Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: розробка сервісу-портфоліо для розробників програмного забезпечення з використанням Flask

Виконав: студент 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Монастирський М.О.
	(підпис)	(прізвище та ініціали)
Керівник		Михалик Д. М.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Розробка сервісу-портфоліо для розробників програмного забезпечення з використанням Flask» написана Монастирським Михайлом Олеговичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, Факультет комп'ютерно- інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42.

Монастирський М. О. Розробка сервісу-портфоліо для розробників програмного забезпечення з використанням Flask: кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю «121 – Інженерія програмного забезпечення» / Монастирський Михайло Олегович. – Тернопіль: ТНТУ, 2025. Сторінок – 51, рисунків – 19, таблиць – 3, частин – 5, додатків – 3, посилань – 17, лістингів – 5.

Метою кваліфікаційної роботи є створення ефективного сервісу-портфоліо для розробників програмного забезпечення, який автоматизує процес презентації професійних навичок, проєктів та досвіду користувачів. Завданням було забезпечити зручне управління даними користувачів, можливість додавання, редагування та демонстрації інформації про їхні проєкти, а також забезпечення безпеки та надійності збереження даних.

В результаті проведеної роботи було не лише досягнуто поставленої мети, а й створено масштабований сервіс з можливістю подальшого розширення функціоналу та інтеграції нових можливостей.

У розробці були використані сучасні технології та інструменти: мови програмування Python та JavaScript, веб-фреймворки Flask для серверної частини, система керування базами даних SQLite, а також допоміжні інструменти для розробки і тестування API — Postman.

Ключові слова: портфоліо-сервіс, Flask, веб-додаток, Python, API, SQLite.

ABSTRACT

Bachelor's qualification work on the topic "Development of a service-portfolio for software developers using Flask" was written by Monastyrsky Mykhailo Olehovych, a student of Ivan Pulyuy Ternopil National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group SP-42.

Monastyrsky M. O. Development of a service-portfolio for software developers using Flask: qualification work for obtaining a bachelor's degree in the specialty "121 - Software Engineering" / Monastyrsky Mykhailo Olehovych. – Ternopil: TNTU, 2025. Pages – 45, figures – 19, tables – 3, parts – 5, appendices – 3, references – 17, listings – 5.

The purpose of the qualification work is to create an effective portfolio service for software developers that automates the process of presenting professional skills, projects and user experience. The task was to provide convenient management of user data, the ability to add, edit and display information about their projects, as well as ensure the security and reliability of data storage.

As a result of the work, not only the set goal was achieved, but also a scalable service was created with the possibility of further expanding the functionality and integrating new features.

Modern technologies and tools were used in the development: Python and JavaScript programming languages, Flask web frameworks for the server side, SQLite database management system, as well as auxiliary tools for developing and testing APIs - Postman.

Keywords: portfolio service, Flask, web application, Python, API, SQLite.

Перелік умовних позначень, символів, скорочень і термінів

Flask – фреймворк для розробки веб-додатків на мові програмування Python.

ІТ – інформаційні технології

ПЗ – програмне забезпечення

БД – база даних

ВВ – варіанти використання

URL – уніфікований покажчик ресурсу

HTTP – протокол передачі гіпертексту

HTML – мова розмітки гіпертексту

Python – мова програмування, один з методів використання якої є конструювання веб-додатків.

CSS – каскадні таблиці стилів

JavaScript – мова програмування, яка використовується для створення динамічних та інтерактивних елементів на веб-сторінках.

JSON – текстовий формат обміну даних

ORM – об'єктно-реляційне відображення

API – інтерфейс програмування застосунків

CSRF – підробка міжсайтових запитів

HTTPS – безпечний протокол передачі гіпертексту

SSL – рівень захищених пакетів

ПК – персональний комп'ютер

Фронтенд – клієнтська частина системи.

Бекенд – серверна частина системи.

ЗМІСТ

ВСТУП

Сучасні технології за останні кілька років почали займати дедалі вагоміше й вагоміше місце у всій сфері людської діяльності, а питання присутності в інтернеті набуло чималого значення. Для спеціалістів у сфері IT, а тимпаче програмістів важливим елементом професійного зростання є можливість ділитись своїми проектами та навичками. Одним з найбільших та найлегших способів зарекомендувати себе як хорошого спеціаліста є створення свого портфоліо в електронному вигляді

Портфоліо в інтернеті являє собою платформу яка дає можливість розробникам ПЗ зберігати приклади своїх власних досягнень, давати опис проєктів, завантажувати скріншоти та архіви з вихідним кодом, вести профіль, та багато іншого. У порівнянні із звичайним резюме чи анкетою інтернет-портфоліо є інтерактивним засобом самопрезентації який полегшує взаємодію з потенційним замовником.

Мета даної роботи – це розробка повноцінного веб-сервісу який надає можливість створення публічного та персонального портфоліо для програмістів. Сервер має підтримувати авторизацію користувачів, організацію та структуризацію інформації про їхні навички. Окрім цього сервіс має приймати мультимедійні файли (зображення) та архіви, а також бути простим і зрозумілим для людей. Адаптація для різних пристроїв також має бути присутня.

Проєкт реалізовано мовою програмування Python на якій написана серверна частина, мікрофреймворком Flask, мовою розмітки сторінок HTML, стилями CSS та базою даних SQLite. Flask дозволяє швидко створювати веб-додатки з чіткою структурою, забезпечує гнучкість у налаштуванні маршрутизації та форм. Системою управління базою даних обрано SQLite яка є легкою та надійною та ідеально підходить для невеликих проєктів.

Для сервісу розроблено кілька ключових моментів такі як:

– реєстрація;

- вхід;
- захищене зберігання паролів;
- додавання проєктів;
- прикріплення файлів;
- зміна аватару;
- можливість перегляду портфоліо іншими користувачами.

Актуальність розробки зумовлена постійним зростанням фахівців IT-сфери, які мають не лише володіти технічними навичками, а й вміти грамотно презентувати себе. Для студентів, початківців і навіть досвідчених розробників важливо мати доступний і зручний інструмент для демонстрації свого прогресу. Наявні сервіси, такі як GitHub або LinkedIn, не завжди надають можливість кастомізованої візуалізації проєктів, а деякі потребують знань у веброзробці для самостійного створення сайту. Запропонований у роботі сервіс є компромісним рішенням між повною кастомізацією та зручністю використання.

Кваліфікаційна робота включає повний цикл проєктування та реалізації вебдодатку: від аналізу потреб цільової аудиторії та постановки задач — до написання коду, тестування, оформлення інтерфейсу й підготовки до публічного представлення. Під час розробки особливу увагу приділялося модульності коду, читабельності, а також можливості подальшого масштабування функціональності. Реалізований прототип може слугувати як основа для повноцінного комерційного або освітнього продукту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ОПИС ЕТАПІВ РОЗРОБКИ СИСТЕМИ

1.1. Аналіз предметної області й актуальність теми

У сфері IT питання постає питання професійної самопрезентації яке набуває особливого значення. Постійно зростаюча конкуренція серед фахівців в даній області окрім наявності знань, вимагає зданість ефективно демонструвати свої проєкти та досвід. Найпоширенішим способом є створення власного портфоліо.

Сервіси по типу GitHub, LinkedIn або персональні сайти частково виконують функцію портфоліо, але мають чимало обмежень. До прикладу GitHub не дозволяє візуально структурувати опис проєктів, а LinkedIn – не призначений для зберігання файлів із кодом, а створення персонального сайту потребує знань у фронтенд-розробці [1, 2].

Виходячи із цього актуальним питанням є створення окремого сервісу який орієнтується саме на програмістів який буде демонструвати реальні навички користувача

У межах даної предметної області портфоліо розгляладається як інструмент:

- для фіксації особистих досягнень та проєктів;
- для демонстрації технічних навичок;
- як посилання при пошуку робити на фрілансі;
- як навчальний ресурс для інших програмістів;

Такий сервіс актуальний для:

- студентів;
- молодих спеціалістів;
- досвідчених розробників;

1.2. Постановка завдання та аналіз вимог до системи

Основним завданням даної роботи є можливість легко створювати та редагувати публічне портфоліо програміста. Сервіс має забезпечити можливість зберігання інформації про реалізовані проєкти та навички, додавання мультимедійного вмісту та архіви вихідного коду, тощо.

Інтерфейс проєкту має бути легким у використанні для новачків, а також для досвідчених спеціалістів. Сервіс має працювати стабільно та мати сучасний інтерфейс який адаптований до різних пристроїв.

Після детального аналізу було виділено такі функціональні вимоги: реєстрація та вхід користувачів, управління профілем, робота з проєктами, публічне портфоліо, адаптивний інтерфейс, базова безпека.

Нефункціональні вимоги:

– Продуктивність: система повинна працювати швидко, без затримок та помилок з оптимальним використанням ресурсів.

– Надійсність: сервіс має функціонувати безперебійно.

– Зручне використання: інтерфейс повинен бути інтуїтивно зрозумілим, адаптивним та не перевантаженим.

Вимоги до захисту та доступу:

– Контроль доступу: система повинна обмежувати підключення лише до обмежених IP-адресів.

– Білий список: має бути доступним для керування адміністратором.

– Блокування: усі спроби доступу з неавторизованих IP-адресів мають блокуватись автоматично [7].

Вимоги до інтерфейсу системи:

– Адаптивність: інтфрейс повинен підлаштовуватись під будь-який розмір екрану.

– Дизайн: мінімалістичне оформлення [5].

– Зручність: інтерфейс має бути структурованим і доступним для користувачів без підготовки.

Вимоги до роботи з даними:

– цілісність даних;

– транзакційність;

– валідація введення [3, 5].

1.3. Основні етапи розробки системи

Процес розробки вебсервісу-портфоліо, було розділено на кілька логічних етапів. Кожен етап охоплював певний спектр завдань і мав чітко визначену мету. Даний підхід дозволив забезпечити поступовий розвиток функціональності та контроль якості на всіх етапах розробки.

Варто підмітити що незалежно від обраної методології розробки ПЗ, створення додатку повинне включати в себе такі етапи:

– Аналіз вимог: детальний опис всіх вимог які стоять перед проектом для розуміння поставлених цілей.

– Проектування архітектури: етап який включає в себе розробку структури системи та визначення основних елементів.

– Розробка та програмування: реалізація серверної логіки яка обробляє запити від клієнта, управління БД, тощо [1, 2, 5].

– Тестування: проведення тестів для перевірки коректної роботи системи.

– Впровадження та підтримка: розгортання проекту.

Дослідивши методології розробки, обрано каскадну модель для розробки ПЗ [10]. Це пояснено послідовним підходом до роботи над проектом впродовж всієї роботи: від аналізу вимог до самого розгортання проекту.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Пошук акторів та варіантів використання

Для того щоб ретельно проаналізувати предметну область та визначити всі її вимоги я виділив одного головного актора – User (авторизований користуач). User це центральна фігура в системі яка постійно взаємодіє із системою. Для цього актора доступні всі сценарії використання які є у діаграмі наведені на рисунку 2.1.

Рисунок 2.1 – Діаграма варіантів використання актора User

Окрім актора User є актор Guest (неавторизований користувач) якому доступні сценарії використання які наведенні на рисунку 2.2

Рисунок 2.2 – Діаграма варіантів використання актора Guest

2.2. Опис варіантів використання

Проаналізувавши діаграми ВВ, зображену на рисунку 2.1 та 2.2, подальший крок передбачає детальне розглядання кожного варіанту використання системи для кожного актора. Для кращого розуміння кожного сценарію взаємодії кожен варіант буде розглянуто окремо. Цей аналіз дозволить глибше зрозуміти різноманітність можливостей використання системи та їх вплив на її функціональність.

Для актора User доступні наступні сценарії:

– Редагувати профіль – даний варіант дозволяє користувачу вносити зміни у своєму профілі, а саме зміна поля bio, та завантаження аватару.

– Додати проєкт – даний варіант дозволяє користувачу додавати проєкти для портфоліо з подальшим відображенням у профілі. При додаванні проєкту користувач завантажує архів проєкту, скріншоти та наводить опис проєкту.

– Видалити проєкт – користувач має можливість видалити проєкт з портфоліо.

– Перегляд свого портфоліо – користувач може переглянути список завантажених проєктів.

– Заванатажити проєкт – користувач має можливість завантажити на свій пристрій архів з файлами проєкту та відкрити його.

Для актора Guest доступні наступні варіанти використання:

– Перегляд портфоліо автора – даний варіант дозволяє неавторизованому користувачу переглянуди розміщені на сайті сторінки портфоліо.

– Перегляд проєктів – актор Guest будучи не авторизованим у системі може перглядати розміщені на сервісі проєкти.

– Завантаження проєкту – актор має можливість завантажити файли проєкту та зі свого ПК переглянути роботу проєкту.

2.3. Архітектура системи

Розроблена система для сервісу-портфоліо реалізована за принципами клієнт-серверної архітектури, що дозволяє користувачу взаємодію з веб-інтерфейсом, який відправляє HTTP-запити на серверну частину яка реалізована за допомогою фреймфворку Flask. На сервері обробляються запити, проходить взаємодія із базою даних, та дається відповідь у вигляді HTML-сторінок або JSON-даних [2].

Система складається з наступних компонентів:

– Клієнтська частина (Frontend).

– Сервена частина (Backend).

– База даних.

– ORM-рівень [3].

Серед основних переваг:

– Мінімалістичність та гнучкість.

– Простота розгортання.

– Інтуїтивний інтерфейс користувача.
– Адаптивність [5].
– Розширюваність.
– Безпека.
– Можливість масштабування.

На рисунку 2.2 зображено загальну схему архітектури розробленої системи

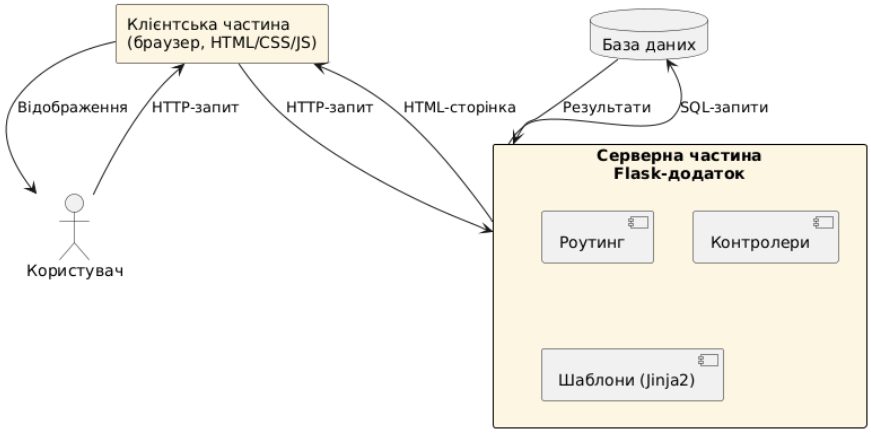

Рисунок 2.2 – Загальна схема архітектури системи

2.4. Вибір технологій та інструментальних засобів розробки системи

При розробці веб-системи було проаналізовано низку сучасних технологій та засобів які використовують у створенні повнофункціональних веб-застосунків. Основними критеріями вибору за якими я рухався були простота у використанні, гнучкість та відповідність функціональним вимогами.

У таблиці 2.1 наведено основні технологій що були обрані мною для реалізації проєкту.

Таблиця 2.1 – Обрані технології

№	Компонент системи	Технологія	Призначення
1	Серверна частина	Python + Flask	Для побудови веб-додатків [2]
2	Рендеринг інтерфейсу	Jinja2	Для динамічного формування HTML-сторінок [2]
3	База даних	SQLite	Для зберігання даних користувачів [4]
4	Зв'язок з БД	SQLAlchemy	Для зручної взаємодії з базою даних [3]
5	Клієнтська частина	HTML, CSS, JavaScript	Для побудови інтерфейсу користувача [1]
6	Контроль версій	Git	Для відстеження змін у коді
7	Розгортання	Gunicorn	Для запуску застосунку
8	Середовище розробки	VS Code	Для написання та налагодження коду

Такий вибір технологій зубомовлений простою у використанні, гнучкістю та адаптивністю.

Такий спектр вибору технологій дозволяє створити масштабовану та легкопідтримувану зрозумілу систему.

2.5. Визначення класів системи

На основі аналізу системи, її вимог і складових було визначено класи додатку й продемонстровано у діаграмі класів (рис. 2.3)

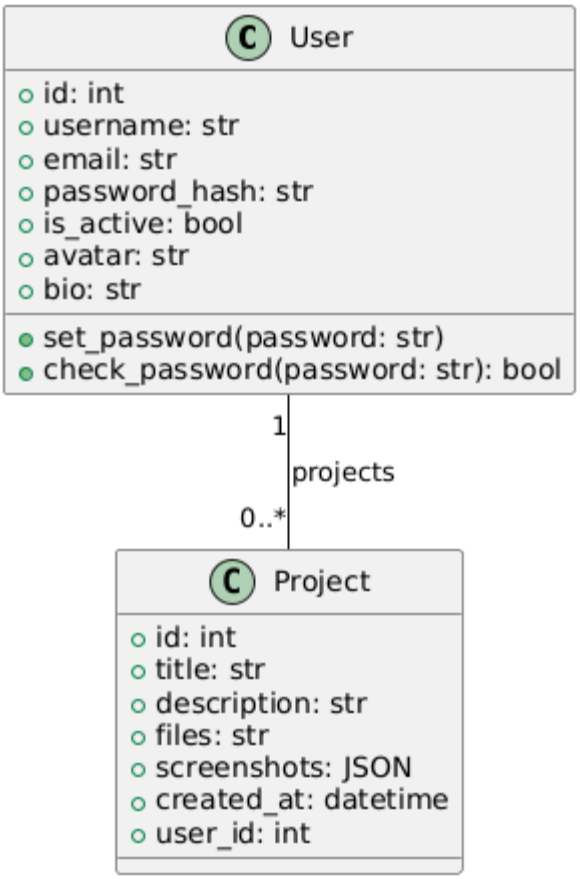

Рисунок 2.3 – Діаграма класів системи

З діаграми видно, що кожен описаний клас має свої набори атрибутів та методів.

Основними класами системи є Project та Use. Між класами User та Project існує зв'язок один до багатьох, тобто користувач може мати декілька проектів. Цей зв'язок реалізований за допомогою ключа user_id у класі Project.

2.6. Керування даними

У системі портфоліо одну з найважливіших ролей відіграє зберігання даних, обробка та їх захист. Для цього мною було реалізовано ефективну та безпечну модель керування даними з використанням сучасних технологій та підходів.

Основними категоріями даними є

- Персональні дані користувачів.
- Дані про проєкти.
- Медіафайли.
- Метадані.

Для збереження даних використовується реляційна база даних SQLite. Таблиці реалізовані за допомогою SQLAlchemy, що забезпечує легку роботу з базою без необхідності писати SQL-запити [4].

SQLAlchemy дозволяє:

- Визначити таблиці як класи
- Здійснювати звязки як сутності
- Виконувати операції через код.

Дані обробляються на стороні сервера у контролерах (перевірка форм введення, хешування паролів, тощо) [7].

Таким чином система забезпечує надійне керування даними з дотриманням вимог безпеки та зручності.

3 РОЗРОБКА СИСТЕМИ

3.1. Створення бази даних

Початковим етапом у розробці системи мною було обрано створення бази даних, оскільки від правильності виконання залежить ефктивне зберігання та доступ до інформації.

Після вдалого проектування та визначення структури бази даних безпосередньо було її створення. Для її реалізації було використано ORM-бібліотеку, а саме SQLAlchemy, що входить до складу системи Flask [3]. Створення бази даних наведено у наведено у лістингу 3.1.1.

```
# Створення директорії instance (для бази)
instance_path = os.path.join(os.getcwd(), 'instance')
os.makedirs(instance_path, exist_ok=True)

# Створення бази даних, якщо не існує
with app.app_context():
    db_path = os.path.join('instance', 'users.db')
    if not os.path.exists(db_path):
        db.create_all()
```

Лістинг 3.1.1 – Створення бази даних

У системі реалізовано 2 основні таблиці:

– User;

– Project.

Оскільки структура бази даних є визначеною, її реалізація наведна у вигляді класів у лістингах 3.1.2 та 3.1.3.

```
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True,
```

```
nullable=False)
        email     =     db.Column(db.String(100),     unique=True,
nullable=False)

        password_hash = db.Column(db.String(200),nullable=False)
# зберігаємо хеш пароля
        is_active = db.Column(db.Boolean, default=True)
        avatar = db.Column(db.String(120))  # Якщо користувач має
аватар
        bio = db.Column(db.String(500), nullable=True)
```

Лістинг 3.1.2 – Створення моделі користувача

```
    class Project(db.Model):
        __tablename__ = 'project'
        id = db.Column(db.Integer, primary_key=True)
        title = db.Column(db.String(100), nullable=False)
        description = db.Column(db.Text, nullable=False)
        files = db.Column(db.String(150), nullable=True)  # архів
ZIP
        screenshots = db.Column(db.JSON, nullable=True)   # список
скріншотів у JSON (підтримується PostgreSQL)
        created_at            =            db.Column(db.DateTime,
default=datetime.utcnow)
        user_id = db.Column(db.Integer, db.ForeignKey('user.id'),
nullable=True)
        user   =   db.relationship('User',   backref='projects',
lazy=True)
```

Лістинг 3.1.3 – Створення моделі користувача

Після проведних операцій було отримано готову базу даних з усіма потрібними класами. Результат виконаної роботи зображений на рисунку 3.1.1.

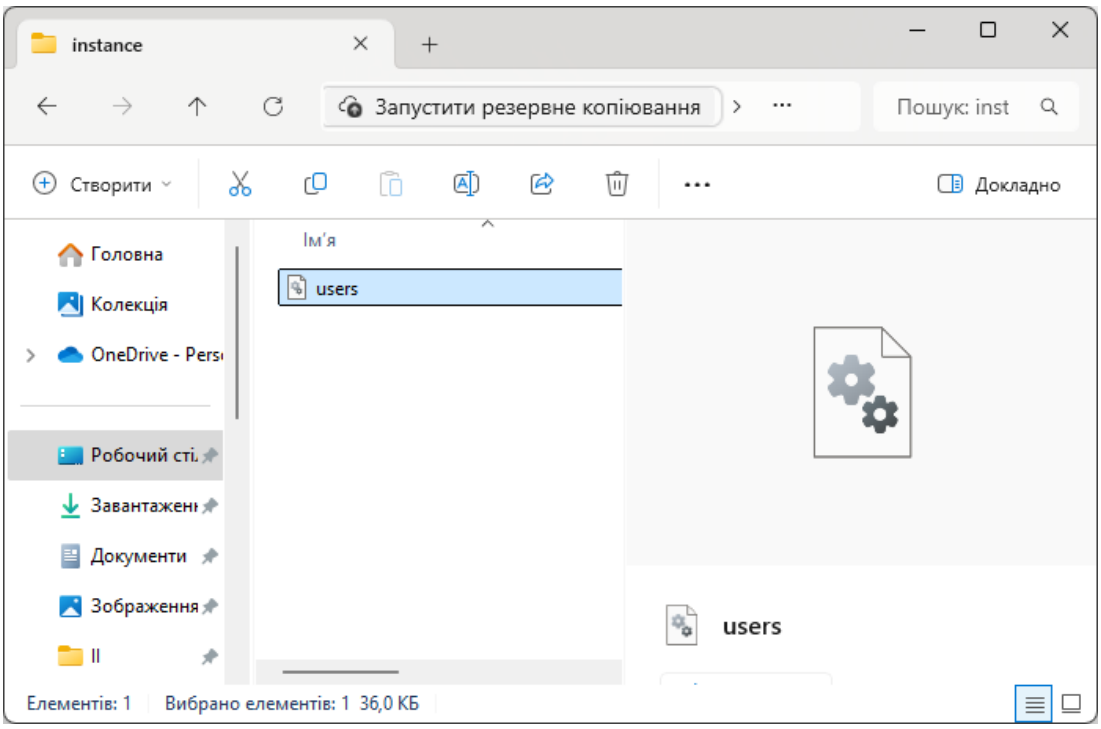

Рисунок 3.1.1 – Створена база даних

3.2. Організація середовища розробки серверної частини

Як сказано раніше, для бекенд-розробки була використана мова програмування Python, зокрема її мікро-фреймворк Flask [6, 1].

Оскільки мова Python специфічна бо для роботи у проекті вимагає наявність віртуального середовища, перед початком роботи було ініціалізовано віртуальне середовище за допомогою команди «python -m myenv6/project/flask_server»

Після успішного виконання команди, у терміналі Visual Studio Code активовано віртуальне середовище Python відкриттям файлу activate, що знаходиться за шляхом .\venv\Scripts\Activate.ps1

Далі було інстальовано фреймворк Flask через відкрите віртуальне середовище за допомогою команди «pip install Flask».

3.3. Етапи розробки серверної частини

При етапі конструювання серверу було визначено кілька ключових етапів:

– Створення базового додатку
– Піідключення бази даних
– Створення класів
– Розробка маршрутів

Даний підхід дозволи послідовно реалізувати функціональність системи для досягнення поставлених задач.

3.4. Конструювання серверної частини

Для виконання заданих завдань створено базовий Flask-додаток, лістинг якого навединий у лістингу 3.4.1

```
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True, use_reloader=False)
```

Лістинг 3.4.1 – Створення базового додатку

Далі було додано логіку ініцілізації бази даних, лістинг якої навединий у лістингу 3.4.2

```
    db.init_app(app)
```

Лістинг 3.4.2 – ініцілізація бази даних

Для кожної сутності системи було створено моделі та форми які розділені по окремих файлах з назвами, що відповідають їхній функції.

Фінальним етапом конструювання серверу була розробка методів маршрутизації та функцій додатку.

При аналізі функціональниї вимог були визначені основні маршрути наведені на рисунку 3.4.1

```
@main.route('/')
def home():···

@main.route('/about')
def about():···

@main.route('/login', methods=['GET', 'POST'])
def login():···

@main.route('/signup', methods=['GET', 'POST'])
def signup():···

@login_manager.user_loader
def load_user(user_id):···

@main.route('/profile', methods=['GET', 'POST'])
@login_required
def profile():···

@main.route('/profile/edit', methods=['GET', 'POST'])
@login_required
def edit_profile():···

def allowed_file(filename):···

@main.route('/upload-avatar', methods=['GET', 'POST'])
def upload_avatar():···

@main.route('/delete-avatar', methods=['POST'])
def delete_avatar():···

def save_file(file):···

@main.route('/create_project', methods=['GET', 'POST'])
def create_project():···

@main.route('/project/<int:project_id>')
def view_project(project_id):···

@main.route('/download_zip/<int:project_id>')
def download_zip(project_id):···

@main.route('/logout')
def logout():···
```

Рисунок 3.4.1 – Основні функції та маршрути системи

Кожен із наведних на рисунку 3.4.1 маршрутів є реалізацією окремих варіантів використання застосунку. Керування даними проходить безпосередньо через прямі запити до бази даних.

Однією з ключових функцій є функцій додавання проекту наведна на рисунку 3.4.2 що реалізовує можливість користувача завантажити проєкт для портфоліо.

```python
@main.route('/create_project', methods=['GET', 'POST'])
def create_project():
    form = CreateProjectForm()

    if form.validate_on_submit():
        title = form.title.data
        description = form.description.data
        # Збереження ZIP файлу
        zip_file = form.files.data
        unique_zip_name = None
        if zip_file:
            filename = secure_filename(zip_file.filename)
            unique_zip_name = f"{uuid.uuid4().hex}_{filename}"
            zip_path = os.path.join(current_app.config['UPLOAD_FOLDER'], unique_zip_name)
            zip_file.save(zip_path)

        # Збереження скріншотів
        screenshots_files = form.screenshots.data
        saved_screenshots = []
        if screenshots_files:
            for screenshot in screenshots_files:
                filename = secure_filename(screenshot.filename)
                unique_screenshot_name = f"{uuid.uuid4().hex}_{filename}"
                screenshot.save(os.path.join(current_app.config['UPLOAD_SCREENSHOTS_FOLDER'], unique_screenshot_name))
                saved_screenshots.append(unique_screenshot_name)

        # Створення нового проекту
        new_project = Project(
            title=title,
            description=description,
            files=unique_zip_name,
            screenshots=saved_screenshots,  # Це список імен файлів
            created_at=datetime.utcnow(),
            user_id=current_user.id if current_user.is_authenticated else None
        )
        db.session.add(new_project)
        db.session.commit()

        return redirect(url_for('main.profile'))

    return render_template('create_project.html', form=form)
```

Рисунок 3.4.2 – Реалізація маршруту create_project()

Також ключовими можливостями є вхід та реєстрація користувача у системі, реалізація чого наведена нижче у рисунках 3.4.3 та 3.4.4.

```python
@main.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()

    if form.validate_on_submit():
        # Перевірка наявності користувача за email
        user = User.query.filter_by(email=form.email.data).first()
        # Перевірка пароля за допомогою методу check_password
        if user and user.check_password(form.password.data):
            login_user(user)

            # Отримуємо параметр 'next' з форми
            next_page = request.args.get('next')

            # Якщо 'next' відсутній або пустий, встановлюємо дефолтний напрямок
            if not next_page or not is_safe_url(next_page):
                next_page = url_for('main.profile')  # Наприклад, на сторінку профілю

            current_app.logger.info(f"Отримано параметр 'next': {next_page}")

            # Після успішного входу перенаправляємо на сторінку, вказану в 'next'
            return redirect(next_page)

    return render_template('login.html', form=form)
```

Рисунок 3.4.3 – Реалізаця маршруту login()

```python
@main.route('/signup', methods=['GET', 'POST'])
def signup():
    form = RegistrationForm()
    if form.validate_on_submit():
        username = form.username.data
        email = form.email.data
        password = form.password.data

        new_user = User(username=username, email=email)
        new_user.set_password(password)  # Хешування пароля

        db.session.add(new_user)
        db.session.commit()

        flash('Registration successful. You can now log in.', 'success')
        return redirect(url_for('main.login'))

    return render_template('signup.html', form=form)
```

Рисунок 3.4.4 – Реалізація маршруту signup()

3.5. Етапи розробки клієнтської частини системи

Як і на етапі розробки серверної частини конструювання клієнтської частини поділено на деякі ключові етапи:

– Організація робочого середовища.
– Підключення клієнтської частини до серверної
– Розробка сторінок додатку

Такий підхід до розробки забезпечив ефективнішу та послідовнішу роботу над проєктом.

При фронтенд-розробці було створено батьківський шаблон що дозволило ефективну створювати веб-сторінки.

3.6. Обробка зображень та файлів

Оскільки однією з вимог є можливість завантаження на сервер зображень та файлів, що дозволяє одразу бачити що візуально являє з себе проєкт наступним етапом була розробка логіка обробки мультимедійних файлів.

Окрім цього при редагуванні профілю користувач має можливість завантажити аватар.

Усі файли зберігаються у локальній файловій системі а доступ до них розміщений у базі даних.

На рисунку 3.6.1 наведено лістинг функції яка дозволяє завантажити аватар.

```python
@main.route('/upload-avatar', methods=['GET', 'POST'])
def upload_avatar():
    form = UploadAvatarForm()
    delete_form = DeleteAvatarForm()

    if form.validate_on_submit():
        file = form.avatar.data
        if file:
            filename = secure_filename(file.filename)
            file_extension = filename.rsplit('.', 1)[1].lower()
            new_filename = f"{current_user.id}_avatar.{file_extension}"
            filepath = os.path.join(UPLOAD_FOLDER, new_filename)
            file.save(filepath)

            current_user.avatar = new_filename
            db.session.commit()

            flash('Avatar updated successfully!', 'success')
            return redirect(url_for('main.profile'))

        flash('Invalid file type', 'danger')
        return redirect(request.url)

    return render_template('upload_avatar.html', form=form, delete_form=delete_form)
```

Рисунок 3.6.1 – Функція upload_avatar()

3.7. Підключення клієнтської частини додатку до серверної

Підключення фронтенд-чатисну до бекенду є одним з ключових етапів при реалізації веб-застосунку. У даній предметній області клієнтська частина відповідає за відображення інтерфейсу, а серверна відповідає за логіку обробки даних, збереження інформації, взаємодію з БД, тощо.

За взаємодію відповідають такі технології:

– HTML + Jinja2 – клієнтські шаблони створюються за допомогою шаблонізатора.

– Flask маршрути – кожен маршрут відповідає за певну сторінку. Маршрут приймає запити з браузера та повертає шаблон після обробки

– Form запити – дані з форм таких як реєстрація передаються на сервер де вони проходять обробку

На рисунках 3.7.1 та 3.7.2 нижче показано лістинг клієнтського шаблону about та маршруту який відповідальний за цю сторінку.

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

{% block title %}About - MyPort{% endblock %}

{% block content %}
<div class="main-text">
    <div style="text-align: center; margin-bottom: 2rem;">
        <a href="/" style="background-color: #529ddb1c; color: white; padding: 0.8rem 2rem; font-size: 1.2rem; border-radius: 8px; text-decora
            Home
        </a>
        <a href="/profile" style="margin-left: 10px; background-color: #586b7c; color: white; padding: 0.8rem 2rem; font-size: 1.2rem; border-
            Profile
        </a>
    </div>

    <h1>About MyPort</h1>
    <p>
        MyPort is an innovative service designed for developers to create and manage their portfolios.
        Our goal is to help developers effectively showcase their skills and projects.
    </p>
    <p>
        This platform is built using modern web technologies like Flask for the backend and HTML/CSS for
        creating the user interface. We offer an easy-to-use interface for adding, editing, and viewing your
        projects.
    </p>
    <p>
        Our mission is to be your partner in showcasing your skills and building success.
    </p>
</div>
{% endblock %}
```

Рисунок 3.7.1 – HTML-шаблон about

```
@main.route('/about')
def about():
    return render_template('about.html')
```

Рисунок 3.7.2 – маршрут about()

3.8. Розробка навігації системи

Ссилаючись на аналіз вимог ло ситеми було розроблено навігацію додатку. Навагаційну систему було поділено на такі основні маршрути:

– Home – вітальна сторінка системи.

– About – інформаційна сторінка.

– Profile – для авторизованих користувачів направлення на сторінку профіль за допомогою ярлика next, якщо ж користувач не ввійшов систему по стандарту відкриється форма входу.

Для реалізації навігації було розроблено HTML-шаблони, які привязані до окремого маршруту.

3.9. Конструювання сторінок та компонентів клієнтської частини додатку

Розробка частини інтерфейсу веб-застосунку передбачала створення легкого та зрозумілого інтерфейсу, що забезпечує зручну навігацію, чітку читабельність та легкий доступ до запропонованих функцій які пропонує система. Основним інструментом реалізації стали технології HTML5, CSS, JavaScript та шаблонізатор Jinja2 [5].

Основні шаблони що були реалізовані:

– base.html – батьківський шаблон.

– view_project.html – сторінка перегляду проєктів.

– profile.html – сторінка пергляду профілю.

– create_project.html – сторінка створення проєкту.

Окрім статичних елементів на сторінках реалізовано динамічні елементи такі як:

– генерація переліку проєктів із БД

– відображення аватару якщо він завантажений

На рисунку 3.9.1 наведено приклад вигляду сторінки профілю

.

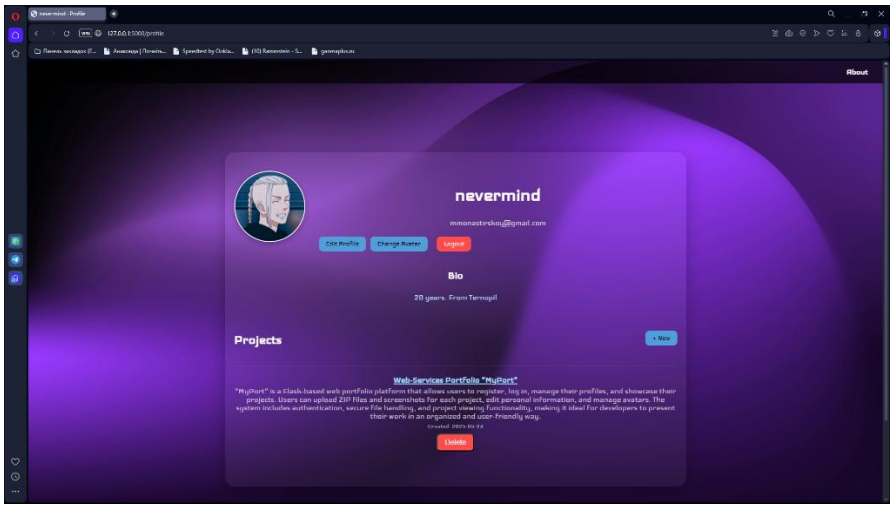

Рисунок 3.9.1 – Вигляд сторінки профілю

Реалізація сторінок була тісно повязана з маршрутами Flask, кожен з яких передає відповідний шаблон та дані для нього через функцію render_template().

Таким способом було досягнено високого рівня узгодженості між бекенд та фронтенд частинами системи.

3.10. Реалізація безпеки

Одним із ключових аспектів при любій розробці є забезпечення безпеки обробки даних користувачів та запобігання несанкціонованому достопу до функцій системи. Для реалізації механізмів автентифікації та авторизації було використано розширення Flask-Login та методи хешування паролів за допомогою бібліотеки werkzeug.security [1, 2, 9].

Основні реалізовані заходи безпеки:

- Хешування паролів
- Керування сесіями
- Обмеження доступу
- Захист форми від CSRF-атак
- Перевірка унікальності при реєстрації
- Валідація введених даних
- HTTPS

У таблиці 3.10.1 наведено способи реалізації та призначеня основних заходів безпеки [7].

Таблиця 3.10.1 – Заходи безпеки

№	Захід безпеки	Реалізація	Призначення
1	Хешування паролів	generate_password_hash, check_password_hash з бібліотеки werkzeug.security	Захист паролів при витоку БД
2	Керування сесіями	Компонент login_manager з розширення Flask-Login	Визначення чи авторизований користувач
3	Обмеження доступу	Декоратор login_required з розширення Flask-Login	Захист внутрішніх сторінок
4	Захист форми від CSRF-атак	Розширення Flask-WTF додає у форму CSRF-токен	Захист від підробки запитів
5	Перевірка унікальності при реєстрації	Перевірка email при реєстрації	Запобігання дублювання акаунтів
6	Валідація введених даних	FlaskForm та вбудовані валідатори	Захист від некоректного вводу
7	HTTPS	SSL-сертифікат	Захист передачі даних між клієнтом та сервером

Завдяки всьому вище сказаному було реалізовано базовий рівень безпеки.

4 ТЕСТУВАННЯ ТА ВІЗУАЛІЗАЦІЯ СИСТЕМИ

4.1. Процес та результати тестування

Для подібних проектів існує безліч інструментів для тестування: від написання unit-тестів до ручного тетсування додатку. Ціль перевірки додатку полягає у перевірці правильностіі роботи маршрутів серверу, оскільки саму там зосереджена найважливіша частина функціоналу системи. Бекенд обробляє запити, керує логікою, взаємодіє з системою тощо [1, 2].

Під час тестування було створено колекцію запитів для первірки маршрутів серверної частин сервісу (таблиця 4.1.1) [3].

Таблиця 4.1.1 – колекція запитів Postman

№	Назва маршруту	Метод	URL	Body	Keys
1	Sign up	POST	http://127.0.0.1:5000/signup	From-data	Username, password, email
2	Login	POST	http://127.0.0.1:5000/login	From-data	Email, password
3	Create Project	POST	http://127.0.0.1:5000/create_project	From-data	Title, description, files, screenshots
4	About	GET	http://127.0.0.1:5000/about	-	-

Використання Postman дозволило ефективно протестувати сервер без потреби використання візуальної частини, перевірити передачу даних через форми, тощо. Усі наведені запити у таблиці 4.1.1 були спішно виконані

4.2. Візуалізація сторінок системи

Візуалізація сторінок системи є також важливим аспектом роботи над сервісом, оскільки сам з інтерфейсом працює користувач. Візуальна частина має бути інтуїтивно-зрозумілою, естетично привабливою, адаптивною та доступною [5].

Дотримуючись таких принципів було створено сторінки сервісу наведні на рисунках 4.2.1 – 4.2.5.

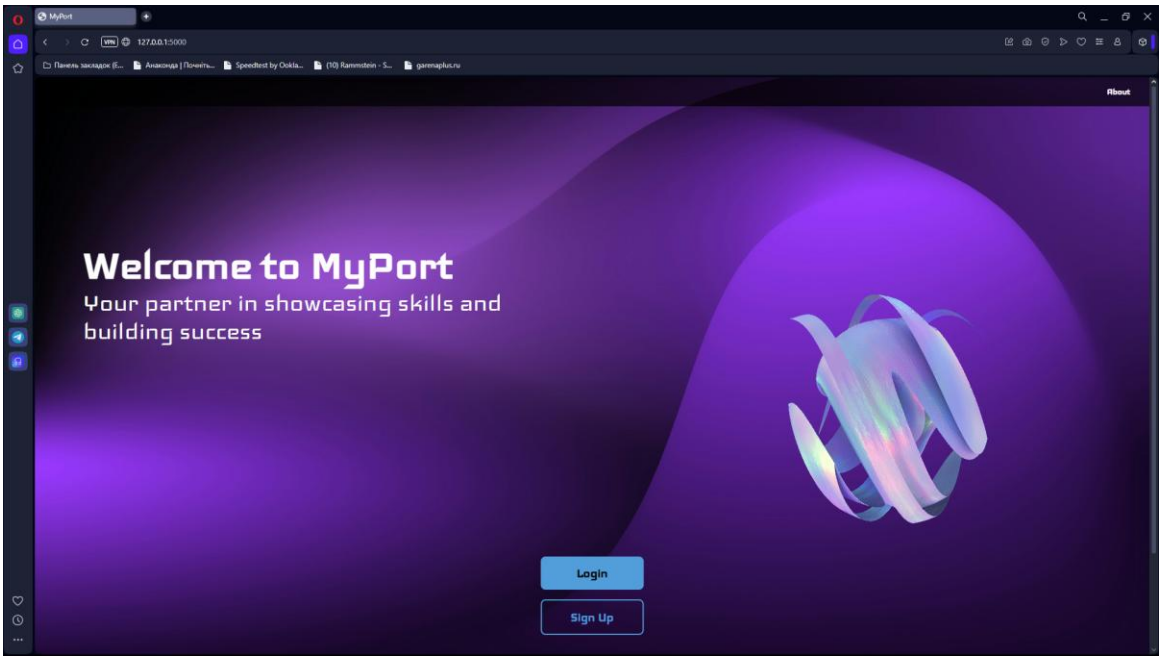

Рисунок 4.2.1 – Вигляд головної сторінки

На головній сторінці користувач бачить вітальне оголошення та містить в собі анімоване зображення та кнопки реєстрації та входу. Дизайн побудований з акцентом на простоту на привабливість, щоб справити на користувача перше враження.

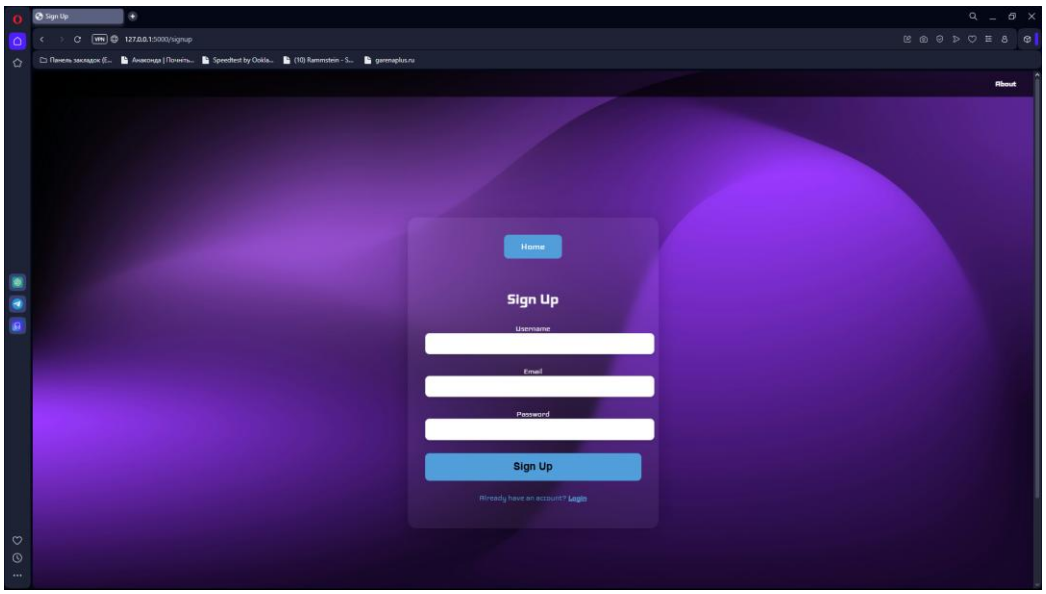

Рисунок 4.2.2 – Форма реєстрації

Форма реєстрації надає можливість новим користувачам створити акаунт у системі. Забезпечено перевірку унікальності та повідомлення про помилки. Форма входу виконана у схожому стилі та містить в собі ті є самі поля вводу за виключенням поля username.

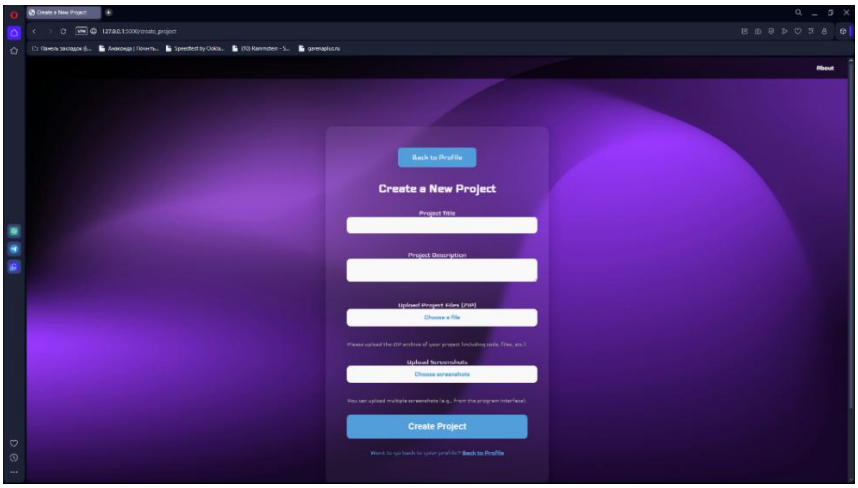

Рисунок 4.2.3 – Форма створення проєкту

Дана форма надає авторизованому користувачу додати нові проєкти до свого портфоліо. Користувач заповнює поля введення, завантажує архів з кодом та скріншоти. Реалізовано перевірку формату файлів та обмеження розміру.

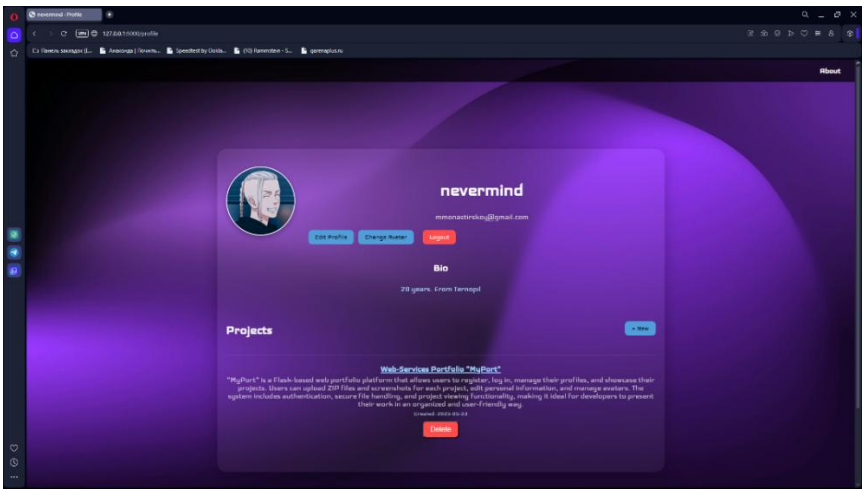

Рисунок 4.2.4 – Сторінка профілю

Користувач який авторизований у системі може переглянути своє портфоліо та переглянути детально проєкти. Для не авторизованих користувачів дана сторінка відображається без кнопок як звязані з зміною портфоліо.

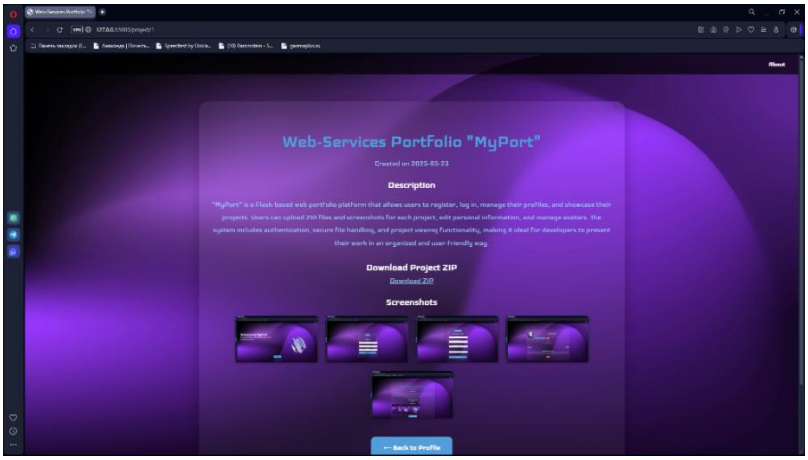

Рисунок 4.2.5 – Сторінка перегляду профілю

Ця сторінка доступна без змін як для авторизованих так і для неавторизованих користувачів. Дає можливість детально переглянути сам проєкт, переглянути скріншоти та завантажити сам проєкт у форматі ZIP-архіву.

Всі сторінки адаптивні та створені з урахуванням доступності.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

5.1. Критичні стани людини.

Підчас процесу професійної діяльності особливого значення набуває питання безпеки життєдіяльності та охорони праці. У межах будь-якого виробничого або навчального процесу існує ризик виникнення надзвичайних ситуацій які можуть призвести людину до критичного стану

Критичний стан – раптове порушення життєво важливих функцій організму, що потребують негайного реагування та допомоги [12, 13].

До критичних станів належать: непритомність, колапс, шок, стенокардію, гіпертонічний криз, інфаркт, гострий живіт гострий апендицит [14].

Непритомність (зімління) – найбілш поширений критичний стан. Є легким проявом гострої судинної недостатності через раптове корткочасне малокрів’я головного мозку. Виникає при:

- Крововтратах
- Травмах
- Хвилюванні
- Духоті
- Голодуванні
- Перевтомі
- Захворюванні серцево-судинної системи

Характерними ознаками є нерухомий стан жертви, відсутність свідомості, часті дихання та пульс, блідий колір шкіри та холодні кінцівки.

При втраті свідомості невідкладною допомогою є горизонтальне положення хворого, доступ до свіжого повітря, звільнення від тісного одягу, дати понюхати ватку змочену нашатирним спиртом, напоїти гарячим чаєм або кавою, розтерти кінцівки, дати грілку [13, 14].

Колапс – більш важка форма судинної недостатності яка виникає при кровотечах, травмах, інтоксикаціях або інфракті міокарду.

Характерними ознаками є вялість, загальмованість, затьмарена свідомість, часте дихання, слабкий але частий пульс, різка блідість шкіри, холодна пітливість, можлива блювота.

Першою допомогою в таких випадках є горизонтальне положення тіла з дещо опощеною головою, ноги потрібно обкласти грілками, накрити потерпілого ковдрою, доступ до свіжого повітря, усунути дію зовнішніх подразників (шум, світло, тощо), при наявності зупинка кровотечі [13].

Шок – комплекс симптомів які супроводжуються різким порушенням регуляції ЦНС. Шок розділяють на 2 фази: еретильна ті торпідна.

Ознаками еретильної фази є: збудливість, рухливість, ясна свідомсть, навязливий контакт, частий пульс, підвищений АТ.

Ознаками торпідної фази є: вялість, затьмарена свідомість, потерпілий неохоче вступає в контакт, часте дихання, блідість, ниткоподібний пульс, холодний піт, низький АТ.

При шоці у потрепілого потрібно усунути джерело паталогічної дії, зупинити кровотечу, ввести знеболювальне, іммобілізувати травмовані кінцівки [12, 14].

Стенокардія – гостра ішемія міокарду, через погіршення його кровопостачання з швидким відновленням кровообігу в зоні ішемії.

Характерними ознаками є: періодичний стискаючий біль, блідість, слабкість, пітливість, пришвидшений пульс, відчуття страху.

В такій ситуації потрібно заспокоїти потерпілого, забезпечити йому доступ до свіжого повітря, надати йому сидяче положення, на груди серцеві каплі, на ділянку серця гірчичники, гарячу ванну для кінцівок, валідол або нітрогліцерин під язик [12].

Гіпертонічний криз – підвищення АТ понад макисмальну норму (140/90). Спричиняється тривалим захворюванням, перенавантаженням ЦНС.

Ознаками гіпертонічного кризу є різкий головний біль, запаморочення, нудота, слабкість, пітливість, шум у вухах.

Першою допомогою є створення повного фізичного та психічного спокою, забезпечення доступу до свіжого повітря, масаж шиї та потилиці, холодний

компрес до голови [14].

Інфаркт – спричиняється гострою та стійкою тривалою ішемією міокарду.

Характерними ознаками є: постійний стискаючий біль за грудною кліткою, відчуття страху, потерпілий ловить ротом повітря, бліда шкіра, тиск падає, розлад мови.

При інфаркті потрібно надати повний спокій та доступ до свіжого повітря, негайно викликати кардіологічну бригаду [13].

Гострий живіт – розвивається при пошкодженнях та гострих захворюваннях органів черевної порожнини та заочеревного простору.

Характерними ознаками є локалізований або по всьому животу біль, шок, ікота, сухий язик, чатий пульс, пронос із кровью.

Перша допомога: транспортування до лікарні на ношах в положенні лежачи на спині [14].

Гострий аппендицит – наслідок порушень функцій нервової системи, через що виникають спазм судин та тромбоз у діляньці апендикса що сприяє заглибленню інфекції.

Характерні ознаки: раптовий біль, безсоння лежачи на правому боці, підвищена температура.

В такому випадку потрібно транспортувати потерпілого в положені лежачи до лікарні а ношах [12, 13, 14].

5.2. Захист електрообладнання від короткого замикання, перенавантаження.

Охорона праці є незамінним складником будь-якої людської діяльності. Головною метою є забезпечення здорових та безпечних умов праці та мінімалізація ризиків виникнення нещасних випадків на роботі. У наш час велика увага приділяється захисту від короткого замикання та перенавантаження що є однією з

найшпоширеніших причин аварійних ситуацій [11, 17].

Коротке замикання – виникає як наслідок зменшення опору в електричному колі до мінімального значення. Призводить до різкого зростання струму. Причини: пошкодження ізоляції, механічні дефекти провідників, неправильне зєднання [11, 15].

Перенавантаження – стан при якому електричне коло тривалий час працює з перевищеним значенням допустимого струму. Як наслідок спричиняє перегрів провідників, старіння ізоляції а також створення умов для короткого замикання [11].

Для запобігання цим випадкам використовують такі пристрої: автоматичні вимикачі, плавкі запобіжники, пристрої захисного відключення(ПЗВ), теплові реле [15,16].

Автоматичні вимикачі — розмикають коло при досягненні граничного струму. Реагують як на перенавантаження, так і на коротке замикання [15].

Плавкі запобіжники — коли є перевищення допустимого струму плавкий елемент розплавляється та перериває коло [15].

ПЗВ — спрацьовують при виявленні витоку струму, запобігаючи ураженню електричним струмом [15].

Теплові реле — реагують на тривале перевищення струму, що характерне для перенавантаження, та вимикають живлення для запобігання перегріву [15].

На рисунку 5.2.1 подана типова схема захисту однофазної електромережі.

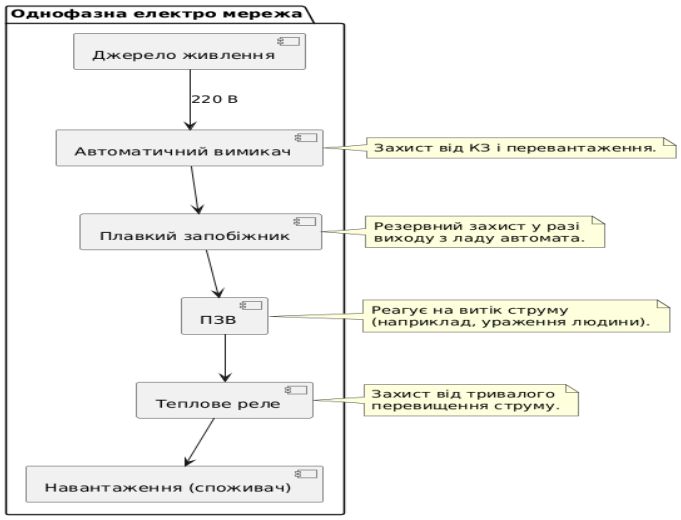

Рисунок 5.2.1 – Типова схема захисту однофазної мережі від КЗ і перенаватаження

Вибір та розрахунок приладів для захисту від аварійних ситуацій проводиться з урахуванням характеристик обладнання, довжини електроліній, типу мережі та умов експлуатації. Усі елементи захисту повинні відповідати вимогам Правил улаштування електроустановок (ПУЕ) [15], ДСТУ EN60204-1:2015 [16], а також Закону України «Про охорону праці» [17].

Крім правильного підбору пристроїв захисту, важливим є їх своєчасне технічне обслуговування та перевірка працездатності. Несправні або застарілі елементи захисту можуть не виконувати свою функцію, що призводить до аварійної ситуації.

Також слід враховувати особливості роботи в умовах підвищеної небезпеки такі як: вологі приміщення, пил, агресивне середовище.

В таких умовах захист повинен бути посилений, з використанням спеціальних герметизованих елементів та пристроїв з підвищеним класом захисту IP (IP44, IP65, тощо) [16].

Під час проєктування та експлуатації електрообладнання необхідно дотримуватись комплексного підходу до електробезпеки: поєднання захисного заземлення, автоматичних вимикачів, ПЗВ, а також інструктажів персоналу з правил безпечної експлуатації [11].

ВИСНОВКИ

У межах виконаня даної кваліфікаційної роботи було успішно реалізовно функціоналний веб-сервіс для створення персонального портфоліо для програмістів.

Проєкт охоплює всі стадії розробки: від аналізу предметної області та постановки завдань до створення інтерфейсу та реалізації серверної частини.

Під час виконання роботи мною досягнуто наступних результатів:

– Аналіз предметної області
– Спроектовано архітектуру системи
– Розроблено БД
– Реалізована основна функціональність
– Забезпечена базова безпека
– Розроблено простий та зручний інтерфейс

У процесі практичного виконання веб-додатку було показано ефективність використання Flask у поєднанні з іншими технологіями зокрема такими як Jinja2, HTML та CSS, Javascript, Git та Postman.

Результат є масштабованим рішенням, що може бути використане як в навчальних так і в комерційних цілях.

Поставлена мета така як створити зручний сервіс для розміщення та перегляду портфоліо була мною досягнута.

Надалі розробка можна розширити шляхом додавання функції підтримки коментарів та інтеграцією з іншими платформами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grinberg M. Flask Web Development: Developing Web Applications with Python. – O'Reilly Media, 2018 – 258c.
2. Янг В. Python. Розробка веб-застосувань за допомогою Flask. — К.: Видавництво «Діалектика», 2021. — 304 с.
3. Документація SQLAlchemy [Електронний ресурс]. — Режим доступу: https://www.sqlalchemy.org/
4. Документація SQLite [Електронний ресурс]. — Режим доступу: https://www.sqlite.org/docs.html
5. Документація Bootstrap [Електронний ресурс]. — Режим доступу: https://getbootstrap.com/
6. Офіційна документація Python [Електронний ресурс]. — Режим доступу: https://docs.python.org/3/
7. OWASP Foundation. Top 10 Web Application Security Risks [Електронний ресурс]. — Режим доступу: https://owasp.org/www-project-top-ten/
8. Richardson L., Amundsen M. RESTful Web APIs. — O'Reilly Media, 2013. — 406 с.
9. Гриценко А.В. Безпека веб-додатків: теорія і практика. — К.: КНЕУ, 2020. — 212 с.
10. Головач О.Ю. Основи розробки веб-застосунків: навч. посіб. — Харків: ХНУРЕ, 2021. — 168 с.
11. 1. Хмельовський В., Мотрич М., Скібчик В., Марчиниша Є., Білько Т. Охорона праці : навч. посіб. для студентів ОПП «Бакалавр». – К. : Центр учбової літератури, 2023. – 594 с.
12. Пістун І.П., Кочубей В.І. Безпека життєдіяльності. – К. : Університетська книга, 2025. – 575 с.
13. Наказ МОЗ України від 09.03.2022 № 441 «Про затвердження Порядків надання домедичної допомоги особам при невідкладних станах» Міністерство

охорони здоров'я України. – [Електронний документ]. – Режим доступу: https://moz.gov.ua/uk/decrees/nakaz-moz-ukraini-vid-09032022--441-pro-zatverdzhennja-porjadkiv-nadannja-domedichnoi-dopomogi-osobam-pri-nevidkladnih-stanah

14. Безпека життєдіяльності. Основи охорони праці : електронний курс лекцій / І.Б. Окіпний, О.Я. Гурик. – Тернопіль : ТНТУ ім. І. Пулюя, 2025. – [Електронний ресурс] – Режим доступу: https://dl.tntu.edu.ua/index.php (доступ для зареєстрованих користувачів).

15. Правила улаштування електроустановок [Електронний ресурс] – Режим доступу: https://zakon.isu.net.ua/sites/default/files/normdocs/pue.pdf

16. ДСТУ EN 60204-1:2015. Безпечність машин. Електрообладнання машин. Частина 1. Загальні вимоги [Електронний ресурс] – Режим доступу: https://www.rts.ua/rus/forpro/613/0/31/#bookmark75

17. Закон України «Про охорону праці» № 2694-XII [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12#Text

18. Методичні вказівки до виконання дипломної роботи освітнього рівня "бакалавр" студентами усіх форм навчання для напряму підготовки 121 – "Інженерія програмного забезпечення" [Електронний документ] – Режим доступу: https://elartu.tntu.edu.ua/bitstream/123456789/17915/1/MV_dyplom_bakalavr_2016_new.pdf

ДОДАТКИ

Додаток А – Тези конференції

УДК 621.326

Монастирський М. – ст. гр. СП-42

Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА СЕРВІСУ-ПОРТФОЛІО ДЛЯ РОЗРОБНИКІВ ПРОГРМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ FLASK

Науковий керівник: к. т. н. Михалик Д.

Monastyrskyi M.

Ternopil Ivan Puluj National TechnicalUniversity

DEVELOPING A PORTFOLIO SERVICE FOR SOFTW ARE DEVELOPERS USING FLASK

Supervisor: Ph. Mykhalyk D

Ключові слова: Вебсервіс, Flask, REST API, база даних, онлайн-портфоліо.

Key words: Web service, Flask, REST API, data base, online portfolio.

У сучасному світі програмістам дедалі важливіше мати персональне портфоліо, яке дозволяє демонструвати свої проєкти, технічні навички, досвід роботи та інші досягнення. Наявність онлайн-сервісу, який дає змогу швидко створити та зручно презентувати власне портфоліо, є актуальною потребою в галузі інформаційних технологій.

У рамках даної роботи розроблено вебсервіс, що надає користувачам можливість створювати персональні сторінки з інформацією про себе, свої проєкти, стек технологій, посилання на репозиторії тощо. Основним інструментом для реалізації серверної логіки обрано мікрофреймворк Flask, що дозволяє швидко будувати вебзастосунки з гнучкою архітектурою.

Сервіс передбачає реалізацію реєстрації та авторизації користувачів, створення, редагування та перегляду портфоліо, а також опціонально — можливість публічного доступу до сторінки користувача.

Ключовими етапами реалізації проєкту стали: проєктування структури бази даних, створення REST API для взаємодії з фронтендом, побудова інтерфейсу користувача, а також забезпечення безпеки збереження та обробки даних.

Результатом роботи став готовий MVP вебсервісу, який може використовуватись як самостійний інструмент для створення портфоліо розробників програмного забезпечення.

Література:

1. Miguel Grinber (2018). "Flask Web Development: Developing Web Applications with Python".

2. Eric Matthes (2019). "Python Crash Course".

3. Robert C. Martin (2017). "Clean Architecture".

Додаток Б – Лістинг файлу forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, FileField, TextAreaField, IntegerField, MultipleFileField
from flask_wtf.file import FileAllowed, FileRequired
from wtforms.validators import DataRequired, Email, Length, ValidationError
from .models import User


class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=3, max=100)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=6)])
    submit = SubmitField('Register')

    def validate_email(self, email):
        # Перевірка наявності користувача з таким email
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('This email is already registered.')
```

Продовження лістингу додатку Б:

```
    class LoginForm(FlaskForm):
        email = StringField('Email', validators=[DataRequired(),
Email()])
        password = PasswordField('Password',
validators=[DataRequired(), Length(min=6)])
        submit = SubmitField('Login')

        def validate_email(self, field):
            user = User.query.filter_by(email=field.data).first()
            if not user:
                raise ValidationError('Email not found.')


    class EditProfileForm(FlaskForm):
        username = StringField('Username',
validators=[DataRequired()])
        bio = StringField('Bio', validators=[DataRequired()])


    class UploadAvatarForm(FlaskForm):
        avatar = FileField('Choose Avatar',
validators=[FileAllowed(['jpg', 'png', 'jpeg'], 'Images only!')])
        submit = SubmitField('Upload Avatar')


    class DeleteAvatarForm(FlaskForm):
        submit = SubmitField('Delete Avatar')
```

Продовження лістингу додатку Б:

```
    class CreateProjectForm(FlaskForm):
        title = StringField('Project Title',
validators=[DataRequired()])
        description = TextAreaField('Project Description',
validators=[DataRequired()])
        files = FileField('Project ZIP File',
validators=[FileRequired(), FileAllowed(['zip'], 'ZIP only!')])
        screenshots = MultipleFileField('Upload Screenshots',
validators=[FileAllowed(['jpg', 'jpeg', 'png', 'gif'], 'Images
only!')])


    class DeleteProjectForm(FlaskForm):
        project_id = IntegerField('Project ID',
validators=[DataRequired()])
        submit = SubmitField('Delete'
```

Додаток В – Диск з роботою