

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

Веб-сайт «Онлайн органайзер»

Виконав студент групи ІПЗ-20-3
СІНКЕВИЧ Микола Антонович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна Вакалюк

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СІНКЕВИЧ Микола Антонович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **Веб-сайт «Онлайн органайзер»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **використання веборієнтованих технологій для розробки, проектування та реалізація вебсайту для створення і редагування завдань**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю. М.	27.02.2024р.	27.02.2024р.
2	Єфремов Ю. М.	27.02.2024р.	27.02.2024р.
3	Єфремов Ю. М.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебсайту онлайн органайзеру та пояснювальної записки. Пояснювальна записка до випускної роботи містить 55 сторінки, 26 ілюстрацій, 3 таблиці, 3 додатки та 24 літературних джерел.

Метою випускної кваліфікаційної роботи є розробка вебсайт, що дозволить користувачу ефективно керувати своїм часом. Вебсайт буде створено на основі фреймворку Django, що забезпечить високу продуктивність, масштабованість та безпеку системи. Сайт буде спрямований на економію часу користувачу.

Інтерфейс вебсайту буде інтуїтивно зрозумілим і зручним для користувачів різного рівня технічної підготовки. Він забезпечить швидкий доступ до основних функцій, таких як створення нотаток, їх редагування, видалення і відновлення

КЛЮЧОВІ СЛОВА: ОРГАНАЙЗЕР, ВЕБСАЙТ, DJANGO, СТВОРЕННЯ, РЕСУРС.

					ІПЗ.КР.Б-121-24-ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Веб-сайт «Онлайн органайзер»	Літ.	Арк.	Аркуші	
Розроб.		Сінкевич М.А.							
Керівник		Єфремов Ю. М.					3	63	
						Житомирська політехніка, група ІПЗ-20-3			
Н. контр.									
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

The graduation thesis of the bachelor consists of the website of the online organizer and an explanatory note. The explanatory note to the final thesis contains 55 pages, 26 illustrations, 3 tables, 3 appendices and 24 literary sources.

The goal of the final qualification work is to develop a web application that will allow the user to effectively manage his time. The web application will be built on the basis of the Django framework, which will ensure high performance, scalability and security of the system. The application will be aimed at saving the user's time.

The interface of the web application will be intuitive and convenient for users of different levels of technical training. It will provide quick access to basic functions such as creating, editing, deleting and restoring notes

KEY WORDS: ORGANIZER, WEB APPLICATION, DJANGO, CREATION, RESOURCE.

					ИПЗ.КР.Б-121-24-ИЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	8
1.1 Постановка задачі	8
1.2 Аналіз аналогів програмного продукту	9
1.3 Визначення архітектури вебсайту.....	15
1.4 Обґрунтування вибору інструментальних засобів.....	19
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБСАЙТУ «ОНЛАЙН ОРГАНАЙЗЕР».....	22
2.1 Визначення варіантів використання та структури системи	22
2.2 Розробка бази даних.....	25
2.3 Проектування та реалізація алгоритмів роботи сайту	27
2.4 Реалізація функціоналу сайту	29
Висновки до другого розділу	36
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН ОРГАНАЙЗЕРОМ.....	38
3.1 Порядок встановлення та налаштування параметрів сайту	38
3.2 Структура інтерфейсу вебсайту.....	39
3.3 Тестування роботи вебсайту	51
Висновки до третього розділу	54
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56
ДОДАТКИ.....	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CSS – Cascading Style Sheets

HTML – Hypertext Markup Language

IDE – Integrated Development Environment

JS – JavaScript

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6

ВСТУП

Актуальність теми. У сучасному світі інформаційні технології проникають у всі сфери життя, забезпечуючи зручність, ефективність та швидкість виконання різноманітних завдань. Однією з актуальних проблем є організація особистого та робочого часу, що часто потребує використання багатьох інструментів та методів. В умовах швидкоплинного інформаційного потоку, важливість наявності зручного та доступного інструменту для планування стає очевидною. Вебсайт онлайн органайзер є саме таким інструментом, який дозволяє користувачам ефективно керувати своїм часом, завданнями та проектами в одному місці, що значно підвищує продуктивність та організованість.

Метою кваліфікаційної роботи є розробка та впровадження вебсайту онлайн органайзера, який надаватиме користувачам можливість зручно та ефективно планувати свій час, керувати завданнями та проектами, а також забезпечить доступ до необхідних функцій з будь-якого пристрою, підключеного до Інтернету. Основною метою є створення функціонального, інтуїтивно зрозумілого та естетично привабливого інструменту для щоденного використання.

Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз вимог до функціональності додатку, враховуючи потреби користувачів.
2. Здійснити вибір засобів проектування.
3. Розробити вебсайт онлайн органайзер.

Об'єктом дослідження є процеси організації та управління особистим і робочим часом користувачів в умовах сучасного інформаційного суспільства. Розглядаються методи та інструменти, які допомагають оптимізувати ці процеси.

Предметом дослідження є розробка та впровадження вебсайту онлайн органайзера, який дозволяє користувачам ефективно планувати свій час, керувати завданнями та проектами.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Планувальник завдань — це комп'ютерна програма для керування виконанням завдань у фоновому режимі без нагляду. Це зазвичай називають пакетним плануванням, оскільки виконання неінтерактивних завдань часто називають пакетною обробкою, хоча традиційне завдання та пакет розрізняють і протиставляють; подробиці дивіться на цій сторінці. Інші синоніми включають пакетну систему, систему управління розподіленими ресурсами (DRMS), менеджер розподілених ресурсів (DRM) і, як правило, сьогодні автоматизацію робочого навантаження (WLA). Структура даних завдань для виконання називається чергою завдань.

Сучасні планувальники завдань зазвичай забезпечують графічний інтерфейс користувача та єдину точку керування для визначення та моніторингу виконання у фоновому режимі в розподіленій мережі комп'ютерів. Від планувальників завдань все частіше вимагається організувати інтеграцію бізнес-діяльності в реальному часі з традиційною фоновією ІТ-обробкою на різних платформах операційних систем і середовищах бізнес-сайтів

Метою даного проекту є розробка зручного та ефективного онлайн органайзера, який допоможе користувачам планувати свій робочий час, управляти задачами та координувати свої дії.

Для досягнення мети проекту необхідно виконати наступні задачі:

- Розробити інтуїтивно зрозумілий інтерфейс користувача.
- Забезпечити можливість створення, редагування та видалення задач і подій.
- Реалізувати функцію спільного використання задач між користувачами.
- Інтегрувати органайзер з популярними календарями та іншими інструментами.
- Забезпечити високу продуктивність та надійність системи.

Результатом проекту стане функціональний прототип онлайн органайзера, що відповідає всім зазначеним вимогам, а також його тестування і аналіз ефективності. Буде створена документація для подальшого розвитку та вдосконалення проекту.

1.2 Аналіз аналогів програмного продукту

Сьогоднішній ритм життя вимагає від нас більшої організованості та управління часом. У зв'язку з цим розвиток онлайн-органайзерів стає все більш актуальним. Онлайн-органайзер - це програма, вебдодаток або вебсайт що дозволяє користувачам планувати свій час, встановлювати завдання та пріоритети, а також вести список справ. Розглянемо деякі з найпопулярніших онлайн-органайзерів на сьогоднішній день, а саме:

- Google Calendar
- Microsoft Outlook
- Todoist
- Trello

Аналіз існуючих програмних продуктів є важливим етапом при розробці нового програмного забезпечення. Він дозволяє визначити ринкові стандарти, уникнути повторення чужих помилок і знайти можливості для інновацій.

- Для аналізу аналогів були визначені наступні критерії:
- Функціональність
- Зручність користування
- Інтеграція з іншими сервісами
- Мобільність
- Продуктивність і надійність
- Відгуки користувачів та репутація

Перший аналог це Google Календар . Google Календар - один з найбільш відомих та популярних онлайн органайзерів. Він дозволяє створювати події, запрошувати учасників, встановлювати нагадування та додавати завдання. Крім

того, Google Календар інтегрований з іншими сервісами Google, що дозволяє легко організовувати свій час.

До основних переваг цього сервісу можна віднести:

- Створення та редагування подій, нагадування, спільне використання календарів, інтеграція з Gmail.
- Інтуїтивно зрозумілий інтерфейс, легкість навігації.
- Глибока інтеграція з іншими сервісами Google.
- Додатки для Android та iOS.
- Висока надійність і швидкодія.
- Безкоштовний з обмеженими функціями, додаткові функції доступні в рамках Google Workspace.

Головна сторінка календаря з відображенням всіх подій(рис1.1).

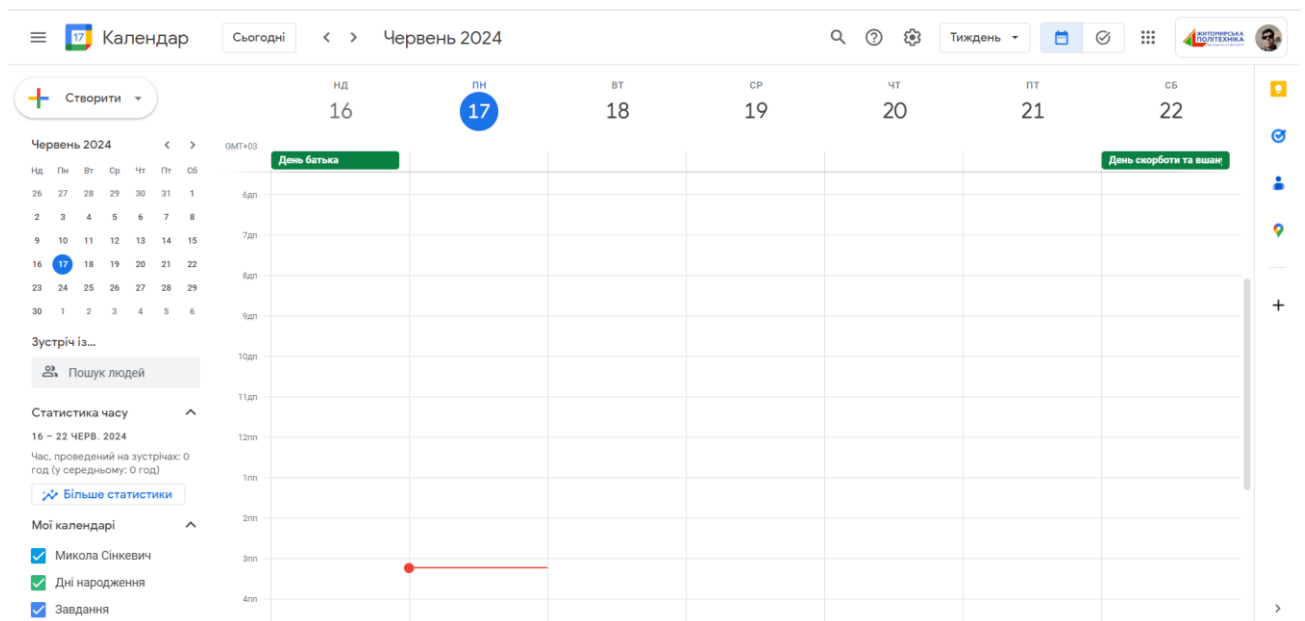


Рисунок 1.1 – Головна сторінка додатку Google Календар

У наступному розділі додатку користувач може додати своє завдання у календар давши їй назву. При створенні завдання доступно багато функцій, таких як: додавання опису завдання, прикріплення відеоконференції і місця події з подальшим сповіщенням у назначений час(рис1.2).

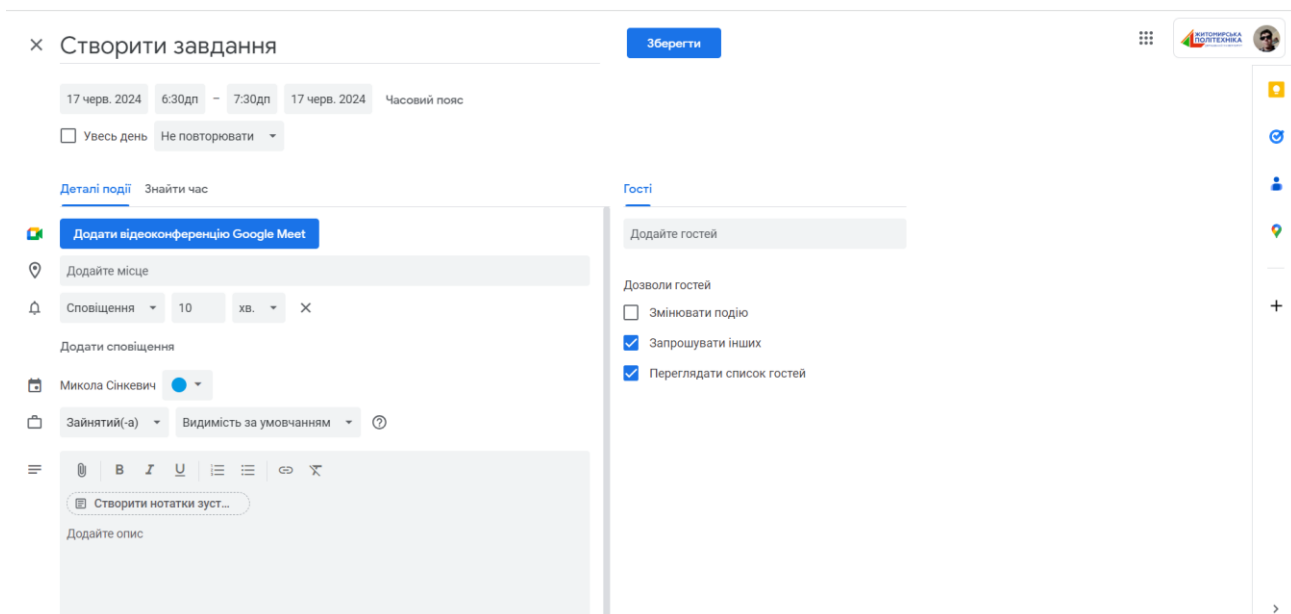


Рисунок 1.2 – Сторінка створення завдання у Google Календар з усіма її функціями

Крім цього є блоки зі статистикою, створенням нових календарів для інших потреб і переглядом календарів інших користувачів, які надали вам доступ.

Microsoft Outlook Календар - це ще один популярний онлайн-органайзер, особливо серед користувачів, які використовують Outlook для управління електронною поштою. Він має схожі можливості з Google Календарем, такі як створення подій та завдань, нагадування та спільний доступ до календаря(рис1.3).

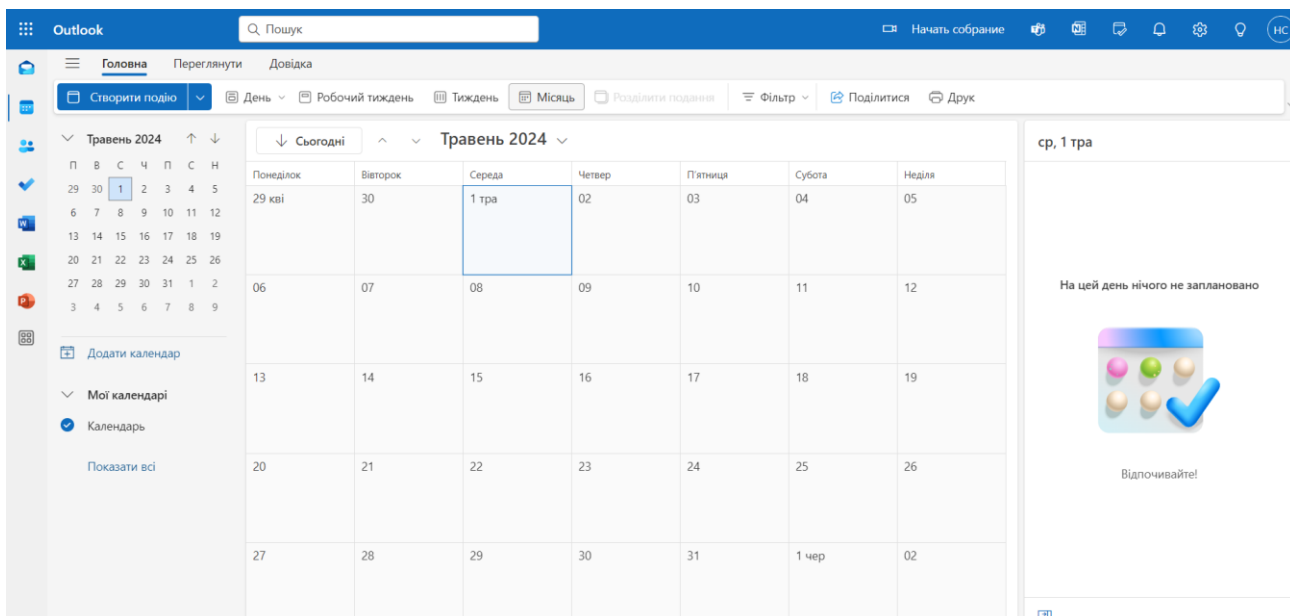


Рисунок 1.3 – Календар у Microsoft Outlook

Порівнюючи цей органайзер з розглянутим раніше «Google Календар» можна помітити більш строгий та корпоративний дизайн. Функції здебільшого схожі до тих що були у Google, але є і свої нововведення(рис.1.4).

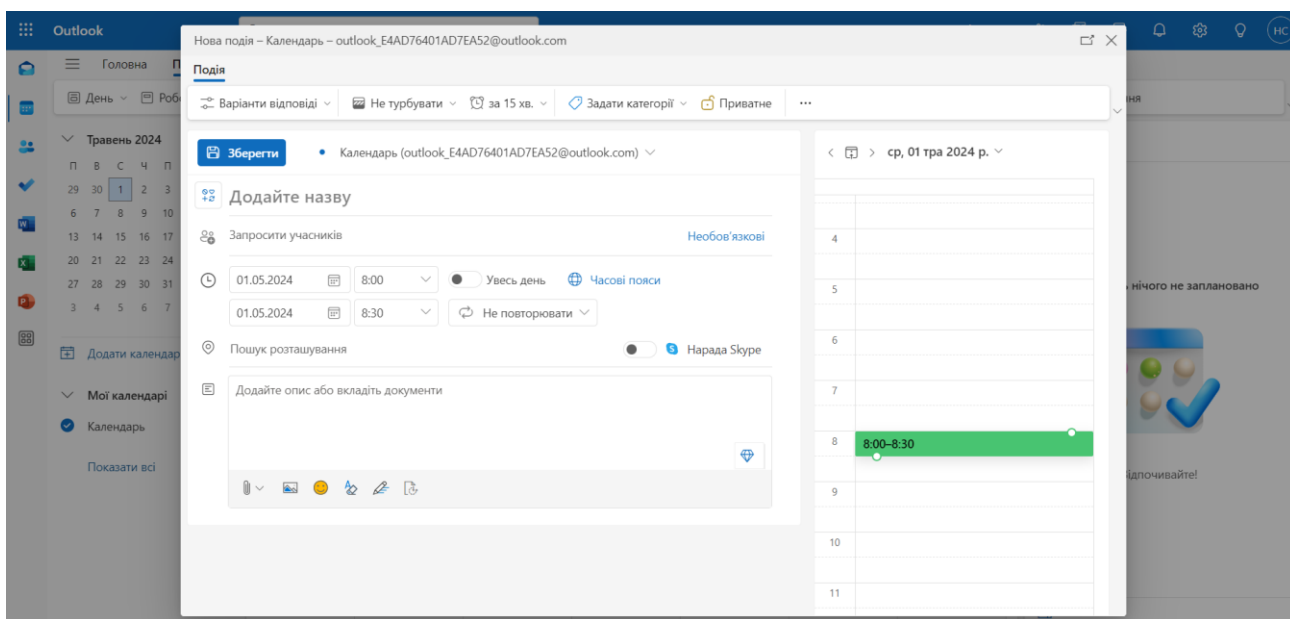


Рисунок 1.4 – Сторінка створення завдання у Microsoft Outlook

До основних переваг даного сервісу можна віднести:

- Функціональність: Календар, електронна пошта, контакти, задачі, інтеграція з Microsoft Teams.

- Професійний інтерфейс, трохи складніший для новачків.
- Глибока інтеграція з іншими сервісами Microsoft.
- Додатки для Android та iOS.
- Висока надійність, особливо в корпоративних середовищах.
- Безкоштовний з обмеженими функціями, додаткові функції доступні в рамках Microsoft 365.

Третім аналогом є Todolist (рис. 1.5). Todoist - це додаток для управління завданнями, який дозволяє створювати списки завдань, встановлювати терміни та пріоритети, а також вести статистику виконаних завдань(рис1.6). Todoist інтегрований з багатьма іншими сервісами, що дозволяє ефективно використовувати його в різних сферах життя

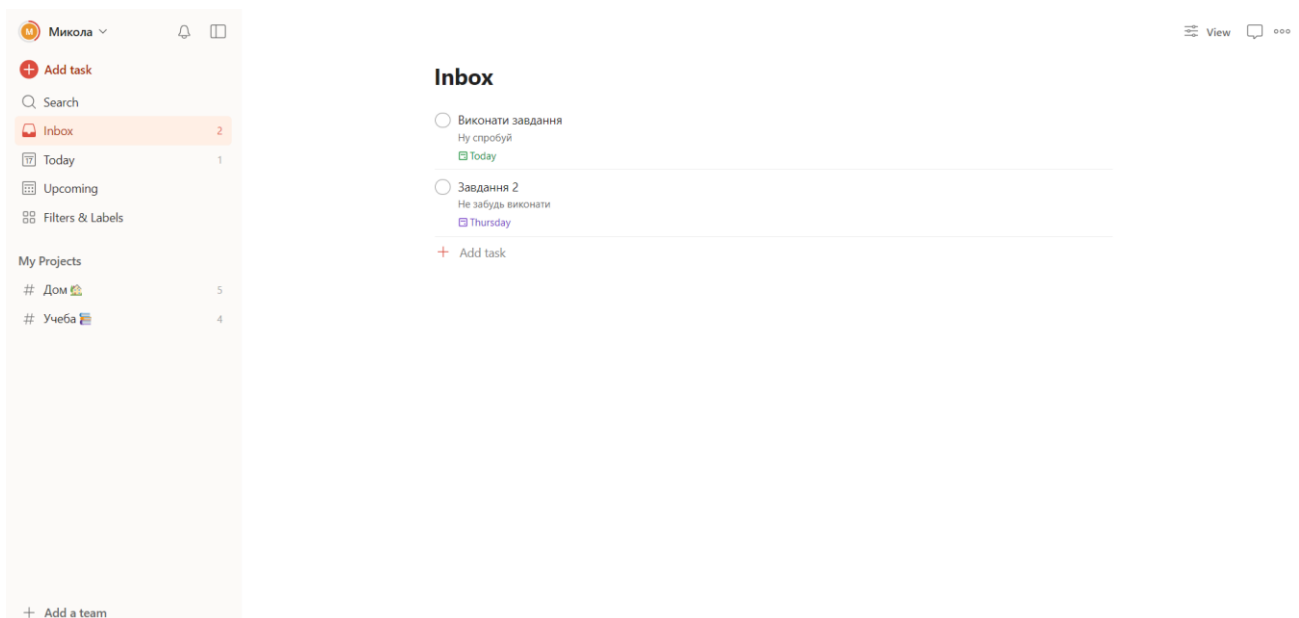


Рисунок 1.5 – Головна сторінка у вебдодатку Todolist

До його основних переваг можна перерахувати:

- Приємний і зрозумілий інтерфейс
- Легкість у користуванні
- Інтеграція з календарями, поштовими сервісами та іншими інструментами.
- Висока продуктивність та надійність.

- Безкоштовний з обмеженими функціями, додаткові функції доступні в рамках платних планів

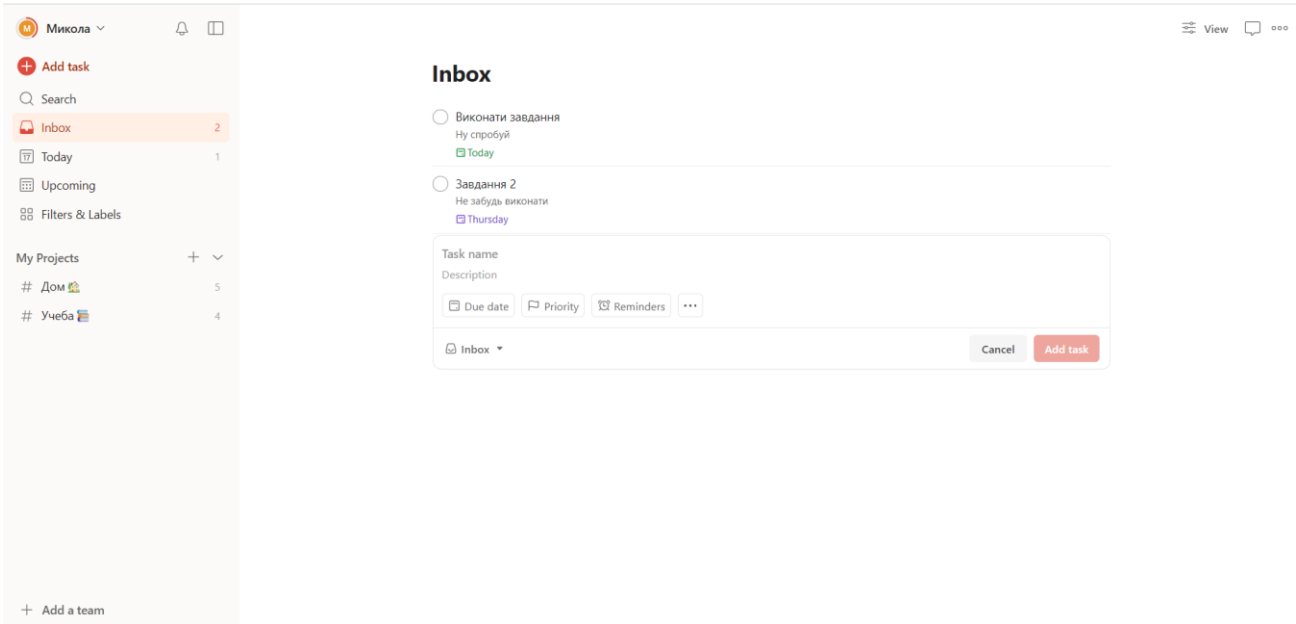


Рисунок 1.6 – Створення завдання у вебдодатку Todoist

Загалом додаток надає всі необхідні можливості органайзеру і є дуже легким для розуміння користувача. Від двох попередніх прикладів відрізняється тим, що зроблений не у вигляді календаря, а у вигляді списку

Щоб порівняти аналоги було створено таблицю (табл. 1.1)

Таблиця 1.1

Порівняння аналогів

Характеристика	Google Календар	Microsoft Outlook	Todolist
Функціональність	Висока	Висока	Висока
Зручність	Зручно для бізнесу і звичайного користувача	Зручно для бізнесу	Зручно для звичайного користувача
Інтеграція	+	+	+

Мобільність	+	+	+
Продуктивність і надійність	+	+	+
Ціна	Умовно безкоштовний	Умовно безкоштовний	Умовно безкоштовний
Відгуки	Позитивні	Позитивні	Позитивні

Провівши аналіз таблиці, можна зробити висновки, що кожен з додатків виконує свої основні функції і надає користувачам багато різних можливостей, але розраховані вони на різну аудиторію. Беручи все це до уваги можна перейняти найкращі моменти з додатків конкурентів і реалізувати їх у власному проекті

Аналіз аналогів показав, що основні функції, такі як управління подіями та задачами, інтеграція з іншими сервісами та мобільність, є критично важливими. Водночас, існує можливість покращення зручності користування та додавання нових функцій, таких як покращена колаборація та інтеграція з сучасними інструментами продуктивності. Це дозволить створити унікальний продукт, який відповідатиме потребам користувачів та виділятиметься на ринку

1.3 Визначення архітектури вебсайту

Вибір архітектури є критичним для успішної реалізації веб-орієнтованих систем, зокрема для систем планування робочого часу. Архітектура включає клієнт-серверну модель, структуру баз даних, вибір технологій і фреймворків, які будуть використовуватись при розробці.

Серед наявних архітектур візьмемо до розгляду

1. Клієнт-серверна архітектура.
2. Мікросервісна архітектура.
3. Монолітна архітектура.

4. MVT архітектура

Клієнт-серверна архітектура — це модель розподіленої обчислювальної системи, в якій завдання і ресурси розподіляються між серверами (постачальниками ресурсів) та клієнтами (споживачами ресурсів). Сервери відповідають за управління, зберігання та надання доступу до даних і сервісів, а клієнти взаємодіють із серверами, відправляючи запити та отримуючи необхідні ресурси чи інформацію. Така архітектура дозволяє покращити продуктивність і ефективність системи, оскільки ресурси централізуються на сервері, а клієнти можуть бути відносно легкими і простими пристроями. Важливими перевагами клієнт-серверної архітектури є можливість легко масштабувати систему та забезпечити централізоване управління безпекою та доступом до даних.

Мікросервісна архітектура — це підхід до розробки програмного забезпечення, в якому застосунок складається з невеликих, незалежних сервісів, кожен з яких виконує конкретну бізнес-функцію. Кожен мікросервіс працює як окрема одиниця, взаємодіючи з іншими через чітко визначені API, зазвичай за допомогою протоколів HTTP/HTTPS або повідомлень. Такий підхід забезпечує гнучкість у розробці та масштабуванні, оскільки кожен мікросервіс можна розробляти, тестувати, розгортати та оновлювати незалежно від інших. Це також сприяє підвищенню надійності системи, оскільки збій одного мікросервісу не призводить до зупинки всього застосунку. Мікросервісна архітектура дозволяє розробникам використовувати різні технології та мови програмування для кожного сервісу, що підвищує адаптивність і ефективність роботи команд розробників.

Монолітна архітектура — це підхід до розробки програмного забезпечення, в якому всі компоненти та функції застосунку об'єднані в єдину, нероздільну програму. Всі частини системи, такі як інтерфейс користувача, бізнес-логіка та доступ до даних, інтегровані та запускаються як єдиний процес. Така архітектура є простою для розробки та тестування на початкових етапах, оскільки всі компоненти

знаходяться в одному кодовому базі, що полегшує керування залежностями та забезпечує цілісність системи. Однак, з часом монолітні застосунки можуть стати громіздкими та складними в обслуговуванні, оскільки зміни в одній частині можуть впливати на інші частини системи. Масштабування моноліту також може бути викликом, оскільки для збільшення продуктивності необхідно розгортати всю програму цілком. Незважаючи на ці недоліки, монолітна архітектура залишається популярною завдяки своїй простоті та зрозумілості, особливо для менших проєктів або на ранніх стадіях розробки.

MVT (Model-View-Template) — це архітектурний шаблон, використовуваний в веб-фреймворках, зокрема в Django. У цій моделі: Модель (Model) відповідає за доступ до даних і бізнес-логіку. Вона визначає структуру даних і правила взаємодії з базою даних. Представлення (View) обробляє запити користувачів і відображає відповіді. В Django, представлення є функціями або класами, які приймають HTTP-запити, взаємодіють з моделями та готують дані для відображення. Шаблон (Template) відповідає за відображення даних користувачеві у вигляді HTML-сторінок. Він використовує дані, підготовлені представленням, для створення динамічного вмісту, який користувач бачить на сторінці. MVP архітектура сприяє чіткому розділенню логіки представлення, бізнес-логіки та відображення, що полегшує розробку, підтримку і масштабування вебсайтів. Вона підходить для проєктів будь-якої складності, забезпечуючи стабільність та ефективність у взаємодії з користувачем і базою даних.

Для реалізації вебсайту було вибрано фреймворк Django , що базується на MVP архітектурі і має впоратися з поставленими задачами.

Вибір MVP архітектури обумовлений її перевагами у масштабованості, централізованому управлінні даними, зручності розробки та мобільності. Це забезпечить високу продуктивність та надійність веб-орієнтованої системи планування робочого часу, а також дозволить легко інтегрувати нові функції та адаптувати систему до змін у вимогах користувачів.

Модель (Model) в MVT відповідає за роботу з даними програми. Вона включає в себе всі необхідні класи і методи для доступу до даних, їх збереження, вибірку та зміну. Модель відокремлює логіку обробки даних від логіки представлення та інтерфейсу користувача.

Представлення (View) відповідає за відображення даних користувачу. Це може бути HTML-шаблон, який відображає сторінку в браузері, або інші форми представлення даних. Представлення мають обмежену логіку і використовуються для відображення даних, які були оброблені моделлю.

Шаблон (Template) в MVT використовується для відображення даних, отриманих з моделі, у представленні. Шаблони дозволяють створювати динамічний контент на веб-сторінках, змінюючи вміст сторінок в залежності від даних, що надходять з сервера.

Один з головних переваг MVT - це чистота коду і відокремлення логіки сайту на окремі компоненти, що спрощує його розробку і підтримку. Крім того, цей шаблон дозволяє покращити безпеку програми, оскільки дані і логіка сайту відокремлені одна від одної.

Однак, використання MVT може бути складним для початківців, оскільки він вимагає розуміння концепцій моделі, представлення та шаблону. Також, в деяких випадках може виникнути перевантаження шаблонів, що може призвести до складності управління сайтом або додатком.

Узагальнюючи, модель MVT є потужним інструментом для розробки вебсайтів, який дозволяє ефективно розділити логіку сайту на окремі компоненти і підвищити його гнучкість і підтримку. Він вимагає певного рівня розуміння технологій розробки вебсайтів, але може забезпечити високу якість і безпеку програмного забезпечення.

1.4 Обґрунтування вибору інструментальних засобів

Кваліфікаційна робота буде будуватися на таких інструментальних засобах як:

- Django - основний фреймворк для роботи.
- Python як основна мова програмування для розробки.
- PyCharm – інтегроване середовище розробки для зручного написання та управління кодом.
- HTML та CSS необхідні для створення вебінтерфейсу.
- SQLite використовуємо для роботи з базою даних для управління даними

1. Фреймворк Django - це високорівневий веб-фреймворк, написаний на мові програмування Python, який дозволяє розробникам швидко створювати вебсайти. Він базується на принципах "швидкості розробки" та "DRY" (Don't Repeat Yourself), спрощуючи процес розробки за рахунок вбудованих компонентів та структурованих шаблонів. Його використання дозволяє швидко розробляти вебсайти, забезпечуючи високу якість, безпеку і ефективність.

2. PhpStorm PyCharm - це інтегроване середовище розробки (IDE) для мови програмування Python, яке розробляється компанією JetBrains. Воно спеціалізується на підтримці розробки Python-проектів, надаючи розширені можливості для зручного написання коду, відлагодження програм, управління проектами та багато іншого. PyCharm і Django відмінно взаємодіють між собою і є популярними інструментами для розробки вебсайтів на Python. PyCharm надає спеціальну підтримку для проектів, створених на базі Django. Він автоматично визначає структуру Django проекту, включаючи файли налаштувань, маршрутизації, моделей та інших складових проекту. PyCharm підтримує вбудований сервер Django для локального розробки та відлагодження, що дозволяє зручно тестувати та відлагоджувати ваші Django додатки без необхідності налаштовувати інший сервер. PyCharm і Django добре сумісні між собою і утворюють потужний інструментарій

для розробки вебсайтів на Python. Використання PyCharm забезпечує зручність розробки, підвищує продуктивність завдяки інтегрованим інструментам для Django, та дозволяє швидко розгортати та тестувати проекти

3. HTML (HyperText Markup Language) є основним будівельним блоком веб-сторінок. Він використовується для структурування контенту та розмітки веб-сторінок, що відображаються в інтернеті. HTML складається з набору елементів, які описують різні частини веб-сторінки, такі як заголовки, абзаци, списки, зображення, посилання та багато інших. Використання HTML дозволяє розробникам створювати структуровані та доступні веб-документи, які можуть включати інші технології, такі як CSS для стилізації та JavaScript для додавання інтерактивності. Основна мета HTML - це забезпечити однорідність відображення контенту на різних пристроях та веб-браузерах, забезпечуючи користувачам зручність у сприйнятті інформації через стандартизовану структуру та семантику елементів. CSS дозволить створити дизайн для додатку, що вплине на зовнішній вигляд та забезпечить єдність візуального стилю.

4. SQLite - це легка вбудовувана база даних, яка підтримує стандарт SQL. Вона відмінно підходить для вбудованих систем, мобільних додатків та невеликих вебсайтів, оскільки не потребує окремого сервера баз даних і може працювати без мережевого доступу. SQLite зберігає всю базу даних в одному файлі, що робить його простим у використанні та налаштуванні. Він підтримує більшість стандартних типів даних, включаючи INTEGER, TEXT, REAL, BLOB, а також підтримує операції зі зв'язками між таблицями. SQLite є популярним вибором для прототипування додатків, а також для ситуацій, де необхідна легкість у використанні та підтримці бази даних без значного навантаження

Системні вимоги, що гарантують стабільну роботу, були сформульовані на основі ретельно підібраного та обґрунтованого інструментарію. Ці вимоги визначають мінімальну конфігурацію системи:

1. Вимоги до середовища виконання:

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						20

Мінімально необхідна конфігурація, при якій система працюватиме стабільно:

- Операційна система: Windows 7 або вище;
- Процесор: 32- або 64-розрядний з тактовою частотою 1.2 ГГц або вище;
- Оперативна пам'ять: 2 ГБ для 32-розрядних процесорів та 4 ГБ для 64-розрядних процесорів;
- Вільне місце на жорсткому диску: 14 ГБ для 32-розрядних систем або 18 ГБ для 64-розрядних систем;

2. Вимоги до СУБД та доступу до даних:

- У ядрі системи повинна бути представлена реляційна система управління базами даних (СУБД);

3. Вимоги до клієнтського пристрою

- Операційна система з підтримкою браузерів;
- 1 ГБ ОЗУ;

Висновки до першого розділу

В рамках першого розділу досліджено вимоги до вебсайту та предметної області, визначено чіткий план дій з розробки, включаючи стек технологій. Проведено порівняльний аналіз аналогів, що дозволило запозичити найкращі практики та вдосконалити власний проект. Обґрунтовано вибір MVT-архітектури, фреймворка Django та мови Python для забезпечення стабільної та продуктивної роботи вебсайту. Інтерфейс користувача буде розроблений з використанням HTML та CSS, що дозволить створити динамічний, візуально привабливий та інтерактивний веб-інтерфейс. База даних SQLite забезпечить надійне зберігання, швидкий доступ та ефективне управління даними, гарантуючи необхідну продуктивність та масштабованість системи. Використання PyCharm як інструмента для розробки оптимізує процес написання, тестування та відлагодження коду, що сприятиме загальній продуктивності та надійності розробки

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБСАЙТУ «ОНЛАЙН ОРГАНАЙЗЕР»

2.1 Визначення варіантів використання та структури системи

З метою гарантування стабільної роботи, відповідності стандартам та ефективної взаємодії з користувачами, сформульовано високорівневі вимоги до онлайн-органайзера. Ці вимоги охоплюють:

- Функціональність: Система повинна забезпечувати всі необхідні функції для зручного ведення календаря, списку завдань, нотаток та спільної роботи.
- Продуктивність: Система повинна швидко реагувати на дії користувача та ефективно використовувати ресурси комп'ютера.
- Надійність: Система повинна бути стійкою до збоїв та гарантувати збереження даних.
- Безпека: Система повинна захищати дані користувачів від несанкціонованого доступу.
- Масштабованість: Система повинна мати можливість розширюватися за рахунок нових користувачів та даних.
- Ергономічність: Система повинна мати зручний та зрозумілий інтерфейс користувача.
- Доступність: Система повинна бути доступною з різних пристроїв та платформ.

Функціональні вимоги:

- Запис та редагування подій з описом, датою, часом, місцем та іншими деталями;
- Налаштування повторюваних подій;
- Створення категорій та міток для подій;

Ведення списку завдань:

- Створення та редагування завдань з описом;
- Налаштування повторюваних завдань;

- Перегляд списку завдань у різних режимах;
- Відмітка завдань як виконаних;

Запис нотаток:

- Створення та редагування текстових нотаток з можливістю форматування;
- Додавання зображень, аудіо та відео до нотаток;
- Створення категорій та міток для нотаток;
- Фільтрація та пошук нотаток;
- Перегляд списку нотаток у різних режимах;

Характеристика об'єкта комп'ютеризації:

Інтуїтивно зрозумілий інтерфейс дозволить користувачам без зусиль знаходити та використовувати необхідні функції, роблячи процес взаємодії більш комфортним та приємним.

Нефункціональні вимоги:

— Продуктивність:

Швидке реагування на дії користувача; Можливість роботи з великою кількістю даних; Економне використання ресурсів комп'ютера;

— Надійність:

Захист даних від втрати та пошкодження; Збереження працездатності системи у разі збоїв; Регулярне резервне копіювання даних;

— Ергономічність:

Зрозумілий та зручний інтерфейс користувача; Можливість налаштування інтерфейсу користувача;

— Масштабованість:

Можливість розширення системи за рахунок нових користувачів та даних; Підтримка додаткових модулів та функцій;

Діаграми варіантів використання (Use Case Diagrams) є важливим інструментом при проектуванні програмного забезпечення (ПЗ), оскільки вони дозволяють моделювати взаємодію між системою та її користувачами або іншими системами. Ці діаграми допомагають визначити функціональність системи, ідентифікувати ролі користувачів та їх взаємодію з системою, а також визначити основні варіанти використання системи.

Однією з основних переваг використання діаграм варіантів використання є їх здатність допомогти зрозуміти потреби і очікування користувачів. Це дозволяє команді проектування та розробки ПЗ краще зрозуміти, як система повинна працювати з точки зору користувачів, і розробити відповідну функціональність. Діаграми варіантів використання також дозволяють ідентифікувати потреби користувачів щодо взаємодії з системою і покращити їх задоволеність від використання ПЗ.

Крім того, діаграми варіантів використання допомагають визначити головні функціональність системи і визначити, які функції повинні бути реалізовані першочергово. Це дозволяє зосередити увагу на ключових аспектах проекту та покращити продуктивність розробки. Діаграми також допомагають уникнути недорозумінь між командами розробників та замовниками проекту, оскільки вони надають зрозуміле та лаконічне відображення функціональних вимог.

У підсумку, діаграми варіантів використання є важливим інструментом при проектуванні програмного забезпечення, оскільки вони дозволяють зрозуміти та відобразити потреби користувачів, ідентифікувати ключові функції системи і уникнути недорозумінь у процесі розробки. Вони є важливим етапом у проектуванні ПЗ і сприяють покращенню якості та ефективності розробки програмних продуктів.

На рисунку 2.1 представлено діаграму класів системи.

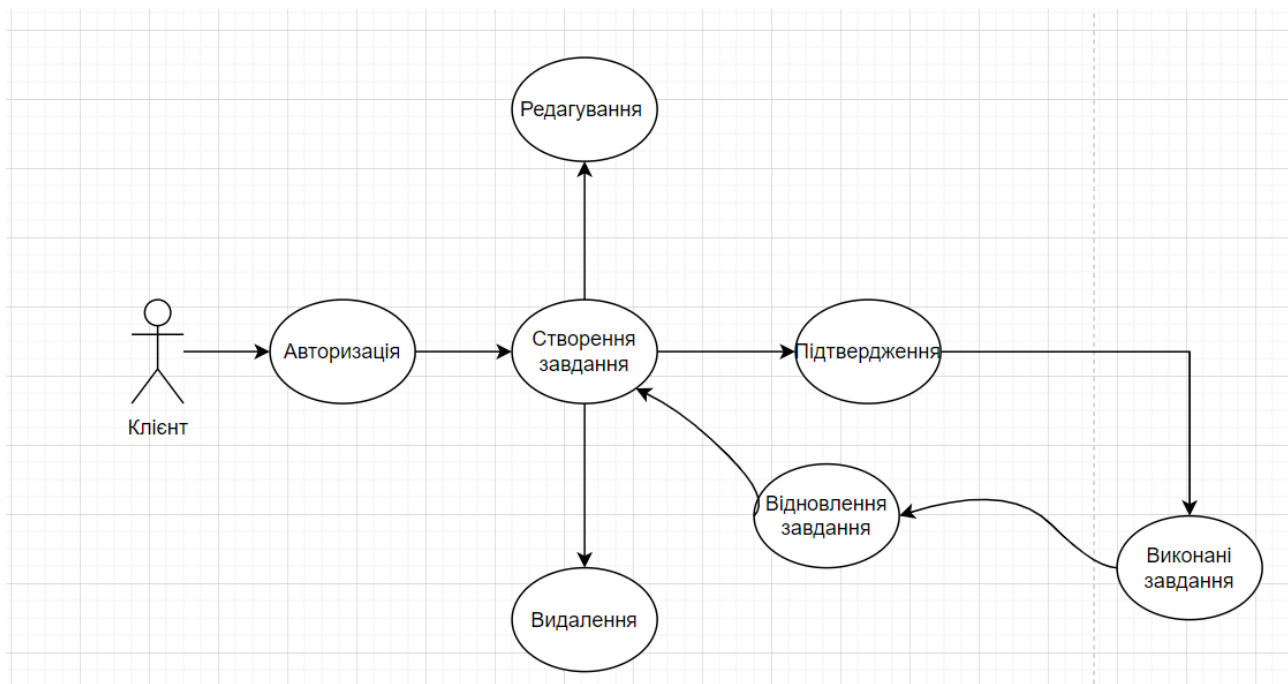


Рисунок 2.1 – Діаграма варіантів використання додатку

Аналіз функціональних вимог допоміг виявити ключові сутності, що описують основні компоненти системи та їх взаємозв'язки. Ці сутності лягли в основу діаграми класів, яка буде використовуватися для розробки програмного забезпечення.

2.2 Розробка бази даних

Проектування баз даних (БД) є ключовим етапом у розробці програмного забезпечення, оскільки від якості структури БД залежить ефективність та надійність системи. Одним з основних інструментів для проектування БД є ER-діаграми (сутність-зв'язок), які відображають структуру даних та їх взаємозв'язки.

ER-діаграми складаються з сутностей (entities), атрибутів (attributes) та зв'язків (relationships) між сутностями. Сутність - це об'єкт або поняття, яке ми хочемо включити в базу даних, наприклад, "користувач", "продукт" або "замовлення". Атрибути представляють властивості сутностей, наприклад, для сутності "користувач" атрибутами можуть бути ім'я, електронна пошта та пароль.

Зв'язки в ER-діаграмах вказують на зв'язок між двома або більше сутностями. Наприклад, зв'язок "має" може вказувати на те, що один об'єкт пов'язаний з іншим, як, наприклад, замовлення має продукти.

Під час проектування БД важливо правильно визначити сутності, атрибути та зв'язки, щоб забезпечити ефективне зберігання та обробку даних. ER-діаграми допомагають уникнути недорозумінь між розробниками та замовниками проекту щодо структури даних та їх взаємозв'язків.

Однією з переваг використання ER-діаграм є їхній інтуїтивно зрозумілий формат, який дозволяє швидко візуалізувати структуру БД. Крім того, ER-діаграми дозволяють виявити потенційні проблеми в структурі даних та внести необхідні зміни ще на ранніх етапах проектування.

У підсумку, ER-діаграми є важливим інструментом для проектування баз даних, оскільки вони дозволяють зрозуміти та відобразити структуру даних та їх взаємозв'язки. Вони допомагають уникнути недорозумінь та покращити якість та ефективність розробки програмного забезпечення

Діаграма бази даних складається з трьох таблиць

1. Users – таблиця, яка містить інформацію про користувачів, їх профілі і дані.

В поля цієї таблиці входять: userID, nickname, password

2. Task – зберігає дані про завдання, які було створено на вебсайті. Поля цієї таблиці включають Date, details, id, image, time та Title.

3. ExTask – зберігає дані про завдання, які було виконано на вебсайті і перенесено для майбутнього перегляду. Поля цієї таблиці включають Date, details, id, image, time та Title

Опис таблиць бази даних вказаний у таблицях 2.1-2.3

Таблиця 2.1

Поле	Тип	Призначення
User ID	int	Ідентифікатор
Nickname	varchar	Назва користувача

password	varchar	Пароль
----------	---------	--------

Таблиця 2.2

Поле	Тип	Призначення
Id	int	Ідентифікатор
Image	image	Зображення
Title	varchar	Назва
Details	varchar	Опис
Date	date	Дата
Time	time	Час

Таблиця 2.3

Поле	Тип	Призначення
Id	int	Ідентифікатор
Image	image	Зображення
Title	varchar	Назва
Details	varchar	Опис
Date	date	Дата
Time	time	Час

2.3 Проектування та реалізація алгоритмів роботи сайту

Проектування та реалізація алгоритмів роботи сайту "Онлайн органайзер" є ключовим етапом у створенні ефективної, функціональної та надійної системи. Алгоритми визначають, як користувачі взаємодіють з додатком, як обробляються дані, як здійснюється авторизація та аутентифікація користувачів, та інші важливі аспекти роботи сайту:

Основними компонентами веб-додатку "Онлайн органайзер" є:

- **Модель (Model):** Визначає структуру даних і забезпечує взаємодію з базою даних.
- **Представлення (View):** Відповідає за відображення даних користувачу та обробку введених даних.
- **Контролер (Controller):** Обробляє запити користувачів, викликає відповідні методи моделей і формує відповіді, які надсилаються користувачам.

Алгоритм реєстрації користувача

- Користувач заповнює форму реєстрації.
- Введені дані перевіряються на коректність (включаючи перевірку унікальності логіна/електронної пошти).
- Якщо дані коректні, створюється новий запис у базі даних користувачів.
- Користувачу надсилається підтвердження реєстрації.

Алгоритм аутентифікації користувача

- Користувач вводить логін та пароль на сторінці входу.
- Система перевіряє відповідність введених даних записам у базі даних.
- У випадку успішної аутентифікації створюється сесія користувача.
- Користувач перенаправляється на головну сторінку органайзера.

Алгоритм створення завдання

- Користувач заповнює форму створення завдання (назва, опис, дата та час, пріоритет тощо).
- Введені дані перевіряються на коректність.
- Якщо дані коректні, створюється новий запис у базі даних завдань.
- Користувачу відображається оновлений список завдань.

Алгоритм редагування завдання

- Користувач обирає завдання для редагування.
- Система завантажує поточні дані завдання у форму редагування.
- Користувач вносить зміни та зберігає їх.

- Введені дані перевіряються на коректність.
- Якщо дані коректні, запис у базі даних оновлюється.
- Користувачу відображається оновлений список завдань.

Алгоритм видалення завдання

- Користувач обирає завдання для видалення.
- Система запитує підтвердження видалення.
- Після підтвердження запис у базі даних видаляється.
- Користувачу відображається оновлений список завдань.

2.4 Реалізація функціоналу сайту

Спочатку створимо модуль реєстрації в додатку, для початку буде створено відповідну форму, у якій користувачі зможуть вводити необхідну інформацію для реєстрації.

Реалізація модулю реєстрації в додатку:

```
class SignUpForm(UserCreationForm):
    class Meta:
        model = User
        fields = [
            'username',
            'password1',
            'password2',
        ]

    username = forms.CharField(max_length=254, widget=forms.TextInput(attrs={
        'class': 'form-control block w-full rounded-md border-0 py-1.5 text-gray-900
shadow-sm ring-1 ring-inset ring-gray-300 placeholder:text-gray-400 focus:ring-2
focus:ring-inset focus:ring-indigo-600 sm:text-sm sm:leading-6',
        'placeholder': 'Введіть ім'я користувача'
    }))
    password1 = forms.CharField(widget=forms.PasswordInput(attrs={
        'class': 'form-control block w-full rounded-md border-0 py-1.5 text-gray-900
shadow-sm ring-1 ring-inset ring-gray-300 placeholder:text-gray-400 focus:ring-2
focus:ring-inset focus:ring-indigo-600 sm:text-sm sm:leading-6',
        'placeholder': 'Введіть пароль'
    }))
    password2 = forms.CharField(widget=forms.PasswordInput(attrs={
        'class': 'form-control block w-full rounded-md border-0 py-1.5 text-gray-900
shadow-sm ring-1 ring-inset ring-gray-300 placeholder:text-gray-400 focus:ring-2
focus:ring-inset focus:ring-indigo-600 sm:text-sm sm:leading-6',
        'placeholder': 'Підтвердіть пароль'
    }))

class LoginForm(AuthenticationForm):
```

```
username = forms.CharField(max_length=254, widget=forms.TextInput(attrs={'class':
'form-input'}))
password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-
input'}))
```

Далі перевіряємо надісланий запит і чи всі дані користувача є правильними.

Перевірка виконується у цьому блоці:

```
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
```

Наступний блок перевіряє чи коректні надіслані користувачем дані. При виконанні умов виконується збереження користувача і перехід на головну сторінку:

```
def login_view(request):
    if request.method == 'POST':
        form = LoginForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('main:index')
    else:
        form = LoginForm()
    return render(request, 'users/login
```

Далі створюємо форму в якій буде відображатися сторінка реєстрації і входу

```
{% extends "base.html" %}
{% block content %}

<div class="flex min-h-full flex-col justify-center px-6 py-12 lg:px-8">
  <div class="sm:mx-auto sm:w-full sm:max-w-sm">
    
    <h2 class="mt-10 text-center text-2xl font-bold leading-9 tracking-tight text-
gray-900">Sign in to your account</h2>
  </div>

  <div class="mt-10 sm:mx-auto sm:w-full sm:max-w-sm">
    <form class="space-y-6 login_form" action="{% url 'users:login' %}"
method="POST">
      {% csrf_token %}

      <div>
        <label for="username" class="block text-sm font-medium leading-6 text-gray-
900">Username</label>
        <div class="mt-2">
          <input id="id_username" name="username" type="text" autocomplete="text"
required class="form-control block w-full rounded-md border-0 py-1.5 text-gray-900
shadow-sm ring-1 ring-inset ring-gray-300 placeholder:text-gray-400 focus:ring-2
focus:ring-inset focus:ring-indigo-600 sm:text-sm sm:leading-6">
        </div>
      </div>
    </form>
  </div>
{% endblock content %}
```

Цей код шифрує пароль користувача після чого іде збереження в базі даних і перенаправлення користувача на головну сторінку

Тепер створюємо шаблон signup.html, який буде відображати форму та повідомляти про помилки

```
{% block content %}
<div class="flex min-h-full flex-col justify-center px-6 py-12 lg:px-8">
  <div class="sm:mx-auto sm:w-full sm:max-w-sm">
    
    <h2 class="mt-10 text-center text-2xl font-bold leading-9 tracking-tight
text-gray-900">Sign up for an account</h2>
  </div>

  <div class="mt-10 sm:mx-auto sm:w-full sm:max-w-sm">
    <form class="space-y-6" action="{% url 'users:signup' %}" method="POST">
      {% csrf_token %}

      <div>
        <label for="id_username" class="block text-sm font-medium leading-6
text-gray-900">Username</label>
        <div class="mt-2">
          {{ form.username }}
        </div>
      </div>

      <div>
        <label for="id_password1" class="block text-sm font-medium leading-6
text-gray-900">Password</label>
        <div class="mt-2">
          {{ form.password1 }}
        </div>
      </div>

      <div>
        <label for="id_password2" class="block text-sm font-medium leading-6
text-gray-900">Password confirmation</label>
        <div class="mt-2">
          {{ form.password2 }}
        </div>
      </div>

      <div>
        <button type="submit" class="flex w-full justify-center rounded-md
bg-indigo-600 px-3 py-1.5 text-sm font-semibold leading-6 text-white shadow-sm
hover:bg-indigo-500 focus-visible:outline focus-visible:outline-2 focus-
visible:outline-offset-2 focus-visible:outline-indigo-600">Sign up</button>
      </div>
    </form>

    <p class="mt-10 text-center text-sm text-gray-500">
      Already have an account? <a href="{% url 'users:login' %}" class="font-
semibold leading-6 text-indigo-600 hover:text-indigo-500">Login</a>
    </p>
  </div>
```

```
</div>
{% endblock content %}
```

Далі створюємо форму для створення і додавання записів. Форма призначена для того, щоб користувач міг створити завдання і зберігати на головній сторінці, куди будуть виводитися всі записи.

```
class TodoForm(ModelForm):
    class Meta:
        model = TodoList
        fields = ['title', 'content', 'due_date', 'image']

        title = forms.CharField(widget=forms.TextInput(attrs={
            'class': 'w-full px-3 py-2 border rounded-md shadow-sm focus:outline-none
focus:ring-2 focus:ring-blue-500',
            'placeholder': 'Введіть назву'
        )))
        content = forms.CharField(widget=forms.Textarea(attrs={
            'class': 'w-full px-3 py-2 border rounded-md shadow-sm focus:outline-none
focus:ring-2 focus:ring-blue-500',
            'placeholder': 'Введіть опис'
        )))
        image = forms.ImageField(widget=forms.FileInput(attrs={
            'class': 'w-full px-3 py-2 border rounded-md shadow-sm focus:outline-none
focus:ring-2 focus:ring-blue-500'
        )))
        due_date = forms.DateField(widget=forms.DateInput(attrs={
            'class': 'w-full px-3 py-2 border rounded-md shadow-sm focus:outline-none
focus:ring-2 focus:ring-blue-500',
            'type': 'date'
        )))
```

Створюємо HTML-форму, яка буде відображати всі завдання. Там будуть міститися назва, опис, фото і час коли має виконатися завдання

```
{% extends "base.html" %}
{% block content %}

<div class="bg-white">

    <div class="relative isolate px-6 pt-14 lg:px-8">
        <div
            class="absolute inset-x-0 -top-40 -z-10 transform-gpu overflow-hidden
blur-3xl sm:-top-80"
            aria-hidden="true">
            <div
                class="relative left-[calc(50%-11rem)] aspect-[1155/678] w-
[36.125rem] -translate-x-1/2 rotate-[30deg] bg-gradient-to-tr from-[#ff80b5] to-
[#9089fc] opacity-30 sm:left-[calc(50%-30rem)] sm:w-[72.1875rem]"
                style="clip-path: polygon(74.1% 44.1%, 100% 61.6%, 97.5% 26.9%, 85.5%
0.1%, 80.7% 2%, 72.5% 32.5%, 60.2% 62.4%, 52.4% 68.1%, 47.5% 58.3%, 45.2% 34.5%,
27.5% 76.7%, 0.1% 64.9%, 17.9% 100%, 27.6% 76.8%, 76.1% 97.7%, 74.1% 44.1%)"></div>
            </div>
            <div class="mx-auto max-w-2xl py-32 sm:py-48 lg:py-56">
                <div class="hidden sm:mb-8 sm:flex sm:justify-center">
```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						32

```

        <div
            class="relative rounded-full px-3 py-1 text-sm leading-6 text-
gray-600 ring-1 ring-gray-900/10 hover:ring-gray-900/20">
            Announcing our next round of funding. <a href="#"
            class="font-semibold text-indigo-600"><span
            class="absolute inset-0" aria-hidden="true"></span>Read
            more <span aria-hidden="true">&rarr;</span></a>
        </div>
    </div>
{% endblock content %}

```

Після створення HTML-структури форми, наступним етапом буде її обробка та збереження даних з форми у відповідному контролері.

```

def create_todo(request):
    if request.method == 'GET':
        return render(request, 'todo/create_todo.html', {'form': TodoForm()})
    else:
        print('ale')
        # try:
        form = TodoForm(request.POST, request.FILES)
        if form.is_valid():
            print('ale2')

            newtodo = form.save(commit=False)
            newtodo.user = request.user
            newtodo.save()

            return redirect('todo:current')
        else:
            return render(request, 'todo/create_todo.html', {'form': form, 'error':
'Невірно заповнене поле'})
        # except ValueError:
        #     return render(request, 'todo/create_todo.html', {'form': TodoForm(),
'error': 'Невірно заповнене поле'})

_todo.html', {'form': form, 'todo': todo, 'error': 'Невірно заповнене поле'})

```

Проводимо налаштування URL-адрес

```

urlpatterns = [
    path('create/', views.create_todo, name='create'),
    path('done/', views.done_todo, name='done'),
    path('current/', views.current_todo, name='current'),
    path('all/', views.all_todo, name='all'),

    path('edit/<int:task_id>', views.edit_todo, name='edit'),

    path('delete/<int:task_id>', views.delete_todo, name='delete'),
    path('mark-as-done/<int:task_id>', views.mark_as_done, name='mark_as_done'),
    path('mark-as-current/<int:task_id>', views.mark_as_current,
name='mark_as_current'),
]

```

Фільтрація завдань:

```

@login_required
def mark_as_done(request, task_id):
    todo = TodoList.objects.filter(id=task_id)
    todo.update(done=True)

```

```

        return redirect('todo:current')

@login_required
def mark_as_current(request, task_id):
    todo = TodoList.objects.filter(id=task_id)
    todo.update(done=False)
    return redirect('todo:done')

@login_required
def delete_todo(request, task_id):
    todo = TodoList.objects.filter(id=task_id)
    todo.delete()
    return redirect('todo:current')

```

Форма виконаних завдань

```

@login_required
def current_todo(request):
    todos = TodoList.objects.filter(user=request.user, done=False).order_by('-created')
    context = {
        'name': 'Current',
        'todolist': todos
    }
    return render(request, 'todo/todo.html', context)

```

Форма яка показує всі завдання:

```

def all_todo(request):
    todos = TodoList.objects.filter(user=request.user).order_by('-created')
    context = {
        'name': 'All',
        'todolist': todos
    }
    return render(request, 'todo/todo.html', context)

```

Видалення завдань:

```

@login_required
def done_todo(request):
    todos = TodoList.objects.filter(user=request.user, done=True)
    context = {
        'name': 'Done',
        'todolist': todos
    }
    return render(request, 'todo/todo.html', context)

```

У проєкті на базі Django файл views.py відповідає за обробку запитів від користувачів та повернення відповідей. Він містить функції або класи, які називаються "видами" (views), і вони визначають, як обробляти різні типи запитів (GET, POST тощо), що надходять до веб-додатка. Основні завдання views включають:

1. Обробка запитів та повернення відповідей: Views отримують

HTTP-запити від клієнтів (наприклад, веб-браузерів) і повертають HTTP-

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						34

відповіді. Це може бути HTML-сторінка, JSON-дані, редирект або інші типи відповідей.

2. Взаємодія з моделями: Views можуть взаємодіяти з моделями для отримання, зміни або збереження даних у базі даних. Вони часто використовують ORM (Object-Relational Mapping) для роботи з базою даних.

3. Рендеринг шаблонів: Views можуть використовувати шаблони для рендерингу HTML-сторінок. Вони передають дані шаблонам, які потім генерують HTML-відповідь, яка відправляється користувачу.

4. Обробка форм: Views можуть обробляти форми, приймаючи дані від користувачів, перевіряючи їхню валідність та зберігаючи ці дані у базі даних або виконуючи інші дії.

Обробка запитів users

```
def login_view(request):
    if request.method == 'POST':
        form = LoginForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('main:index')
    else:
        form = LoginForm()
    return render(request, 'users/login.html', {'form': form})
```

Обробка запитів форми

```
def create_todo(request):
    if request.method == 'GET':
        return render(request, 'todo/create_todo.html', {'form': TodoForm()})
    else:
        print('ale')
        # try:
        form = TodoForm(request.POST, request.FILES)
```

```

if form.is_valid():
    print('ale2')

    newtodo = form.save(commit=False)
    newtodo.user = request.user

```

Міграції в django

Міграції в Django — це механізм для управління змінами в схемі бази даних. Вони дозволяють зручно створювати, змінювати та видаляти таблиці та поля в базі даних, відстежуючи ці зміни у вигляді окремих файлів міграцій. Використання міграцій спрощує процес підтримки та оновлення структури бази даних в проект

Міграції в проекті

```

from django.db import migrations, models
import django.db.models.deletion

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Category',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=100)),
            ],
            options={
                'verbose_name': 'Category',
                'verbose_name_plural': 'Categories',
            }
        )
    ]

```

Висновки до другого розділу

У другому розділі було представлено детальний опис проектування онлайн-органайзера. Було визначено функціональні можливості додатку, розроблено алгоритми роботи з платформою та обрано технології реалізації. Отримані результати свідчать про те, що розроблений онлайн-органайзер буде простим у використанні, матиме всі необхідні функції та буде зручним для користувачів.

Розроблений онлайн-органайзер має потенціал стати цінним інструментом для управління завданнями, часом та інформацією. Він буде корисним для людей з різних сфер діяльності, включаючи студентів, працівників, бізнесменів та домогосподарок.

Наступним кроком у проектуванні онлайн-органайзера буде його реалізація. Після реалізації буде проведено тестування та доопрацювання додатку.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН ОРГАНАЙЗЕРОМ

3.1 Порядок встановлення та налаштування параметрів сайту

Встановлення та налаштування платформи потребує розгортання всіх її компонентів та поетапного налаштування як кожного елемента окремо, так і всієї системи в цілому. Для зручності відображення цього процесу створимо діаграму розгортання(рис3.1).

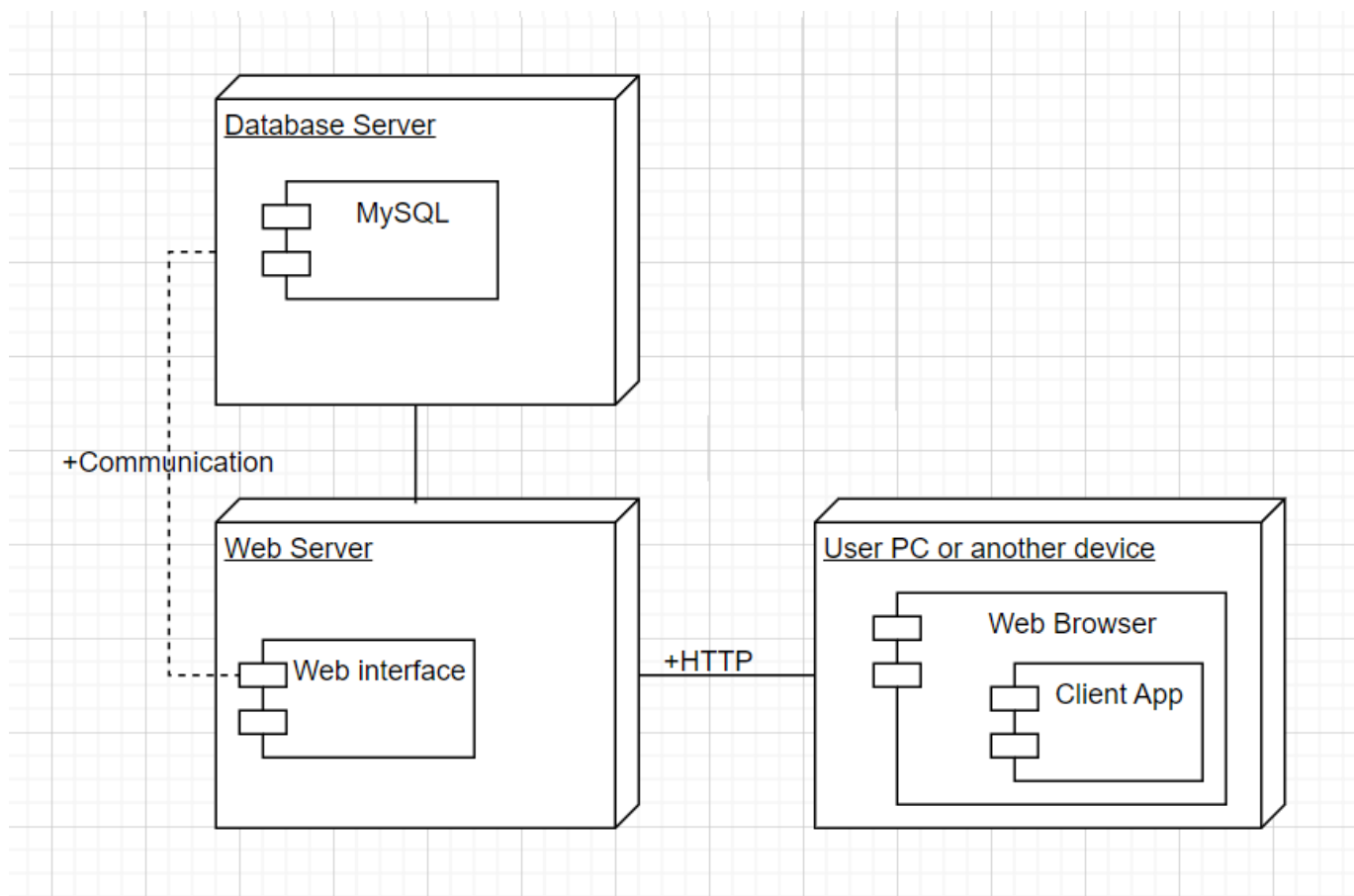


Рисунок 3.1 – Діаграма розгортання

Опис діаграми розгортання:

Ця діаграма розгортання ілюструє систему, що складається з трьох компонентів: веб-сервера, сервера бази даних та клієнтського пристрою.

Взаємодія між клієнтським пристроєм та веб-сервером відбувається за допомогою протоколу HTTP, який використовує модель запит-відповідь, базовану на стандартах TCP. Цей протокол забезпечує надійну передачу даних без втрат.

Обмін інформацією між веб-сервером та сервером бази даних здійснюється за допомогою протоколу TCP/IP, який гарантує стабільне та безперервне з'єднання.

3.2 Структура інтерфейсу вебсайту

Графічний інтерфейс користувача, або GUI, представляє собою специфічну форму інтерфейсу, яка дає можливість користувачам взаємодіяти з електронними пристроями за допомогою графічних елементів, таких як піктограми та аудіоіндикатори, замість текстових інтерфейсів користувача, які передбачають введення команд через клавіатуру комп'ютера.

Взаємодія в GUI зазвичай здійснюється шляхом безпосереднього взаємодії з графічними елементами, а не шляхом введення текстових команд. Інтерфейси командного рядка (CLI) були поширені раніше, і GUI були введені відповідно до спрощення використання пристроїв та програм, оскільки CLI вимагали введення текстових команд.

GUI застосовуються не лише на комп'ютерах, але й на різних кишенькових пристроях, таких як MP3-плеєри, портативні медіапрогравачі, ігрові пристрої, смартфони і різноманітні побутові, офісні та промислові пристрої для управління. GUI часто не використовуються у випадках, коли дисплей має низьку роздільну здатність, як у відеоіграх (де використовується інтерфейс зі складнішою внутрішньою структурою), або коли пристрої мають інші типи дисплеїв, як от об'ємні дисплеї.

Розробка візуальної композиції та тимчасової поведінки GUI є ключовою частиною програмування прикладних програм, які спрямовані на взаємодію людини з комп'ютером. Вона спрямована на полегшення використання для користувачів, покращення ефективності та дотримання принципів юзабіліті. Дизайн, орієнтований

на користувача, використовується для того, щоб забезпечити адаптованість візуального інтерфейсу до потреб користувачів. Модель адаптивної реалізації контенту веб сторінки дозволяє створювати гнучкі структури, в яких інтерфейс незалежний від функцій програми та опосередковано пов'язаний із ними, що дозволяє легко налаштувати графічний інтерфейс. Це робить можливим вибір або створення іншої обгортки вебсторінки і полегшує зміну інтерфейсу з урахуванням потреб користувача.

У більшості випадків, великі віджети, такі як вікна, надають рамку або контейнер для відображення основного вмісту презентації, такого як веб-сторінка, електронна пошта або малюнок. Менші віджети служать для введення користувачем і взаємодії з інформацією.

GUI може бути спеціально розробленим для конкретних програм та використовувати різноманітні види екранів і пристроїв для управління, такі як АТМ, POS-термінали в ресторанах, каси самообслуговування у роздрібних магазинах, інформаційні кіоски в громадських місцях та багато інших. GUI також використовуються в автомобільних навігаційних системах та мультимедійних центрах, а також на сенсорних екранах кишенькових пристроїв та ігрових систем.

Завдяки GUI користувачі можуть інтуїтивно взаємодіяти з програмами, не вдаючись до командного рядка або текстових інтерфейсів. Це значно полегшує роботу з програмами, робить їх доступнішими для широкого кола користувачів та покращує загальний користувацький досвід. Добре продуманий і розроблений GUI є ключовим фактором успіху програмного продукту, оскільки він визначає зручність та ефективність використання програми.

В даному випадку інтерфейс користувача складається з кількох окремих сторінок.

Починається сайт з головної сторінки(рис.3.2):

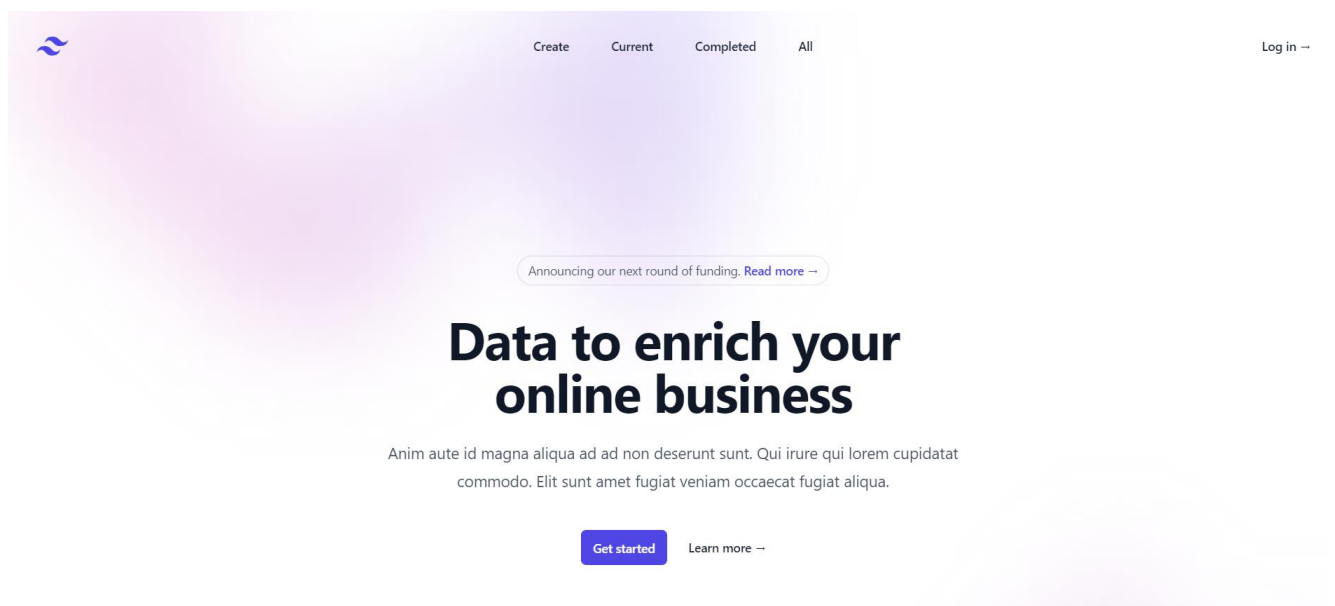


Рисунок 3.2 – Головна сторінка

У шапці додатку розташоване навігаційне меню, що містить основні розділи сайту: “Create”, “Current”, “Completed” і “All” (рис. 3.3) і кнопку щоб увійти в систему(рис3,4). Також зверху справа розташований до форми реєстрації

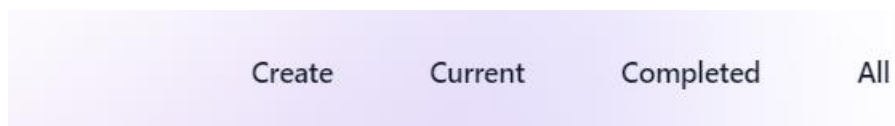


Рисунок 3.3 – Навігаційне меню

Log in →

Рисунок 3.4 – Кнопка log in для входу в систему

Після натискання кнопки "Увійти" користувачеві буде запропоновано ввести свої дані для авторизації на сайті, за умови, що він вже має зареєстрований обліковий запис. (рис. 3.5)

Sign in to your account

Username

Password

[Forgot password?](#)

Sign in

Рисунок 3.5 – Форма для входу в обліковий запис

У випадку необхідності зареєструвати обліковий запис користувач натискає кнопку “Зареєструватися” у низу форми (рис. 3.5). При реєстрації відвідувачу буде слід вказати своє ім'я, пошту, номер телефону та придумати пароль (рис. 3.6).

Якщо у клієнта немає зареєстрованого акаунту йому необхідно натиснути кнопку signup внизу форми (рис. 3.6)

[Signup](#)

Рисунок 3.6 – Кнопка signup

При реєстрації клієнту необхідно вказати назву акаунту та двічі ввести пароль, перший раз для створення, другий для підтвердження (рис. 3.7)

Sign up for an account

Username

Password

Password confirmation

Sign up

Already have an account? [Login](#)

Рисунок 3.7 – форма реєстрації

У випадку, якщо клієнт вже зареєстрований або в нього не співпали паролі, то введенні дані буде скинуто.

Після авторизації користувач може створювати завдання (рис. 3.8)

Додати задачу

Image:

Выберите файл Файл не выбран

Title:

Введіть назву

Content:

Введіть опис

Due date:

ДД.ММ.ГГГГ



Додати

Рисунок 3.8 – форма створення завдання

Додати задачу

Image:

Виберите файл

25528187d8a32d1c...a63e3b301de86.jpg

Title:

Завдання 1

Content:

Виконати

Due date:

20.06.2024

Додати

Рисунок 3.9 – Заповнена форма створення завдання

Після того як завдання було створено воно потрапляє на сторінку Current(рис3.10-3.11).

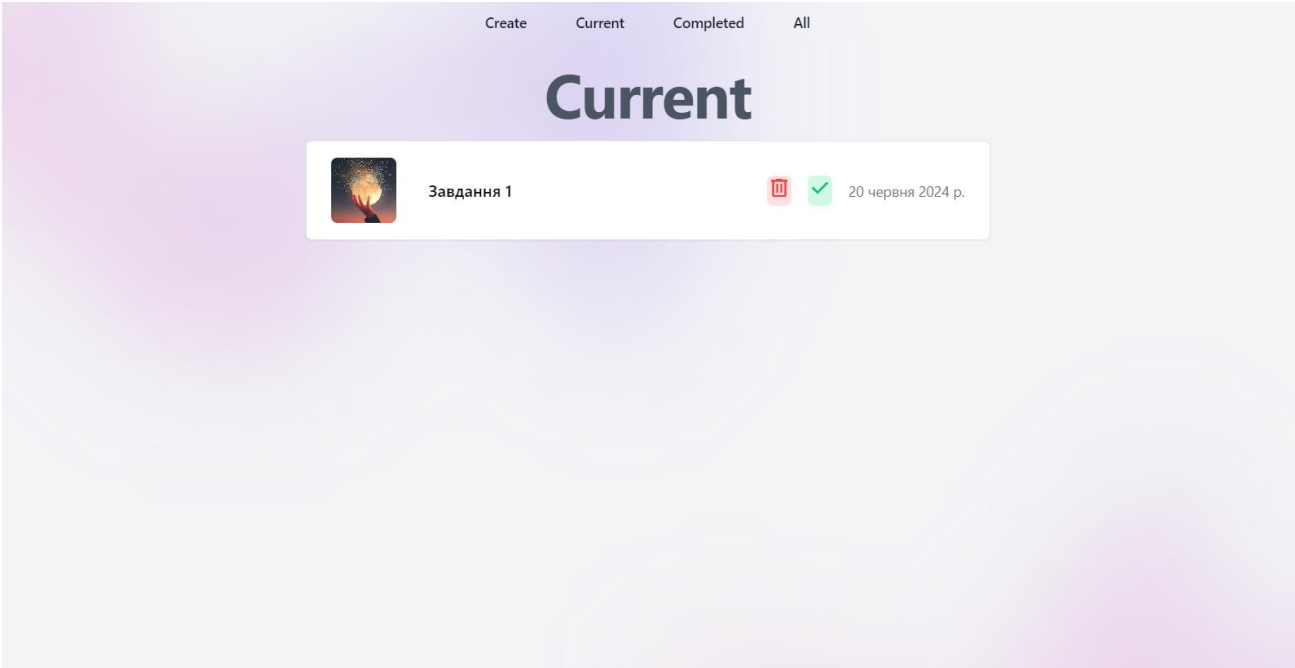


Рисунок 3.10 – Сторінка з поточними завданнями

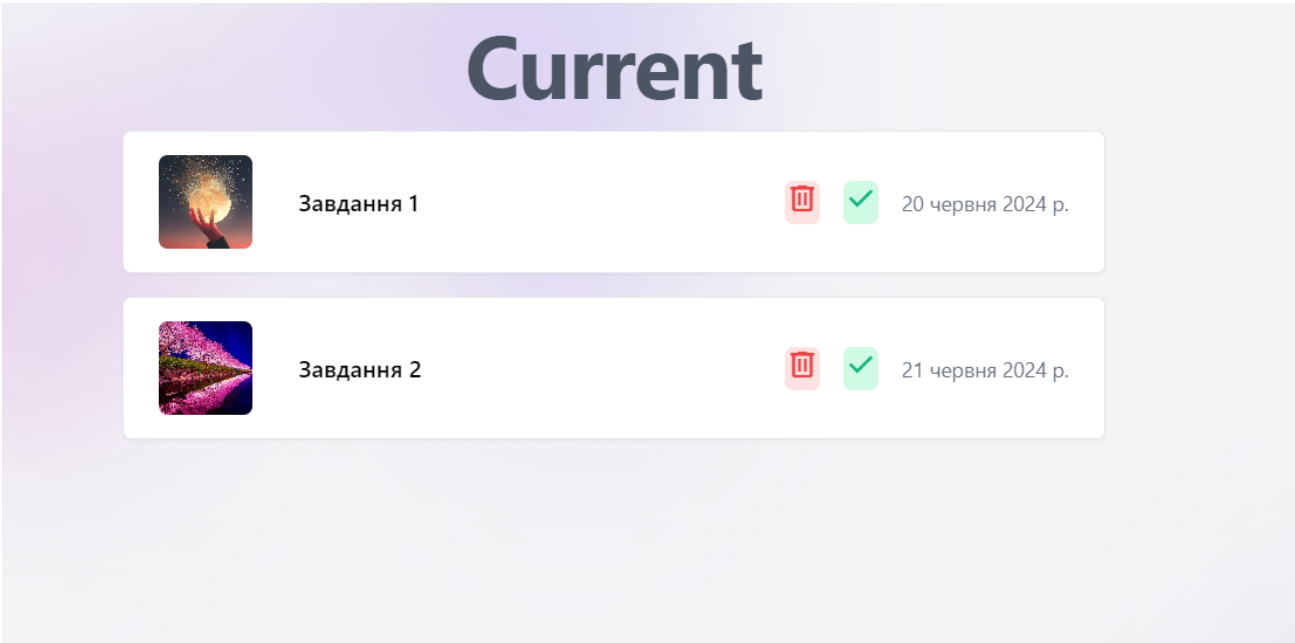


Рисунок 3.11 – Декілька завдань на сторінці

У випадку якщо у завданні потрібно щось змінити можна перейти на сторінку редагування. Відкривається вона натисканням по назві завдання(рис.3.12-3.14).

Редагувати задачу

Image:



Выберите файл Файл не выбран

Title:

Завдання 1

Content:

ВИКОНАТИ

Due date:

ДД.ММ.ГГГГ



Зберегти

Відміна

Рисунок 3.12 – Сторінка редагування

Редагувати задачу

Image:



Выберите файл Файл не выбран

Title:

Редаговане завдання

Content:

перероблено

Due date:

23.06.2024



Зберегти

Відміна

Рисунок 3.13 – Сторінка відредагована

Current



Редаговане завдання

 23 червня 2024 р.

Рисунок 3.14 – Відредаговане завдання на сторінці поточних завдань

Після виконання завдання ми можемо його підтвердити і воно перенесеться на сторінку completed(рис3.15). Якщо завдання видалити, то воно просто зникне(рис3.16). На прикладі двох створених завдань можна побачити, що редаговане завдання, яке ми підтвердили перемістилось на сторінку виконаних, а на сторінці поточних завдань нічого немає, бо завдання 2 було видалено з системи.

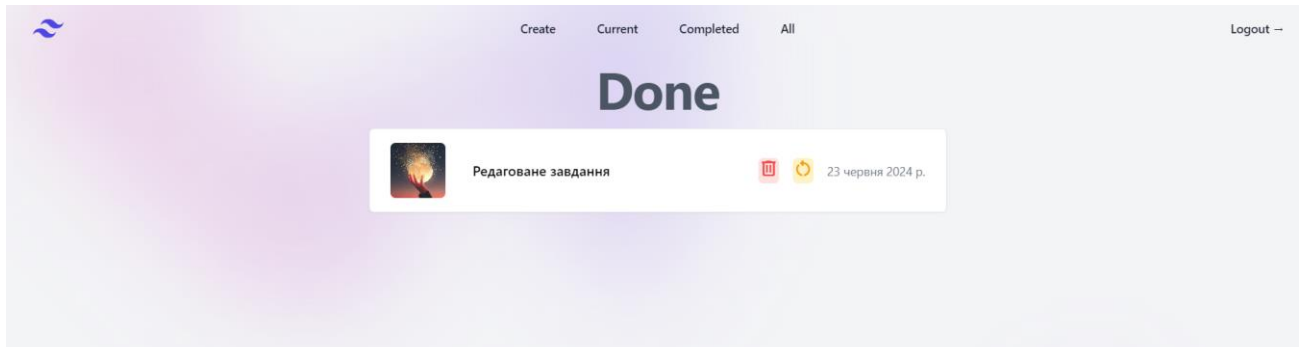


Рисунок 3.15 – Сторінка виконаних завдань

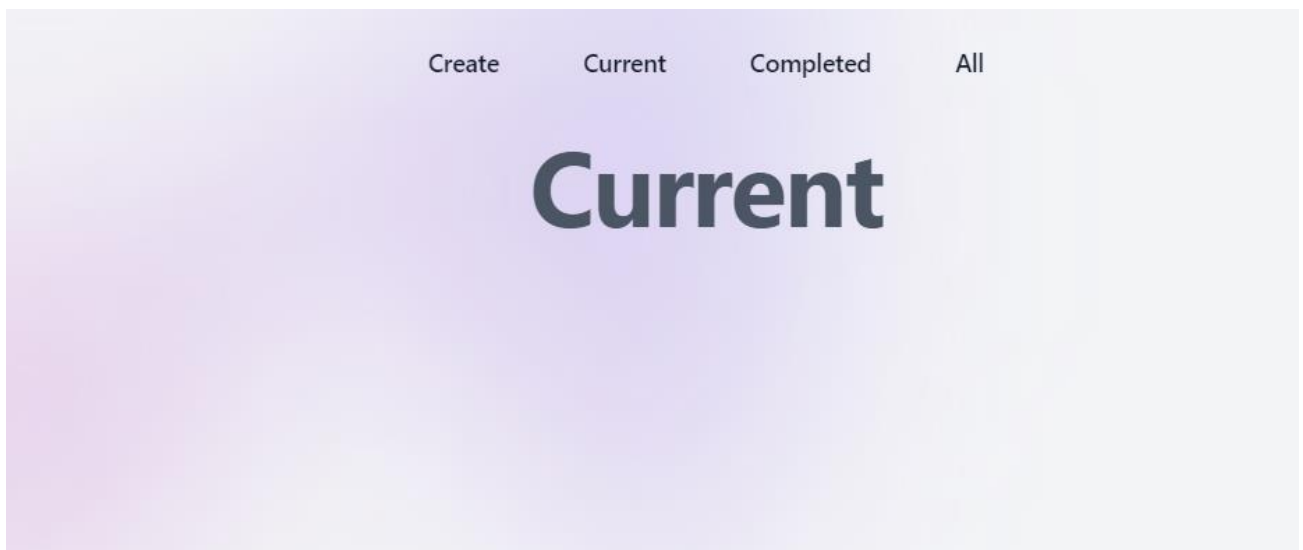


Рисунок 3.16 – Сторінка поточних завдань після проведених дій

На сторінці All(рис.3.17) показано всі поточні завдання і всі виконані завдання

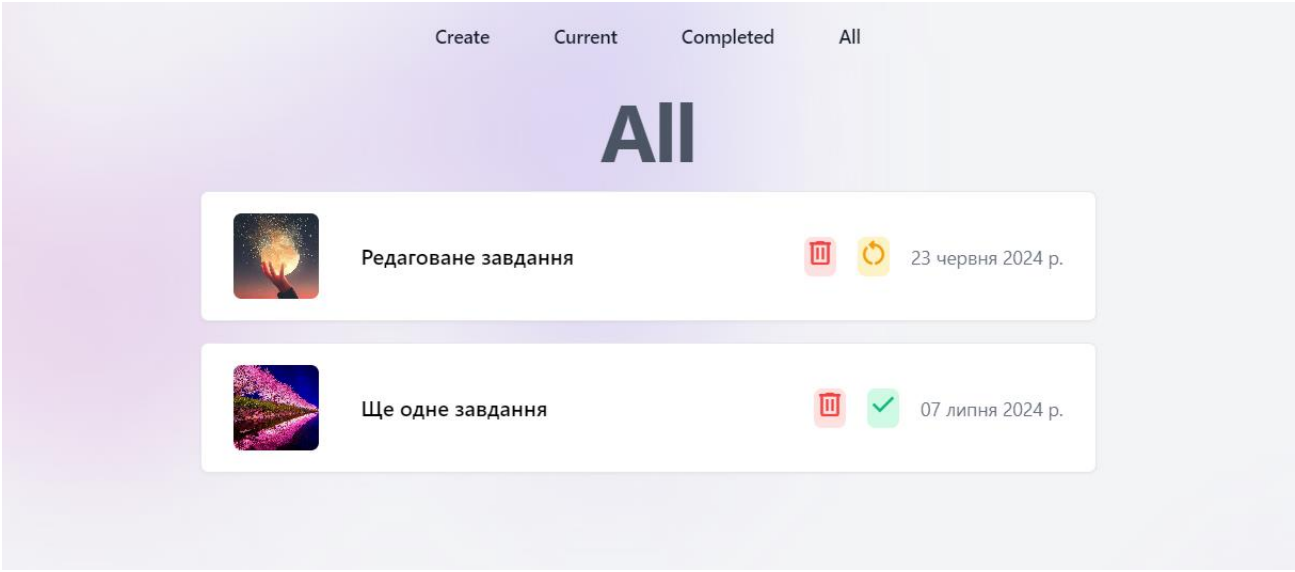


Рисунок 3.17 – Сторінка зі всіма завданнями

Виконані завдання в подальшому можна редагувати і відновлювати(рис.3.18). Після натискання кнопки відновлення задача повернеться на сторінку current(рис.3.19).



Рисунок 3.18 – Кнопка для відновлення завдання

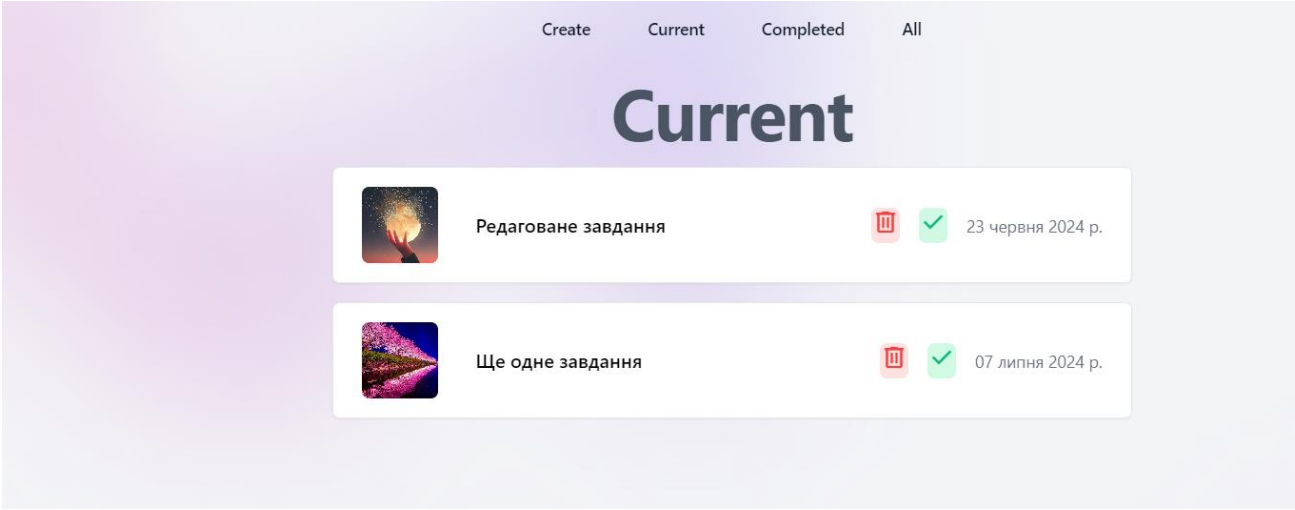


Рисунок 3.19 – Відновлення завдання

3.3 Тестування роботи вебсайту

Тестування програмного забезпечення (Software Testing) - це процес перевірки програмного продукту з метою виявлення дефектів і переконання в його відповідності вимогам і очікуванням користувачів. Цей процес включає в себе виконання програми або системи з метою ідентифікації помилок, визначення їхніх причин і підготовки звіту про результати тестування.

Тестування програмного забезпечення важливо з кількох причин. По-перше, воно допомагає виявити помилки і дефекти в програмному продукті на ранніх етапах розробки, що дозволяє виправити їх до випуску продукту на ринок. Це допомагає зменшити витрати на виправлення помилок у майбутньому і підвищує якість програмного забезпечення.

По-друге, тестування дозволяє підтвердити, що програмний продукт відповідає вимогам і очікуванням користувачів. Це допомагає підтримувати високий рівень задоволеності користувачів і збільшує ймовірність успіху програмного продукту на ринку.

По-третє, тестування дозволяє покращити процес розробки програмного забезпечення, зменшуючи кількість помилок і дефектів, що потребують виправлення. Це дозволяє прискорити час розробки програмного продукту і зменшити його вартість.

Узагальнюючи, тестування програмного забезпечення є важливим етапом у розробці програмного продукту, оскільки воно дозволяє виявити помилки, підтвердити відповідність вимогам і покращити процес розробки. Воно є необхідною складовою успішного програмного продукту і визначає його якість і надійність.

В даному випадку було вирішено використати тестування "чорного ящика" для перевірки програмного забезпечення. Цей метод тестування ґрунтується на тому, що тестувальник не знає внутрішньої реалізації програми і працює лише з її зовнішнім інтерфейсом.

Під час тестування "чорного ящика" тестувальник перевіряє реакцію програми на різні вхідні дані і взаємодію з іншими компонентами системи. Він створює тестові випадки на основі специфікації вимог і очікуваних результатів, не звертаючи уваги на внутрішню реалізацію програми.

Один з головних плюсів тестування "чорного ящика" - це можливість виявлення дефектів і проблем, які можуть виникнути через несправність або невірну взаємодію між компонентами програми. Також цей метод дозволяє тестувальнику думати як потенційний користувач програми, що допомагає покращити її користувацький досвід.

Однак, тестування "чорного ящика" може бути менш ефективним у виявленні складних помилок, що пов'язані з внутрішньою реалізацією програми. Також важливо мати точну і повну специфікацію вимог для ефективного використання цього методу тестування.

У підсумку, тестування "чорного ящика" є важливим методом для перевірки функціональності програмного забезпечення з точки зору його користувачів. Воно дозволяє виявляти дефекти і покращувати якість програми, але потребує точної специфікації вимог і дбайливого підходу до створення тестових випадків.

Під час розробки функціоналу додатку був створений набір тестових сценаріїв, спрямованих на верифікацію коректної роботи його ключових можливостей

1. Реєстрація користувача:

Передумови: Користувач хоче створити обліковий запис.

Кроки:

- Користувач відкриває вікно з реєстрацією;
- Користувач вказує необхідні дані;
- Користувач натискає кнопку "Реєстрація";

Очікуваний результат: Користувач реєструється в системі та зберігається у сесії зі своїм токеном.

Фактичний результат: Користувач реєструється в системі та зберігається у сесії зі своїм токеном.

2. Створення завдання:

Передумови: Користувач хоче додати до списку завдання.

Кроки:

- Користувач відкриває форму створення завдання;
- Користувач заповнює форму і натискає кнопку створити

Очікуваний результат: Завдання успішно створюється і переноситься до списку.

Фактичний результат: Завдання успішно створюється і переноситься до списку.

3. Видалення завдання:

Передумови: Користувачу необхідно видалити завдання через його неактуальність.

Кроки:

- Користувач натискає кнопку видалення;

Очікуваний результат: Завдання успішно зникає зі сторінки створених завдань.

Фактичний результат: Завдання успішно зникає зі сторінки створених завдань.

4. Редагування завдання:

Передумови: Користувач бажає редагувати вже створене завдання.

Кроки:

- Користувач натискає кнопку редагування;
- Користувач редагує форму;

Очікуваний результат: Завдання користувача було перероблено

Фактичний результат: Очікуваний результат: Завдання користувача було перероблено

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						53

5. Відновлення виконаного завдання:

Передумови: Користувач бажає відновити вже виконане завдання.

Кроки:

- Користувач переходить на сторінку виконаних завдань;
- Користувач натискає кнопку відновити;

Очікуваний результат: Завдання успішно відновлено.

Фактичний результат: Завдання успішно відновлено.

Висновки до третього розділу

У третьому розділі було проведено глибокий аналіз та опис аспектів розробки вебдодатку для бухгалтерської компанії. Представлена діаграма розгортання чітко ілюструє архітектуру системи, роблячи її зрозумілою для користувачів. Детально описано інтерфейс користувача, включаючи елементи навігації

Проведено ретельне тестування додатку, що дозволило виявити та виправити можливі помилки, гарантуючи надійну роботу системи. Для наочності представлені приклади тест-кейсів, які перевіряють основні функції системи, такі як авторизація, редагування даних користувачів та додавання новин адміністратором.

В результаті виконаної роботи було створено вебдодаток для оптимізації часу, який відповідає всім поставленим вимогам та очікуванням. Цей вебдодаток дозволить користувачам оптимізувати робочі процеси.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра на тему вебдодаток онлайн-органайзер було здійснено значний прогрес у напрямку розробки та впровадження програмного продукту, спрямованого на покращення управління часом та завданнями. Результати дослідження та розробки вказують на важливість подальшого розвитку таких інструментів для підвищення продуктивності та ефективності в сучасному світі.

Розроблений онлайн-органайзер може бути використаний як в особистих, так і в робочих цілях. Він допоможе користувачам планувати свій час, вносити пріоритети та контролювати виконання завдань, що зробить їх більш організованими та продуктивними.

У майбутньому планується розширення функціоналу онлайн-органайзера шляхом додавання нових можливостей, таких як підтримка повідомлень та нагадувань, інтеграція з календарями та іншими програмами, а також покращення інтерфейсу користувача для більшої зручності використання.

Отже, розробка та дослідження онлайн-органайзера є актуальним та перспективним напрямком у сучасному програмному розвитку. Результати цієї роботи свідчать про можливості покращення управління часом та завданнями за допомогою інноваційних програмних рішень. Дана робота відкриває шлях до подальших досліджень та розвитку в цій галузі, що може призвести до створення ще більш ефективних та зручних інструментів для організації життя та роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Django [Електронний ресурс] - Режим доступу до ресурсу: <https://www.djangoproject.com/start/>
2. Django documentation [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/>
3. Django basics [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/topics/templates/>
4. Django forms and models [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/topics/forms/modelforms/>
5. Django manage.py [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/ref/django-admin/>
6. Django testing [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/topics/testing/>
7. Python [Електронний ресурс] - Режим доступу до ресурсу: <https://www.python.org/>
8. Python Docs [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.python.org/3/>
9. Python tutorial [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.python.org/3/tutorial/index.html>
10. Python controll [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.python.org/3/tutorial/controlflow.html>
11. SQLite [Електронний ресурс] - Режим доступу до ресурсу: <https://sqlite.org/>
12. SQLite source code [Електронний ресурс] - Режим доступу до ресурсу: <https://sqlite.org/src/doc/trunk/README.md>
13. SQLite documentation [Електронний ресурс] - Режим доступу до ресурсу: <https://www.sqlite.org/docs.html>

14. Bootstrap [Электронный ресурс] - Режим доступа до ресурсу: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
15. Bootstrap [Электронный ресурс] - Режим доступа до ресурсу: <https://getbootstrap.com/docs/4.1/content/reboot/>
16. Bootstrap [Электронный ресурс] - Режим доступа до ресурсу: <https://getbootstrap.com/docs/4.1/utilities/borders/>
17. CSS [Электронный ресурс] - Режим доступа до ресурсу: <https://devdocs.io/css/>
18. HTML [Электронный ресурс] - Режим доступа до ресурсу: <https://devdocs.io/html/>
19. TWIG [Электронный ресурс] - Режим доступа до ресурсу: <https://twig.symfony.com/doc/3.x/>
20. Nixon R. Learning PHP, MySQL & JavaScript. Boston, USA, 2021. 709 p.
21. Majida L. A comparative study of laravel and symfony PHP frameworks. Tetouan, Morocco, 2019. 9 p.
22. Sotnik S. Manakov V. Lyashenko V. Overview: PHP and MySQL Features for Creating Modern Web Projects. Washington, USA, 2023. 17p.
23. M'hamed Rahmouni. Model-driven architecture: generating models from Symfony framework. Rabat, Morocco, 2023. 10 p.
24. Gunnard Engebret. Introduction to Symfony. Madison, USA, 2022. 551 p.

ДОДАТКИ

```

from django.shortcuts import render, redirect, get_object_or_404
from django.http import HttpResponse, HttpRequest
from django.contrib.auth.decorators import login_required
from .forms import TodoForm
from .models import TodoList
# Create your views here.

@login_required
def create_todo(request):
    if request.method == 'GET':
        return render(request, 'todo/create_todo.html', {'form': TodoForm()})
    else:
        print('ale')
        # try:
        form = TodoForm(request.POST, request.FILES)
        if form.is_valid():
            print('ale2')

            newtodo = form.save(commit=False)
            newtodo.user = request.user
            newtodo.save()

            return redirect('todo:current')
        else:
            return render(request, 'todo/create_todo.html', {'form': form, 'error':
'Невірно заповнене поле'})
            # except ValueError:
            #     return render(request, 'todo/create_todo.html', {'form': TodoForm(),
'error': 'Невірно заповнене поле'})

@login_required
def edit_todo(request, task_id):
    todo = get_object_or_404(TodoList, id=task_id, user=request.user)
    if request.method == 'GET':
        form = TodoForm(instance=todo)
        return render(request, 'todo/edit_todo.html', {'form': form, 'todo': todo})
    else:
        try:
            form = TodoForm(request.POST, request.FILES, instance=todo)
            if form.is_valid():
                form.save()
                return redirect('todo:current')
            else:
                return render(request, 'todo/edit_todo.html', {'form': form, 'todo':
todo, 'error': 'Невірно заповнене поле'})
        except ValueError:
            return render(request, 'todo/edit_todo.html', {'form': form, 'todo':
todo, 'error': 'Невірно заповнене поле'})

@login_required
def mark_as_done(request, task_id):
    todo = TodoList.objects.filter(id=task_id)
    todo.update(done=True)
    return redirect('todo:current')

```

```

@login_required
def mark_as_current(request, task_id):
    todo = TodoList.objects.filter(id=task_id)
    todo.update(done=False)
    return redirect('todo:done')

@login_required
def delete_todo(request, task_id):
    todo = TodoList.objects.filter(id=task_id)
    todo.delete()
    return redirect('todo:current')

# -----

@login_required
def current_todo(request):
    todos = TodoList.objects.filter(user=request.user, done=False).order_by('-created')
    context = {
        'name': 'Current',
        'todolist': todos
    }
    return render(request, 'todo/todo.html', context)

@login_required
def all_todo(request):
    todos = TodoList.objects.filter(user=request.user).order_by('-created')
    context = {
        'name': 'All',
        'todolist': todos
    }
    return render(request, 'todo/todo.html', context)

@login_required
def done_todo(request):
    todos = TodoList.objects.filter(user=request.user, done=True)
    context = {
        'name': 'Done',
        'todolist': todos
    }
    return render(request, 'todo/todo.html', context)

```

Додаток Б

```

from django.contrib.auth import login, authenticate
from django.contrib.auth.forms import AuthenticationForm
from django.shortcuts import render, redirect
from .forms import SignUpForm, LoginForm
from django.contrib import auth, messages
from django.contrib.auth.decorators import login_required
from django.urls import reverse
from .forms import LoginForm, SignUpForm
from django.http import HttpResponseRedirect

def signup_view(request):
    if request.method == 'POST':
        form = SignUpForm(data=request.POST)
        if form.is_valid():
            form.save()
            user = form.instance

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						60

```

        auth.login(request, user)
        return HttpResponseRedirect(reverse('main:index'))
    else:
        form = SignUpForm()
        return render(request, 'users/signup.html', {'form': form})

def login_view(request):
    if request.method == 'POST':
        form = LoginForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('main:index')
        else:
            form = LoginForm()
            return render(request, 'users/login.html', {'form': form})

# Create your views here.
def index(request):
    context = {
        "title": 'Online Organizer',
    }
    return render(request, 'users/index.html', context)

@login_required
def logout(request):
    auth.logout(request)
    return redirect(reverse('main:index'))

```

Додаток В

```

{%load static%}

<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
      {% block title %}{% endblock title %}
    </title>

    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FIL
LL,GRAD@24,400,0,0" />
    <link rel="stylesheet" href="{% static 'css/output.css' %}">
    <link
href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
rel="stylesheet">
    <script src="https://kit.fontawesome.com/a076d05399.js"
crossorigin="anonymous"></script>
  </head>
  <body>
    <header class="absolute inset-x-0 top-0 z-50">
      <nav class="flex items-center justify-between p-6 lg:px-8"
        aria-label="Global">
        <div class="flex lg:flex-1">
          <a href="#" class="-m-1.5 p-1.5">
            <span class="sr-only">Online Organizer</span>
            
  </a>
</div>
<div class="flex lg:hidden">
  <button type="button"
    class="-m-2.5 inline-flex items-center justify-center
rounded-md p-2.5 text-gray-700">
    <span class="sr-only">Open main menu</span>
    <svg class="h-6 w-6" fill="none" viewBox="0 0 24 24"
      stroke-width="1.5" stroke="currentColor"
      aria-hidden="true">
      <path stroke-linecap="round" stroke-linejoin="round"
        d="M3.75 6.75h16.5M3.75 12h16.5m-16.5 5.25h16.5" />
    </svg>
  </button>
</div>
<div class="hidden lg:flex lg:gap-x-12">
  <a href="{% url 'todo:create' %}"
    class="text-sm font-semibold leading-6 text-gray-
900">Create</a>
  <a href="{% url 'todo:current' %}"
    class="text-sm font-semibold leading-6 text-gray-
900">Current</a>
  <a href="{% url 'todo:done' %}"
    class="text-sm font-semibold leading-6 text-gray-
900">Completed</a>
  <a href="{% url 'todo:all' %}"
    class="text-sm font-semibold leading-6 text-gray-900">All</a>

</div>
<div class="hidden lg:flex lg:flex-1 lg:justify-end">
  {% if not user.is_authenticated %}

  <a href="{% url 'users:login'%}"
    class="text-sm font-semibold leading-6 text-gray-900">Log in
    <span aria-hidden="true">&rarr;</span></a>
  {% else %}

  <a href="{% url 'users:logout'%}"
    class="text-sm font-semibold leading-6 text-gray-900">Logout
    <span aria-hidden="true">&rarr;</span></a>
  {% endif %}

</div>
</nav>
<!-- Mobile menu, show/hide based on menu open state. -->
<div class="lg:hidden" role="dialog" aria-modal="true">
  <!-- Background backdrop, show/hide based on slide-over state. -->
  <div class="fixed inset-0 z-50"></div>
  <div
    class="fixed inset-y-0 right-0 z-50 w-full overflow-y-auto bg-
white px-6 py-6 sm:max-w-sm sm:ring-1 sm:ring-gray-900/10">
    <div class="flex items-center justify-between">
      <a href="#" class="-m-1.5 p-1.5">
        <span class="sr-only">Your Company</span>
        <img class="h-8 w-auto"

```

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						63