**Отчет по лабораторным работам 3-4: Подключение к базе данных MySQL в Flask-приложении**

**Выполнил студент:** Ованнисян Петрос Абгарович

**Группа:** СГН3-64Б

**Обзор:**

Данный отчет фокусируется на конфигурации и подключении к базе данных MySQL в представленном Flask-приложении на Python. Анализируются этапы настройки соединения, создания таблиц, а также обеспечения безопасности посредством переменных окружения и хэширования паролей.

**Ключевые компоненты и этапы подключения к MySQL:**

**1. Зависимости:**

• flask\_sqlalchemy: Расширение Flask, упрощающее работу с SQLAlchemy (ORM).

• mysql-connector-python: Драйвер Python для соединения с MySQL. В коде он упоминается через mysql+mysqlconnector в строке подключения.

• flask\_bcrypt: Используется для безопасного хранения паролей пользователей.

• dotenv: Для загрузки конфигурационных параметров из .env файла.

• sqlalchemy: Python SQL Toolkit и Object Relational Mapper (ORM).

**2. Конфигурация через переменные окружения:**

• os.environ.get('MYSQL\_USER'): Имя пользователя MySQL.

• os.environ.get('MYSQL\_PASSWORD'): Пароль пользователя MySQL.

• os.environ.get('MYSQL\_HOST'): Хост MySQL-сервера.

• os.environ.get('MYSQL\_DB'): Имя базы данных MySQL.

• ADMIN\_LOGIN (с дефолтом 'admin') и ADMIN\_PASSWORD (с дефолтом 'adminpass'): Используются для создания администратора, если база пуста.

Использование переменных окружения является лучшей практикой для хранения конфиденциальной информации (например, учетных данных базы данных) отдельно от кода. Файл .env (не должен коммититься в репозиторий) содержит эти значения. Библиотека dotenv загружает их в окружение, откуда они доступны через os.environ.get().

**3. Строка подключения SQLAlchemy:**

app.config['SQLALCHEMY\_DATABASE\_URI'] = (

f"mysql+mysqlconnector://{os.environ.get('MYSQL\_USER')}:{os.environ.get('MYSQL\_PASSWORD')}@{os.environ.get('MYSQL\_HOST')}/{os.environ.get('MYSQL\_DB')}"

)

Эта строка определяет, как Flask-SQLAlchemy будет подключаться к MySQL. Она содержит:

• mysql+mysqlconnector: Указывает, что используется драйвер mysql-connector-python.

• Учетные данные: Имя пользователя, пароль, хост и имя базы данных, полученные из переменных окружения.

**4. Инициализация Flask-SQLAlchemy:**

db.init\_app(app)

Инициализирует расширение Flask-SQLAlchemy с приложением Flask, связывая его с базой данных, указанной в SQLALCHEMY\_DATABASE\_URI.

**5. Функция create\_db():**

• Проверка соединения: con.execute(text("SELECT 1")) Выполняет простой SQL-запрос для проверки успешности соединения с базой данных.

• Создание таблиц: db.create\_all() создает таблицы в базе данных на основе моделей SQLAlchemy, определенных в файле models.py.

• Создание администратора (если необходимо):

\* Проверяет, есть ли пользователи в базе данных (User.query.count() == 0).

\* Если пользователей нет, создает администратора с логином, паролем, именем и фамилией, полученными из переменных окружения (или значениями по умолчанию).

\* Пароль администратора хэшируется с использованием bcrypt.generate\_password\_hash() перед сохранением в базе данных.

• Обрабатывает исключения: Оборачивает процесс создания БД в try...except, чтобы при возникновении ошибок (например, неверные учетные данные) приложение завершалось с понятным сообщением.

**6. Запуск create\_db() при запуске приложения:**

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

with app.app\_context():

try:

create\_db()

except Exception as e:

print(f"Failed to initialize database: {e}")

sys.exit(1)

app.run(debug=True)

При запуске приложения, код в if \_\_name\_\_ == '\_\_main\_\_': выполняет функцию create\_db() для инициализации базы данных. app.app\_context() создает контекст приложения Flask, необходимый для работы с базой данных внутри функции.

**Анализ и выводы:**

• Безопасность: Использование переменных окружения для хранения учетных данных БД является хорошей практикой безопасности. Хэширование паролей с использованием bcrypt необходимо для защиты учетных записей пользователей.

• Надежность: Код проверяет соединение с базой данных и обрабатывает исключения, что делает приложение более устойчивым.

• Удобство: Flask-SQLAlchemy упрощает взаимодействие с базой данных, предоставляя ORM для работы с моделями данных.

• Автоматическая инициализация: Функция create\_db() автоматизирует создание таблиц и создание администратора, что упрощает развертывание приложения.

**Рекомендации:**

• Миграции базы данных: Рассмотреть использование Alembic или Flask-Migrate для управления изменениями схемы базы данных. Это облегчит обновление базы данных при изменении моделей SQLAlchemy.

• Валидация переменных окружения: Добавить проверку наличия и корректности переменных окружения перед использованием.

• Более сложная обработка ошибок: Обеспечить более подробную информацию об ошибках, возникающих при подключении к базе данных. Логирование ошибок будет полезным.

• Подумать о Dependency Injection: Вместо глобальной переменной db, лучше внедрять экземпляр db в функции, которые в нем нуждаются. Это облегчит тестирование и сделает код более модульным.

• Более сложная настройка администратора: Предоставить более гибкий способ настройки администратора (например, через командную строку или веб-интерфейс).