**АНОТАЦІЯ**

Кваліфікаційна робота включає пояснювальну записку.

Цей фрагмент коду представляє веб-додаток, розроблений за допомогою Flask, популярного мікро веб-фреймворку, написаного на Python. Додаток служить системою для керування прийомами в ветеринарній клініці, керуючи користувачами, ветеринарами, прийомами та іншими пов'язаними сутностями.

Додаток працює шляхом надання маршрутів (або кінцевих точок), з якими користувачі можуть взаємодіяти через веб-браузер. Коли користувач відвідує певну URL, сервер Flask виконує функцію, асоційовану з цією URL, і повертає відповідь, зазвичай відтворений шаблон HTML.

Дані додатка, такі як користувачі, домашні тварини, прийоми та хвороби, зберігаються в базі даних SQLite, яка є легкою, файловою базою даних. Ці дані структуровані за допомогою SQLAlchemy, бібліотеки Object-Relational Mapping (ORM) для Python, яка дозволяє додатку взаємодіяти з базою даних за допомогою об'єктів Python.

Додаток підтримує два типи користувачів - звичайних користувачів та ветеринарів. Звичайні користувачі можуть реєструватися, входити в систему, створювати прийоми для своїх домашніх тварин та переглядати ці прийоми. Ветеринари, з іншого боку, можуть реєструватися, входити в систему, переглядати прийоми, заплановані з ними, та оновлювати свою доступність.

Дані обробляються та відтворюються в браузері за допомогою шаблонів Flask, механізму шаблонів, який дозволяє генерувати динамічний вміст. Користувачі вводять дані через форми (створені з Flask-WTF, обгорткою Flask для бібліотеки WTForms), які потім перевіряються та зберігаються в базі даних.

Користувачі зазвичай взаємодіють з цією системою через веб-браузер. Після переходу на сайт користувач може вибрати реєстрацію як звичайний користувач або як ветеринар. Після реєстрації та входу в систему користувач перенаправляється на інформаційну панель, де він може переглядати та створювати прийоми.

Під час створення прийому користувачи заповнюють форму, надаючи деталі про свою домашню тварину, хворобу, бажаного ветеринара та час. Ця інформація використовується для створення нового прийому, який потім зберігається в базі даних.

Ветеринари, після входу в систему, можуть переглядати свою інформаційну панель, на якій відображаються майбутні прийоми. Вони також мають можливість оновити свою доступність.

Розробка цього додатка вимагала глибоких знань Python, Flask, SQLAlchemy та HTML/CSS для шаблонів фронтенду. Спочатку були визначені моделі бази даних для структурування даних. Далі були написані маршрути Flask для обробки HTTP-запитів. Кожен маршрут відповідає різному функціоналу додатка, наприклад, перегляду інформаційної панелі або створенню прийому. Наостанок, були створені шаблони HTML для кожного маршруту, щоб відобразити дані в браузері.

Для обробки сеансів користувачів та входів в систему використовувався Flask-Login. Це бібліотека, яка забезпечує керування сеансами та дозволяє легко отримати доступ до поточного користувача в маршрутах та шаблонах.

В цілому, цей код - це чудовий приклад того, наскільки потужним і універсальним може бути фреймворк Flask при створенні веб-додатків. Чітке розмежування зон відповідальності, логічна структура та використання добре встановлених бібліотек (наприклад, Flask-SQLAlchemy, Flask-WTF та Flask-Login) роблять код легким для розуміння, підтримки та розширення.

**Ключові слова**: Flask, Python, SQLite, SQLAlchemy, Flask-SQLAlchemy, Flask-WTF, WTForms, Flask-Login, UserMixin, LoginManager, routing, decorators, HTML templates, GET, POST, CSRF protection, secret key, running Flask application.

**ANNOTATION**

Qualification work includes an explanatory note.

This piece of code is a web application developed using Flask, a popular micro web-framework written in Python. The application serves as a system to manage appointments in a veterinary clinic, managing users, veterinarians, appointments, and other related entities.

The application works by providing routes (or endpoints) that users can interact with via a web browser. When a user visits a particular URL, the Flask server executes a function associated with that URL and returns a response, usually a rendered HTML template.

The application’s data, such as users, pets, appointments, and diseases, are stored in an SQLite database, which is a lightweight, file-based database. This data is structured using SQLAlchemy, an Object-Relational Mapping (ORM) library for Python, which enables the application to interact with the database using Python objects.

The app supports two types of users – regular users and veterinarians. Regular users can register, log in, create appointments for their pets, and view those appointments. Veterinarians, on the other hand, can register, log in, view the appointments scheduled with them, and update their availability.

The data is processed and rendered in the browser using Flask templates, a templating engine that allows for dynamic content generation. Users input data via forms (built with Flask-WTF, a Flask wrapper for the WTForms library) which is then validated and stored in the database.

Users would typically interact with this system via a web browser. After navigating to the site, a user can choose to register either as a regular user or as a veterinarian. Once registered and logged in, the user is redirected to a dashboard where they can view and create appointments.

When creating an appointment, users fill out a form providing details about their pet, the disease, and their preferred veterinarian and timings. This information is used to create a new appointment, which is then stored in the database.

Veterinarians, after logging in, can view their dashboard showing upcoming appointments. They also have the option to update their availability.

Developing this application required a strong knowledge of Python, Flask, SQLAlchemy, and HTML/CSS for the frontend templates. Initially, the database models were defined to structure the data. Next, Flask routes were written to handle HTTP requests. Each route corresponds to a different functionality of the application, such as viewing the dashboard or creating an appointment. Finally, HTML templates were created for each route to display the data in the browser.

For handling user sessions and logins, Flask-Login was used. It’s a library that provides session management and allows for the current user to be easily accessed in routes and templates.

**Keywords**: Flask, Python, SQLite, SQLAlchemy, Flask-SQLAlchemy, Flask-WTF, WTForms, Flask-Login, UserMixin, LoginManager, routing, decorators, HTML templates, GET, POST, CSRF protection, secret key, running Flask application.