# Введение

Сфера логистики и доставки активно развивается, требуя создания удобных инструментов для управления процессами. Одним из таких инструментов является сервисное приложение для курьеров. Целью данного курсового проекта является разработка веб-приложения, которое обеспечивает удобный интерфейс для взаимодействия курьеров, менеджеров и клиентов, позволяя эффективно управлять процессом доставки.  
  
Основные задачи проекта:  
1. Реализовать регистрацию и авторизацию пользователей с разделением ролей (курьер, менеджер, посторонний).  
2. Создать интерфейс для управления доставками менеджером.  
3. Разработать функционал для курьеров, обеспечивающий обработку активных и доступных доставок.  
4. Обеспечить безопасность данных и защиту от основных уязвимостей.

# 1. Выбранные технологии разработки

Для реализации приложения были выбраны следующие технологии:  
1. \*\*Flask\*\* — легковесный веб-фреймворк на Python, который позволяет быстро разрабатывать веб-приложения благодаря простоте и гибкости.  
2. \*\*SQLAlchemy\*\* — библиотека ORM, которая предоставляет удобные инструменты для работы с базами данных.  
3. \*\*SQLite\*\* — легковесная встроенная база данных, подходящая для небольших проектов.  
4. \*\*HTML, CSS\*\* — для создания пользовательского интерфейса, адаптированного к потребностям пользователей.  
5. \*\*Jinja2\*\* — шаблонизатор, встроенный в Flask, для динамического формирования HTML-страниц.  
  
Такая комбинация технологий была выбрана из-за их простоты, масштабируемости и хорошей документации.

# 2. Описание разработанного приложения

Разработанное приложение имеет следующую структуру:  
  
1. Основные роли пользователей:  
- Посторонний: доступ только к статической странице с информацией о регистрации.  
- Курьер: просмотр активных доставок и возможность подтверждения их выполнения.  
- Менеджер: управление доставками, курьерами и посторонними.  
  
2. Основной функционал:  
- Регистрация пользователей с заполнением данных (ФИО, телефон, дата рождения, логин, пароль).  
- Вход с проверкой логина и пароля, предоставление доступа в зависимости от роли.  
- Для курьеров: отображение доступных и активных доставок, подтверждение завершения доставок.  
- Для менеджеров: создание, редактирование и удаление доставок, управление курьерами и посторонними, просмотр логов.

Маршрут регистрации пользователей.

@app.route('/register', methods=['GET', 'POST'])  
def register():  
 if request.method == 'POST':  
 full\_name = request.form['full\_name']  
 phone\_number = request.form['phone\_number']  
 birth\_date = datetime.strptime(request.form['birth\_date'], '%Y-%m-%d')  
 login = request.form['login']  
 password = bcrypt.generate\_password\_hash(request.form['password']).decode('utf-8')  
  
 new\_user = User(full\_name=full\_name, phone\_number=phone\_number, birth\_date=birth\_date,  
 login=login, password=password, role='postoronniy')  
 db.session.add(new\_user)  
 db.session.commit()  
 flash('Вы успешно зарегистрированы', 'success')  
 return redirect(url\_for('login'))  
 return render\_template('register.html')

Модель пользователя для базы данных.

class User(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 full\_name = db.Column(db.String(100), nullable=False)  
 phone\_number = db.Column(db.String(15), nullable=False, unique=True)  
 birth\_date = db.Column(db.Date, nullable=False)  
 login = db.Column(db.String(50), nullable=False, unique=True)  
 password = db.Column(db.String(100), nullable=False)  
 role = db.Column(db.String(50), nullable=False, default='postoronniy')

# 3. Тестирование приложения

Для проверки корректной работы приложения были проведены следующие тесты:  
  
1. Функциональные тесты:  
- Проверка регистрации, входа и авторизации для всех ролей.  
- Проверка создания, редактирования и удаления доставок менеджером.  
- Подтверждение доставки курьером.  
  
2. Модульные тесты:  
- Тестирование маршрутов Flask и взаимодействия с базой данных.  
3. Тестирование безопасности: защита от SQL-инъекций, проверка прав доступа.

# 4. Уязвимости и их защита

Для обеспечения безопасности приложения были реализованы следующие меры:  
- Использование ORM для защиты от SQL-инъекций.  
- Хэширование паролей с использованием bcrypt.  
- Разделение прав доступа для ролей.  
- Экранирование данных в шаблонах для защиты от XSS-атак.

# 5. Выводы и результаты

Разработанное приложение успешно решает поставленные задачи и обладает гибкостью для дальнейшего развития.

# 6. Список использованных источников

1. Flask Documentation - https://flask.palletsprojects.com/  
2. SQLAlchemy Documentation - https://docs.sqlalchemy.org/  
3. Jinja2 Documentation - https://jinja.palletsprojects.com/  
4. HTML и CSS - https://developer.mozilla.org/ru/  
5. bcrypt - https://pypi.org/project/bcrypt/