ЗМІСТ

[Вступ 3](#_Toc166429702)

[1. Постановка задачі 5](#_Toc166429703)

[1.1 Дослідження існуючих методів та алгоритмів до розв’язання поставленої задачі 5](#_Toc166429704)

[1.2 Уточнена постановка задачі на розробку програмного продукту 7](#_Toc166429705)

[2 Розробка програмного продукту 8](#_Toc166429706)

[2.1 Метод розв’язку задачі 8](#_Toc166429707)

[2.2 Алгоритм розв’язку задачі 11](#_Toc166429708)

[3 Опис розробленого програмного продукту 14](#_Toc166429709)

[3.1 Опис головних структур і змінних програми 14](#_Toc166429710)

[3.2 Опис головних функцій програми 19](#_Toc166429712)

[3.3 Опис інтерфейсу 24](#_Toc166429713)

[3.4 Результати роботи програмного продукту 26](#_Toc166429714)

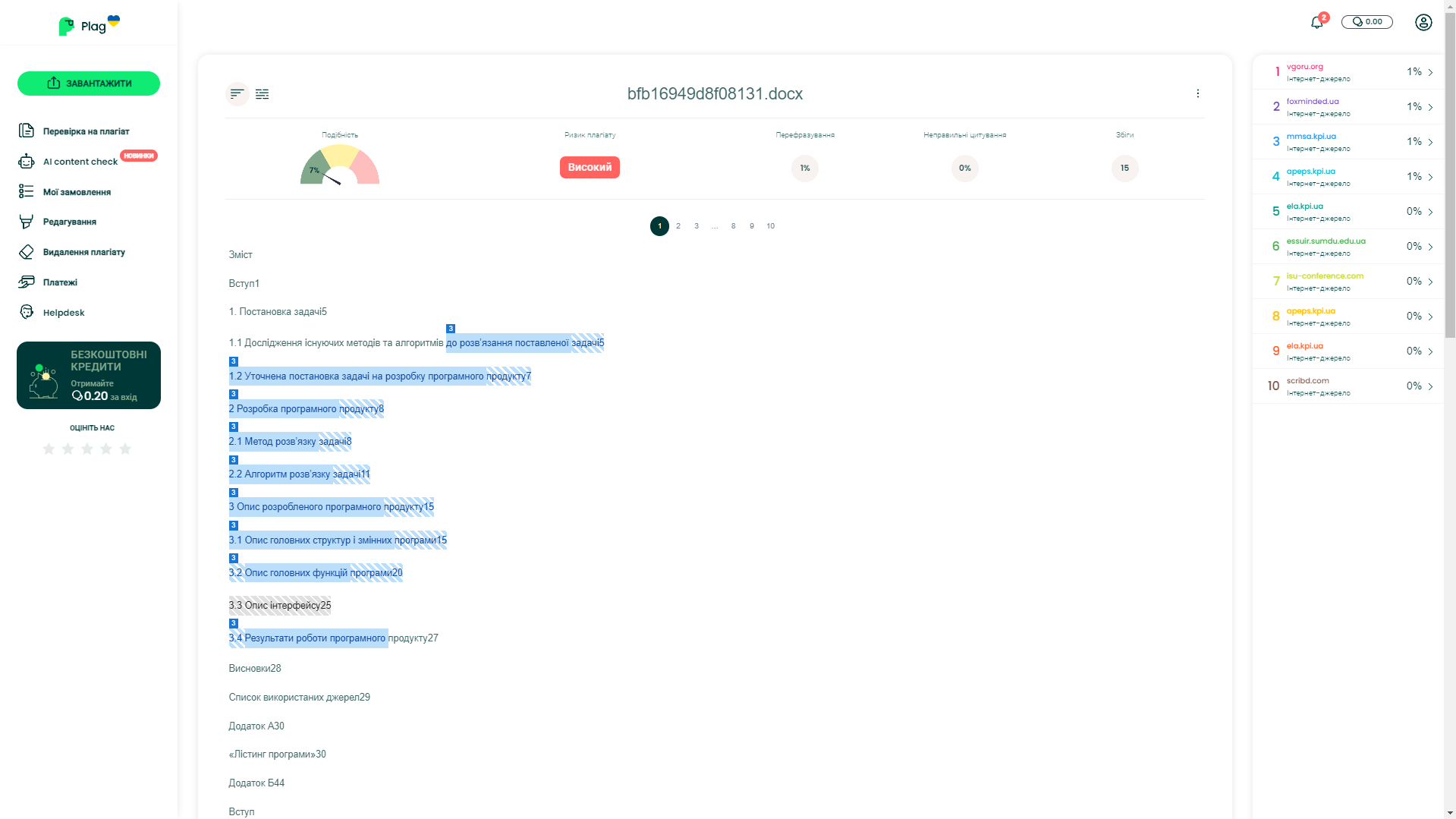
[Висновки 27](#_Toc166429715)

[Список використаних джерел 28](#_Toc166429716)

[Додаток А 30](#_Toc166429717)

[«Лістинг програми» 30](#_Toc166429718)

[Додаток Б 40](#_Toc166429719)

Перевірка на антиплагіат

# Вступ

У наш час вебпрограмування стає все більш важливою сферою діяльності. Навіть найменший бізнес без власного вебсайту може відстати від конкурентів. Для створення інтерактивних та ефективних вебдодатків важливо мати розуміння сучасних вебстандартів.

Швидкий розвиток Інтернету призвів до зростання числа онлайн-сервісів, і він став невід'ємною частиною сучасного бізнесу. Багато компаній переходять на електронний бізнес, створюючи свої вебсайти, рекламуючи товари та послуги в Інтернеті та здійснюючи продажі онлайн. Такий розвиток відкриває безліч можливостей для новаторських підходів та веброзробок.

Вебпрограмування - це процес створення сайтів та програм, які функціонують у мережі Інтернет. Для цього створюються спеціальні комп'ютерні програми, які називаються скриптами і діляться на два типи: серверні і клієнтські. Сервер - це комп'ютер, на якому знаходяться файли сайту, тоді як клієнт - це користувач, або, конкретніше, веббраузер, що встановлено на ПК. Коли клієнт надсилає запит до сервера, він виступає як клієнт.

Мова вебпрограмування є ключовим елементом, який зв'язує веброзробника з сайтами. Існує багато мов, на яких можна писати як серверні, так і клієнтські скрипти. Однак деякі мови використовуються виключно для серверної або клієнтської частини веброзробки.

Python є ідеальним вибором для створення вебдодатків завдяки двом популярним фреймворкам - Django та Flask. Django дозволяє швидко розпочати розробку великих та складних вебпроєктів, надаючи вбудовані інструменти для автентифікації, роботи з базами даних та іншими функціями. З іншого боку, Flask надає розробникам більше волі у створенні легших та спеціалізованих додатків, призначених для вирішення конкретних завдань. Обидва ці фреймворки сприяють швидкій розробці вебдодатків та API, надаючи розробникам відмінні інструменти для втілення їхніх ідей.

Предметом дослідження є вебсайт з функціоналом адресної книги.

Об’єктом дослідження є сайт адресної книги

Завданням роботи є:

* дослідження існуючих методів та алгоритмів до розв'язання поставленої задачі;

- створення алгоритму розв’язку задачі;

- реалізація файлової структури програми;

- написання коду;

- тестування програми.

Мета курсової роботи полягає в розробці програмного забезпечення, яке забезпечує зручний та ефективний спосіб зберігання, організації та керування контактною інформацією користувачів.

Завдання роботи полягає у створенні вебдодатку, який буде реалізовувати просту книгу контактів з базовим функціоналом.

Методологічна база дослідження:

- аналіз літературних джерел;

- емпіричні дослідження;

- аналіз існуючих програмних засобів;

- проведення експериментів з реалізації та тестування різних аспектів програми.

Обсяг курсової роботи становить 40 аркушів.

# 1. Постановка задачі

# 1.1 Дослідження існуючих методів та алгоритмів до розв’язання поставленої задачі

Оскільки, за реалізацію інтерфейсу буде відповідати браузер, то для створення сайту потрібно використовувати відповідні фреймворки. Для Python основними варіантами для реалізації даної задачі слугують Flask та Django[4].

Flask - це легкий та гнучкий вебфреймворк для Python, який дозволяє швидко створювати вебдодатки. Він базується на принципах простоти і прозорості, що дозволяє розробникам швидко створювати вебдодатки без зайвого навантаження. Flask надає базові можливості для створення вебдодатків, такі як маршрутизація URL, обробка запитів та відповідей, шаблонізація і т.д. Однак він також дозволяє розширювати функціоналість за допомогою різноманітних розширень (extensions) або плагінів. Flask ідеально підходить для створення невеликих або середніх вебдодатків, а також для прототипування та швидкого розгортання проектів[6].

Django - це повноцінний вебфреймворк для Python, який надає комплексний набір інструментів для створення складних вебдодатків. Він має вбудовану підтримку для бази даних, аутентифікації користувачів, адміністративного інтерфейсу та багато іншого. Django базується на концепції "зроблено правильно" (batteries-included), що означає, що він надає готові рішення для багатьох типових задач веброзробки. Це робить Django ідеальним вибором для великих та складних вебпроектів, де потрібно враховувати багато аспектів розробки, таких як безпека, масштабованість та продуктивність.

Тому у випадку з створення простого додатку який буде реалізовувати базовий CRUD функціонал ідеально підійде Flask.

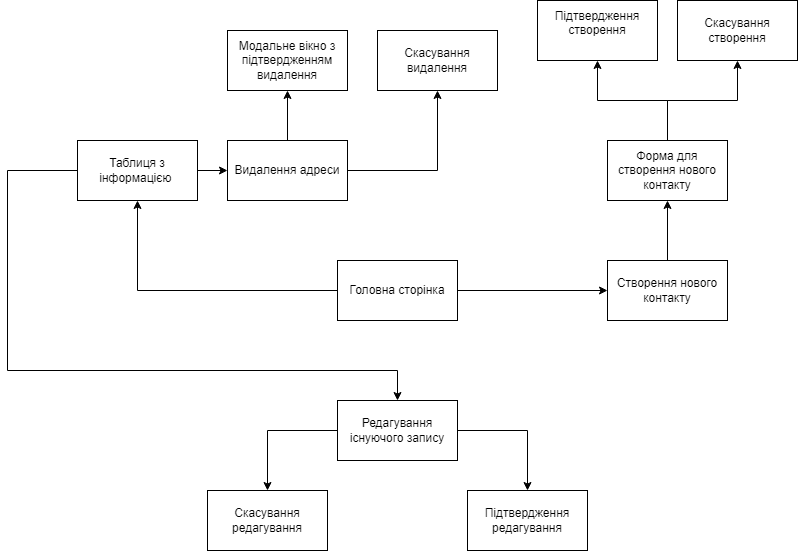
Візуальне меню програми зображено на структурній схемі на рисунку 1.1.

Рисунок 1.1 – Структурна схема вебсайту

Використання фреймворків може значно спростити та прискорити процес розробки вебсайту.

# 1.2 Уточнена постановка задачі на розробку програмного продукту

Головною задачею курсової роботи є створення адресної книги.

Програма повинна реалізовувати простий довідник, що зберігає різні дані (наприклад номери телефонів, ПІП, адреса і т.д.). Повинні забезпечуватися: введення нових даних перевіркою їх коректності, перегляд довідника, пошук необхідних даних за зразком, коригування даних, видалення непотрібних записів, збереження даних у файлі та читання даних файлу. Довідник можна реалізувати у вигляді як консольного так і візуального застосування.

Але саме через покращення структури та досвід користувача з використанням довідника, було вирішено замінити деякі пункти.

На початку роботи програми повинно бути забезпечено виведення графічного меню у вигляді кнопок та таблиці, що реалізують відповідні дії при натисканні на них.

Отже сторінка повинна містити наступні пункти та відповідні функції

1. Таблицю зі всіма записами які вже є в базі даних
2. Створення нового запису
3. Редагування існуючого запису
4. Видалення існуючого запису

Збереження даних буде реалізовано не до файлу, а в SQLite базу даних.

Для реалізації інтерфейсу як середовище було обрано браузер.

# 2 Розробка програмного продукту

# 2.1 Метод розв’язку задачі

За час розвитку мови Python було створено багато різних методологію за якими будують додатки, у кожних з них є як свої мінуси так і плюси. Ось деякі з них:

Водопад - це лінійний підхід, де кожен етап розробки виконується послідовно, один за одним. Етапи включають в себе аналіз, проектування, реалізацію, тестування та впровадження.

Прототипування - методологія, що передбачає швидке створення прототипів програмного забезпечення для зрозуміння вимог та отримання зворотного зв'язку від користувачів.

Ітераційний та інкрементний розвиток (Iterative and Incremental Development) передбачає поділ процесу розробки на невеликі ітерації. Кожна ітерація включає в себе аналіз, проектування, реалізацію та тестування, але тільки для конкретної частини функціоналу.

Функціональне програмування – це парадигма програмування, заснована на використанні функцій як основного будівельного блоку програми. У функціональному програмуванні дані вважаються незмінними, а функції – чистими, тобто такими, що не мають побічних ефектів і завжди повертають однаковий результат для заданих вхідних параметрів. Функціональне програмування дає змогу створювати надійніші та стійкіші програми, які простіше тестувати та підтримувати.

Спрямована на об'єкти розробка (Object-Oriented Development) використовує концепції об'єктно-орієнтованого програмування для створення програмного забезпечення. Класи, об'єкти, спадковість та інші концепції ООП використовуються для організації коду та забезпечення повторного використання.

Саме для роботи над програмою обраний об’єктно-орієнтований підхід поєднаний з функціональним підходом, оскільки, Flask в своїй основі опирається на парадигму ООП, але для маршрутизації використовуються функції, які опираються на декоратори з відповідним маршрутом.

Щоб реалізувати весь функціонал додатку в браузері на мові Python, потрібно спочатку обрати фреймворк для роботи з браузером. Взагалі є два основні варіанти це Django та Flask. Але програма буде реалізовано саме на Flask через те, що для новачків легше з ним працювати. З плюсів можна виділити те, що браузер надає додаткові функції та інформацію, яка може бути корисна при розширені функціоналу сайту, також Flask надає багато методів якими зручно користуватися при розробці сайту, та структура додатків які побудовані на фреймворках зазвичай дуже хороша, оскільки при їх побудові використовують патерни та методології такі як SOLID, GoF та інші.

Всі данні зберігаються у базі даних SQLite оскільки збереження даних саме в базах даних є більш структурованих, прогнозованим та надійним.

SQLite - це система управління базами даних (СУБД), яка використовує мову SQL для зберігання та управління даними. В SQLite дані організовані у вигляді таблиць, які складаються з рядків та колонок. Кожна колонка в таблиці має свій тип даних (наприклад, ціле число, рядок, дата тощо). SQLite підтримує транзакції, які дозволяють групувати декілька операцій в один блок, який виконується або виконується повністю, або не виконується взагалі. Це важливо для забезпечення цілісності та надійності даних. підтримує можливість створення індексів для швидкого доступу до даних. Індекси допомагають зменшити час виконання запитів, особливо для великих таблиць.

Для побудови інтерфейсу використано HTML, CSS та фреймворк CSS Bootstrap, який включає в себе JavaScript файли.

HTML (HyperText Markup Language), CSS (Cascading Style Sheets) і Bootstrap - це три ключових компоненти для розробки стильних і адаптивних вебсайтів. Вони працюють разом для створення зручного вебінтерфейсу.

HTML використовується для створення структури вебсторінки. Він визначає елементи, такі як заголовки, параграфи, таблиці, форми, зображення тощо.

CSS використовується для стилізації HTML-елементів. За допомогою CSS можна задавати кольори, шрифти, розміри, відступи та інші властивості елементів[5].

Мова програмування JavaScript використовується для створення динамічних та інтерактивних вебсторінок, а також для додавання функціональності, анімації та взаємодії до вебдодатків, що робить їх більш динамічними та привабливими для користувачів[1].

Bootstrap – це потужний інструмент для швидкого створення адаптивних та стильних вебінтерфейсів. Він надає набір зрозумілих класів CSS та компонентів JavaScript, які можна використовувати для розробки шаблонів вебсторінок[3].

Для зберігання коду та розподілення його версії використано Git.

Git розподілена система керування версіями, яка використовується для керування змінами у програмному коді. Вона дозволяє розробникам спільно працювати над проектами, відстежувати історію змін, вирішувати конфлікти, відмічати версії та багато іншого.

За рахунок великої кількості інструментів при розробців вебсайтів, кінечний продукт буде збалансований, як за функціональною частиною, так і за дизайном.

# 2.2 Алгоритм розв’язку задачі

Алгоритм - це точно визначений набір інструкцій, які можуть бути виконані для вирішення певної задачі чи досягнення певної мети. Він є фундаментальним поняттям в області програмування та комп'ютерних наук. Ось деякі ключові аспекти алгоритмів:

- коректність;

- чіткість;

- детермінованість;

- кінцевість;

- ефективність;

- масштабованість;

- оптимальність.

Існує безліч типів алгоритмів, які використовуються для вирішення різних задач у різних галузях. Ось кілька загальних типів алгоритмів:

- сортування;

- пошук;

- графові алгоритми;

- деревові алгоритми;

- рекурсивні алгоритми;

- жадібні алгоритми;

- динамічне програмування.

Створення алгоритму для CRUD (створення, читання, оновлення, видалення) сайту вимагає декількох кроків.

Потрібно обрати тип бази даних (наприклад, SQLite, MySQL, PostgreSQL). Створити схему бази даних з таблицями для об'єктів, які будуть зберігатися.

Встановитт Flask та його розширення, такі як Flask-SQLAlchemy для роботи з базою даних.

Визначити моделі для об'єктів, які будуть зберігатися в базі даних, використовуючи ORM (об'єктно-реляційне відображення) SQLAlchemy.

Створити маршрути для кожної операції CRUD GET для читання, POST для створення оновлення, DELETE для видалення. Кожен маршрут повинен викликати відповідну функцію-відображення, яка буде обробляти запити.

Для кожної операції CRUD створити відповідні функції-відображення, які будуть обробляти запити. У цих функціях виконати відповідні операції з об'єктами бази даних (створення, читання, оновлення, видалення).

Створити HTML-шаблони для кожної сторінки сайту. Відобразити дані з бази даних у шаблонах, використовуючи шаблонізатор Flask, такий як Jinja2.

Створити HTML-форми для взаємодії з користувачем та відправити дані з цих форм на сервер за допомогою методів POST або PUT.

Обробити дані, надіслані з форм, у функціях-відображеннях. Використовувати відповідні операції CRUD з об'єктами бази даних залежно від типу запиту (наприклад, додавання нового запису, оновлення існуючого тощо).

Додати валідацію даних на клієнтському та серверному боці для запобігання некоректним даним. Перевіряти дані, що надходять з форм, перед їх збереженням у базі даних.

Додавати обробку помилок для відображення користувачам зрозумілих повідомлень про помилки при взаємодії з сайтом.

Провести тестування веб-сайту, переконавшись, що всі операції CRUD працюють належним чином та, що дані коректно зберігаються в базі даних.

Алгоритм роботи програми відображений в Додатку Б.

Блок-схема описує роботу маршрутизатора який по своїй суті і повертає результат виконання програми для всіх сторінок. Блок схема складається з 5-ти умов, які формують CRUD додаток.

CRUD – це абревіатура сreate read update delete, тобто це форма програми у якої є можливості для створення, читання, оновлення та видалення даних. Також у програмування до кожної дії йде створення шаблонів через які користувач може взаємодіяти з інтерфейсом додатку.

Отже, блок-схема як і програма відвовідає патерну CRUD.

# 3 Опис розробленого програмного продукту

# 3.1 Опис головних структур і змінних програми

У Python основні структури даних, які можна використовувати для організації даних, включають списки (lists), кортежі (tuples), множини (sets) та словники (dictionaries). Кожна з цих структур даних має свої особливості і призначення.

При написанні програми використані такі структури як словники, списки та класи, а також об’єкти класів.

Списки є однією з основних структур даних, яка дозволяє зберігати колекцію об'єктів у впорядкованій послідовності. Списки є динамічними, тобто можна додавати, видаляти і змінювати їх елементи.

Приклад використання:

fruits = []

fruits.append("apple")

fruits.append("banana")

fruits.append("orange")

fruits.remove("banana")

Словник (dictionary) є структурою даних, що дозволяє зберігати дані у вигляді пар ключ-значення. Це одна з основних вбудованих структур даних і дозволяє ефективно зберігати і відновлювати дані за допомогою унікальних ключів. Прикладом використання словників в програмі може слугувати доступ до даних які передаються через об’єкт request. Щоб отримати ім’я контантакту достатньо прописати request.form['name'] . Приклад словника:

students = {

"John": 90,

"Alice": 85,

"Bob": 88,"Emily": 92}

Класи це фундаментальна концепція об'єктно-орієнтованого програмування (ООП), яка дозволяє створювати структуровані моделі даних з атрибутами і методами, що працюють з цими даними. Приклад класу в програмі:

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(50), nullable=False)

surname = db.Column(db.String(50), nullable=False)

phone\_number = db.Column(db.Integer(), nullable=False)

address = db.Column(db.String(256), nullable=False)

addition\_info = db.Column(db.String(256))

date = db.Column(db.DateTime, default=datetime.utcnow)

def \_\_repr\_\_(self):

return '<User %r>' % self.id

Об'єкти в контексті програмування є основним поняттям об'єктно-орієнтованого програмування (ООП). Об'єкт представляє конкретний екземпляр класу і є основним будівельним блоком ООП. Основні концепції, пов'язані з об'єктами, включають наступне:

Клас визначає структуру та поведінку об'єктів, включаючи атрибути (змінні стану) і методи (функції). Об'єкт є конкретним екземпляром класу.

Інкапсуляція принцип ООП, за яким об'єкт зберігає свій стан (атрибути) в приватних полях і використовує методи (гетери і сетери) для управління доступом до цих даних.

Абстракція означає виділення основних характеристик об'єкта і відокремлення їх від деталей реалізації. Це дозволяє розглядати об'єкт на високому рівні абстракції, ігноруючи деталі внутрішньої реалізації.

Наслідування дозволяє створювати нові класи на основі існуючих (базових) класів. Клас, який успадковує функціональність, називається підкласом, а клас, від якого успадковується, - батьківським класом.

Поліморфізм здатність об'єктів різних класів використовувати однаковий інтерфейс (методи) з різною реалізацією. Поліморфізм дозволяє здійснювати однакові операції на різних типах об'єктів.

Функція - це блок коду в програмуванні, який призначений для виконання конкретної задачі або обчислення певного результату. Вона використовується для організації і структурування програмного коду шляхом виділення певної логіки в окремі підпрограми, які можуть бути викликані з різних частин програми. Приклад функції з коду програми

def update\_record(record\_id):

user = User.query.get\_or\_404(record\_id)

if (len(request.form['name']) > 3 and len(request.form['surname']) > 3 and len(request.form['phone\_number']) > 10 and len(request.form['add\_info']) > 3):

user.name = request.form['name']

user.surname = request.form['surname']

user.phone\_number = request.form['phone\_number']

user.address = request.form['address']

user.addition\_info = request.form['add\_info']

db.session.commit()

flash('Контакт успішно оновлено', 'success')

return redirect("/")

else:

flash('Помилка валідації', 'danger')

return redirect(request.referrer or "/")

У Flask для маршрутизації використовуються декоратори, які прикріплюють URL-шляхи до функцій відображення. В таблиці 3.1 наведені основні декоратори для маршрутизації у Flask. Таблиця декораторів (див. Табл. 3.2)

*ВСП РФК КНЕУ ПКР 000 013 ПЗ*

*Вим*

Арк.

№ докум.

Підпис

### *Дата*

Таблиця 3.1 – Маршрутизація в фреймворку Flask

|  |  |
| --- | --- |
| Маршрут | Опис декоратора |
| @app.route('/') | Вказує, що функція відображення повинна відповідати кореневому URL-шляху веб-додатка. |
| @app.route('/about') | Вказує, що функція відображення повинна відповідати URL-шляху /about |
| @app.route('/user/<post>') | Вказує, що функція відображення повинна приймати параметр <username> у URL-шляху. |
| @app.route('/post/<int:post\_id>') | Вказує, що параметр <post\_id> у URL-шляху має бути цілим числом. |
| @app.route('/login', methods=['GET', 'POST']) | Вказує, що функція відображення повинна обробляти як GET, так і POST запити. |

Для повної роботи з Flask потрібно підключити деякі бібліотеки, які розширюють можливості фреймворку. (див. Табл. 3.2)

Таблиця 3.2 – Опис підключених компонентів

|  |  |
| --- | --- |
| Import | Опис |
| Flask | Вебфреймворк для Python, який дозволяє швидко створювати веб-додатки |
| render\_template | Функція Flask використовується для відображення HTML-шаблонів |
| request | Об'єкт Flask містить дані про поточний HTTP-запит, такі як дані форми, параметри URL тощо. |
| redirect | Функція Flask використовується для перенаправлення користувача на іншу сторінку |
| flash | Механізм Flask для передачі повідомлень користувачеві після виконання певної дії |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| Flask-SQLAlchemy | Розширення Flask – надає підтримку роботи з базою даних SQLAlchemy у Flask додатках. Дозволяє легко взаємодіяти з базою даних, використовуючи ORM |
| datetime | Модуль Python надає функціональність для роботи з датою та часом. |

За допомогою цих функцій, об’єктів та бібліотек створення вебсайту на Flask стає набагато легше та структорованіше, за рахунок уже ідеально складених об’єктів та функцій, які використовують при підключенні.

# 3.2 Опис головних функцій програми

Весь функціонал сайту забезпечується декораторами до яких під’єднані функції. Ці функції відповідні за видалення відображення, редагування та створення в базі даних. Декоратори з себе представляють просту директиву в якій вказано маршрут до якого буде прикріплена ця функція, параметри request та методи якими буде опрацьовано запит для функцій.

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(50), nullable=False)

surname = db.Column(db.String(50), nullable=False)

phone\_number = db.Column(db.Integer(), nullable=False)

address = db.Column(db.String(256), nullable=False)

addition\_info = db.Column(db.String(256))

date = db.Column(db.DateTime, default=datetime.utcnow)

def \_\_repr\_\_(self):

return '<User %r>' % self.id

Клас User є моделлю користувача для бази даних в застосунку Flask, який використовує SQLAlchemy для взаємодії з базою даних.

Основні атрибути класу User відображаються як стовпці в таблиці бази даних:

id – унікальний ідентифікатор користувача, який є первинним ключем для таблиці.

name – поле, яке зберігає ім'я користувача. Встановлено, що воно не може бути порожнім (nullable=False).

Surname – поле, яке зберігає прізвище користувача. Також не може бути порожнім.

phone\_number – поле для зберігання номера телефону користувача.

Address – поле для зберігання адреси користувача.

addition\_info – додаткова інформація про користувача, яка може бути введена за бажанням. Це необов'язкове поле.

Date – час створення запису користувача. За замовчуванням встановлено поточну дату та час за допомогою datetime.utcnow.

Метод \_\_repr\_\_ використовується для представлення об'єктів класу у рядковому форматі, в якому виводиться ідентифікатор користувача.

Цей клас дозволяє створювати, зчитувати, оновлювати та видаляти записи користувачів у базі даних, а також виконувати різноманітні операції з ними, такі як пошук, сортування тощо.

Коли користувач відкриває головну сторінку вебсайту, виконується функція index().

@app.route('/')

def index():

records = User.query.order\_by(User.id).all()

return render\_template("index.html", records=records)

В даній функції спочатку дані про всіх користувачів беруться з бази даних і потім повертаються уже в шаблоні index.html до якого підключений файл стилів. Результатом буде таблиця записів про всіх користувачів.

@app.route('/store-record', methods=["POST"])

def store\_record():

if request.method == "POST":

name = request.form['name']

surname = request.form['surname']

phone\_number = request.form['phone\_number']

address = request.form['address']

add\_info = request.form['add\_info']

if(len(name) > 3 and len(surname) > 3 and len(phone\_number) >10 and len(address) > 3):

user = User(name=name, surname=surname, phone\_number=phone\_number, address=address, addition\_info=add\_info)

try:

db.session.add(user)

db.session.commit()

flash('Новий контакт успішно додано', 'success')

return redirect("/")

except:

flash("Сталася помилка при валідації нового контакту. Заповніть поля коректно", "danger")

return redirect("/")

Оскільки в головній сторінці вже додано кнопку, яка відкриває модальне меню з формою створення нового запису, то в програмі не потрібно створювати відповідний шаблон і функцію для рендеру відповідно цього шаблону. Ця функція store\_record є маршрутом POST запиту, що обробляє надходження нових записів про користувачів і зберігає їх у базі даних. Функція перевіряє чи саме відправлення форми відбулося через метод POST, отримує та валідує дані і вже через сутність User додає дані до бази даних

Можливих результатів 2:

* Валідація форми пройде успішно і результатів буде створення нового запису в базі даних, створення повідомлення про успішно доданий запис та перенаправлення з цим повідомлення на головну.
* При валідації трапиться помилка, в такому разі буде повернено помилку, про те що для полів не виконалися правила валідації.

@app.route('/edit-record/<int:record\_id>')

def edit\_record(record\_id):

user = User.query.get\_or\_404(record\_id)

return render\_template("edit\_record.html", user=user)

Функція edit\_record відповідає за повернення шаблону edit.html разом зі всією інформацією про контакт який було обрано. Отримується ідентифікатор запису користувача з URL-адреси <int:record\_id>, який отримує ціле число (ідентифікатор запису) з URL. Це робиться за допомогою динамічного маршруту результатом буде форма зі всіма даними про контакт, які можна буде редагувати відповідно зі встановленими правилами

@app.route('/update-record/<int:record\_id>', methods=['POST'])

def update\_record(record\_id):

user = User.query.get\_or\_404(record\_id)

if (len(request.form['name']) > 3 and len(request.form['surname']) > 3 and len(request.form['phone\_number']) > 10 and len(request.form['add\_info']) > 3):

user.name = request.form['name']

user.surname = request.form['surname']

user.phone\_number = request.form['phone\_number']

user.address = request.form['address']

user.addition\_info = request.form['add\_info']

db.session.commit()

flash('Контакт успішно оновлено', 'success')

return redirect("/")

else:

flash('Помилка валідації', 'danger')

return redirect(request.referrer or "/")

Функція update\_record є маршрутом, який оновлює існуючий запис про користувача у базі даних. Відповідно як і в edit функції, update\_record використовує ідентифікатор запису користувача з URL-адреси <int:record\_id>. При етапі оновлення даних також присутня така ж валідація, як і в функції зберігання контакту (store\_record). І відповідно до результату буде виведено результат зберігання.

@app.route('/delete-record/<int:record\_id>', methods=['POST'])

def delete\_record(record\_id):

user = User.query.get\_or\_404(record\_id)

db.session.delete(user)

db.session.commit()

return redirect("/")

Щоб видалити використовується функція delete\_record. Користувача якого потрібно видалити отримується через рядок пошук, якщо за таким id не знайдено користувача, то буде повернено 404 помилка. В іншому випадку буде відображено модальне вікно з підтвердженням видалення контакту.

Для розробки даного проєкту було використано досить велику кількість різних типів та структур даних, які забезпечують відповідний функціонал для повноцінної роботи сайту. Через коректність їх використання код проєкту стає більше читабельним та структурованим.

# 3.3 Опис інтерфейсу

Щоб отримати доступ до інтерфейсу потрібно через термінал перейти у директорію де лежить файл main.py командою cd example/directory/ та відкрити цей файл за допомогою команди Python main.py. Після запуску з’явиться посилання на локальний вебсайт. Щоб відкрити вебсайт в браузері потрібно скопіювати посилання та вставити його в пошуковий рядок зазвичай це буде <http://127.0.0.1:5000>.

Після відкриття сторінки з’явиться головна сторінка з даними, якщо раніше вони були збережені в базі даних

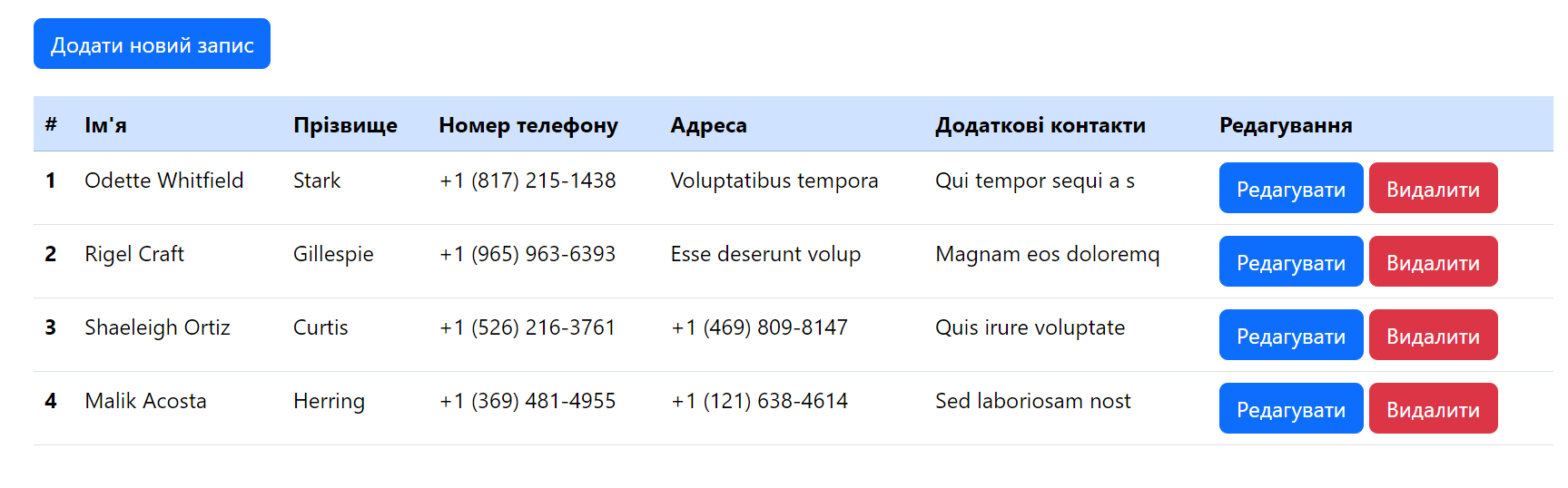


Рисунок 3.1 – Вигляд головної сторінки

Таблиця 3.3 – компоненти інтерфейсу

|  |  |
| --- | --- |
| Опис | Зображення |
| Кнопка створення нового запису |  |
| Кнопка видалення запису |  |
| Кнопка редагування запису |  |
| Модальне вікно для видалення запису |  |

Продовження таблиці 3.3

|  |  |
| --- | --- |
| Форма створення запису |  |
| Форма редагування запису |  |

# 3.4 Результати роботи програмного продукту

Фінальним результатом коду це повністю функціонуючий сайт, який включає в себе всі функції які повинні бути в адресній книзі. До форм додано валідацію, яка буде перевіряти на коректність заповнених форм за для цілісності даних в таблиці.

Результатом роботи є папка з такою файловою системою

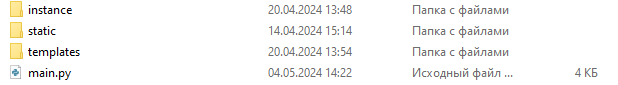


Рисунок 3.1 – Файлова система сайту

Instance – папка в якій зберігається база даних.

static – папка з всіма css та js файлами, файлова структура цієї папки складається з templates css та js папок.



Рисунок 3.2 – Файлова система папки static

tempaltes – папка зі всіма html шаблонами.

main.py головний файл, який відповідає за базу даних, маршрутизацію та вебсайт.

# Висновки

Курсова робота з дисципліни «Програмування» на тему "Адресна книга" розкриває можливості створення вебдодатків мовою програмування Python та фреймворку Flask. Дана програма є прикладом використання Flask для реалізації функціональності адресної книги. За допомогою вебдодатку користувач може зручно керувати своїми контактами, додавати нові, видаляти, редагувати та шукати існуючі.

Суть розробленого вебдодатку полягає в створенні інтерфейсу, який дозволяє користувачам здійснювати різноманітні операції з контактами, такі як додавання нового контакту, видалення, редагування та пошук. Інтерфейс додатку повинен бути зручним та інтуїтивно зрозумілим, щоб користувачі могли легко користуватися всіма його можливостями.

Перевагами вебдодатку є привабливий та простий інтерфейс, який забезпечує зручність у користуванні. Крім того, використання Flask дозволяє легко розширювати функціональність додатку за необхідності, що відкриває широкі можливості для подальшого вдосконалення.

Недоліками можна вважати те, що вебдодаток може бути не дуже гнучким у плані функціональності порівняно з більш складними рішеннями.

Узагальнюючи, робота над курсовою роботою дала можливість глибше ознайомитись з розробкою вебдодатків на Flask, а також покращити навички програмування на Python. Подальше вивчення цих тем дозволить подальшому вдосконаленню вебдодатку та розвитку в області веброзробки.

# Список використаних джерел

1. 3.12.3 python documentation. 3.12.3 Documentation. URL: https://docs.python.org/3/ (date of access: 12.05.2024).

2. Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/ (date of access: 05.04.2024).

3.Flask documentation (3.0.x). Flask Documentation (3.0.x). URL: https://flask.palletsprojects.com/en/3.0.x/ (date of access: 12.04.2024).

4. Matthes М. Python Crash Course: A Hands-On, Project-Based Introduction to Programming 1st Edition : No Starch Press, 2019. 544 p.

5. MDN web docs. MDN Web Docs. URL: https://developer.mozilla.org/en-US/ (date of access: 04.04.2024).6. The Flask Mega-Tutorial. miguelgrinberg.com. URL: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (date of access: 13.04.2024).

6. The Flask Mega-Tutorial. miguelgrinberg.com. URL: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (date of access: 13.04.2024).

7. Мелешко Є. В. Програмування на мові Python. Методичні вказівки до виконання лабораторних робіт студентами денної та заочної форми навчання спеціальностей 123 Комп’ютерна інженерія 125 Кібербезпека. Кропивницький: ЦНТУ, 2017. 58 с.

8. Методичні вказівки до курсової роботи з навчальної дисципліни «Програмування» для студентів спеціальності 123 «Комп`ютерна інженерія» /уклад. Ткаченко В.М. Ромни : 2020. 26 с.

9. Сучасний підручник з JavaScript. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/ (дата звернення: 12.04.2024).

10. Руденко В. Д., Речич Н. В., Потієнко В. О. Інформатика для загальноосвітніх навчальних закладів з поглибленим вивченням інформатики: підруч. для 9 кл.загальноосвіт. навч. закл. Харків : Вид-во «Ранок», 2017. 240 с.

# Додаток А

# (Лістинг програми)

Файл main.py

#підключення всіх потрібних компонентів

from flask import Flask, render\_template, request, redirect, url\_for, flash

from flask\_sqlalchemy import SQLAlchemy

from datetime import datetime

app = Flask(\_\_name\_\_)

app.secret\_key = "jirewrlwerkm" # Секретний ключ для захисту сесій

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///kyrsova.db' # URI бази даних SQLite

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False # Вимкнення відстеження змін

db = SQLAlchemy(app) # Ініціалізація розширення для роботи з базою даних

app.app\_context().push() # Створення контексту додатка для роботи з базою даних

# Клас моделі користувача

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True) # Унікальний ідентифікатор користувача

name = db.Column(db.String(50), nullable=False) # Ім'я користувача

surname = db.Column(db.String(50), nullable=False) # Прізвище користувача

phone\_number = db.Column(db.Integer(), nullable=False) # Номер телефону користувача

address = db.Column(db.String(256), nullable=False) # Адреса користувача

addition\_info = db.Column(db.String(256)) # Додаткова інформація про користувача

date = db.Column(db.DateTime, default=datetime.utcnow) # Дата створення запису

def \_\_repr\_\_(self):

return '<User %r>' % self.id # Представлення об'єкту класу у вигляді рядка

# Головна сторінка, яка відображає всі записи користувачів

@app.route('/')

def index():

records = User.query.order\_by(User.id).all() # Отримання всіх записів з бази даних

return render\_template("index.html", records=records) # Передача записів у шаблон для відображення

# Маршрут для збереження нового запису користувача

@app.route('/store-record', methods=["POST"])

def store\_record():

if request.method == "POST":

# Отримання даних про нового користувача з POST-запиту

name = request.form['name']

surname = request.form['surname']

phone\_number = request.form['phone\_number']

address = request.form['address']

add\_info = request.form['add\_info']

# Перевірка коректності введених даних

if(len(name) > 3 and len(surname) > 3 and len(phone\_number) > 10 and len(address) > 3):

# Створення нового об'єкту користувача з отриманими даними

user = User(name=name, surname=surname, phone\_number=phone\_number, address=address, addition\_info=add\_info)

try:

# Додавання нового користувача до бази даних та збереження змін

db.session.add(user)

db.session.commit()

flash('Новий контакт успішно додано', 'success') # Відображення повідомлення про успішну операцію

return redirect("/") # Перенаправлення на головну сторінку

except:

flash("Сталася помилка при валідації нового контакту. Заповніть поля коректно", "danger")

else:

flash("Введені дані некоректні. Будь ласка, перевірте форму!", "danger")

return redirect("/")

# Маршрут для редагування запису користувача

@app.route('/edit-record/<int:record\_id>')

def edit\_record(record\_id):

user = User.query.get\_or\_404(record\_id) # Отримання запису користувача з бази даних за його ідентифікатором

return render\_template("edit\_record.html", user=user) # Передача даних про користувача у шаблон для відображення

# Маршрут для оновлення запису користувача

@app.route('/update-record/<int:record\_id>', methods=['POST'])

def update\_record(record\_id):

# Отримання запису користувача за його ідентифікатором або виведення статусу 404, якщо запис не знайдено

user = User.query.get\_or\_404(record\_id)

# Перевірка коректності введених даних для оновлення

if (len(request.form['name']) > 3 and len(request.form['surname']) > 3 and len(request.form['phone\_number']) > 10 and len(request.form['add\_info']) > 3):

# Оновлення полів запису користувача новими значеннями з POST-запиту

user.name = request.form['name']

user.surname = request.form['surname']

user.phone\_number = request.form['phone\_number']

user.address = request.form['address']

user.addition\_info = request.form['add\_info']

# Збереження змін у базі даних та відображення повідомлення про успішну операцію

db.session.commit()

flash('Контакт успішно оновлено', 'success')

return redirect("/") # Перенаправлення на головну сторінку

else:

# Відображення повідомлення про помилку валідації та перенаправлення користувача на попередню сторінку

flash('Помилка валідації', 'danger')

return redirect(request.referrer or "/")

# Маршрут для видалення запису користувача

@app.route('/delete-record/<int:record\_id>', methods=['POST'])

def delete\_record(record\_id):

# Отримання запису користувача за його ідентифікатором або виведення статусу 404, якщо запис не знайдено

user = User.query.get\_or\_404(record\_id)

# Видалення запису користувача з бази даних та збереження змін

db.session.delete(user)

db.session.commit()

return redirect("/") # Перенаправлення на головну сторінку

# Початок роботи додатка, який буде запущений у режимі debug=False

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=False)

templates – папка з html файлами

base.html – базовий файл для всіх інших html файлів, підключає js, css та bootstrapl.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>{% block title%} {% endblock title%}</title>

<meta name="viewport" content="width=device-width, initial-scale=1" />

// Підключення бутстрапу

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">

// Підключення css файлу

<link rel="stylesheet" href="static/css/style.css">

</head>

<body>

// Блок з контентом

{% block content%} {% endblock content%}

// Підключення JavaScript

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>

<script src="static/js/delete.js"></script>

</body>

</html>

Index.html – головний файл, який відповідає за відображення таблиці з контактами

// Наслідуванян від головного файлу

{% extends 'base.html'%}

// Блок назви сторінки

{% block title%} Головна {% endblock title%}

//Блок контету

{% block content%}

<div class="container">

// кнопка нового запису

<button type="button" class="btn btn-primary add-button " data-bs-toggle="modal" data-bs-target="#modal">

Додати новий запис

</button>

{% with messages = get\_flashed\_messages(with\_categories=true) %}

// Умова при якій будуть перебиратися повідомлення

{% if messages %}

//Початок перебору повідомлень

{% for category, message in messages %}

<div class="d-flex ">

<div class="alert alert-{{ category }}" role="alert">

//Виведення повідомлення

{{ message }}

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

</div>

{% endfor %}

{% endif %}

{% endwith %}

// Модальне вікно з формою створення

<div class="modal fade" id="modal" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1"

aria-labelledby="staticBackdropLabel" aria-hidden="true">

<div class="modal-dialog">

<div class="modal-content">

//Тег форми з посиланням та методом

<form action="/store-record" method="POST">

<div class="modal-header">

<h1 class="modal-title fs-5" id="staticBackdropLabel">Новий запис</h1>

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

// Тіло модального вікна з полями для вводу

<div class="modal-body">

<div class="mb-3">

<label for="name" class="form-label">Ім'я</label>

<input type="text" class="form-control" id="name" name="name"

placeholder="Ім'я">

</div>

<div class="mb-3">

<label for="surname" class="form-label">Прізвище</label>

<input type="text" class="form-control" id="surname" name="surname"

placeholder="Прізвище">

</div>

<div class="mb-3">

<label for="phone-number" class="form-label">Номер телефону</label>

<input type="text" class="form-control" id="phone-number" name="phone\_number"

placeholder="Номер телефону">

</div>

<div class="mb-3">

<label for="address" class="form-label">Адреса</label>

<input type="text" class="form-control" id="address" name="address"

placeholder="Адреса">

</div>

<div class="mb-3">

<label for="additional-contacts" class="form-label">Додаткові контакти</label>

<input type="text" class="form-control" id="additional-contacts" name="add\_info"

placeholder="Додаткові контакти">

</div>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Закрити</button>

<button type="submit" class="btn btn-primary">Додати</button>

</div>

</form>

</div>

</div>

</div>

// Таблиця з даними

<table class="table">

<thead>

<tr class="table-primary">

<th scope="col column-titles">#</th>

<th scope="col column-titles">Ім'я</th>

<th scope="col column-titles">Прізвище</th>

<th scope="col column-titles">Номер телефону</th>

<th scope="col column-titles">Адреса</th>

<th scope="col column-titles">Додаткові контакти</th>

<th scope="col column-titles">Редагування</th>

</tr>

</thead>

<tbody>

// Перебір всіх записів

{% for record in records %}

<tr>

<th scope="row">{{record.id}}</th>

<td>{{record.name}}</td>

<td>{{record.surname}}</td>

<td>{{record.phone\_number}}</td>

<td>{{record.address}}</td>

<td>{{record.addition\_info}}</td>

<td>

<a href="{{url\_for('edit\_record', record\_id=record.id)}}" class="btn btn-primary">Редагувати</a>

<button type="button" class="btn btn-danger" data-bs-toggle="modal" data-bs-target="#deleteModal"

data-name="{{record.name}}" data-surname="{{record.surname}}" data-id="{{record.id}}"

id="deleteButton">

Видалити

</button>

</td>

</tr>

{% endfor %}

</table>

</div>

<div class="modal fade" id="deleteModal" tabindex="-1" aria-labelledby="DeleteLabel" aria-hidden="true">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h1 class="modal-title fs-5" id="DeleteLabel">Видалення контакту</h1>

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div class="modal-body">

Ви впевнені що хочете видалити цей контакт?

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-bs-dismiss="modal">Закрити</button>

<form action="" id="deleteForm" method="POST">

<button type="submit" class="btn btn-danger">Видалити</button>

</form>

</div>

</div>

</div>

</div>

{% endblock content%}

edit\_record.html – відображає форму з даними контакту для редагування

// Наслідування від головного файлу

{% extends 'base.html'%}

// блок назви

{% block title%} Редагувати {% endblock title%}

{% block content%}

<div class="container mt-4">

// повідмлення

{% with messages = get\_flashed\_messages(with\_categories=true) %}

// умова перебору

{% if messages %}

{% for category, message in messages %}

<div class="d-flex ">

<div class="alert alert-{{ category }}" role="alert">

{{ message }}

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

</div>

{% endfor %}

{% endif %}

{% endwith %}

// форма редагування

<form action="{{url\_for('update\_record', record\_id=user.id)}}" method="POST">

<div class="modal-header">

<h1 class="modal-title fs-5" id="staticBackdropLabel">Новий запис</h1>

</div>

<div class="modal-body">

// Поля для редагування інформації

<div class="mb-3">

<label for="name" class="form-label">Ім'я</label>

<input type="text" class="form-control" id="name" name="name"

placeholder="Ім'я" value="{{user.name}}">

</div>

<div class="mb-3">

<label for="surname" class="form-label">Прізвище</label>

<input type="text" class="form-control" id="surname" name="surname"

placeholder="Прізвище" value="{{user.surname}}">

</div>

<div class="mb-3">

<label for="phone-number" class="form-label">Номер телефону</label>

<input type="text" class="form-control" id="phone-number" name="phone\_number"

placeholder="Номер телефону" value="{{user.phone\_number}}">

</div>

<div class="mb-3">

<label for="address" class="form-label">Адреса</label>

<input type="text" class="form-control" id="address" name="address"

placeholder="Адреса" value="{{user.address}}" >

</div>

<div class="mb-3">

<label for="additional-contacts" class="form-label">Додаткові контакти</label>

<input type="text" class="form-control" id="additional-contacts" name="add\_info"

placeholder="Додаткові контакти" value="{{user.addition\_info}}">

</div>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary mx-2" data-bs-dismiss="modal">Закрити</button>

<button type="submit" class="btn btn-primary">Додати</button>

</div>

</form>

</div>

{% endblock content%}

static – папка в якій зберігаються js та css файли.

style.css – файл зі стилями

// додавання відступів

.add-button{

margin: 20px 0;

}

delete.js – відповідає за відображення модального вікна для видалення

// отримання елементу с класом контейнер

const container = document.querySelector(".container");

// додавання обробника на елемент

container.addEventListener("click", function (e){

if (e.target.id === "deleteButton"){

\\ отримання даних з атрибутів

let id = e.target.getAttribute("data-id");

let name = e.target.getAttribute("data-name");

let surname = e.target.getAttribute("data-surname");

\\ Місце де буде повідомлення

const deleteLabel = document.querySelector("#DeleteLabel")

deleteLabel.innerHTML = `Видалення контакту ${name} ${surname}`

\\ зміна посилання на видалення елементу

const deleteForm = document.querySelector("#deleteForm");

deleteForm.action = `delete-record/${id}`;

}

})

# Додаток Б

(Блок-схема програми)

