки, може забезпечувати більшу зрозумілість роботи деяких функцій;
- автоматичне оновлення всієї веб-сторінки за потреби, наприклад для погодних додатків чи відображення сповіщень про нові повідомлення.

Особливості використання AJAX:

- забезпечення асинхронності: запити працюють у фоновому режимі, що дозволяє веб-сторінці опрацьовувати інші операцію, які виконує користувач, на чекаючи завершення запиту;
- дана технологія підтримує передачу та отримання різних типів даних, таких як JSON;

- запити працюють завдяки використанню JavaScript, що дозволяє створювати складні веб системи з підтримкою інтерактиву для користувачів.

Навіть при схожих процесах обміну даними та потоці інформації, виявляється, що використання AJAX є більш ефективним у порівнянні зі звичайними веб-запитами[9]. Під час використання AJAX браузер оновлює лише конкретний вміст веб-сторінки, враховуючи лише запитані дані і уникає зайвих оновлень інших елементів на сторінці. Це призводить до прискорення та ефективності роботи додатків AJAX, які краще адаптуються до змін, ніж звичайні веб-додатки.

Для забезпечення асинхронного обміну даними з веб-сервером AJAX використовує певні веб-технології та програмні інструменти.

- XHTML (Extensible Hypertext Markup Language): HTML і CSS виступають у ролі мов розмітки, що дозволяють визначити структуру та оформлення вмісту веб-сторінки. Наприклад, за допомогою XHTML чи HTML можна впорядковувати текст та вставляти зображення, а CSS використовується для визначення шрифтів та кольорів фону;
- XML: є мовою програмування, яка дозволяє різним застосункам обмінюватися даними. Оскільки різні застосунки можуть представляти дані по-різному, використання XML дозволяє подавати дані у вигляді тексту, що може бути зрозумілим для інших застосунків чи систем, які їх обробляють. Формат XML дозволяє структурувати дані у деревоподібній ієрархії тегів, що забезпечує відображення їхньої логічної структуру. Додатки AJAX використовують XML для обміну та обробки даних в єдиному форматі;
- XMLHttpRequest: інтерфейс програмування додатків (API), який дає змогу веб-браузерам взаємодіяти з веб-серверами асинхронно. Об'єкт XMLHttpRequest дозволяє відправляти часткову інформацію про веб-сторінку на сервер у форматі XML. Це сприяє асинхронному обміну даними між веб-сторінкою та сервером без повного її перезавантаження;

- об'єктна модель документа (DOM): структурує сторінки HTML і XML у вигляді ієрархічної структури. DOM складається з вузлів, які розгалужуються на більшу кількість дочірніх вузлів або об'єктів. Це дозволяє ефективно застосовувати стилі або вносити зміни в коди конкретних сторінок;
- JavaScript – це скриптова мова, яку можна використовувати для динамічного відображення контенту на веб-сторінках. Динамічний контент являє собою інформацію на веб-сторінці, що оновлюється в реальному часі або залежить від взаємодії з користувачем, може бути інтерактивними ефектами елементів. Наприклад, у використанні технології AJAX, JavaScript взаємодіє з іншими веб-технологіями, для асинхронного оновлення вмісту сторінок.

Фреймворк Django для розробки веб-додатків на мові програмування Python, надає зручні інструменти для роботи з AJAX, що дозволяє забезпечити ефективну взаємодію між клієнтом та сервером та полегшити роботу розробникам завдяки готовій структурі відповідних файлів та коду[11].

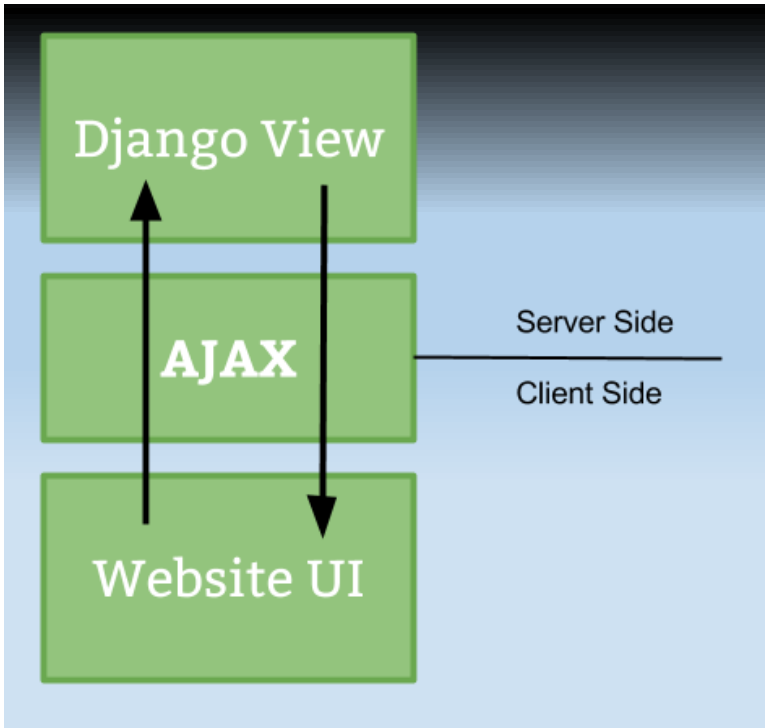


Рисунок 1.6 Взаємодія Django з клієнтом використовуючи AJAX

У веб-додатках, побудованих на Django, взаємодія з клієнтом через AJAX може включати в себе визначення відповідних URL-шляхів, створення відображень, які обробляють AJAX-запити, відправлення та отримання даних у форматі, який обирається, зазвичай використовують JSON.

На стороні клієнта, JavaScript використовується для виклику AJAX-запитів для обробки вхідних даних та оновлення DOM сторінки без її повного оновлення. Більшість браузерів мають вбудовану підтримку AJAX, але можуть використовуватися і бібліотеки, такі як jQuery або Axios, для спрощення взаємодії.

Використання AJAX в Django включає в себе:

- створення AJAX-видів (Views): Django дозволяє створювати власні види, які оброблять AJAX-запити та повертають відповідні дані у форматі JSON за допомогою класу JSONResponse або функції JsonResponse. Це дозволяє передавати добре структуровані дані між сервером та клієнтом;
- AJAX можна використовувати для асинхронного завантаження окремих частин веб-сторінок, що дозволяє покращити роботу користувацького інтерфейсу веб-сайту шляхом його інтеграції у Django-шаблони;
- використання Django REST Framework: Для створення API, яке може використовуватися для AJAX-запитів, можна використовувати Django REST Framework.

Загальна інтеграція AJAX з Django дозволяє покращити ефективність та зручність взаємодії клієнта та сервера у веб-додатку, а також оптимізувати його роботу знизивши ресурсозатратність на певних етапах взаємодії та обміну даними з клієнтською стороною.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Специфікація вимог до створення веб-додатку

У цьому підрозділі детально розглядається специфікація вимог до створення веб-додатку для управління завданнями. Визначаються ключові функціональні та нефункціональні, а також бізнес вимоги, які повинні бути враховані при розробці програмного продукту.

Для початку реалізації веб-додатку управління завданнями відповідно до основних етапів каскадної моделі було визначено та сформовано вимоги, що представлені на рис.2.1.

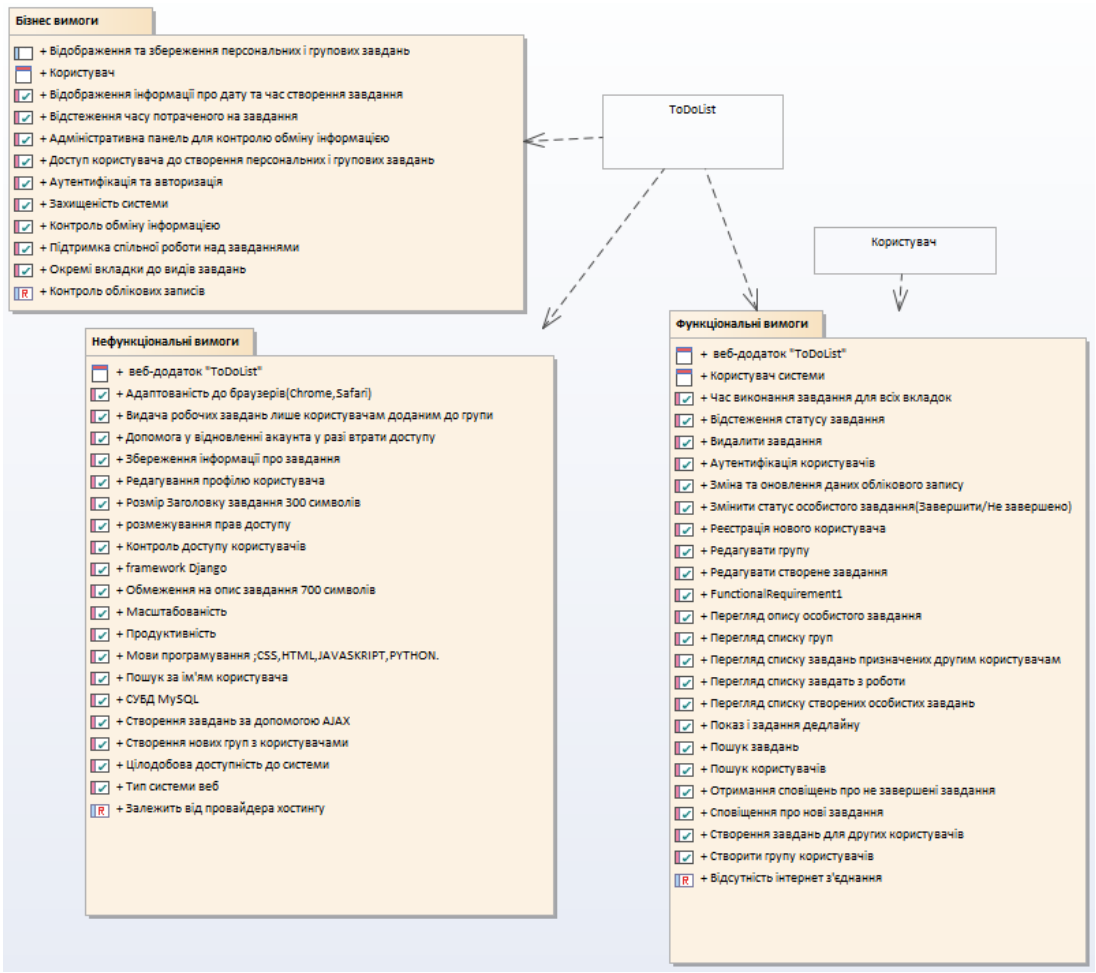


Рисунок 2.1 Загальна діаграма вимог до веб-додатку для управління завданнями

Першочерговим завданням є чітке визначення основних функціональних вимог до веб-додатку. Програмний продукт має надавати можливість створення персональних або командних завдань шляхом їх призначення учасникам команди, надавати можливість їх редагування та видалення. Також потрібно включати функціональність, яка забезпечує можливість відстеження статусу персональних та призначених другим користувачам завдань, моніторинг дедлайну, дати створення, інформації про користувача, який створив завдання та для кого призначив. Для зручності передбачено можливість створення груп користувачів та розподіл завдань між ними. Додатковою та не менш важливою вимогою є додавання таймера до кожного завдання для ефективного тайм-менеджменту.

Невід'ємною частиною специфікації є визначення нефункціональних вимог, що включають в себе вимоги до продуктивності, розмежування доступу та масштабованості, яка дозволяє забезпечувати підтримку зростання обсягу даних запобігаючи програмним збоям і перевантаженню мережі. Створення завдань за допомогою технологій AJAX у взаємодії з фреймворком Django, що дозволить оптимізувати роботу веб-додатку для більш швидкої та якісної роботи.

Бізнес-вимоги включають в себе забезпечення безпеки обміну інформацією, наявність адміністративної панелі для здійснення контролю за завданнями та виконанням звичних адміністративних обов'язків з обслуговування веб-додатку. Для розробки потрібно використовувати Python Django так як це забезпечить легке створення системи управління завданнями та чітко структурований код, який в подальшому буде зручно підтримувати і покращувати впроваджуючи нові функції, що дозволить проекту розвиватись далі та ставати більш популярним з часом.

Окремо розглядаються вимоги до інтерфейсу, що включають в себе створення та реалізацію дизайну користувацького інтерфейсу та його ергономіку. В розробці повинні враховуватись інтуїтивно зрозумілий інфтерфейс, адаптивність до різних пристроїв та браузерів.

Система, що розробляється, передбачає дві основні ролі для осіб, що взаємодіють з продуктом:

- користувач: основна дійова особа, що використовує програмний продукт для створення, редагування та відстеження особистих і групових завдань. Користувач має доступ до основного функціоналу системи, необхідного для ефективного управління завданнями;
- адміністратор: дійова особа, що має повний доступ та контроль над усіма аспектами системи. Адміністратор відповідає за управління користувачами, безпекою, конфігурацією та іншими ключовими аспектами.

Веб-додаток може стикатись з рядом потенційних ризиків, які можуть впливати на працездатність:

- виникнення проблем з доступом до мережевих ресурсів може обмежити функціональність додатку;
- програмні збої можуть призвести до некоректної роботи системи, порушити її продуктивність та надійність;
- високий рівень конкуренції може вплинути на успішність проекту, оскільки він маловідомий на ранніх етапах;
- порушення встановлених правил та норм безпеки може призвести до порушення конфіденційності та цілісності даних, що може значно нашкодити подальшому розвитку веб-додатку;
- недостатнє фінансування може призвести до обмеження ресурсів, впливати на розвиток та підтримку веб-системи, що може призвести до зниження темпів розвитку та покращення.

Визначені потенційні ризики можуть викликати труднощі в роботі веб-додатку для управління завданнями, але їх аналіз, готовність та вживання відповідних заходів дозволяють знизити рівень загрози та негативного впливу на програмний продукт. Дотримання всіх правил та політик безпеки даних, а також вчасне обслуговування та постійне покращення додатку у поєднанні з його якісною оптимізацією дозволять уникнути більшості можливих ризиків.

Для більш точного та наглядного визначення вимог до веб-додатку для управління завданнями, була створена UML діаграма прецедентів, що

представлена на рис.2.2. Такий підхід дозволяє відобразити взаємодію між акторами та прецедентами, що дає змогу вставити різні сценарії використання програмного продукту.

UML (Unified Modeling Language) – це мова, яка призначена для виконання побудови моделей систем, застосовується для створення візуальних представлень, розробки та можливості ведення документації програмних продуктів відповідно до потреб. В рамках UML визначено уніфікований набір графічних символів і нотацій, які дозволяють стандартизовано описувати різні аспекти моделі програмного забезпечення.

Діаграма прецедентів є одним з ключових компонентів визначення функціональності та взаємодії системи з користувачами. Це важливий інструмент, який використовується в області розробки програмного забезпечення для моделювання різних варіантів взаємодії між користувачами та системою. Основними компонентами даної діаграми є прецеденти, актори, відношення між ними та сама система.

Прецеденти є важливими елементами uml-діаграми, що дозволяють більш точно визначити функціональні вимоги до системи шляхом детального опису взаємодії між користувачами та самою системою, завдяки зручному графічному представленню. Кожен прецедент має описувати конкретний сценарій використання, що надає уявлення про те, як система повинна себе поводити при відповідній взаємодії користувача з її функціональними елементами.

Актори, у свою чергу, є дійовими особами, що взаємодіють із системою. Кожен актор має визначену роль та рівень доступу до різних функціональних можливостей системи. Їхні дії та взаємодія з системою моделюються через визначені прецеденти, що дозволяє встановити майже всі варіанти використання системи та її поведінку на кожному етапі.

Взаємодія між акторами та прецедентами відображається за допомогою стрілок. Стрілки вказують напрямок взаємодії та показують, хто ініціює конкретний прецедент.

Такий підхід дозволяє чітко визначити очікувані результати взаємодії, спростити розуміння функціональності системи та враховувати різноманітні сценарії використання, що сприяє ефективному проектуванню та розробці програмного продукту, який в результаті вийде якісним та оптимізованим, що забезпечить добру роботу веб-додатку та забезпечить потужну відправну точку у його подальшому розвитку та покращенню адже вже буде напрацьована відповідна база.

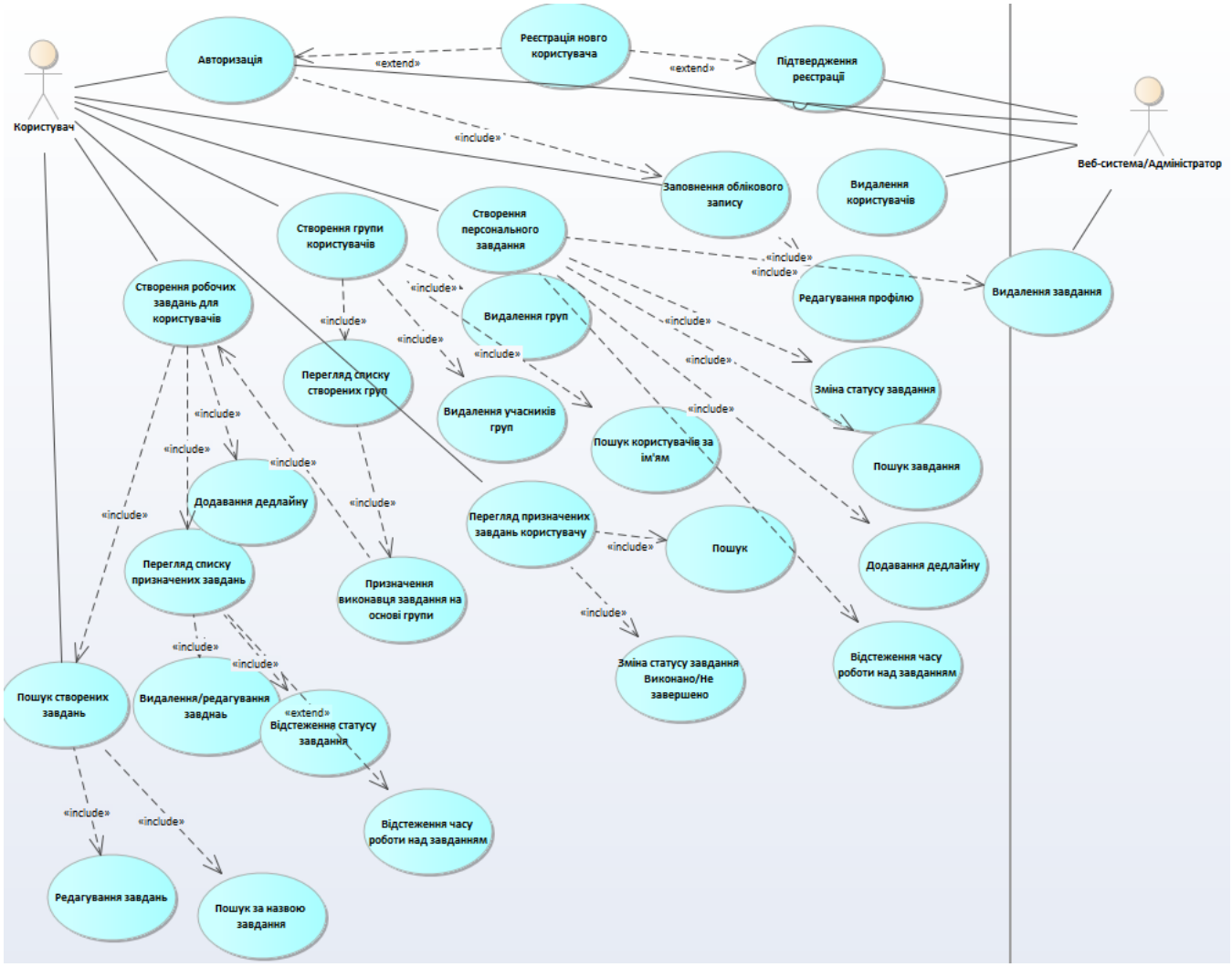


Рисунок 2.2 UML діаграма прецедентів

Ця діаграма відображає зв’язки між акторами, що використовують веб-систему для управління завданнями. Кожна дійова особа має свій рівень доступу до певного функціоналу веб-додатку та взаємодії з ним.

Щоб розпочати роботу в додатку для управління завданнями користувачу потрібно пройти авторизацію після успішності якої буде здійснено перенаправлення на домашню сторінку з персональними завданнями. У випадку невдалої спроби авторизуватись в системі або відсутності створеного облікового запису потрібно пройти процес реєстрації.

Після успішної автентифікації користувача в системі, він отримує доступ до функціоналу визначеного відповідно до вимог і може використовувати веб-додаток відповідно до своїх потреб.

Сучасний світ інформаційних систем та технологій на стільки стрімко змінюється і розвивається, що різні додатки та системи доволі швидко можуть застарівати. Це може призводити до втрати програмними продуктами актуальності для споживачів, особливо, якщо є велика кількість конкурентів, які продовжують крокувати в ногу з часом, а також швидко та гнучко пристосовуватись до нових технологій і впроваджувати їх у свої аналоги. Жоден продукт не є кінцевим і остаточним варіантом, усі додатки з моменту релізу допрацьовуються, щось може змінюватись в їх функціональних особливостях відповідно до технологічного та інноваційного ринку розробок. Тому, важливо відзначити, що в міру подальшого розвитку проекту, покращення та модифікації його функціональних можливостей, а також зростання числа користувачів, кількість дійових осіб та варіантів використання представлених на даній діаграмі (див.рис.2.2) може суттєво збільшуватись та відповідно розширюватись, тому це не є кінцевим варіантом. Саме по цій причині враховується можливість подальшого впровадження нових ідей для програмного продукту та внесення змін до існуючої архітектури.

2.2 Аналіз та вибір архітектурних рішень

В контексті аналізу, встановлення вимог та розробки веб-додатка для управління завданнями з’являється необхідність вирішення архітектурних питань,

що є ключовим етапом для забезпечення ефективності, масштабованості та легкості розширення програмного продукту. Для цього потрібно розглянути різні архітектурні рішення, що мають свої переваги та недоліки, після чого обрати конкретно ті, що найкраще підійдуть для виконання поставленого завдання.

Архітектура веб-додатків – це концептуальне планування та організація всіх складових веб-додатка з метою забезпечення ефективності, простоти в розгортанні, масштабованості та відповідності вимогам безпеки та продуктивності[19]. Вона визначає, як компоненти взаємодіють між собою, як обробляються дані та як взаємодіє з користувачем.

В сучасному світі вибір оптимальної архітектури для веб-додатка стає стратегічним завданням через велике розмаїття доступних варіантів.

У цьому контексті, архітектура веб-додатків стає не лише технічним плануванням, але і стратегічним вибором, що враховує як технічні, так і бізнес-аспекти розвитку продукту. Вона визначає спосіб організації та взаємодії всіх елементів системи для забезпечення її оптимального функціонування, легкості обслуговування та задоволення потреб користувачів.

Монолітна архітектура – це найпростіший і найпоширеніший тип архітектури для розробки веб-додатків[20]. Вона представляє собою простий та широко використовуваний підхід до створення додатків. Усі компоненти програмного продукту включно з фронтендом, бекендом та базою даних інтегровані в єдиний блок. Зазначена модель здійснює обробку запитів та передбачає збереження даних в одному місці. Дана архітектура та принцип її роботи представлені на рисунку 2.3.

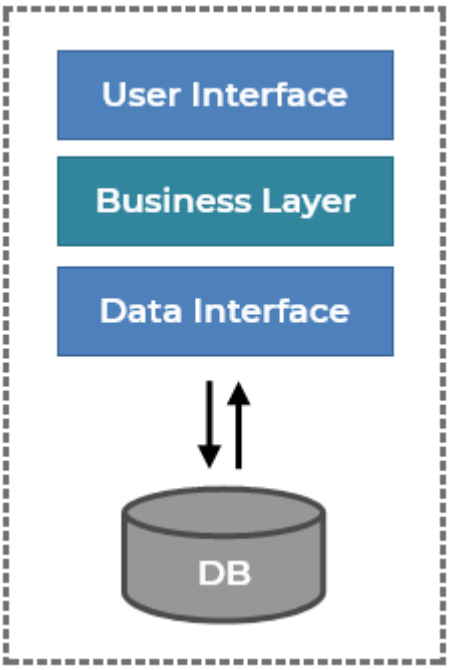


Рисунок 2.3 Принцип роботи монолітної архітектури

В даній архітектурі взаємодія між сервером, базою даних та клієнтською частиною відбувається в межах однієї системи. Запити від користувачів передаються з клієнтської частини до серверної, після чого здійснюється їх обробка та повернення відповіді.

Монолітна архітектура поділяється на кілька основних компонентів:

- клієнтський додаток, що відповідає за відображення інтерфейсу користувача та взаємодію з ним;
- серверний додаток, який відповідає за логіку роботи додатка та обробку даних для взаємодії з клієнтською частиною;
- база даних, яка містить дані, що використовуються додатком та забезпечують його роботу.

Всі компоненти в даній архітектурі працюють як єдина система, що дозволяє забезпечити правильну роботу веб-додатку. Клієнтська частина передає відповідні запити до серверу, який здійснює їх обробку і передає відповідний результат згідно логіки, що написана в коді, після чого клієнту відображається відповідний результат.

До основних переваг даної архітектури можна віднести наступні:

- просте розгортання: можливість розгортати додаток у формі одного виконуваного файлу чи каталогу спрощує процес розгортання;
- зручна розробка: процес створення та модифікації стає більш спрощеним, коли додаток написаний одним кодом;
- продуктивність розробки: у випадку централізованої бази коду та сховища, часто можна використовувати один API для виконання тих самих функцій, що вимагають численні API з сервісами;
- спрощене тестування: завдяки тому, що монолітна система рахується єдиним централізованим блоком, проведення тестування на всій системі буде швидше у порівнянні з розподіленою програмою;
- налагодження: процес виконання запитів та виявлення проблем стає більш спрощеним, так як увесь код зосереджено в одному місці.

Основні недоліки архітектури:

- масштабування є доволі складним для великих та складних проектів;
- в міру розвитку проекту та збільшення його функціональних можливостей стає складно забезпечувати підтримку в подальшому, що може негативно відобразитись надалі;
- повільна швидкість розробки через складність процесів;
- низька надійність так як помилка в одному модулі може вплинути на працездатність всього додатку;

може бути менш ефективним, ніж інші архітектури.

Монолітна архітектура має ряд переваг, вона проста у розробці та підтримці, оскільки всі компоненти додатка взаємодіють один з одним за допомогою стандартних інтерфейсів. Крім того, монолітну архітектуру можна швидко розгорнути, оскільки всі компоненти додатка розгортаються разом, що дозволяє спростити початковий процес реалізації чи покращення системи.

Однак, дана архітектура має і ряд недоліків. Її важко масштабувати для великих та складних проектів. Наприклад, якщо додаток стає популярнішим і починає отримувати більше запитів, серверний додаток може не впоратися з навантаженням, що призводитиме до програмних збоїв та неналежної роботи

програмного продукту в цілому. Крім того, монолітну архітектуру може бути складно підтримувати, якщо додаток стає більшим і складнішим в процесі свого розвитку. Це пов'язано з тим, що зміни в одному компоненті системи можуть призвести до необхідності змін і в інших, що в кінцевому результаті може ускладнити процес покращення чи навіть заплутати розробників в певний момент.

Далі досліджувалась не менш популярна багаторівнева архітектура, що представлена на рисунку 2.4.

Архітектура шарів - це більш складний тип архітектури, ніж монолітна архітектура. Вона також відома як n-рівнева архітектура та описує архітектурний шаблон, що складається з кількох окремих горизонтальних рівнів, які функціонують разом як єдина одиниця програмного забезпечення.

Рівень – це логічне розділення компонентів або коду. компоненти, які пов’язані або подібні, зазвичай розміщуються на тих самих шарах. Однак кожен рівень відрізняється і вносить свій внесок у іншу частину загальної системи.

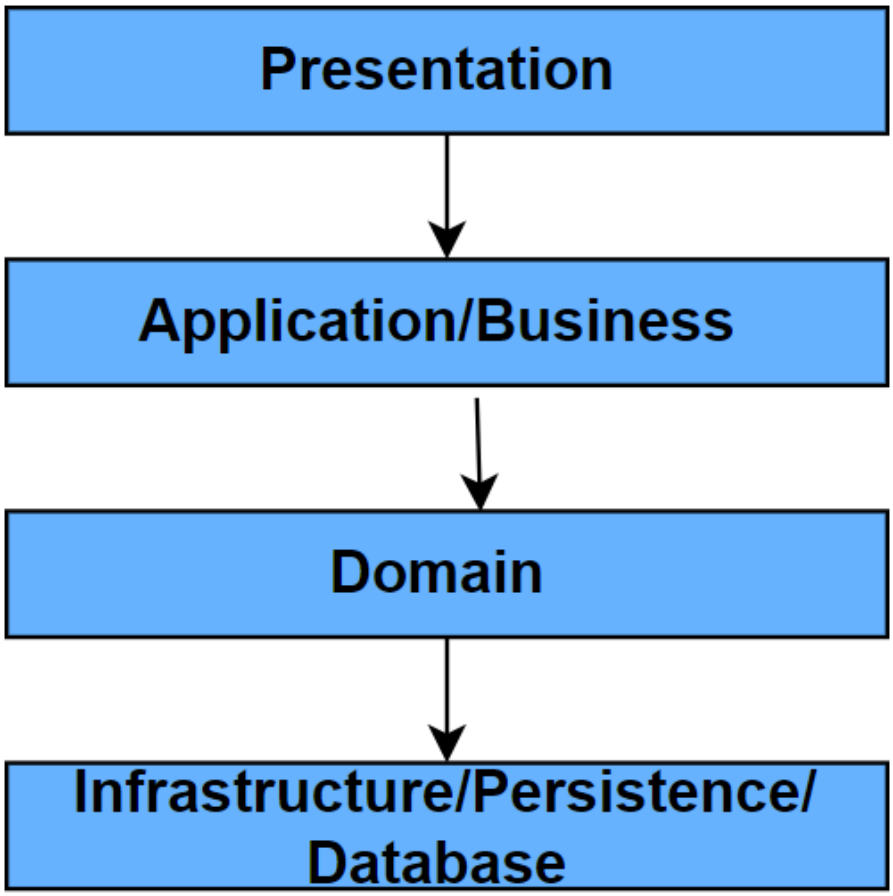


Рисунок 2.4 Принцип роботи багатошарової архітектури

Головною особливістю є те, що кожен шар взаємодіє лише з шарами, які безпосередньо розташовані під ним також важлива концепція ізоляції шарів, що дозволяє змінювати окремі шари без впливу на інші зміни котрі вносяться. Обмежуються конкретним шаром, який піддається змінам. Ця концепція розділення проблем є ще однією визначальною особливістю, що вказує на те, як модулі на одному рівні спільно виконують конкретну функцію.

Кількість рівнів у багатошаровій архітектурі не фіксується конкретним числом і зазвичай залежить від вибору розробника або архітектора програмного забезпечення важливо відзначити що в цій структурі зазвичай завжди присутній рівень взаємодії з користувачем та рівень обробки даних.

Є кілька основних шарів багаторівневої системи:

- презентаційний рівень, що відповідає за взаємодію користувача з програмним продуктом;
- прикладний/бізнес-рівень обробляє аспекти, пов’язані з виконанням функціональних вимог та відповідає за бізнес-логіку додатка в цілому;
- доменний рівень відповідає за алгоритми та компоненти програмування;
- рівень інфраструктури/постійності/бази даних відповідає за обробку даних.

У деяких програмах ці шари можуть поєднуватися, їх функції та обов’язки групуються для виконання на одному рівні. Також важливо враховувати, що концепція багаторівневої архітектури дозволяє створити чітке розділення обов'язків між різними рівнями. Кожен рівень виконує свою унікальну роль у системі, забезпечуючи оптимальну організацію та структуру коду.

Зокрема, принцип ізоляції шарів робить процес змін великої системи більш контрольованим. Якщо необхідно внести зміни в логіку бізнес-рівня, наприклад, це можна робити без впливу на презентаційний рівень або рівень доступу до даних. Це полегшує розробку, підтримку та масштабування системи в цілому.

Невизначеність кількості рівнів у багатошаровій архітектурі також підкреслює адаптивність цього підходу до різноманітних вимог проекту та стилів

розробки. Розробники можуть вільно вибирати кількість та природу рівнів відповідно до конкретних потреб і контексту свого проекту.

Такий підхід також надає можливість легше розподіляти завдання між командами розробників, які можуть концентруватися на певних рівнях без необхідності вдаватися в деталі інших частин системи. Це сприяє ефективності розробки та підтримці довгострокового проекту.

Переваги використання багаторівневої архітектури включають простоту вивчення та впровадження, зменшення залежностей, полегшення тестування та покращену безпеку.

Недоліки включають ускладнену масштабованість, важкість в підтримці через взаємозалежність шарів та можливу повільність порівняно з монолітною архітектурою. Також, хоча розділення на шарів сприяє ізоляції, великі зміни на одному рівні можуть вплинути на всю систему.

Наступна, також доволі ефективна архітектура для розробки певних додатків – це мікросервісна архітектура, що представлена на рисунку 2.5.

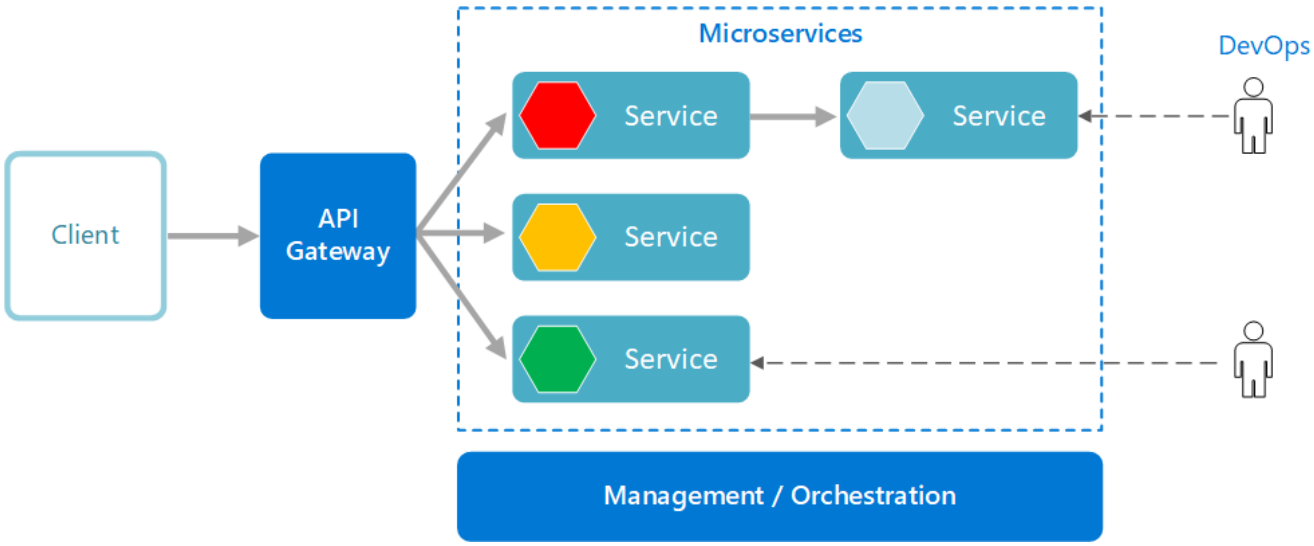


Рисунок 2.5 Принцип роботи мікросервісної архітектури

Мікросервісна архітектура – це інноваційний підхід до побудови програмних систем, де функціональні частини або служби працюють незалежно одна від одної. Кожна мікрослужба виконує конкретний аспект бізнес-логіки та має власну базу даних[21]. Головна мета полягає в розбитті системи на невеликі, самостійні

компоненти, які можна оновлювати, тестувати та масштабувати незалежно одне від одного.

Ключові аспекти мікросервісної архітектури визначають її унікальність та ефективність:

- автономія та ізоляція:
  - кожен мікросервіс є окремим модулем, що дозволяє їм розвиватися, оновлюватися та розгортатися ізольовано;
  - незалежність технологій дозволяє кожному мікросервісу використовувати власний стек технологій, що підвищує гнучкість системи.
- масштабованість та збереженість:
  - мікросервіси можуть бути масштабовані незалежно, оптимізуючи використання ресурсів;
  - відмова в одному мікросервісі не впливає на решту системи, забезпечуючи збереженість.
- інтеграція та комунікація:
  - спілкування мікросервісів здійснюється через чітко визначені API, спрощуючи інтеграцію та зменшуючи залежності;
  - мінімізація прямих залежностей дозволяє мікросервісам функціонувати незалежно один від одного.
- база даних та доступ до даних:
  - кожен мікросервіс може мати власну базу даних або використовувати різні технології зберігання даних;
  - доступ до даних здійснюється через чітко визначені API, забезпечуючи гнучкість та безпеку.
- моніторинг та управління:
  - централізована система моніторингу дозволяє відстежувати стан та продуктивність кожного мікросервісу;
  - централізована панель керування допомагає ефективно управляти та масштабувати окремі мікросервіси.

- інструменти розробки та доставки:
  - застосування CI/CD для кожного сервісу дозволяє швидке розгортання та впровадження змін в окремих мікросервісах;
  - ізоляція та розвиток кожного мікросервісу окремо забезпечують ефективний процес розробки та підтримки.

До недоліків мікросервісів можна віднести:

- розростання розробок: мікросервіси додають більше складності порівняно з монолітною архітектурою, оскільки є більше сервісів у багатьох місцях, створених декількома командами. Якщо розповсюдження розробки не управляється належним чином, це призводить до уповільнення швидкості розробки та низької операційної продуктивності;
- витрати на інфраструктуру: Кожна нова мікрослужба може мати власну вартість набір тестів, посібники з розгортання, інфраструктуру хостингу, інструменти моніторингу тощо;
- додаткові організаційні витрати: командам потрібно додати інший рівень спілкування та співпраці, щоб координувати оновлення та інтерфейси;
- проблеми з налагодженням: Кожен мікросервіс має власний набір журналів, що ускладнює налагодження. Крім того, один бізнес-процес може працювати на кількох машинах, що ще більше ускладнює налагодження;
- відсутність стандартизації: без спільної платформи можуть використовуватися різні мови програмування для написання різних мікросервісів, що в майбутньому впливатиме на їхню взаємодію та ускладнювати обслуговування, а також ж стандартів журналів змін та моніторингу;
- відсутність чіткої приналежності: у міру впровадження нових послуг зростає кількість команд, які керують цими послугами. З часом стає важко знати, якими послугами може скористатися команда та до кого звернутися за підтримкою.

Мікросервіси надають певні переваги для компаній та програмного забезпечення, які невпинно розвиваються та збільшуються. Оскільки вони працюють незалежно один від одного, кожен сервіс може бути розроблений, оновлений, розгорнутий та масштабований без впливу на інші служби. Це дозволяє частіше впроваджувати оновлення програмного забезпечення із покращеною надійністю та продуктивністю.

При переході до мікросервісної архітектури компанії отримують можливість використовувати DevOps-практики, що забезпечує неперервну доставку та адаптацію до вимог користувачів. Цей підхід особливо корисний для компаній, що динамічно розвиваються і прагнуть забезпечити масштабованість команд та географічних розташувань, забезпечуючи нові можливості для оперативного оновлення продукту та збільшення його ефективності та конкурентоспроможності.

Також, не можна оминути стороною архітектуру хмарних сервісів, що має практичне застосування, переваги і недоліки подібно до інших архітектур. Дана архітектура представлена на рисунку 2.6.

Хмарні обчислення зараз є однією з найпопулярніших технологічних концепцій[23]. Завдяки цьому кожна організація тепер приймає новий погляд на віртуалізовані послуги та ресурси на вимогу. Незалежно від розміру, кожна організація використовує хмарні служби для доступу або зберігання інформації, яка завжди легко доступна з будь-якої точки світу через Інтернет.

Основа хмарної архітектури об'єднує різноманітні технології, роблячи їх доступними для всіх та звідусіль. У визначенні хмари як послуг, доступних клієнтам на вимогу, незалежно від місця та часу, хмарна інфраструктура представляє собою набір технологій, що забезпечує ефективну роботу всієї системи обчислень в хмарі. Вона може використовуватися в приватних, публічних та гібридних хмарових середовищах, є витратно-ефективною, гнучкою та простою у масштабуванні.

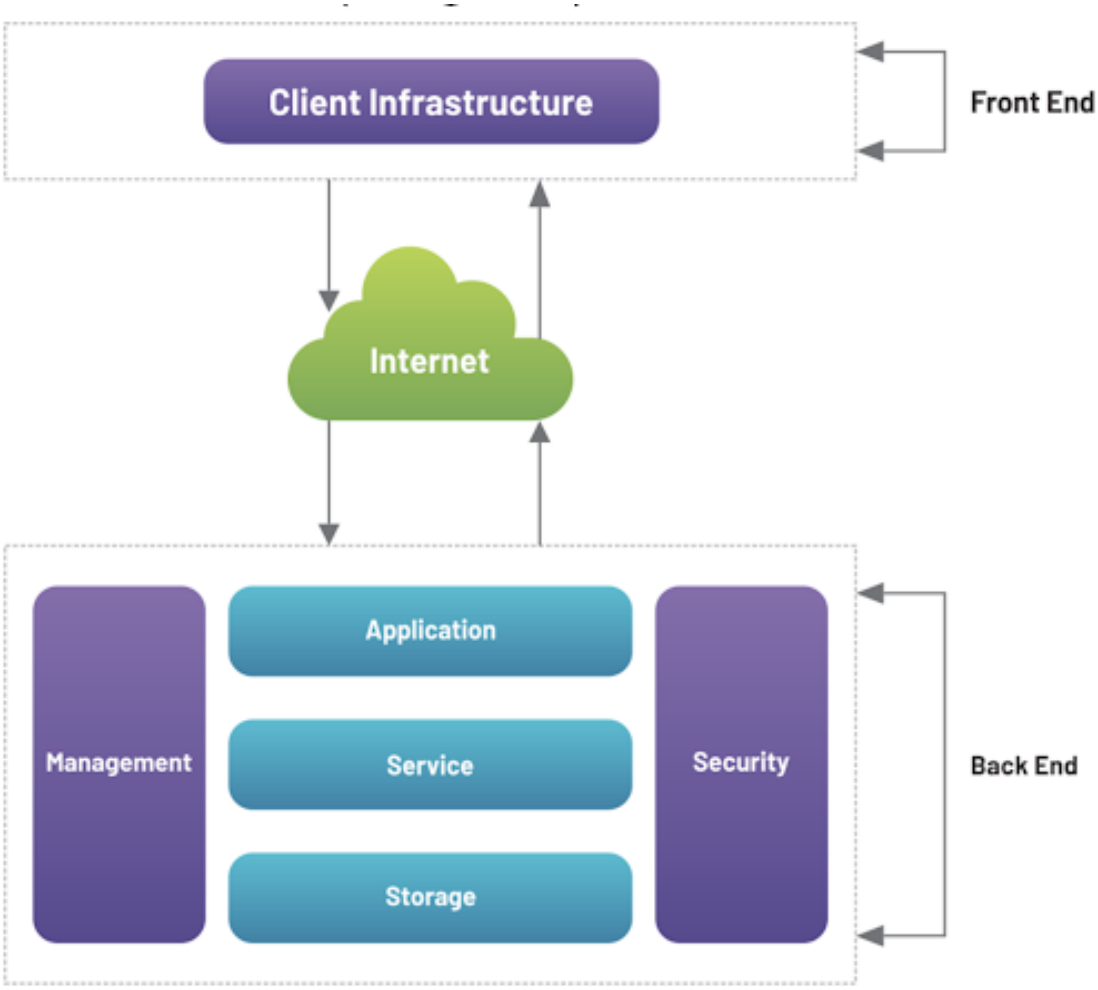


Рисунок 2.6 Принцип роботи хмарної архітектури

Зазвичай хмарна структура включає в себе модель доставки, мережу, рівні безпеки та керування, і, що найважливіше, бекенд і інтерфейс.

- інтерфейс: набір елементів, з якими взаємодіє клієнт є інтерфейсом користувача та програмою, які використовуються для доступу до хмарних служб;
- бекенд: хмара, яку використовує хмарний постачальник, який зберігає, керує та захищає ресурси. на додаток до цього, він містить сховище, віртуальні машини, способи контролю трафіку, моделі розгортання тощо.;
- зберігання: віртуальне місце для керування та безпечного зберігання даних;

- керування: елементи, які допомагають піклуватися про програму, службу, час виконання, сховище, інфраструктуру та інші механізми безпеки;
- безпека: практика та інструменти для використання різноманітних механізмів безпеки для збереження хмарних ресурсів, систем і файлів у безпеці для кінцевих користувачів;
- мережа: підключення до інтернету, яке діє як міст між інтерфейсом і сервером, дозволяючи їм спілкуватися та взаємодіяти один з одним.

Хмарна архітектура дозволяє компаніям відмовитися від підходу із початковими витратами та перейти до стратегії операційних витрат. Цей перехід в основному характеризується трьома основними типами хмарної архітектури:

- програмне забезпечення як послуга (SaaS) представляє собою метод доставки та управління програмами, де постачальник відповідає за оновлення та підтримку клієнтських програм. Користувачі уникають необхідності самостійного завантаження та розгортання хмарної архітектури, оскільки більшість платформ SaaS пропонують веб-інтерфейс, сумісний з різними операційними системами і браузерами;
- платформа як послуга (PaaS) описує хмарний підхід, де постачальник надає доступ до стеку програмного забезпечення, включаючи обчислювальну платформу. У цьому випадку користувачі відповідають за встановлення та конфігурацію програмного забезпечення, в той час як постачальник хмарних послуг відповідає за обслуговування серверів, мережі та сховища;
- інфраструктура як послуга (IaaS) визначається стороннім постачальником, який забезпечує обслуговування серверів, мереж і засобів зберігання для компаній. Цей підхід ідеально підходить для організацій, які бажають оплачувати лише за ресурси, які їм реально потрібні.

Хмарні архітектури надають широкий спектр переваг, серед яких варто відзначити:

- збільшення швидкості впровадження нових програм;
- використання власних хмарних архітектур, таких як kubernetes, для оновлення програм і прискорення цифрової трансформації;
- спрощення загальної структури хмарних систем;
- підвищення вимог до обробки даних;
- забезпечення вищого рівня безпеки;
- покращене відновлення після аварійних ситуацій;
- зменшення операційних витрат в галузі іт;
- забезпечення відповідності останнім вимогам та стандартам;
- запобігання витоку даних;
- отримання ресурсів набагато швидше;
- використання гібридної хмарної архітектури для масштабування програм у реальному часі;
- постійне досягнення бізнес-цілей.

Незважаючи на зазначені переваги, хмарні архітектури також мають свої недоліки, включаючи:

- питання щодо конфіденційності даних;
- можливість виникнення проблем з безпекою та приватністю;
- залежність від доступу до інтернету для роботи;
- потенційні обмеження управління та контролю над інфраструктурою;
- ризик виникнення непередбачених ситуацій із збереженням даних.

Хмарна програма створена з урахуванням певних методів розробки, які добре спрацювали на деяких найбільших і найпопулярніших веб-сайтах у світі. Багато з цих підходів, хоча не широко використовуються, проявили свою ефективність в забезпеченні високої масштабованості та продуктивності. Це призвело до їхнього впровадження у обмежене число компаній, які дійсно мали потребу в таких рішеннях. Коли певний метод використовується достатньо часто і продемонструє високий рівень ефективності, він стає патерном для досягнення аналогічного результату. У зведеному вигляді, патерн – це рекомендований спосіб виконання конкретної задачі чи завдання, який можна успішно використовувати повторно.

Методологія хмарної архітектури є випробуваною та надійною стратегією для вирішення різноманітних завдань у хмарних програмах. Хмарні патерни використовуються у розробці хмарних додатків, призначених для використання в різних хмарових середовищах, таких як AWS, Azure чи приватні хмари. Про шаблони проектування для хмарних додатків не завжди говорять, особливо досягнувши певного розміру компанії. Незважаючи на наявність численних шаблонів проектування для вибору, взаємодія з масштабуванням, коли воно є необхідним, може стати однією з найскладніших аспектів. Справжньою суттю шаблонів хмарної архітектури є їх роль як будівельних блоків для хмарних програм, які можна інтегрувати та ефективно управляти на різних хмарових платформах.

Далі розглядається архітектура безсерверних сервісів, що представлена на рисунку 2.7.

Рисунок 2.7 Принцип роботи безсерверної архітектури

Безсерверна архітектура – це спосіб розробки програмного забезпечення, який дозволяє розробникам створювати та запускати сервіси, не витрачаючи час на управління базовою інфраструктурою. У цьому підході розробники можуть писати та розгортати код, отримуючи від хмарного постачальника сервери для виконання їхніх програм, доступ до баз даних та систем зберігання, і все це легко масштабується.

Сервери забезпечують взаємодію з користувачами, надаючи можливість користуватися функціоналом програм. Але управління серверами вимагає значних зусиль та ресурсів. Команди повинні взяти на себе обов'язки щодо апаратного забезпечення сервера, підтримки програмного забезпечення та своєчасного

оновлення безпеки, а також регулярного створення резервних копій на випадок непередбачуваних ситуацій. Застосування безсерверної архітектури дозволяє розробникам передавати ці обов'язки сторонньому постачальнику, що дає можливість їм зосередитися на написанні програмного коду.

Однією з найпопулярніших форм безсерверної архітектури є концепція "функції як послуга" (FaaS) (див.рис.2.7), де розробники створюють програмний код у вигляді набору індивідуальних функцій. Кожна функція виконує конкретне завдання, спрацьовуючи при настанні певних подій, таких як вхідна електронна пошта або HTTP-запит. Після перевірки та тестування розробники розгортають свої функції разом із тригерами у хмарному середовищі. Коли викликається функція, хмарний постачальник виконує її на активному сервері або, якщо жоден сервер не працює, автоматично створює новий сервер для виконання функції. Цей процес виконання відділений від розробників, які можуть зосередитися на написанні та розгортанні коду програми.

Навіть при відсутності необхідності управління сервером у безсерверній архітектурі, існує певна крута крива навчання, особливо коли розробники комбінують кілька функцій для створення складних робочих процесів у програмі. Саме тому корисно ознайомитися із ключовими термінами в безсерверному середовищі:

- виклик: виконання окремої функції;
- тривалість: час, необхідний для виконання конкретної безсерверної функції;
- холодний старт: затримка, що виникає при першому запуску функції або після певного періоду бездіяльності;
- обмеження паралельності: визначає кількість одночасно працюючих екземплярів функцій у конкретному регіоні, встановлену провайдером хмарних послуг. якщо функція перевищить це обмеження, її виконання буде призупинено;
- час вийшов: визначає період, протягом якого постачальник хмарних послуг дозволяє функції виконуватися перед її припиненням. зазвичай,

провайдери встановлюють час очікування за замовчуванням та максимальний час очікування.

Обов’язково варто враховувати, що кожен провайдер хмарних послуг може мати свою власну термінологію та встановлювати унікальні обмеження для безсерверних функцій. Проте викладений вище перелік визначає основні концепції, не зважаючи на можливі відмінності в термінології та обмеженнях.

Останніми роками впровадження безсерверних технологій значно зростає в геометричній прогресії, і практично 40 відсотків компаній у всьому світі впроваджують їх у різних формах.

Невеликі стартапи і глобальні корпорації вдаються до використання безсерверних архітектур з різних причин:

- витрати: за використання хмарних послуг обчислення стягуються оплати за кожен окремий виклик, забезпечуючи відсутність оплати за неактивні сервери чи віртуальні машини;
- масштабованість: екземпляри функцій автоматично створюються чи видаляються в залежності від змін в трафіку, дотримуючись обмежень паралельності;
- ефективність: інженери, які працюють у безсерверному режимі, можуть просто розгортати свій код, не обтяжуючись керуванням серверами, що сприяє прискоренню циклів доставки та оперативному масштабуванню операцій організації.

Існують і деякі труднощі, пов'язані із застосуванням безсерверних архітектур, що може негативно вплинути на проект, а саме:

- втрата контролю: у безсерверних середовищах ви не маєте повного контролю над програмним стеком, на якому виконується ваш код. В разі збою обладнання, неполадок у центрі обробки даних чи іншої проблеми, що впливає на один із ваших серверів, ви залежите від постачальника хмарних послуг для вирішення цих проблем;

- безпека: хмарний постачальник може запускати код кількох клієнтів на одному сервері одночасно. Якщо спільний сервер налаштовано невірно, ваші дані програми можуть бути вразливими для розкриття;
- вплив на продуктивність: холодний запуск є загальним явищем у безсерверних середовищах, що додає деяку затримку до виконання коду, коли функції викликаються після перерви в активності;
- тестування: хоча розробники можуть виконувати модульні тести функціонального коду, інтеграційні тести, що оцінюють взаємодію інтерфейсних та серверних компонентів, можуть бути складнішими для проведення в безсерверному середовищі;
- блокування постачальника: великі постачальники хмарних послуг, такі як AWS, пропонують різноманітні послуги, такі як бази даних, черги обміну повідомленнями та API, які можна використовувати для спрощення запуску безсерверних програм. Навіть якщо можна комбінувати елементи від різних постачальників, послуги одного постачальника спрямовані на максимально зручну інтеграцію.

Підприємства, які націлені на мінімізацію часу виходу на ринок та розробку легких та масштабованих програм, можуть здобути значні переваги від використання безсерверних підходів. Проте, у випадку, якщо програми включають в себе велику кількість тривалих процесів, вигідніше може бути використання віртуальних машин або контейнерів. У гібридній інфраструктурі розробники можуть застосовувати контейнери або віртуальні машини для обробки основної частини запитів і в той же час передавати конкретні короткострокові завдання, такі як зберігання баз даних, функціоналу безсерверних компонентів.

Безсерверну архітектуру найбільш доцільно використовувати для виконання короткострокових завдань і обробки робочих навантажень, які можуть зазнавати різкого або непередбачуваного трафіку.

Основні випадки використання безсерверного підходу включають:

- завдання на основі тригерів: будь-яка діяльність користувача, що викликає подію або серію подій, стає відмінним варіантом для

застосування безсерверної архітектури. Наприклад, реєстрація користувача на веб-сайті може ініціювати зміну бази даних, яка, в свою чергу, може активувати відправку привітального електронного листа. Роботу серверної частини можна організувати через ланцюжок безсерверних функцій;

- створення RESTful API: використання Amazon API Gateway разом із безсерверними функціями дозволяє створювати масштабовані RESTful API, які автоматично адаптуються до обсягу запитів;
- асинхронна обробка: безсерверні функції ефективно вирішують фонові завдання програми, такі як рендеринг інформації про продукт або перекодування відео після завантаження, не впливаючи на основний потік програми і не спричиняючи затримок для користувача;
- перевірка безпеки: при запуску нового контейнера можна активувати функцію для перевірки наявності неправильних конфігурацій чи вразливостей. Безсерверні функції також можуть служити більш безпечною опцією для перевірки SSH та двофакторної автентифікації;
- безперервна інтеграція (CI) і безперервна доставка (CD): безсерверні архітектури дозволяють автоматизувати багато етапів конвеєрів CI/CD. Наприклад, зміни в коді можуть ініціювати функцію для створення збірки, а запити на злиття можуть автоматично сприйматися як сигнали для проведення автоматичних тестів.

Багато розробників поступово переходять до безсерверного режиму, поступово переносячи деякі частини своїх програмних продуктів на безсерверну архітектуру, а решту залишають на традиційних серверах. Безсерверні рішення легко масштабуються, що дозволяє поетапно впроваджувати додаткові функції, коли з’являються відповідні можливості.

На завершення слід розглянути ще одну популярну архітектуру клієнт-сервер представлену на рисунку 2.8.

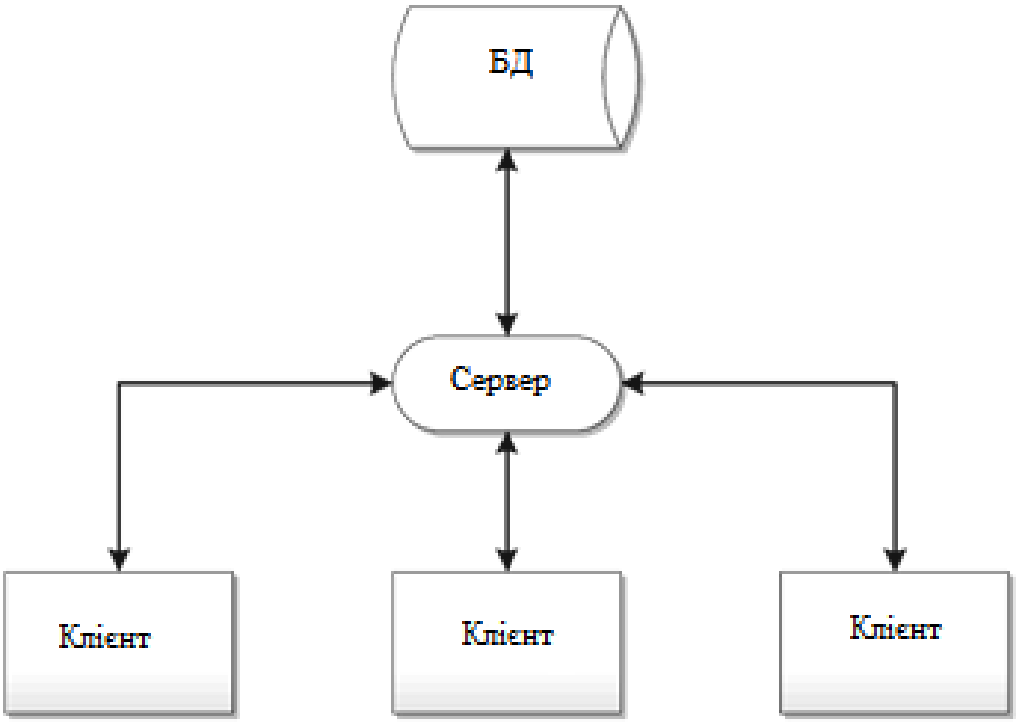


Рисунок 2.8 Принцип роботи клієнт-серверної архітектури

Архітектура клієнт-сервер відноситься до системи, яка розміщує, надає та керує більшістю ресурсів і послуг, що запитується клієнтом. У цій моделі всі запити та послуги передаються через мережу, і цю модель також називають моделлю мережевих обчислень або мережею клієнт-сервер.

Архітектура клієнт-сервер, відома також як модель клієнт-сервер, є мережевою програмою, яка розподіляє завдання та навантаження між клієнтами та серверами, які можуть бути в одній системі або об'єднані комп'ютерною мережею.

Зазвичай клієнт-серверна архітектура включає декілька робочих станцій користувачів, комп'ютерів чи інших пристроїв, які підключені до центрального сервера через Інтернет або іншу мережу. Клієнт ініціює запит на отримання даних, а сервер приймає та обробляє цей запит, відправляючи пакети даних користувачеві, якому вони необхідні.

Цю архітектуру також іноді називають як мережа клієнт-сервер або модель мережевих обчислень.

Процес роботи:

- починаючи, клієнт направляє свій запит через мережевий пристрій;
- далі мережевий сервер приймає та обробляє запит від користувача;

- нарешті, сервер надсилає відповідь клієнту.

Сучасним компаніям необхідна система, яка спрощує збір, обробку та використання корпоративних даних, сприяючи підвищенню ефективності бізнес-процесів і забезпечуючи конкурентоздатність на сучасному світовому ринку.

Модель мережі "клієнт-сервер" забезпечує високий рівень обробки, що сприяє підвищенню продуктивності робочих станцій, розширенню можливостей робочих груп, віддаленому управлінню мережею, а також орієнтації на ринок бізнесу та збереженню наявних інвестицій.

Характеристики архітектури клієнт-сервер зазвичай включають:

- клієнтські та серверні машини часто вимагають різних апаратних та програмних ресурсів і можуть бути поставлені від різних постачальників;
- мережа підтримує горизонтальне масштабування, що дозволяє збільшувати кількість клієнтських машин, і вертикальне масштабування, коли весь процес може бути переміщений на більш потужні сервери або конфігурації з кількома серверами;
- один сервер може обслуговувати кілька служб одночасно, проте для кожної служби може бути використана окрема серверна програма;
- клієнтські та серверні програми взаємодіють безпосередньо з протоколом транспортного рівня, який встановлює зв'язок та забезпечує передачу інформації;
- як клієнтські, так і серверні комп'ютери використовують повний стек протоколів, де транспортний протокол використовує протоколи нижчого рівня для передачі окремих повідомлень.

Архітектура клієнт-сервер має свої переваги і недоліки для споживачів.

Головні переваги використання даної архітектури:

- централізована система, що забезпечує об'єднане зберігання всіх даних та елементів керування;
- високий рівень масштабованості, організації та ефективності завдяки централізованій структурі;

- можливість окремо налаштовувати потужності клієнтів та серверів іт-персоналом;
- економічно вигідна, зокрема з точки зору обслуговування;
- забезпечує можливість відновлення даних;
- можливість балансування навантаження для оптимізації продуктивності;
- дозволяє різним платформам обмінюватися ресурсами;
- не вимагає від користувачів входу в термінал або інший процесор для доступу до корпоративної інформації чи інструментів;
- мінімізує випадки реплікації даних за допомогою налаштувань.

Також слід враховувати, що існують певні обмеження та негативні аспекти використання цієї архітектури:

- у разі наявності вірусів, хробаків чи троянів на сервері може виникнути загроза їх поширення серед користувачів, оскільки мережа взаємопов'язаних клієнтів і серверів сприяє цьому;
- сервер стає уразливим до атак типу "відмова в обслуговуванні" (DoS) ;
- можливість підроблення або зміни пакетів даних під час їх передачі;
- запуск та початкове впровадження є високими за вартістю;
- в разі виходу з ладу критично важливого сервера клієнти залишаються без обслуговування;
- налаштування вразливе до фішингу та атак типу "людина посередині" (MITM).

Виходячи з аналізу різних архітектур, а також встановлення їхніх основних переваг та недоліків було визначено, що найкраще використовувати монолітну архітектуру для розробки веб-додатку управління завданнями на основі Django та Python. Django є фреймворком, який добре підходить для монолітних додатків, і він сприяє швидкому розгортанню та розробці. Такий підхід забезпечить простоту управління кодом, а також зручність та продуктивність у процесі розробки веб-додатку.

Інші архітектури, такі як мікросервісна, хмарні сервіси, і безсерверні сервіси, зазвичай використовуються для більших та складніших систем, де важлива

гнучкість та масштабованість. Однак вони можуть бути занадто складними для невеликих веб-додатків та принести зайву складність у розробці та управлінні.

Сервер-клієнтська архітектура теж може використовуватися для веб-додатків, але вона частіше використовується в контексті розподіленого програмного забезпечення та не є основним вибором для бекенду Django.

UML-діаграму компонентів веб-додатку для управління завданнями та взаємодію модулів і інтерфейсів представлено на рисунку 2.9.

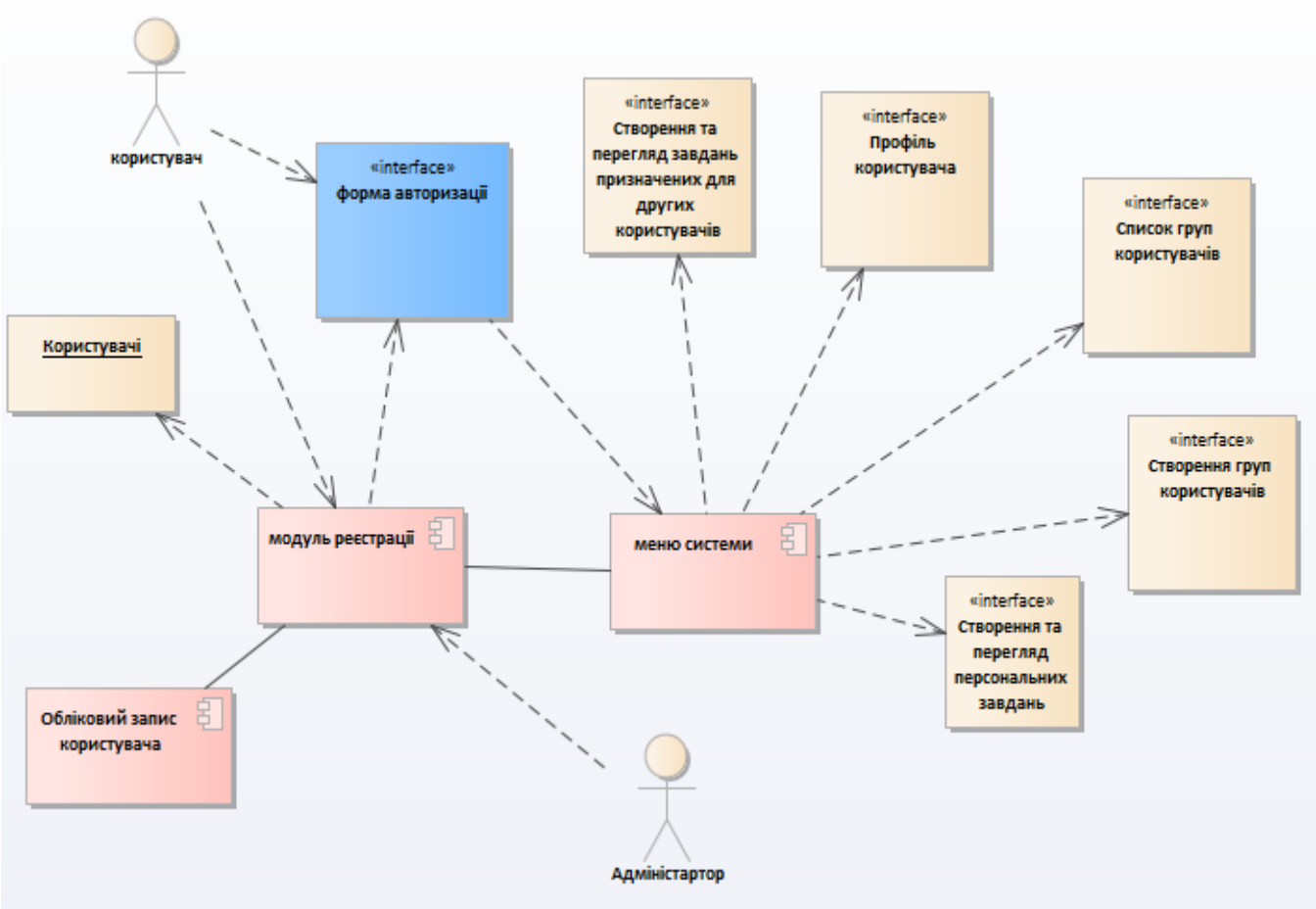


Рисунок 2.9 UML-діаграма компонентів веб-додатку для управління завданнями

2.3 Алгоритм роботи програмного продукту

Перед початком розробки веб-додатку для управління завданнями на основі Django та Python, потрібно продумати алгоритм роботи програмного продукту, його основну логіку, враховуючи вже визначені вимоги до проекту.

Алгоритм роботи програмного продукту - це систематично описаний набір інструкцій, процедур та правил, які визначають послідовність операцій, що виконуються програмою для досягнення певної мети чи вирішення конкретної задачі. Це концептуальний план дій, який визначає логіку та порядок взаємодії між компонентами програмного продукту з метою отримання очікуваного результату. Алгоритм роботи є ключовим елементом в процесі програмування та визначає основні етапи обробки даних та прийняття рішень в межах програмного застосунку.

Для побудови алгоритму роботи веб-додатку можна виділити наступні етапи:

- аналіз вимог:
  - детальний розгляд вимог до системи та їх узгодження зі стекхолдерами;
  - визначення функціональних та нефункціональних вимог.
- архітектура системи:
  - обрання архітектурного підходу.
- визначення сутностей:
  - ідентифікація основних сутностей системи (користувачі, завдання, проекти тощо);
  - визначення взаємозв'язків між сутностями.
- планування інтерфейсу:
  - проектування користувацького інтерфейсу;
  - розгортання шляхів навігації та взаємодії з користувачем.
- визначення бізнес-логіки:
  - опис логіки обробки завдань та подій;

- визначення прав доступу користувачів.

- безпека:
  - визначення стратегії безпеки;
  - розгляд заходів для захисту від атак та зловживань.
- тестування:
  - планування тестових сценаріїв та їх виконання;
  - розгортання системи тестування та валідації.
- документація:
  - створення технічної документації для розробників та адміністраторів;
  - опис алгоритму роботи програмного продукту у вигляді структурованої документації.
- розгортання та моніторинг:
  - розробка плану розгортання та стратегії моніторингу продукту в реальному часі;
  - визначення механізмів збору та аналізу логів.
- підтримка та післязапусковий етап:
  - визначення стратегії технічної підтримки;
  - забезпечення можливостей для майбутніх розширень та оновлень.

Додавання цих етапів допоможе створити повний план роботи алгоритму системи для розробки веб-додатку, забезпечити його стабільність та відповідність встановленим раніше вимогам користувачів.

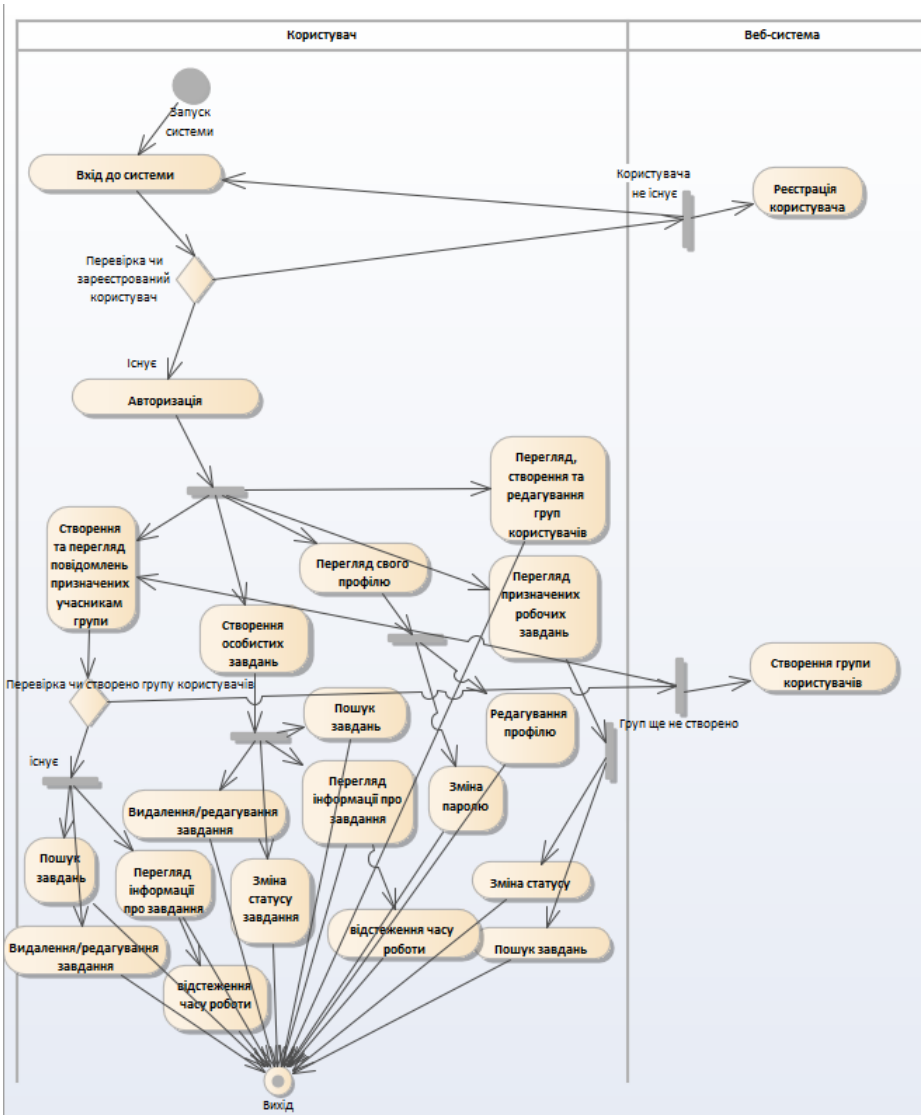


Рисунок 2.10 Алгоритм роботи системи

Створений алгоритм роботи програмного продукту дозволить реалізувати веб-додаток для управління завданнями на основі Django та Python. Передбачені основні дії користувача і продемонстровані події, які виникатимуть при взаємодії з однією із функцій, були враховані функціональні, нефункціональні враховані вимоги до проекту для більш точного відображення поведінки системи. Веб-додаток може здійснювати перевірку на існування облікового запису користувача для доступу до основного функціоналу, а у випадку невдалої спроби авторизації в системі пропонуватиме спочатку пройти процес реєстрації і повторити спробу входу. Присутня перевірка на існування груп, що може бути корисно, коли користувач бажає створити завдання для когось з других людей, колег, знайомих, друзів, але у нього відсутня створена група з ними.

3 РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ЗАВДАННЯМИ

3.1 Обґрунтування вибору інструментальних засобів розроблення

Для розробки веб-додатку управління завданнями було обрано конкретні інструменти та мови програмування, які забезпечать його легку та ефективну реалізацію та працездатність.

Python є важливою мовою програмування, що отримала значну популярність у сфері веб-розробки[26]. Потужні та привабливі характеристики роблять його відмінним вибором для створення надійних та масштабованих веб-додатків.

Основні причини чому Python широко використовується у веб-розробці та є ефективним:

- читабельність і простота: синтаксис Python призначений для легкості читання та написання коду, що спрощує обслуговування та робить його доступним для розробників будь-якого рівня. Простота мови дозволяє висловлювати ідеї меншим обсягом коду, що сприяє швидкій та ефективній розробці;
- велика та активна спільнота: Python має потужну спільноту розробників, яка забезпечує велику кількість бібліотек і фреймворків для веб-розробки. Це робить мову ефективним інструментом для широкого спектру завдань;
- великі бібліотеки та фреймворки: Python пропонує багатий вибір бібліотек і фреймворків, таких як Django та Flask, які спрощують створення складних веб-додатків. Ці інструменти забезпечують потужну основу для продуктивної розробки;
- масштабованість і продуктивність: Python відомий своєю здатністю ефективно обробляти великий трафік завдяки таким вдосконаленням, як асинхронне програмування. Це робить його придатним для веб-додатків із високою навантаженістю;

- інтеграція та сумісність: Python легко інтегрується з різними технологіями, включаючи різні бази даних та інші сервіси, розширюючи його можливості та гнучкість;
- тестування та налагодження: Python має потужні інструменти для тестування, такі як unittest та pytest, і забезпечує зручні засоби для налагодження, сприяючи виявленню та усуненню помилок;
- швидка розробка: Простота та гнучкість Python дозволяють розробникам швидко створювати веб-додатки, що робить його ідеальним вибором для проектів з короткими термінами виходу на ринок.

Python має хороший вибір веб-фреймворків для розробки додатків, кожен з яких підходить під конкретні цілі та завдання, відповідно – це в рази полегшує реалізацію проекту.

Django є повнофункціональним фреймворком високого рівня, відомим своїм принципом "батарейки включені", так як при створені проекту він вже забезпечує чітку структуру та багато готових реалізованих функцій. Цей фреймворк пропонує обширний набір інструментів і функціональностей для швидкої розробки складних веб-додатків. Django включає в себе систему ORM (Object-Relational Mapping) для управління базами даних, маршрутизацію URL-адрес, обробку форм, автентифікацію та інші функції. Він дотримується архітектурного шаблону Model-View-Controller (MVC)[27].

До основних переваг можна віднести:

- продуктивність: завдяки технологіям фреймворку та наявності значної кількості бібліотек і обширній документації;
- простота: завдяки готовому набору інструментів та чіткій структурі проекту, розробка стає значно простішою;
- безпека: даний фреймворк має вбудований захист вже при першому створенні проекту, наявні функції захист запобігають більшості поширених типів атак на систему;

- наявність вбудованої адміністративної панелі значно спрощує розробку, адже не потрібно її розробляти з повного нуля, якщо нема таких вимог до проекту;
- масштабованість: на відміну від більшості фреймворків, додавання апаратного забезпечення не впливає на роботу вже існуючих компонентів, а модульну структури завжди можна легко розширити або видалити не вплинувши на решту проекту;
- великій вибір бібліотек, практично на всі випадки життя.

Однією з ключових переваг, яка робить Django відмінним вибором для проекту, є його широка сумісність. Це означає, що фреймворк може бути успішно використаний в різних проектах, незалежно від їхньої бізнес-спрямованості або технологічних вимог. Чи ви прагнете створити унікальну веб-платформу, чи шукаєте надійне рішення з відкритим вихідним кодом, Django надає потрібну гнучкість та ефективність у розробці програмного забезпечення.

В якості бази даних для працездатності веб-додатку була вибрана SQLite3, так як вона вже встановлена та забезпечує необхідний рівень функціоналу додатку на початковому етапі.

Для реалізації динамічних змін при роботі з завданнями використовується JavaScript.

JavaScript – це мова програмування, яка використовується для додавання динаміки та інтерактивності на веб-сторінках. Зазвичай використовується для обробки подій, маніпуляції DOM (Document Object Model) та взаємодії з користувачем.

Однією з переваг використання даної мови програмування є можливість розробки веб-додатків реалізовуючи динамічний інтерфейс взаємодії з користувачем без перезавантаження сторінки. Що покращує продуктивність та користувацький досвід.

Також, для створення та забезпечення зручного та ергономічного інтерфейсу користувача, в проекті використовуються HTML, CSS та Bootstrap, що допоможуть

візуально модифікувати систему управляння завданнями для взаємодії на клієнтські стороні.

HTML (Hypertext Markup Language) є стандартною мовою розмітки для створення структури веб-сторінок. Використовуючи теги та їхню вкладеність, ви визначаєте елементи сторінки, такі як заголовки, абзаци, таблиці, форми і багато інших.

Переваги: простота використання, широка підтримка браузерами, ключова складова веб-розробки.

CSS (Cascading Style Sheets) відповідає за стиль та вигляд веб-сторінок, дозволяючи задавати кольори, шрифти, відступи, розташування елементів тощо.

Переваги: покращує управління та підтримку коду, можливість створення адаптивного та привабливого дизайну.

Bootstrap – це відкритий фреймворк для розробки веб-сайтів з використанням HTML, CSS та JS. Він містить зручні компоненти та стилі, що допомагають створювати адаптивний та стильний дизайн.

Переваги: прискорює розробку завдяки готовим компонентам, забезпечує адаптивність для різних пристроїв, сприяє консистентності та зручності сумісності.

Як це згодиться в розробці веб-додатку для управління завданнями:

- HTML визначатиме структуру сторінок, таку як форми для додавання завдань, списки завдань тощо;
- CSS відповідатиме за зовнішній вигляд, забезпечуючи зручний та привабливий дизайн;
- JavaScript використовуватиметься для реалізації динамічних функцій, таких як створення завдань, фільтрація користувачів за групами до яких вони належать, оновлення статусу завдань без перезавантаження сторінки;
- Bootstrap спростить створення адаптивного та консистентного дизайну, забезпечуючи готові компоненти для ефективної розробки та взаємодії користувача з компонентами веб-додатка.

Всі ці технології добре інтегруються з Django, оскільки фреймворк використовує спеціальні шаблони для роботи з фронтендом.

3.2 Забезпечення безпеки даних у веб-додатку

У сучасному світі технологій існує безліч різноманітних атак, які спрямовані на порушення безпеки та викрадення інформації з веб-додатків. Ці атаки використовують різні вразливості програмного забезпечення.

Безпечний веб-додаток – це той, який може ефективно працювати навіть при наявності різних атак, забезпечуючи захист конфіденційної інформації[24]. Незважаючи на те, що сам програмний продукт може вживати заходи для своєї безпеки, також необхідні додаткові інструменти для гарантування безпеки та захисту від можливих атак. Недоліки у механізмах безпеки можуть впливати на всі аспекти функціонування додатка, включаючи його доступність.

Веб-загрози постійно змінюються, тому необхідно стежити за ними та вживати необхідних заходів, що вбережуть систему від витоку даних.

До основних небезпек які становлять загрозу для веб-додатків відносяться:

- порушення контролю доступу: атаки можуть відбутися внаслідок неправильно налаштованих дозволів для автентифікованого користувача, що може призвести до надмірного обсягу дозволів і, відповідно, втрати конфіденційних даних;
- криптографічні збої: проблеми з шифруванням під час передачі, такі як використання незахищеного протоколу HTTP, можуть викликати криптографічні збої, що спричиняють витік конфіденційних даних через мережу;
- ін'єкційні атаки: атаки, спрямовані на введення зловмисником в шляхи введення даних, можуть викликати підробку чи видалення даних, що призводить до великих ризиків безпеки;

- ненадійний дизайн: проблеми з дизайном контролю доступу можуть викликати різноманітні атаки, які порушують безпеку системи через недостатність архітектурних заходів;
- неправильна конфігурація безпеки: використання стандартних паролів, наявність шкідливих бібліотек чи непотрібних компонентів може призвести до небезпеки та вразливостей;
- уразливості програми та застарілі компоненти: використання застарілих компонентів чи програмного забезпечення може відкривати двері для атак, які використовують відомі вразливості;
- помилки ідентифікації та автентифікації: використання слабких паролів або неефективних методів автентифікації може призвести до неправильного управління сеансами та витоку облікових даних;
- порушення цілісності програмного забезпечення та даних: недостатня увага до цілісності програм та використання ненадійних репозиторіїв може стати джерелом серйозних загроз безпеці;
- збої реєстрації та моніторингу безпеки: недостатній моніторинг може призвести до невчасного виявлення аномальних дій та атак, що загрожують безпеці системи;
- підробка запитів на стороні сервера (SSRF): використання зловмисником URL-адрес, які ініціюють небезпечні дії на сервері, може призвести до серйозних наслідків, таких як розкриття конфіденційних даних чи втрата контролю.

Django має вбудовані функції захисту від більшості поширених атак. Він здійснює обробку вхідних даних від користувачів, забезпечуючи їхню безпеку для подальшого використання в програмі. Ці функції дозволяють уникнути потенційно небезпечних символів у запитах, що робить дані безпечними для подальшого використання в програмному коді.

Ось як деякі з цих атак можуть бути захищені за допомогою функцій безпеки Django:

- порушення контролю доступу:

  - Django використовує систему дозволів та груп для контролю доступу. Користувачам надаються конкретні дозволи, і їх членство в групах визначає їх права доступу;
  - використання декораторів, таких як @login_required, дозволяє обмежити доступ до конкретних частин додатку тільки для автентифікованих користувачів;
  - використання системи ролей для приведення відповідних дозволів до користувачів.
- криптографічні збої:
  - Django автоматично переводить з'єднання на захищений протокол HTTPS для ефективного шифрування даних, які передаються між клієнтом і сервером;
  - використання бібліотек криптографії Django, таких як django.crypto, для шифрування конфіденційних даних перед збереженням у базі даних.
- ін'єкційні атаки:
  - використання параметризованих запитів та ORM Django допомагає уникнути SQL ін'єкцій, оскільки дані автоматично екрануються та валідуються;
  - використання функцій Django для валідації та фільтрації введених даних для запобігання вразливості до ін'єкцій через різні канали.
- ненадійний дизайн:
  - використання вбудованих функцій Django для контролю доступу, таких як система дозволів та груп, допомагає уникнути атак, пов'язаних з ненадійним дизайном контролю доступу.
- неправильна конфігурація безпеки:
  - повідомлення про стандартні паролі та рекомендації щодо безпеки паролів для користувачів;
  - використання безпечних методів аутентифікації та регулярне оновлення налаштувань безпеки в конфігураційних файлах.

- уразливості програми та застарілі компоненти:
  - Django сприяє використанню механізмів автоматичного оновлення та має вбудовані інструменти для виявлення та оновлення застарілих компонентів;
  - моніторинг загроз безпеки та реагування на оголошені вразливості у використовуваних бібліотеках та компонентах.
- помилки ідентифікації та автентифікації:
  - використання безпечних методів автентифікації, таких як двофакторна аутентифікація (2FA), для ускладнення процесу неправомірного доступу;
  - встановлення політик паролів та механізмів безпеки сеансів для ефективного управління доступом.
- порушення цілісності програмного забезпечення та даних:
  - використання систем контролю версій та регулярне оновлення програм для запобігання використанню застарілих версій існуючих компонентів.
- збої реєстрації та моніторингу безпеки:
  - використання систем журналювання та моніторингу, які слідкують за подіями в додатку для виявлення аномалій та атак;
  - регулярний аналіз журналів для вчасного виявлення та відповіді на підозрілі дії.
- підробка запитів на стороні сервера (SSRF):
  - використання валідації та фільтрації введених URL-адрес для запобігання виконанню небезпечних дій на сервері через невірний ввід.

Варто відзначити, що це не єдині вбудовані функції захисту, які забезпечує даний фреймворк. До них відносяться:

- захист від CSRF (Cross-Site Request Forgery):

    - Django включає вбудований захист CSRF, який генерує та вимагає унікальний токен для кожного запиту форми. Це ускладнює атакам CSRF подражання використання користувача.
- захист від XSS (Cross-Site Scripting):
    - використання автоматичної екранізації даних при виведенні на сторінки допомагає уникнути XSS-атак, де зловмисник вставляє в шаблони вредоносний код.
- захист від небезпечних файлів:
    - Django може обмежувати типи файлів, які можуть бути завантажені, та перевіряти їх вміст на предмет можливих загроз.
- безпека сесій:
    - використання безпечних механізмів сесій, таких як використання підписаних кукі для зберігання ідентифікатора сесії, допомагає уникнути атак на сесії.
- захист від небезпечних HTTP-запитів:
    - Django може включати заголовки безпеки, такі як Content Security Policy (CSP) та HTTP Strict Transport Security (HSTS), щоб ускладнити атаки, пов'язані із змістом сторінок та безпекою транспортного рівня.
- сканування на вразливості:
    - використання зовнішніх інструментів або служб для регулярного сканування на вразливості та виявлення потенційних проблем безпеки.
- моніторинг безпеки:
    - встановлення систем моніторингу та журналювання для вчасного виявлення та реагування на події, які можуть свідчити про атаки або аномальну поведінку.
- додаткові заходи безпеки для аутентифікації:
    - використання більш складних методів аутентифікації, таких як біометричні дані чи аутентифікація на основі апаратних ключів.
- захист від переповнення буфера:

  - використання безпечних механізмів обробки введених даних для уникнення атак на переповнення буфера.
- аудит безпеки:
  - проведення регулярних аудитів безпеки для виявлення та усунення потенційних загроз безпеці.

У Django використовуються деякі бібліотеки безпеки, серед яких:

- безпечна автентифікація: ця бібліотека забезпечує безпечний процес автентифікації, використовуючи багатофакторну аутентифікацію (MFA) з одноразовими паролями (TOTP), службами коротких повідомлень, кодами безпеки та запитання-відповідь. Також можливість поліпшення безпеки автентифікації за допомогою діапазонів IP та капч;
- Django Defender: бібліотека блокує спроби несанкціонованого входу в систему, використовуючи захист від методу грубої сили;
- активність сеансу Django: допомагає виявити підозрілу або зловмисну поведінку, відстежуючи дії в обліковому записі та виходи з усіх сеансів на різних пристроях;
- обмежені сеанси: ця бібліотека обмежує сеанси Django за IP-адресою та/або агентами користувача, аутентифікує їх і, у разі зміни IP або агента користувача, сеанс очищається;
- правила Django: надає можливості дозволів на рівні об'єкта для Django;
- Admin Honeypot і Django Honeypot: створюють приманки для виявлення зловмисної активності в адміністративних панелях та загалом;
- криптографія Django: використовується для шифрування даних в програмах Django.

Загалом, даний фреймворк вже забезпечує необхідними функціями захисту на початковому етапі розробки, що допоможе створити безпечний та надійний програмний продукт для управління завданнями. Проте, варто пам’ятати, що найкращою безпека може бути лише тоді, коли вона вище еталонного рівня та налаштована спеціально для додатку. Саме тому, розвиток, модифікації і впровадження нових функцій безпеки на всьому життєвому шляху веб-додатку

повинні невпинно та активно вдосконалюватись враховуючи існуючі проблеми та недоліки.

3.3 Опис компонентів програмного продукту

На рис 3.1 наведено ієрархію екранних форм веб-додатку для управління завданнями, що містить сім основних сторінок, з якими взаємодіє користувач:

- форма реєстрації;
- форма авторизації;
- мої завдання;
- робочі завдання;
- завдання для команди;
- список груп;
- профіль користувача.

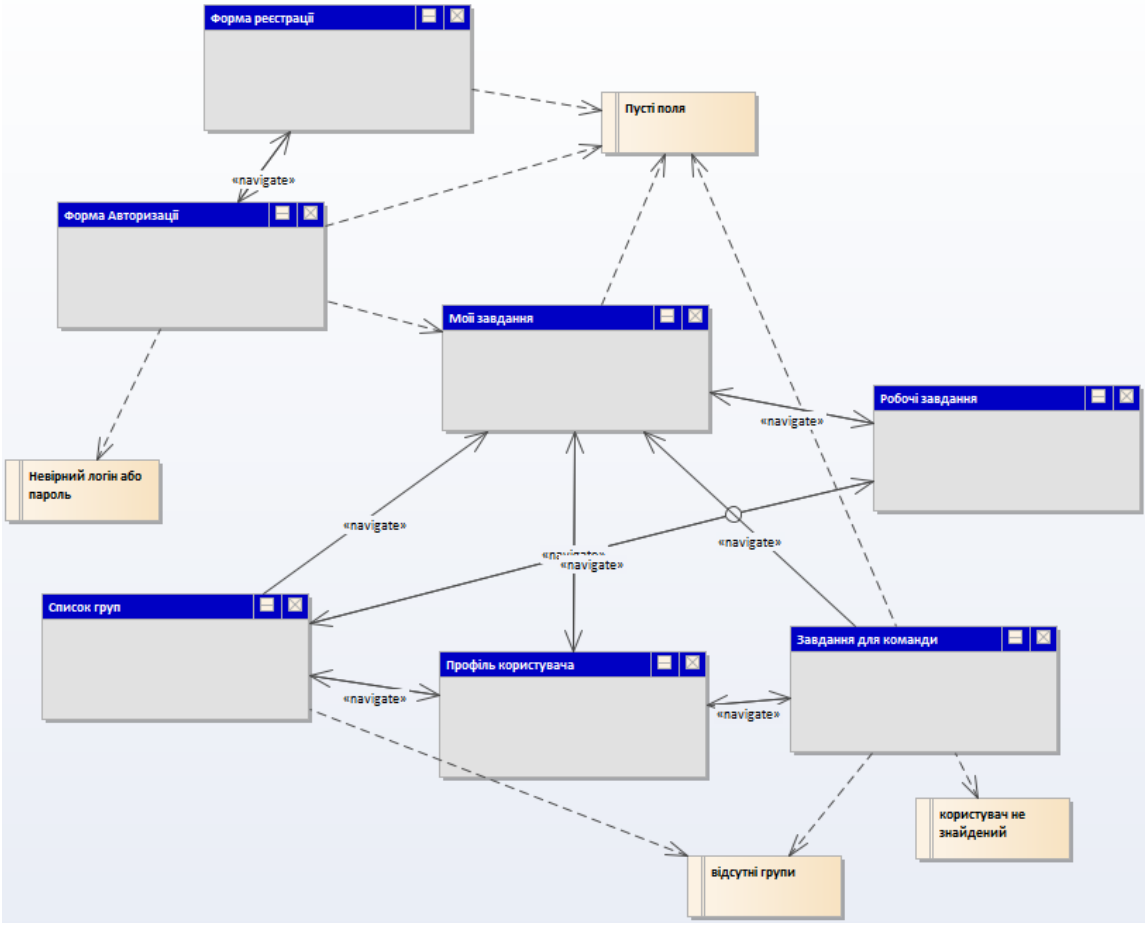


Рисунок 3.1 Ієрархія екранних форм

Фрагменти створеної бази даних для забезпечення працездатності веб-додатку представлено на рис. 3.2 – 3.7.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	title	varchar (500)								NULL
3	description	text								NULL
4	created_at	datetime								NULL
5	is_complete	bool								NULL
6	user_id	bigint								NULL
7	deadline	datetime								NULL
8	elapsed_time	real								NULL
9	start_time	datetime								NULL

Рисунок 3.2 таблиця «Мої завдання»

Дана таблиця (див.рис 3.2) містить поля: назва завдання, опис, дата створення, перевірочне поле статусу завдання (завершено чи ні), id користувача якому це все належить, поле для дедлайну, два поля для роботи з часом, що

забезпечують відстеження затраченого часу на завдання та обрахунок сумарного часу, який був використаний.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	name	varchar (100)								NULL
3	creator_id	bigint								NULL

Рисунок 3.3 таблиця «Групи користувачів»

Таблиця, що представлена на рис 3.3 містить дані про назву групи та користувача, якому вона належить, варто відзначити, список груп є персональним і доступний лише користувачам, які їх створювали.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	usergroup_id	bigint								NULL
3	customuser_id	bigint								NULL

Рисунок 3.4 таблиця «проміжна таблиця групи користувачів»

Дані, що представлені на рис 3.4 є проміжними і використовуються для фільтрації користувачів системи і їхньої належності до конкретних груп. В подальшому це дозволяє фільтрувати користувачів за їх приналежністю до конкретної групи.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	title	varchar (100)								NULL
3	description	text								NULL
4	deadline	datetime								NULL
5	is_complete	bool								NULL
6	created_at	datetime								NULL
7	assigned_to_id	bigint								NULL
8	created_by_id	bigint								NULL
9	elapsed_time_task	real								NULL
10	start_time_task	datetime								NULL

Рисунок 3.5 таблиця «Завдання для команди та Робочі завдання»

Таблиця показана на рис 3.5 містить в собі дані для створювання та призначення завдання конкретному користувачу, а також їх відображення на сторінці користувача в окремій вкладці «Робочі завдання». До таблиці входять поля: id, назва завдання, опис, дедлайн, перевірочне поле статусу завдання

(завершено чи ні), дату створення, кому призначено, хто призначив та два поля для роботи з часом, що забезпечують відстеження затраченого часу на завдання та обрахунок сумарного часу, який був використаний в процесі роботи.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	user_id	bigint								NULL
3	user_group_id	bigint								NULL

Рисунок 3.6 допоміжна таблиця вибору користувачів

На рис 3.6 зображена допоміжна таблиця, яка є доповненням таблиці, що представлена на рис 3.5. Її завдання полягає в фільтрації користувачів за групами до яких вони належать, що дає змогу при створення та призначені завдання здійснити вибірку користувача з поміж створених груп і списку користувачів які відносяться до конкретних груп.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
2	password	varchar (128)								NULL
3	last_login	datetime								NULL
4	is_superuser	bool								NULL
5	username	varchar (150)								NULL
6	is_staff	bool								NULL
7	is_active	bool								NULL
8	date_joined	datetime								NULL
9	first_name	varchar (30)								NULL
10	last_name	varchar (30)								NULL
11	email	varchar (254)								NULL
12	profile_picture	varchar (100)								NULL

Рисунок 3.7 таблиця «Авторизації та Реєстрації»

Таблиця на рис 3.7 є кастомізованою, використовується для збереження даних про облікові записи користувачів в системі.

Дану базу даних було створено з використанням міграції, роботу яких забезпечує фреймворк Django. Це дозволяє значно спростити процес розробки, так як моделі даних прописуються в окремих файлах проекту з використанням програмного коду.

3.4 Опис роботи програмного продукту

Розроблений веб-додаток для управління завданнями – це система, що значно спрощує процес виконання рутинних повсякденних справ, які потребують нагадування, а також надає можливість для управління командами та цілими відділами працівників. Значно підвищує ефективність та планування управління завданнями.

Для початку роботи з веб-додатком та ефективного управління завданнями, необхідно в будь-якому наявному браузері перейти по посиланню за яким знаходиться програмний продукт. Після переходу за адресою, користувача зустріне вікно авторизації в системі, у випадку, якщо облікового запису в системі ще не існує, є вкладка для проходження процедури реєстрації. Варто зазначити, що у випадку корпоративного використання додатку на власних серверах компанії, для забезпечення належної безпеки, процес реєстрації працівників в системі рекомендується здійснювати адміністратору або представнику відділу кібербезпеки.

У випадку успішної авторизації в системі, користувач отримує доступ до усього функціоналі, який надає веб-додаток для управління завданнями, що описано у пункті 2.3 Алгоритм роботи програмного продукту.

Після завантаження веб-додатку для управління завданнями користувачу будуть доступні форми, що представлені на рис. 3.1.

Результати розробки веб-додатку для управління завданнями за допомогою framework Django представлено на рис. 3.8 – 3.21.

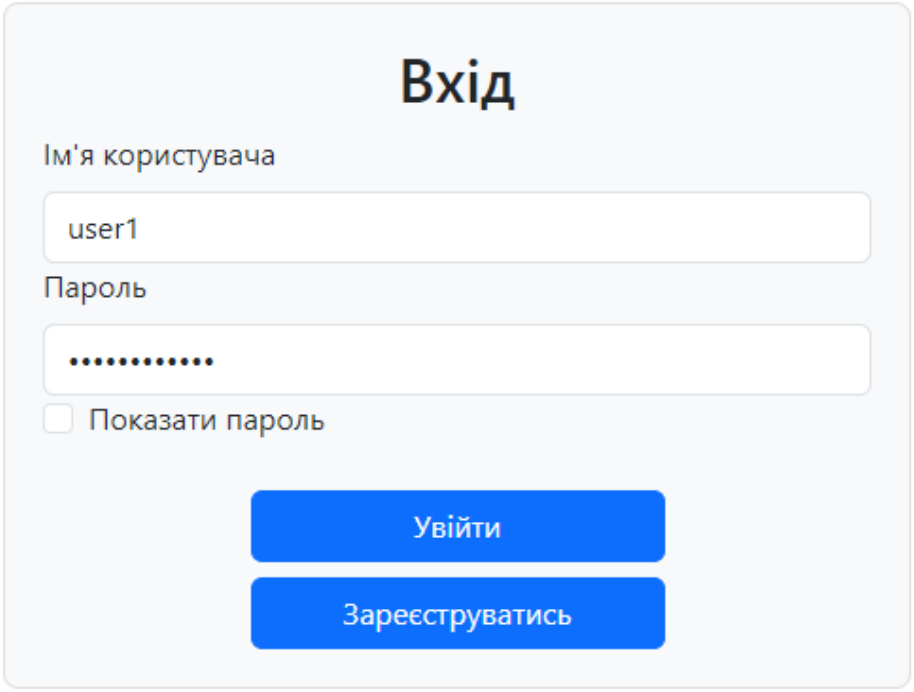


Рисунок 3.8 Сторінка авторизації користувача

На рис 3.8 представлена сторінка для здійснення авторизації користувача в системі. Вона має два поля, а саме: унікальне ім’я користувача, яка в подальшому використовуватиметься для його ідентифікації (username) та пароль. При цьому система здійснює перевірку в моделі CustomUser на наявність такого користувача та співставляє пароль з хешем збереженим в системі при реєстрації. Дана форма авторизації потрібна для можливості входу до ситеми, якщо обліковий запис існує.

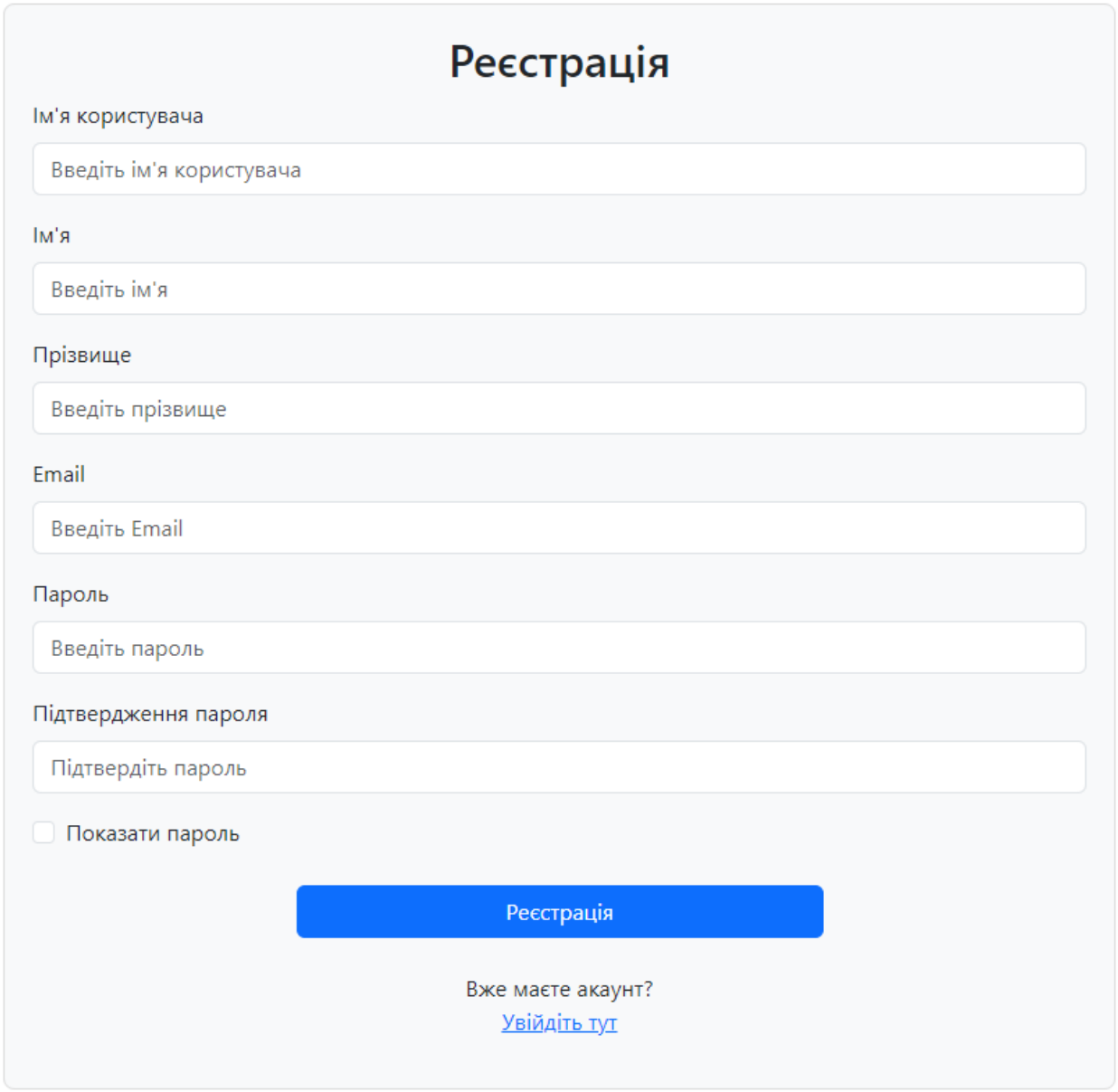


Рисунок 3.9 Сторінка реєстрації користувача

Рис 3.9 включає форму реєстрації нового користувача, вона є загальнодоступно і будь-який новий користувач може створити свій обліковий запис за допомогою неї, після чого пройти автентифікацію представлену на рис 3.8 та отримати доступ до функціоналу веб-додатку.

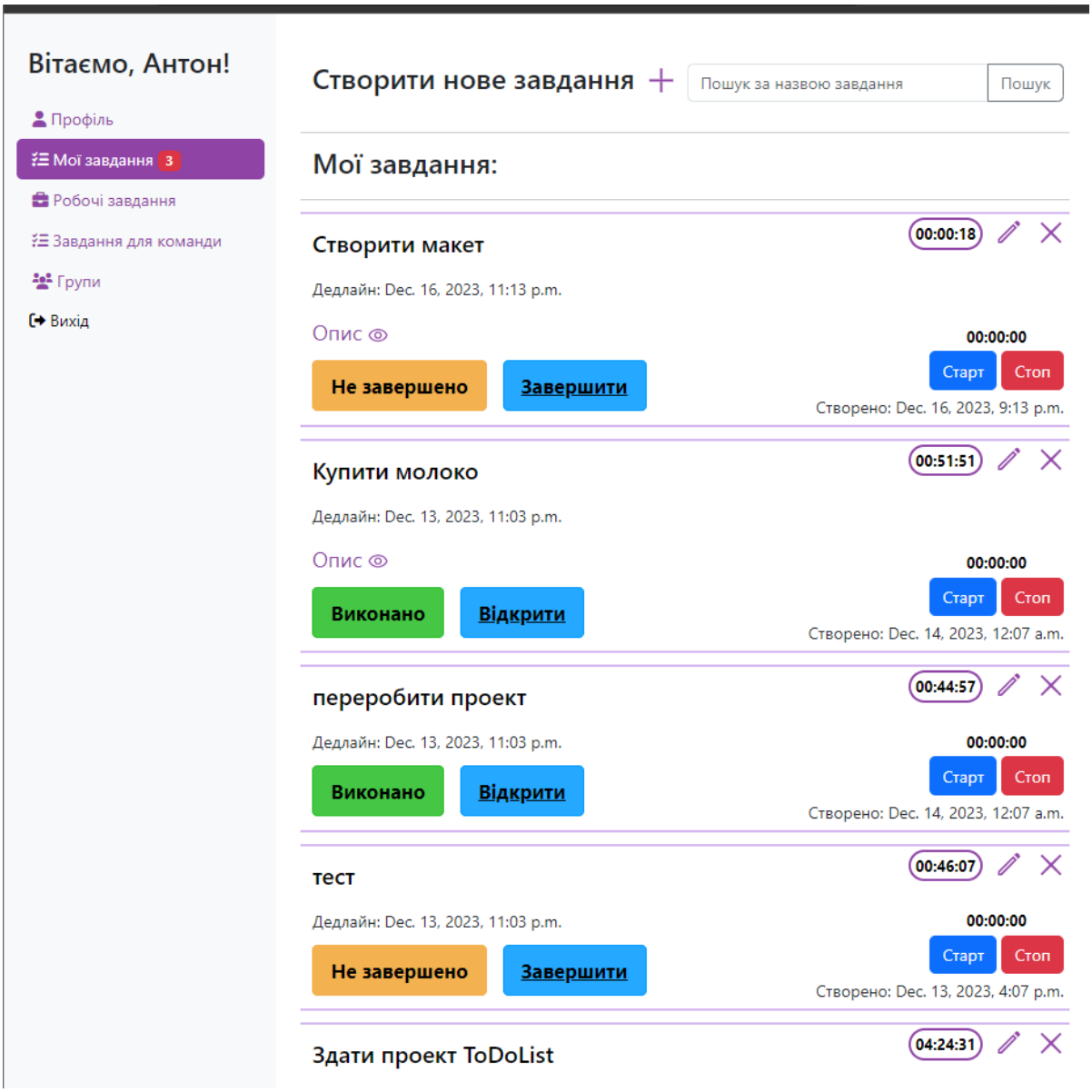


Рисунок 3.10 Сторінка «Мої завдання»

На рис 3.10 представлена сторінка «Мої завдання», вона стає доступна користувачеві після успішної авторизації в системі. Дана сторінка містить персональні завдання користувача, які він для себе створив. Надається функціонал для відстеження та управління завданнями, а саме:

- зміна статусу завдання виконано/не завершено з відповідним виділенням, що дозволяє легко розрізняти завдання;
- можливість перегляду опису завдання, якщо його додав користувач;
- встановлений дедлайн на виконання поставленого завдання;
- є функція редагування створених завдань;

- надана можливість видаляти завдання;
- здійснення пошуку за назвою завдання;
- відстеження дати створення завдання;
- присутність окремих таймерів для кожного завдання, що дозволяє відстежувати продуктивність та затрату часу на окреме завдання;
- додана функція створення нових завдань;
- присутній пагінатор, який розділяє сторінку у випадку, коли кількість завдань стає більше 6.

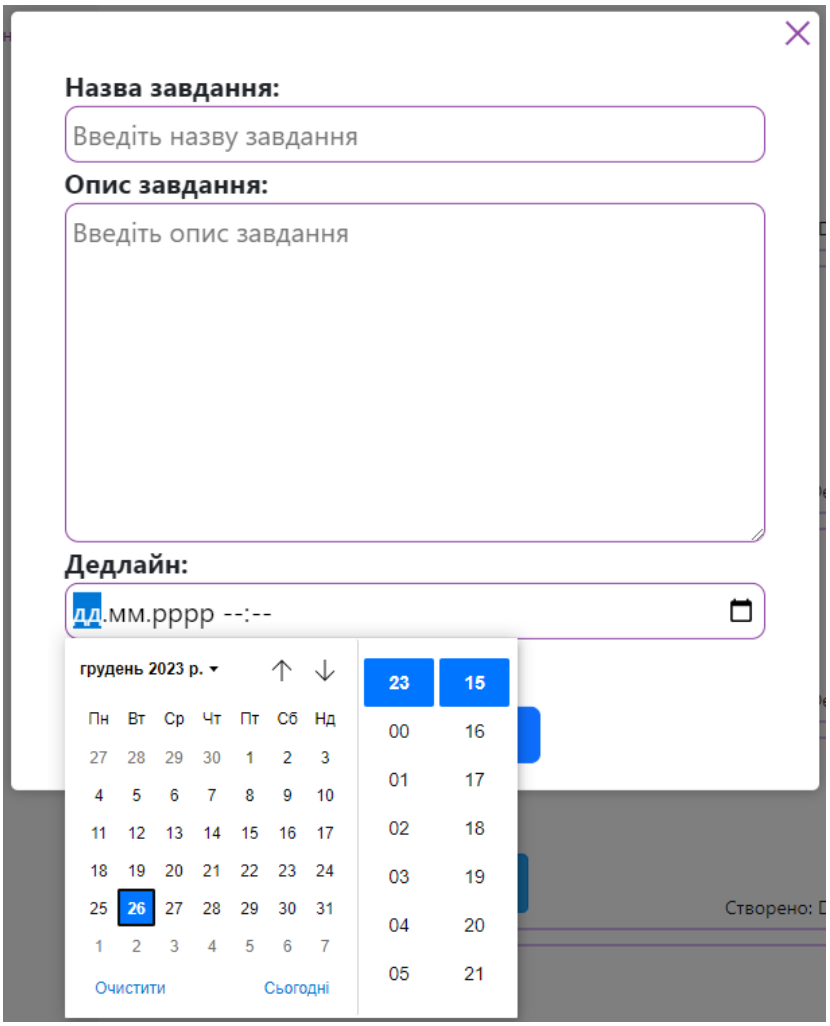


Рисунок 3.11 Попап-вікно для створення персонального завдання

На рис 3.11 зображене попап-вікно для створення персонального завдання користувача. Форма має поле для введення назви завдання, при потребі є не обов’язкове поле для введення опису, яке має обмеження у 700 символів. Також є поле дедлайну створене з використанням засобів HTML, яке надає зручний спосіб

вибору дати та часу до якого терміну потрібно виконати роботу, поле є обов'язковим для заповнення. Після внесення всіх даних користувач натискає на кнопку «Створити завдання» і воно додається до загального списку. Попап-вікно відкривається над всіма елементами і унеможливлює випадкову взаємодію користувача з елементами сторінки, доки форма не буде закрита або збережена.

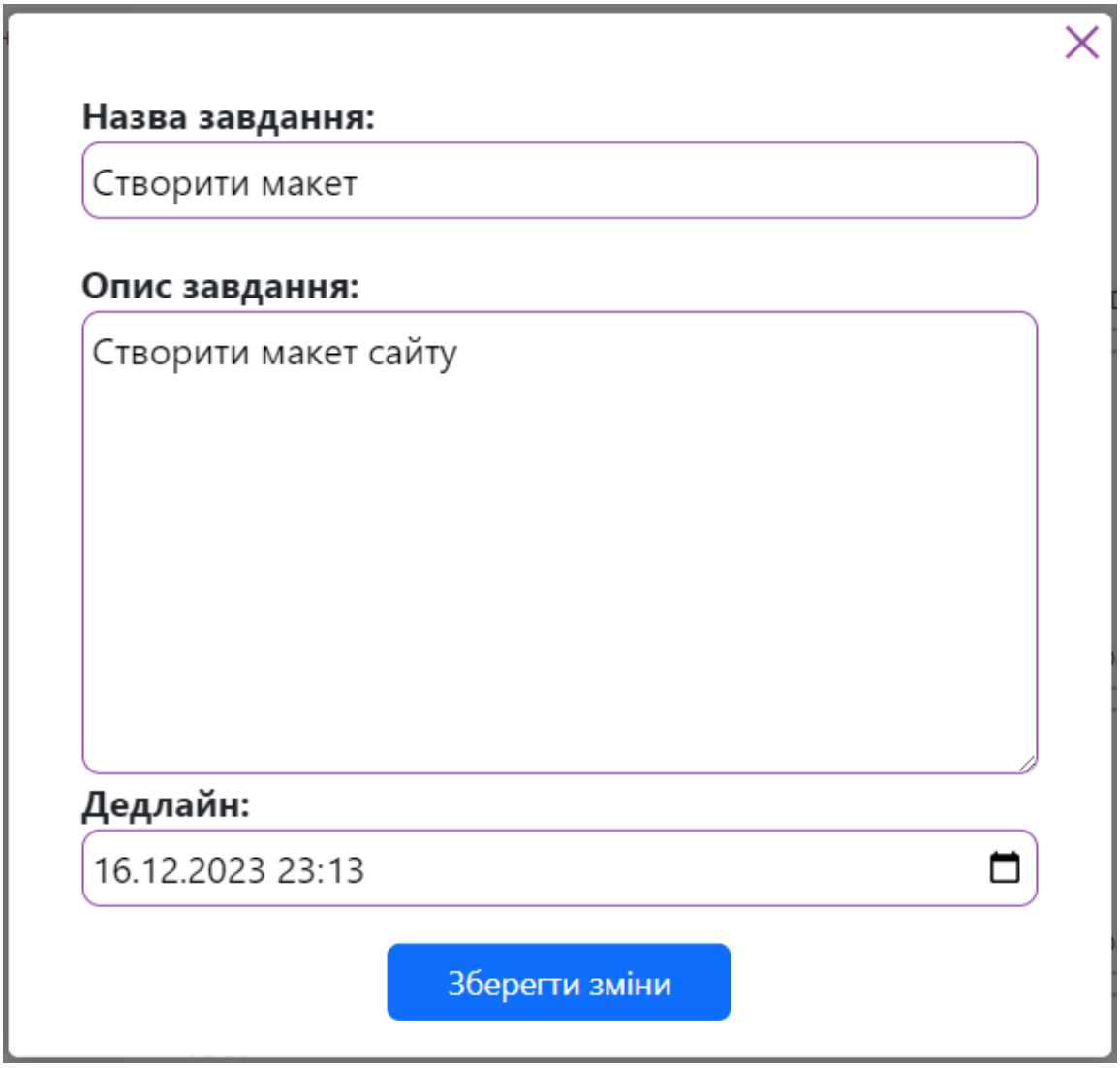


Рисунок 3.12 Попап-вікно для редагування персонального завдання

Форма для редагування завдання представлена на рис 3.12 дозволяє змінювати інформацію про завдання в усіх полях. Можна додавати або видаляти опис, змінювати повністю назву завдання та дедлайн. Після внесення та збереження змін дані по завдання оновлюються та відображаються у списку поряд з попередньо створеними.

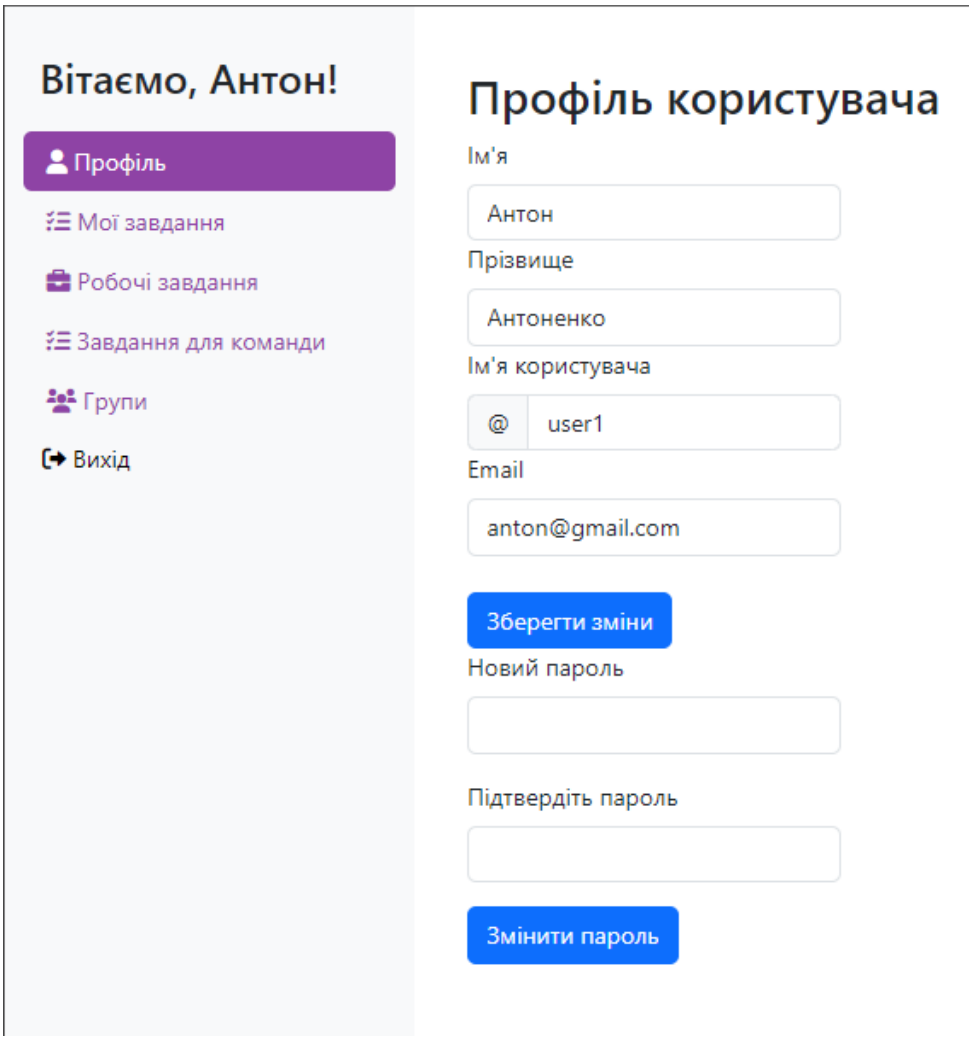


Рисунок 3.13 Вкладка «Профіль»

Рисунок 3.13 відображає вміст вкладки «Профіль». Дана вкладка представляє для користувача його дані облікового запису, відображається його ім’я, прізвище, унікальне ім’я користувача з яким взаємодіє система, електронна пошта. Усі ці дані власник може при необхідності самостійно зміни чи обновити. Також, дана вкладка передбачає можливість зміни актуального паролю користувача. По правилах безпеки, рекомендується змінювати пароль хоча б один раз на шість місяців. В подальшому планується покращити дану вкладку шляхом додавання функції відображення та зміни фото профіля користувача, а також додавання нового функціоналу, який би надавав можливість забороняти отримання завдань від других користувачів, які не є знайомими з власником акаунту та випадково або цілеспрямовано призначають йому певні завдання.

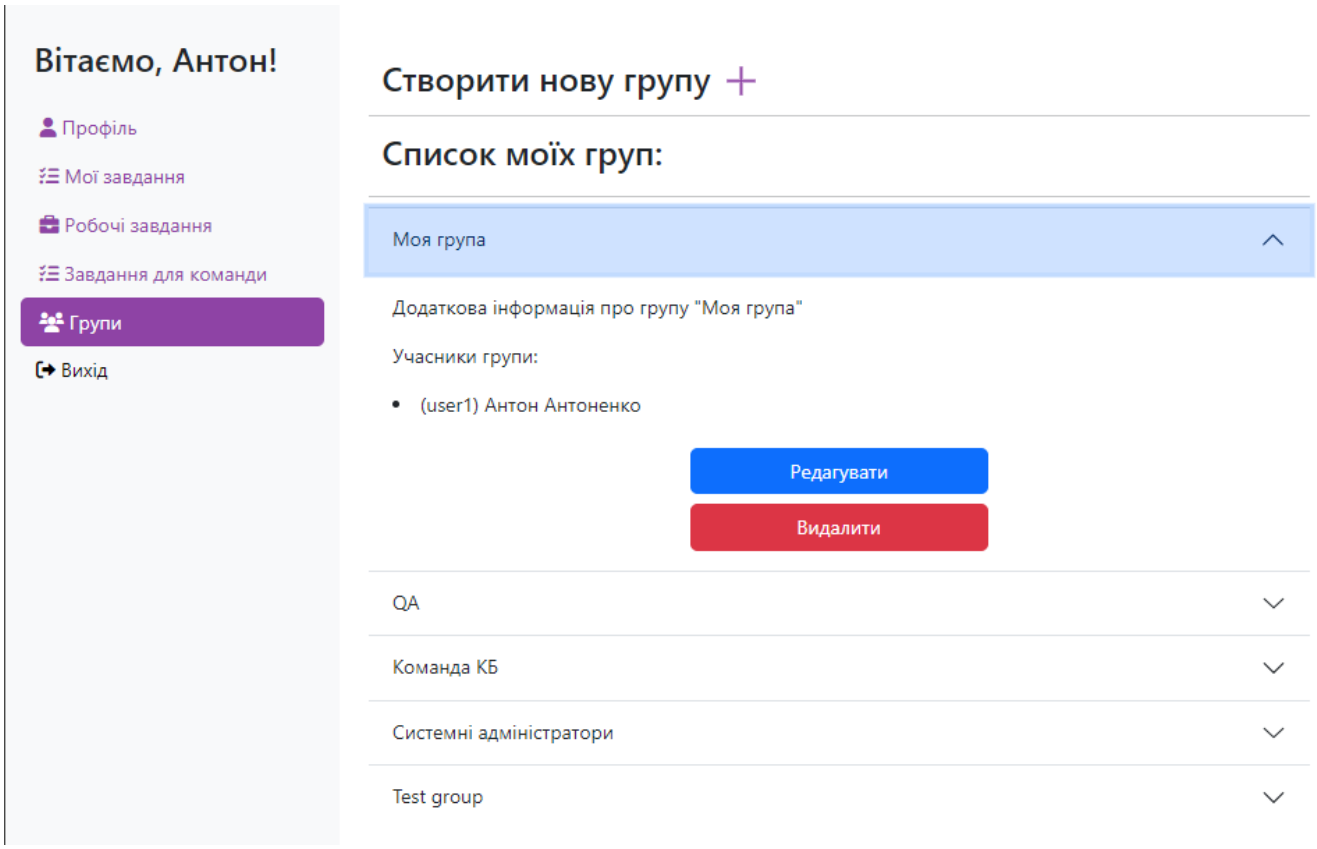


Рисунок 3.14 Вкладка «Групи»

На рисунку 3.14 представлено вкладку «Групи». Дана вкладка призначена та надає відповідний функціонал для створення нових груп користувачів і перегляду деталей про вже існуючі. Як можна побачити, надається список вже створених груп, який реалізовано засобами Bootstrap5 та надає зручний і ергономічний функціонал. Можна розгорнути групу та переглянути список її учасників, видалити чи відредагувати. Є можливість створити нову групу та додати туди учасників чи працівників проекту, відповідно до потреб. Ця вкладка буде особливо корисною для сортування користувачів по відділах чи розмежуванню учасників різних проектів за їх назвами. Завдяки цьому можна легко та в зручний спосіб відстежувати кількість працівників задіяних в роботі на певному проекті, бачити загальник список усіх проектів. Список груп та розподілені в них користувачів доступні лише власнику цього облікового запису, передбачається, що таку функцію використовуватимуть керівники проектів чи менеджери, які займаються структуруванням проектів та користувачів для подальшої взаємодії з ними. В подальшому, планується покращити даний функціонал зробивши групи

доступними не лише в рамках одного користувача, але і для керівників організації чи персоналу, який туди входить аби мати спільну та єдину базу проектів і користувачів, що на них задіяні. Такі зміни дозволять покращити організацію управління проектами всередині компанії.

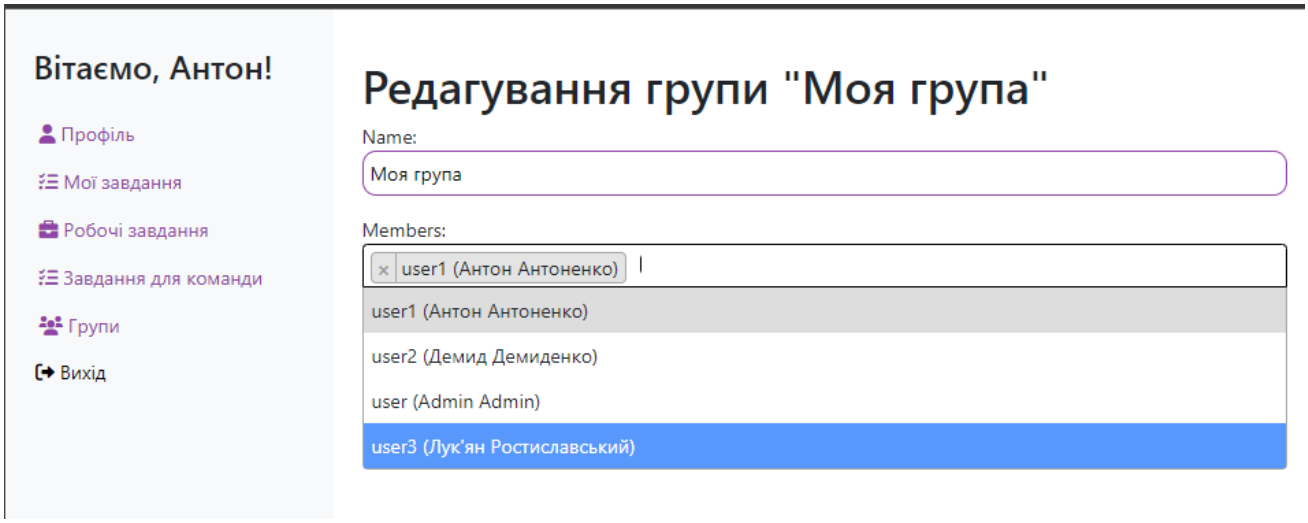


Рисунок 3.15 Функція редагування створеної групи

На рис 3.15 представлена функція редагування вже існуючих груп. Можна змінювати назву групи, додавати та видалити користувачів, а також здійснювати їх пошук з використанням унікальних username. Працездатність забезпечує віджет DjangoSelect2, який дозволяє обирати зі списку кілька користувачів. Кількість учасників групи не є жорстко обмеженою та дозволяє додавати потрібну кількість людей.

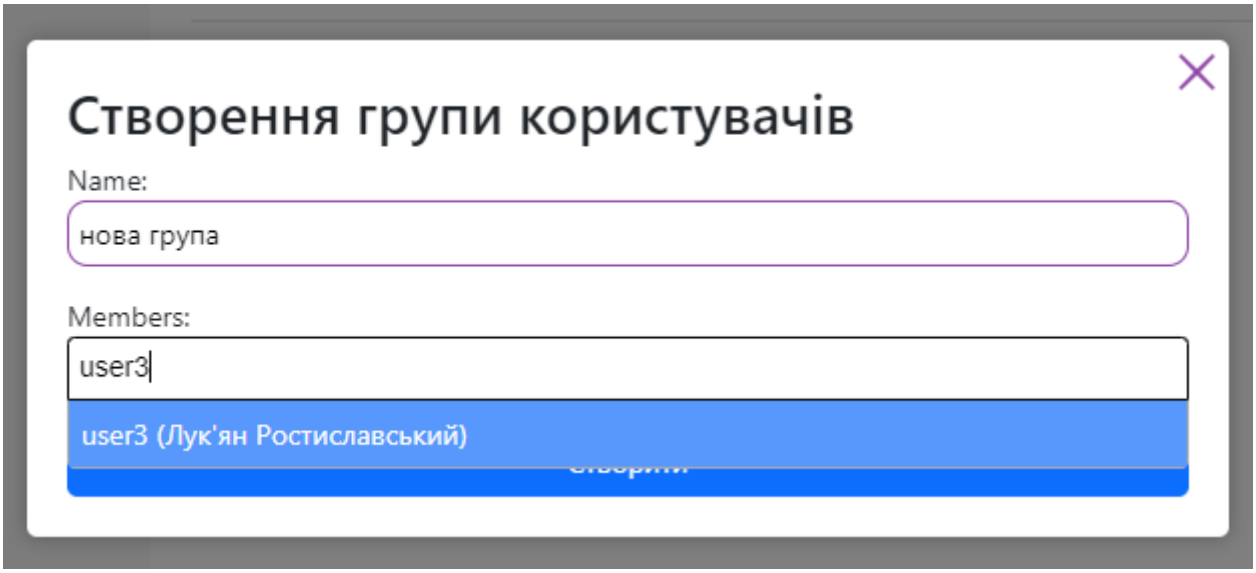


Рисунок 3.16 Створення нової групи

Рисунок 3.16 демонструє можливість створювати нову групу користувачів, якщо є потреба в цьому. Вказується назва групи та здійснюється пошук користувачів по їх username.

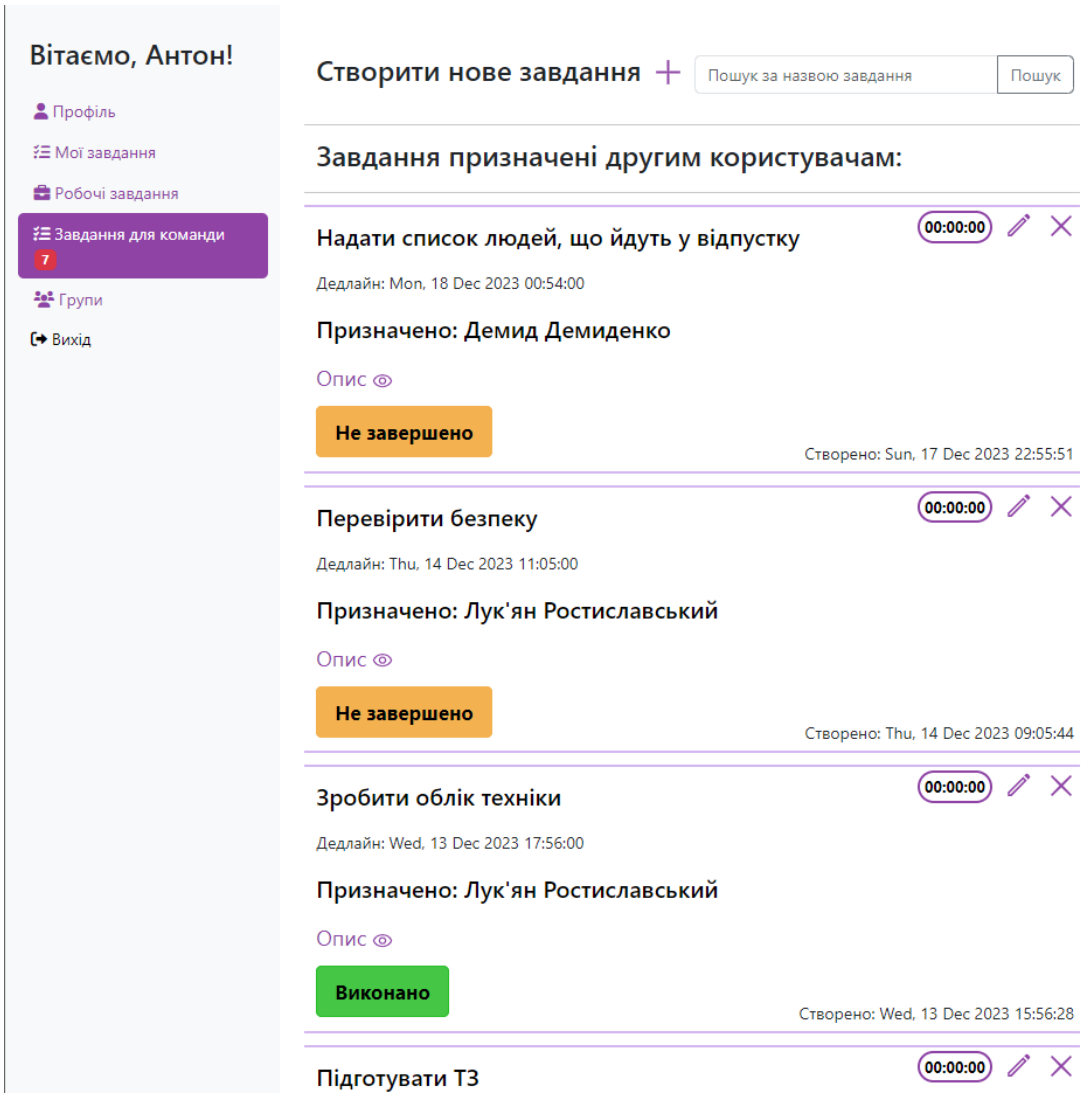


Рисунок 3.17 Вкладка «Завдання для команди»

На вкладці «Завдання для команди» представленій на рис 3.17 є доволі широкий функціонал, який забезпечує управління командними завданнями. До функцій, які надає дана вкладка для здійснення контролю виконання можна віднести:

- пошук завдань за назвою;
- створення нових завдання;
- перегляд опису завдань;
- відображення дедлайну;

- є показ дати створення завдання;
- можливість редагування завдань;
- видалення;
- перегляд статусу завдання: не завершено/виконано, що виділяється відповідними кольорами для кращого розуміння;
- відображення інформації про користувача, якому було призначене завдання;
- додається пагінатор, якщо кількість завдань більше шести;
- таймер для можливості відстеження часу виконання поставленого завдання, особливо буде корисним, якщо є робота на погодинну оплату.

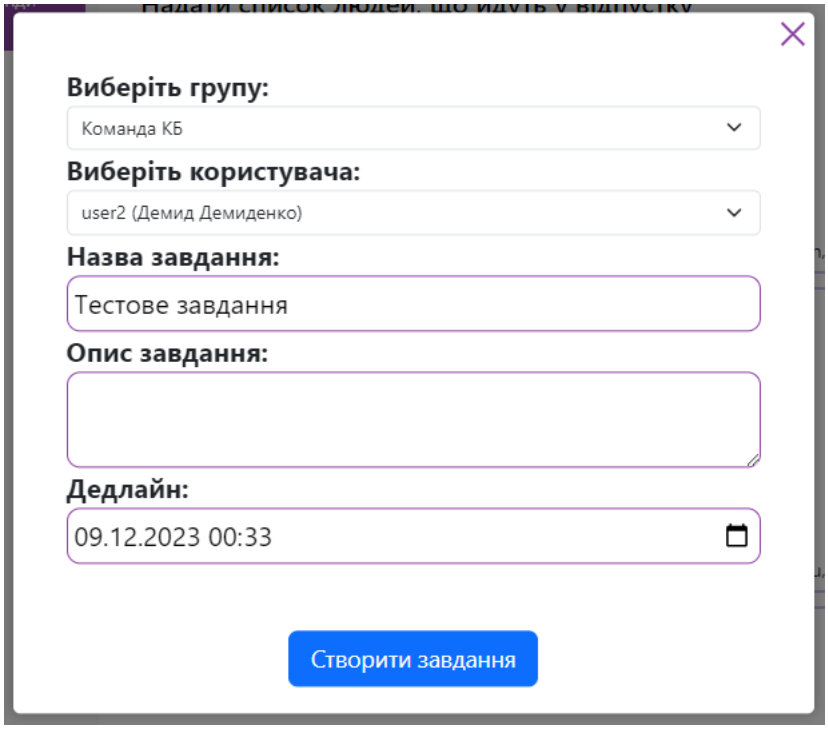


Рисунок 3.18 Створення завдання для учасника команди

Рисунок 3.18 демонструє форму створення завдання для учасника команди. Форма розміщена в попап-вікні та не дозволяє взаємодіяти з другими елементами системи допоки активна. Процес призначення завдання полягає у виборі користувача на основі групи до якої він входить на вкладці «Групи» (див.рис 3.14). Спочатку вибирається група зі списку наявних, далі пропонується вибір користувача зі списку тих, хто входить до цієї групи. Поля групи та користувача є обов’язковими, так дана функція передбачає саме командну роботу над

завданнями. Вказується назва завдання, опис (за потреби), дедлайн на виконання. завдання призначається вибраному користувачу та закріпляється за ним.

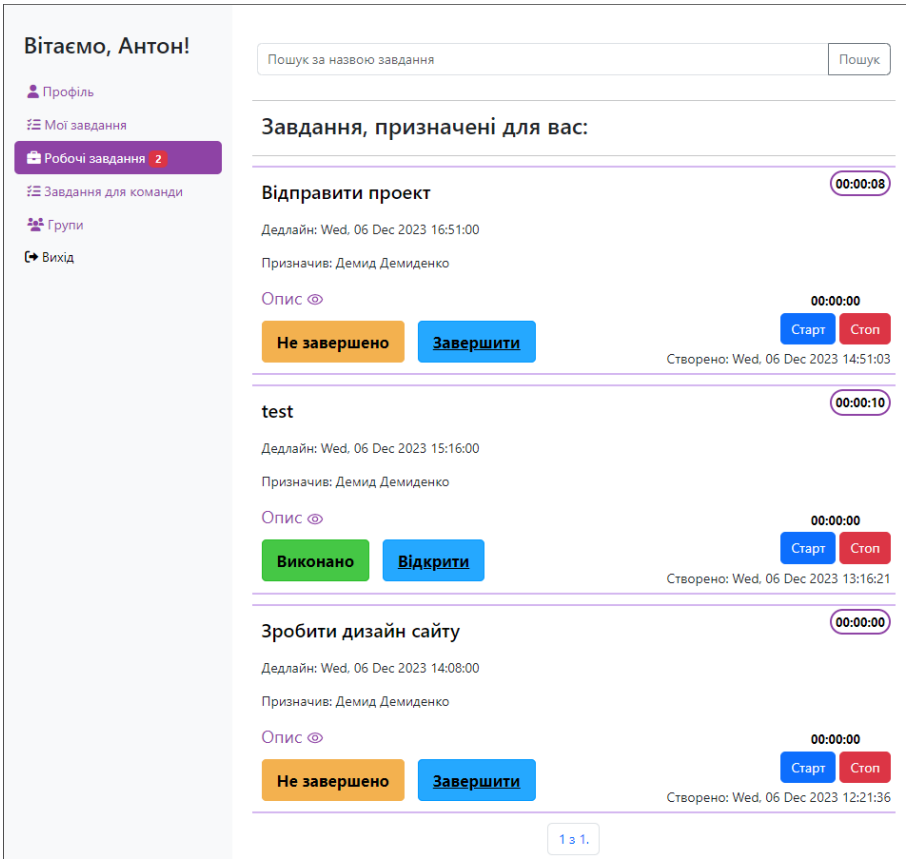


Рисунок 3.19 Вкладка «Робочі завдання»

Вкладка «Робочі завдання», що представлена на рис 3.19 відображає список завдань, які були призначені для користувача. Наприклад, колегою зі спільного проекту чи керівником відділу організації на яку працює даний користувач. Вкладка відображає необхідну інформацію про призначені завдання, а саме: назву завдань, опис (за наявності), дедлайн виконання, інформацію про того, хто призначив завдання, відображається інформація про дату створення завдання, додані кнопки для зміни статусу завдання, пошук за назвою завдання. Не менш важливою функцією на даній вкладці є наявність таймерів для кожного окремого завдання – це може бути корисно для відстеження своєї продуктивності виконання поточного завдання. В одне поле рахується сумарний час проведений над роботою та час конкретний, який нарахувався з моменту запуску таймера і до його зупинки. Дані таймерів, а також статус завдання відображаються користувачу, який створив це завдання. Таймер може добре згодитись не лише для відстеження своєї

швидкості роботи, але і для співпраці з клієнтами, коли присутня погодинна оплата. Це дозволить і користувачу і клієнту відстежувати потрачений час на виконання.

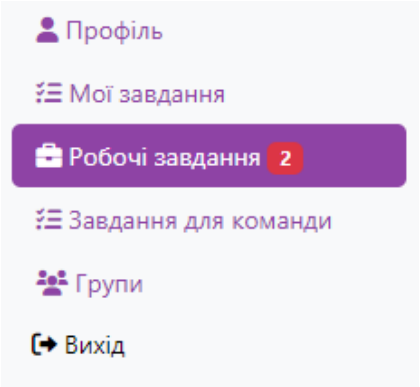


Рисунок 3.20 Меню веб-додатку

Меню веб-додатку представлене на рис 3.20 демонструє вкладки, які присутні на проекті, а також, лічильники на активних сторінках. Лічильники відображаються кількість завдань, які користувач ще не зміг завершити.

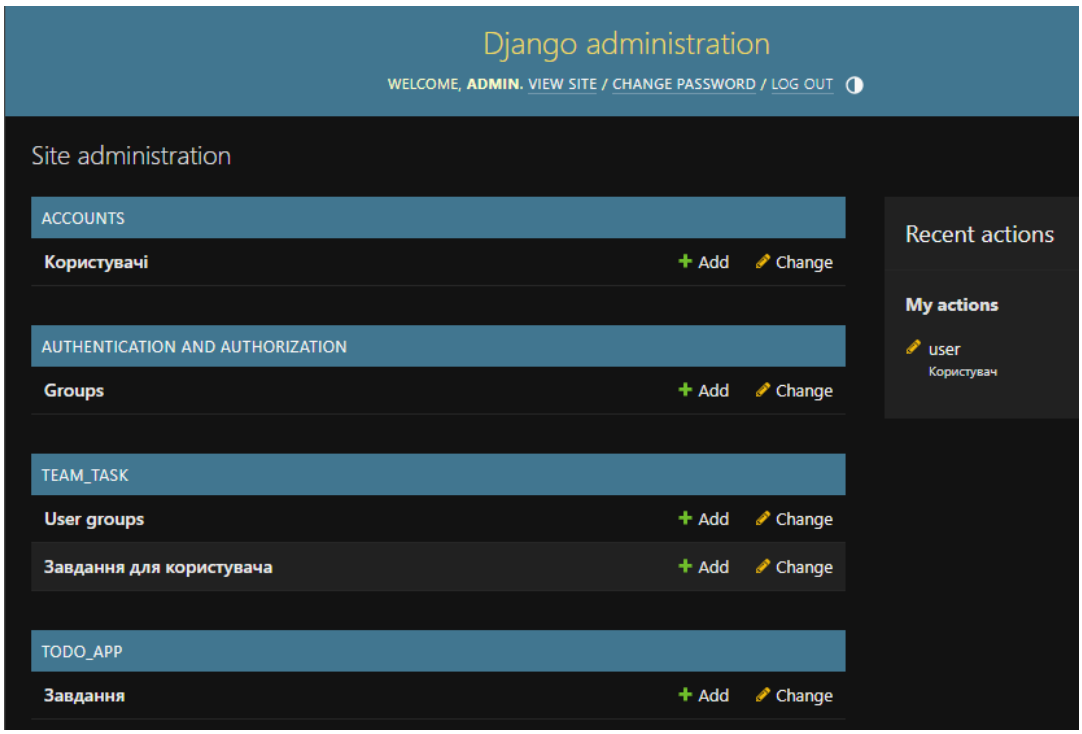


Рисунок 3.21 Адміністративна панель веб-додатку

Рисунок 3.21 відображає адміністративну панель веб-додатку для управління завданнями. Django надає готову панель та інструменти для управління веб-додатком, що значно полегшує розробки, а також дозволяє здійснювати контроль та підтримку роботи програмного продукту. У випадку корпоративного використання дозволяє керувати правами доступу користувачів.

ВИСНОВКИ

У першому розділі була проведена детальна характеристика та аналіз предметної області, включаючи дослідження актуальних аспектів, культури безпеки даних та аналіз існуючих рішень. Вибір підходів та технологій для розробки веб-додатку управління завданнями обґрунтовано врахуванням виявлених особливостей та вимог. Також проведено опис та принцип роботи протоколів HTTP, а також висвітлено особливості використання AJAX запитів.

У другому розділі була проведена робота з моделювання та проектування програмного продукту. Специфікація вимог визначає основні критерії для створення програмного продукту, а архітектурні рішення та алгоритм роботи встановлюють основні принципи його функціонування.

У третьому розділі було обґрунтовано вибір інструментальних засобів розроблення та розглянуто аспекти забезпечення безпеки даних у веб-додатку. Детально описано компоненти програмного продукту та його роботу, забезпечуючи чітке уявлення про функціонал системи.

Розроблений веб-додаток для управління завданнями на основі Django та Python підійде, як для використання користувачами в особистих цілях, так і для застосування та впровадження у середньому та малому бізнесі, державних організаціях чи командах початківців, що створюються спільні проекти. У порівнянні з конкурентами, веб-додаток є більш доступний для користувачів, а також надає безкоштовно основний функціонал, тоді як існуючі аналоги на ринку надають його за окрему плату, що не завжди підійде для використання малим та середнім бізнесом. Завдяки використанню Python та Django, веб-додаток має хороший захист від поширених кіберзагроз і є потужним інструментом для управління завданнями. Завдяки функціям відстеження часу роботи та управління завданнями для других користувачів, веб-додаток дозволяє спростити виконання робочих процесів, покращувати продуктивність та якість виконання роботи. Даний веб-додаток не є кінцевим варіантом та може бути покращеним з часом.

ПЕРЕЛІК ПОСИЛАНЬ

1. Task (project management) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Task_(project_management).

2. What is Cyber Security Culture and why does it matter for your organisation? [Електронний ресурс] – Режим доступу до ресурсу: https://www.paconsulting.com/insights/what-is-cyber-security-culture-and-why-does-it-matter-for-your-organisation.

3. Grubb S. How Cybersecurity Really Works: A Hands-On Guide for Total Beginners / Sam Grubb., 2021. – 216 с.

4. What Is Security Culture? [Електронний ресурс] – Режим доступу до ресурсу: https://www.knowbe4.com/security-culture.

5. Information security [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Information_security

6. Протоколи HTTP та HTTPS [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukraine.com.ua/uk/blog/seo-optimization/chto-takoe-https-i-zachem-on-nuzhen-kazhdomu-sajtu.html.

7. Що таке https? І навіщо він потрібен кожному сайту [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukraine.com.ua/uk/blog/seo-optimization/chto-takoe-https-i-zachem-on-nuzhen-kazhdomu-sajtu.html.

8. What is AJAX? [Електронний ресурс] – Режим доступу до ресурсу: https://testdriven.io/blog/django-ajax-xhr/.

9. Powell T. Ajax: The Complete Reference / Thomas Powell., 2008. – 654 с. – (McGraw Hill).

10. Modern Software Architecture Patterns: The Main Things to Know [Електронний ресурс] – Режим доступу до ресурсу: https://ideasoft.io/blog/modern-software-architecture-patterns/

11. Django and AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://devzone.org.ua/post/django-ajax

12. XHTML [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/XHTML

13. HTTP (Hypertext Transfer Protocol) [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol

14. Agile Method: Understanding the differences between stories, epics and initiatives [Електронний ресурс] – Режим доступу до ресурсу: https://www.bocasay.com/agile-method-understanding-the-differences-between-stories-epics-and-initiatives/.

15. Каскадна модель (waterfall model) [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/kaskadna-model-waterfall-model/.

16. Ітеративна модель (iterative model) [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/.

17. What Is Waterfall Methodology? Here’s How It Can Help Your Project Management Strategy [Електронний ресурс] – Режим доступу до ресурсу: https://www.forbes.com/advisor/business/what-is-waterfall-methodology/.

18. What Is Hybrid Project Management? [Електронний ресурс] – Режим доступу до ресурсу: https://www.wrike.com/blog/what-hybrid-project-management/.

19. Common web application architectures [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures.

20. Pattern: Monolithic Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://microservices.io/patterns/monolithic.html.

21. Software Architecture Types: Monolith vs Microservices [Електронний ресурс] – Режим доступу до ресурсу: https://apiko.com/blog/software-architecture-types-monolith-vs-microservices/.

22. Коли, чому та як здійснити перехід [Електронний ресурс] – Режим доступу до ресурсу: https://techukraine.net/%D0%BA%D0%BE%D0%BB%D0%B8-%D1%87%D0%BE%D0%BC%D1%83-%D1%82%D0%B0-%D1%8F%D0%BA-

%D0%B7%D0%B4%D1%96%D0%B9%D1%81%D0%BD%D0%B8%D1%82%D0%B8-%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D1%96%D0%B4/.

23. What is cloud architecture? [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/learn/what-is-cloud-architecture.

24. Web Application Security: The Ultimate Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.codica.com/blog/web-application-security/.

25. Security in Django [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/4.2/topics/security/.

26. What is Python? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/python/?nc1=h_ls.

27. Django (web framework) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Django_(web_framework).

28. Django Web Framework (Python) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django.

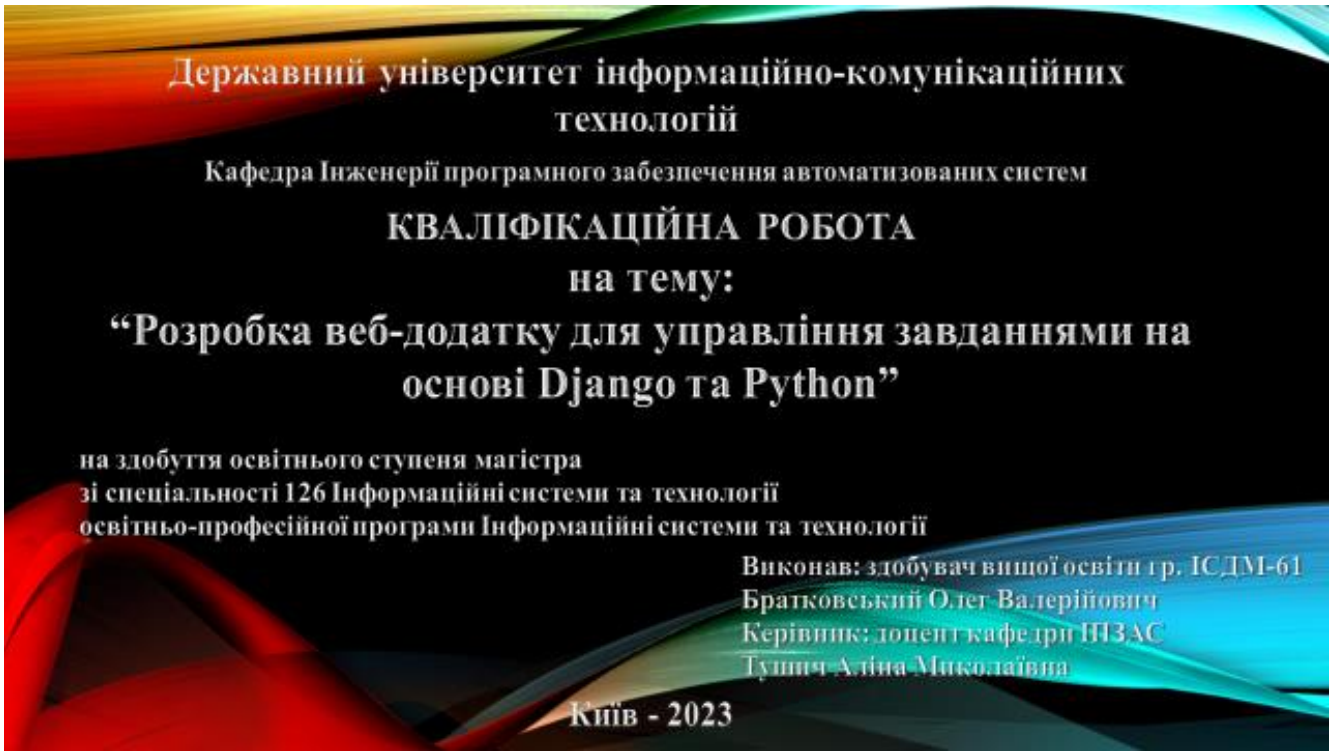
29. A Tutorial to Python for Web Development [Електронний ресурс] – Режим доступу до ресурсу: https://radixweb.com/blog/python-for-web-development.

30. Django Tutorial Part 8: User authentication and permissions [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication.

31. What Is Web Application Security? [Електронний ресурс] – Режим доступу до ресурсу: https://www.f5.com/glossary/web-application-security.

32. Django: how to pass the user object into form classes [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/django-how-to-pass-the-user-object-into-form-classes-ee322f02948c.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)



Інструментальні та програмні засоби для розробки веб-додатку управління завданнями на основі Django та Python

3

- ✓ Мова програмування – Python;
- ✓ Фреймворк – Django;
- ✓ Front-end частина: HTML, CSS, JavaScript, Bootstrap;
- ✓ СУБД – SQLite;
- ✓ Case-засоби – Enterprise Architect;
- ✓ Операційна система – Windows 10;
- ✓ Прикладне ПЗ: MS Word, Power point, Google Chrome, Visual Studio Code;
- ✓ Мови моделювання: SysML, UML;
- ✓ Технології: AJAX, jQuery, HTTP та розширена версія з врахуванням аспектів безпеки.

Безпека даних в Django

4

- ✓ Вбудований захист від SQL-ін'єкцій;
- ✓ Надійна автентифікація та авторизація на основі фреймворку сеансів;
- ✓ Хешування паролів;
- ✓ Захист від атак типу Cross-Site Scripting (XSS);
- ✓ Захист від атак типу CSRF;
- ✓ Захист від клікджекінгу.

```
<form method="post" action="{% url 'create_user_group' %}">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Створити групу</button>
</form>
```

Захист форм від CSRF атак.

Change Користувач

user1

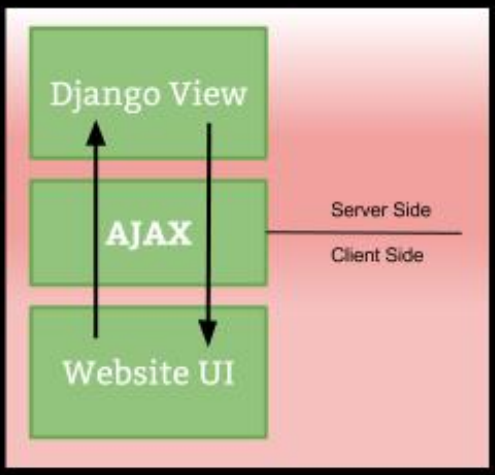
Password: pbkdf2_sha256$720000$UJbMi1UrYtvDNc

Паролі користувачів автоматично хешуються. що ускладнює злочинцям можливість проникнути в акаунт користувача, знаючи хеш неможливо проникнути в акаунт.

HTTP ТА AJAX ЗАПИТИ: РІЗНИЦЯ ТА ЗАСТОСУВАННЯ

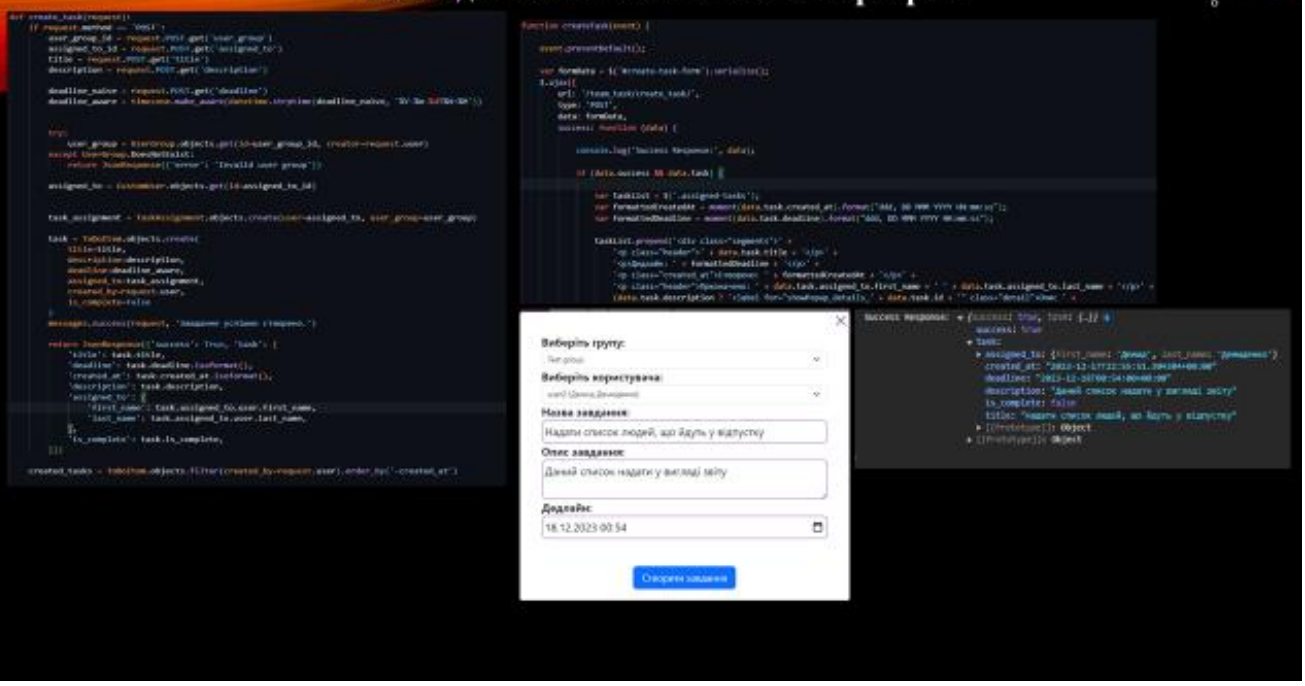
5

- ✓ AJAX дозволяє створювати більш динамічні та ефективні веб-сайти, забезпечуючи високу швидкість та зручність для користувачів;
- ✓ HTTP використовує синхронний підхід, де запит відправляється на сервер, а потім очікується відповідь перед відображенням результату. Призводить до перезавантаження сторінки при кожному запиті, що може затримати користувача та спричинити гальмування веб-сайту.

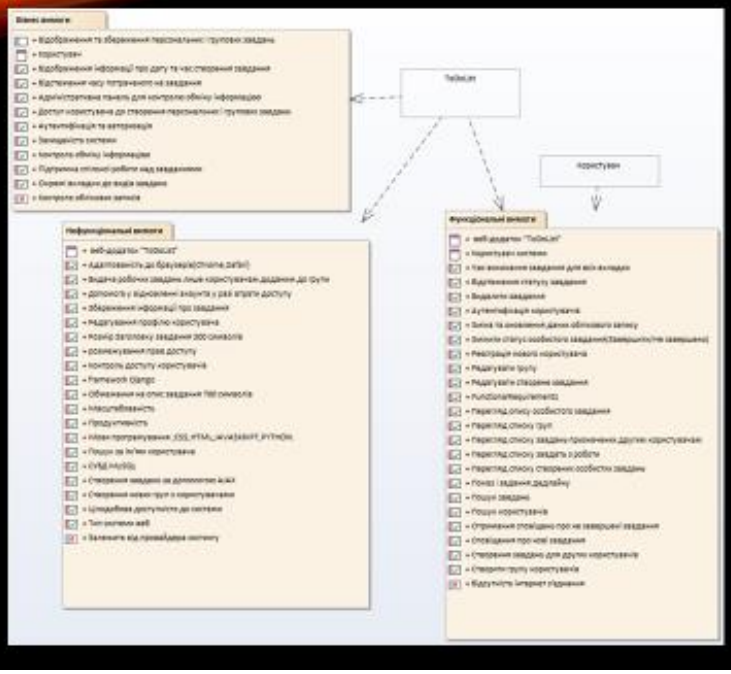


Обмін даними між клієнтом та сервером

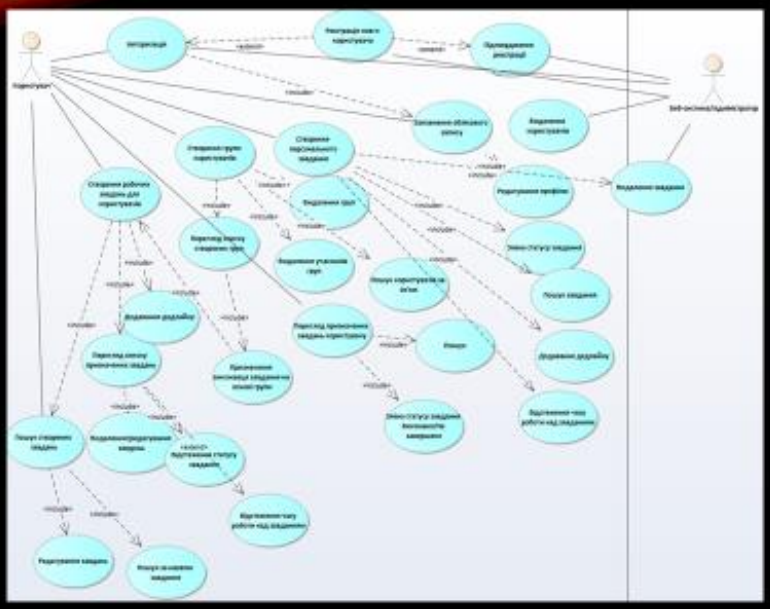
6



Для розробки веб-системи було створено вимоги:

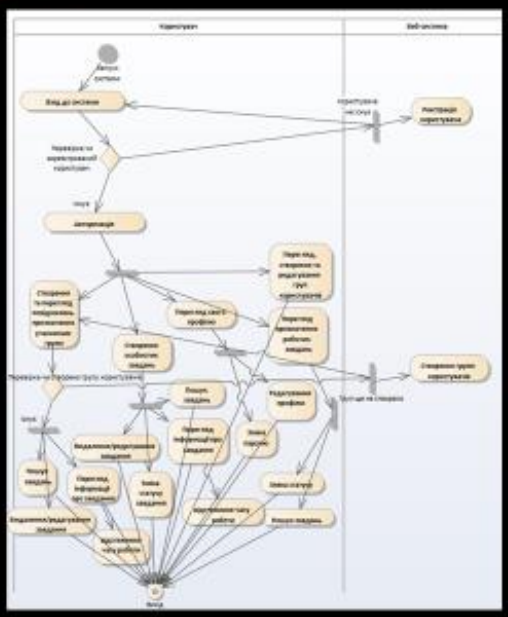


Функціональна модель системи



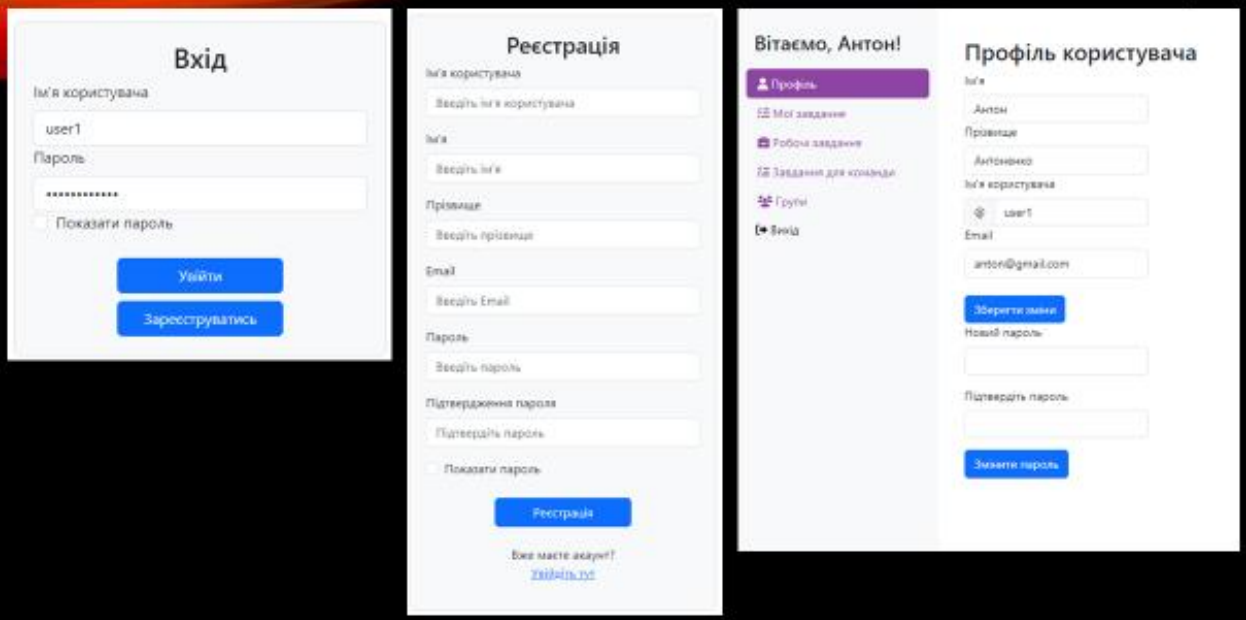
Алгоритм роботи веб-системи

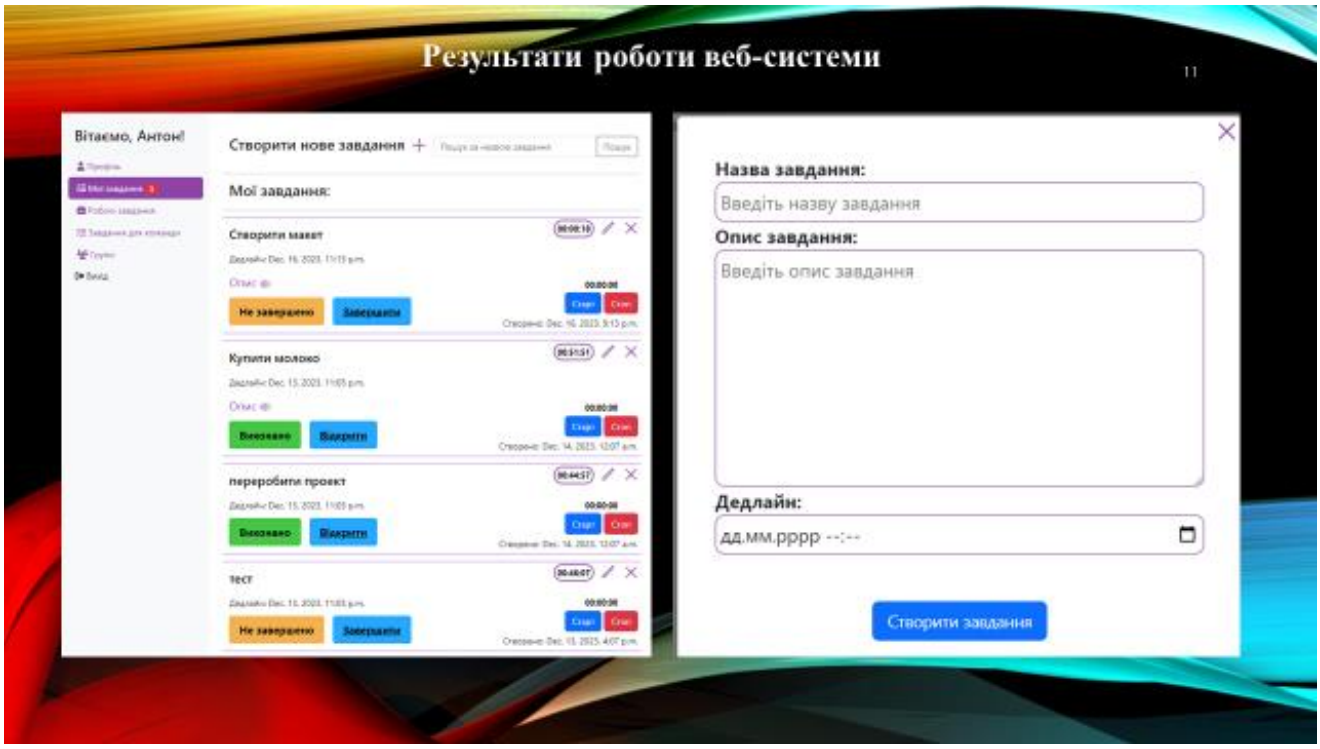
9

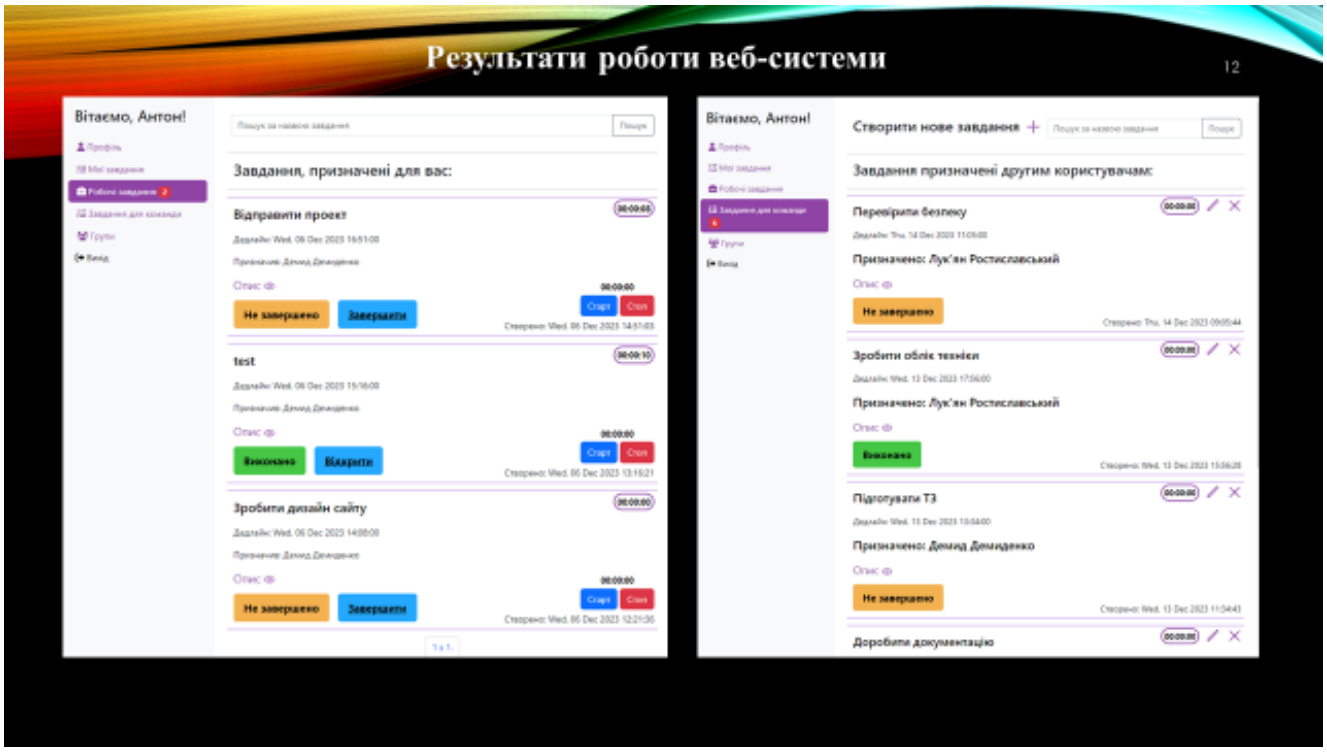


Результати роботи веб-системи

10

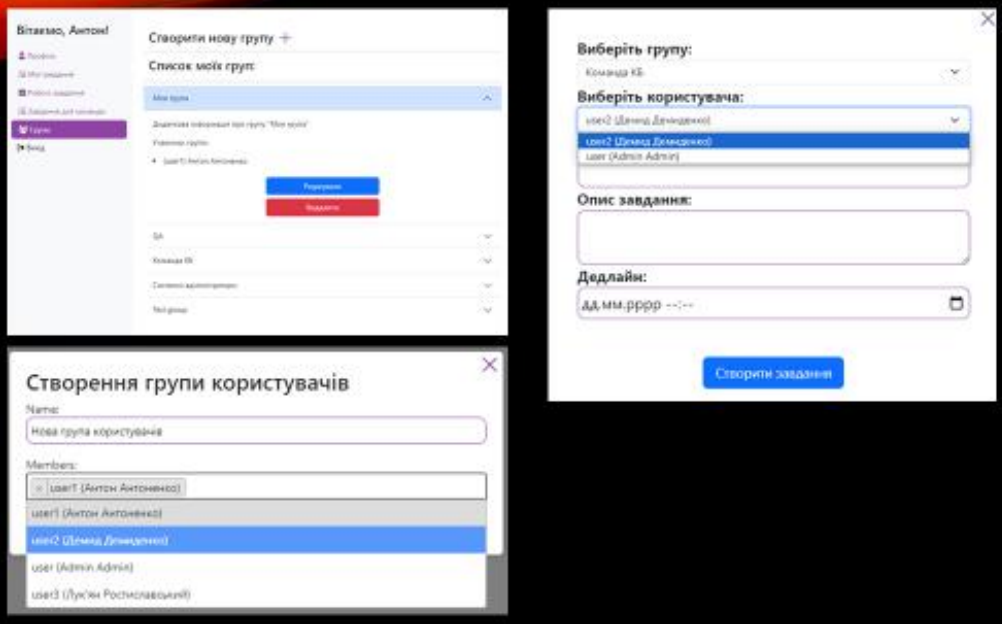



Результати роботи веб-системи
11
Вітаємо, Антон!
Створити нове завдання
Мої завдання:
Створити макет
Не завершено
Купити молоко
Виконано
Відкрити
переробити проект
Виконано
Відкрити
тест
Не завершено
Назва завдання:
Введіть назву завдання
Опис завдання:
Введіть опис завдання
Дедлайн:
дд.мм.рррр --:--
Створити завдання


Результати роботи веб-системи
12
Вітаємо, Антон!
Завдання, призначені для вас:
Відправити проект
Не завершено
test
Виконано
Відкрити
Зробити дизайн сайту
Не завершено
Вітаємо, Антон!
Створити нове завдання
Завдання призначені другим користувачам:
Перевірити безпеку
Призначено: Лук'ян Ростиславський
Не завершено
Зробити облік техніки
Призначено: Лук'ян Ростиславський
Виконано
Підготувати ТЗ
Призначено: Демид Демиденко
Не завершено
Доробити документацію

Результати роботи веб-системи

13

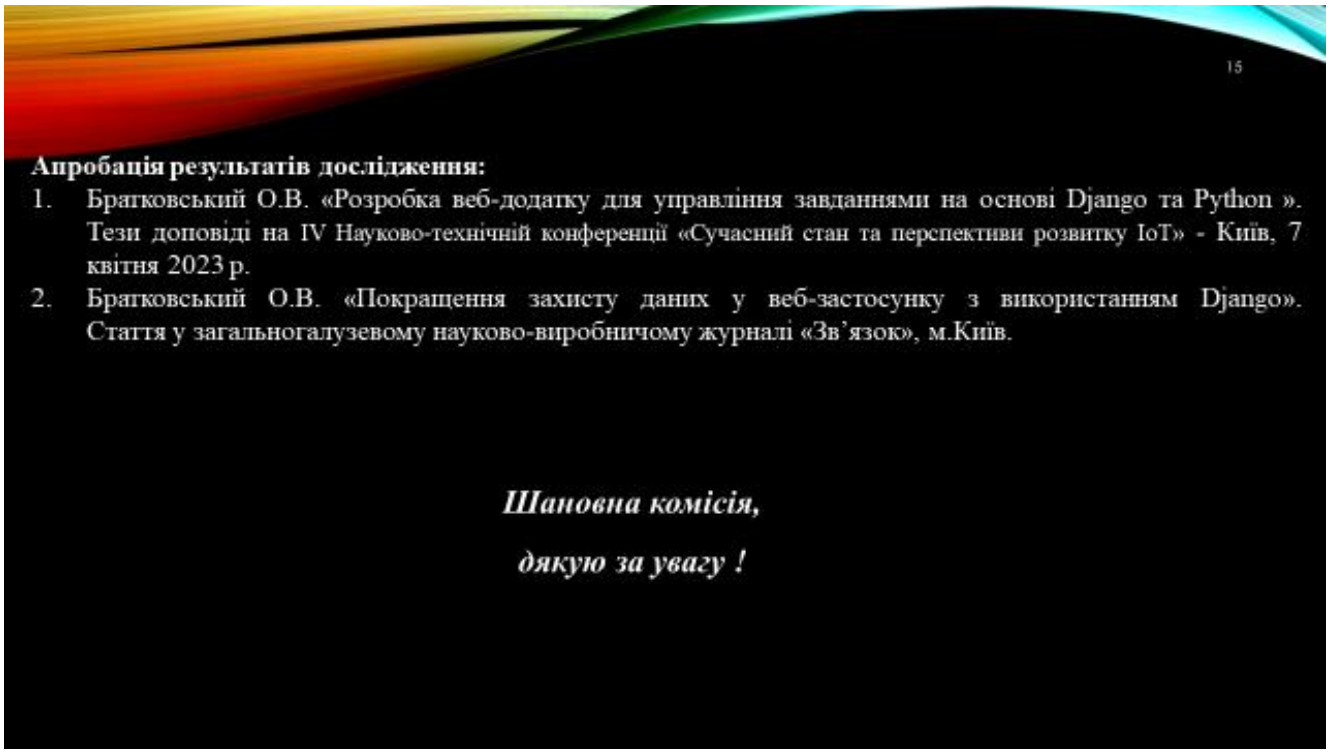


Висновки

14

- Проаналізовано сферу управління завданнями;
- Було здійснено аналіз існуючих рішень на ринку;
- Проаналізовано основні правила культури безпеки та забезпечення безпеки веб-додатків;
- Проаналізовано основні моделі розробки;
- Досліджено та створено діаграми вимог та варіантів використання веб-додатку;
- Розроблено алгоритм використання додатка;
- Здійснено аналіз існуючих архітектурних рішень для розробки проекту;
- Досліджено основні технології веб-розробки для реалізації додатка;
- Розроблено веб-додаток для управління завданнями на основі Django та Python.

- Веб-додаток є універсальним інструментом, що спрощує рутинні завдання для кожного користувача та стає затребуваним у розвитку і досягненні успіху в бізнесі, управлінні проектами чи в повсякденному керуванні особистими справами. Завдяки своїй гнучкості, з часом може бути легко модифікований відповідно до вимог бізнес-середовища.


15
Апробація результатів дослідження:
1. Братковський О.В. «Розробка веб-додатку для управління завданнями на основі Django та Python ». Тези доповіді на IV Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT» - Київ, 7 квітня 2023 р.
2. Братковський О.В. «Покращення захисту даних у веб-застосунку з використанням Django». Стаття у загальногалузевому науково-виробничому журналі «Зв'язок», м.Київ.
Шановна комісія,
дякую за увагу !