Geekbrains

**Веб сайт Визитка с использованием технологии языка программирования Python и микрофрейворка Flask**

IT-Специалист

Python – программист

Иванов А. А

Москва

2024

**Содержание**

Введение 2

Техническое задание 4

Основы языка программирования Python 6

Основы микрофреймворка Flask 10

Базы данных и Flask-SQLAlchemy 13

Разработка веб-приложения "Визитка" 18

Тестирование веб-приложения 20

Оптимизация и масштабирование 27

Внедрение и деплой веб-приложения 32

Заключение 50

Список использованных источников 52

Приложения 53

**1. Введение**

1.1. Актуальность темы

Современный мир в сфере информационных технологий стремительно развивается, и веб-приложения играют ключевую роль в повседневной жизни. В условиях постоянных изменений и роста конкуренции, создание привлекательных и функциональных веб-приложений становится важным фактором успешного представления бизнеса и личных проектов в онлайн-среде. Данная работа посвящена разработке веб-приложения "Визитка", которое может служить эффективным инструментом для представления личной информации, портфолио, или бизнеса в интернете.

1.2. Цели и задачи исследования

**Целью** данной работы является разработка и исследование веб-приложения "Визитка" с использованием языка программирования Python и микрофреймворка Flask.

**Задачи** работы включают в себя анализ требований к приложению, изучение основ Python и Flask, проектирование базы данных, реализацию функционала, тестирование, оптимизацию, и деплой приложения.

1.3. Обзор существующих решений

Проведен анализ существующих веб-приложений "визитка", выявлены их преимущества и недостатки. Работы посвящены выявлению современных трендов в дизайне и функционале веб-приложений подобного типа. Рассмотрены популярные решения и их технологические особенности.

1.4. Методология исследования

В работе использована комбинация методов исследования, включая аналитический обзор литературы, анализ требований, программирование, тестирование и оптимизацию. Процесс разработки веб-приложения будет осуществляться с использованием принципов гибкой методологии разработки ПО (Agile).

1.5. Структура работы

Работа организована следующим образом: во втором разделе представлено техническое задание, определяющее функциональные и нефункциональные требования к приложению. Третий и четвёртый разделы посвящены основам языка программирования Python и микрофреймворка Flask соответственно. В пятом разделе рассматривается вопрос взаимодействия приложения с базой данных, а в шестом - сам процесс разработки "Визитки". Тестирование веб-приложения представлено в седьмом разделе. В восьмом и девятом разделах обсуждаются вопросы оптимизации и деплоя приложения. Заключение, список использованных источников, а также приложения завершают данную работу.

1.6. Обоснование выбора Flask для разработки

Выбор микрофреймворка Flask обоснован его легкостью в освоении, простотой в использовании и гибкостью для создания различных типов веб-приложений. Flask также обладает активным сообществом и широким функционалом благодаря множеству расширений. В данной работе Flask выбран в качестве инструмента для разработки веб-приложения "Визитка", с учетом его современности и популярности в сообществе разработчиков.

**2. Техническое задание**

2.1. Функциональные требования

**2.1.1. Регистрация пользователя**

* Возможность создания аккаунта для пользователя.
* Защита пароля и хранение учетных данных в безопасной форме.
* Валидация данных при регистрации.

**2.1.2. Создание и редактирование визитки**

* Возможность пользователя создавать и редактировать свою визитку.
* Добавление информации о себе, контактных данных, ссылках на социальные сети.
* Возможность загрузки и управления изображением (фотографией) пользователя.

**2.1.3. Просмотр и поиск визиток**

* Возможность просмотра отдельных визиток для получения подробной информации.

**2.1.4. Аутентификация и авторизация**

* Защищенный вход в систему с использованием учетных данных.
* Разграничение прав доступа для пользователей.
* Возможность управления своей визиткой только зарегистрированными пользователями.

2.2. Нефункциональные требования

**2.2.1. Производительность**

* Время загрузки страницы не должно превышать 2 секунды.
* Поддержка работы с приложением до 1000 одновременных пользователей.

**2.2.2. Безопасность**

* Защита от SQL-инъекций и других видов атак.
* Хранение паролей в виде хэшей.
* Использование токенов для сессий и передачи данных.

**2.2.3. Масштабируемость**

* Возможность легкого масштабирования системы для увеличения числа пользователей.
* Поддержка кэширования для улучшения производительности.

2.3. Архитектура веб-приложения

**2.3.1. MVC-архитектура**

* Использование модели-представление-контроллер (MVC) для структурирования кода.
* Разделение логики бизнес-процессов, представления и управления данными.

**2.3.2. RESTful API**

* Построение веб-приложения с использованием RESTful принципов для обеспечения легкости взаимодействия с клиентской частью.

2.4. Взаимодействие с базой данных

**2.4.1. Выбор СУБД**

* Использование СУБД PostgreSQL для хранения данных приложения.

**2.4.2. ORM SQLAlchemy**

* Использование SQLAlchemy в качестве ORM (Object-Relational Mapping) для более удобной работы с базой данных через Python-код.

**2.4.3. Структура базы данных**

* Определение структуры базы данных для хранения информации о пользователях и их визитках.
* Реализация связей между таблицами для эффективного запроса данных.

**2.4.4. Миграции данных**

* Использование механизма миграций для обновления структуры базы данных при изменении моделей.

**3. Основы языка программирования Python**

3.1. Основные принципы языка Python

Python - высокоуровневый язык программирования, ориентированный на читаемость кода и удобство разработчика. Основные принципы Python включают:

**3.1.1. Простота и ясность кода**

* Отсутствие необходимости в явных объявлениях типов делает код более читаемым и понятным.
* Использование отступов вместо фигурных скобок для обозначения блоков кода способствует ясности и структурированности.

**3.1.2. Динамическая типизация**

* Переменные не требуют объявления типа заранее, что упрощает и ускоряет процесс разработки.

**3.1.3. Обширная стандартная библиотека**

* Python поставляется с обширной библиотекой, предоставляющей широкий набор инструментов для разработки.

**3.1.4. Объектно-ориентированное программирование (ООП)**

* Поддержка ООП в Python позволяет создавать модульный и масштабируемый код.

Классы и объекты:

В Python все является объектом, и каждый объект принадлежит к какому-то классу. Класс определяет свойства и методы объектов.

Пример создания класса и объекта:

python

Copy code

class Dog:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def bark(self):

print(f"{self.name} говорит: Гав!")

# Создание объекта класса Dog

my\_dog = Dog("Шарик", 3)

my\_dog.bark()

3.2. Структуры данных в Python

**3.2.1. Списки (Lists) и Кортежи (Tuples)**

* Основные структуры данных для хранения упорядоченных коллекций элементов.
* Списки могут изменяться, кортежи - нет.

**3.2.2. Словари (Dictionaries)**

* Хранят данные в форме ключ-значение, обеспечивая быстрый доступ к значениям.

**3.2.3. Множества (Sets)**

* Коллекции уникальных элементов без порядка.

**3.2.4. Строки (Strings)**

* Операции со строками и методы облегчают работу с текстовой информацией.

**3.2.5. Стеки и очереди**

* Использование списков для реализации стеков и очередей.

3.3. ООП в Python

**3.3.1. Классы и объекты**

* Классы представляют собой шаблоны для создания объектов, которые могут содержать атрибуты и методы.

**3.3.2. Наследование и полиморфизм**

* Возможность создавать новые классы на основе существующих и использовать их функциональность.

**3.3.3. Инкапсуляция**

* Скрытие деталей реализации внутри класса, обеспечивая доступ только к необходимым частям.

**3.3.4. Абстракция**

* Способность представлять объекты исключительно в терминах их важных характеристик, игнорируя мелкие детали.

**3.3.5. Методы и свойства**

* Использование методов для описания действий объекта и свойств для представления его состояния.

**Выводы**

Основы языка программирования Python предоставляют разработчикам мощный инструментарий для создания эффективных и читаемых программ. В следующих разделах дипломной работы мы рассмотрим, как эти основы могут быть применены при разработке веб-приложения "Визитка" с использованием микрофреймворка Flask.

**4. Основы микрофреймворка Flask**

**Flask - это легкий микрофреймворк для создания веб-приложений на языке программирования Python. Вместо того чтобы предоставлять все необходимые компоненты для создания сложных приложений, Flask предоставляет минимальный набор инструментов, оставляя разработчику свободу выбора библиотек и инструментов для реализации конкретных функций. Вот основные аспекты Flask:**

4.1. Установка и настройка Flask

**4.1.1. Установка Python и pip**

* Инструкции по установке Python и инструмента pip для управления пакетами.

Чтобы установить Flask, можно воспользоваться инструментом управления пакетами Python - pip. Выполните следующую команду в командной строке:

pip install Flask

**4.1.2. Установка и виртуальное окружение**

* Создание виртуального окружения для изоляции зависимостей проекта.
* Установка Flask через pip.

**4.1.3. Структура проекта**

* Рекомендации по организации файлов и каталогов в проекте Flask.

4.2. Роутинг и представления

**4.2.1. Определение маршрутов**

* Создание маршрутов для различных страниц приложения.
* Использование декораторов для привязки функций-обработчиков к URL-адресам.

Flask использует декораторы для определения маршрутов и обработчиков запросов. Например:

@app.route('/about')

def about():

return 'Это страница "О нас".'

**4.2.2. Передача параметров**

* Получение параметров из URL и передача их в представления.

**4.2.3. Работа с запросами и ответами**

* Обработка HTTP-запросов и формирование HTTP-ответов.

Flask предоставляет инструменты для обработки данных, отправленных через формы. Пример:

from flask import Flask, render\_template, request

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

# Обработка данных

return render\_template('home.html')

4.3. Шаблонизация в Flask

**4.3.1. Использование шаблонов Jinja2**

* Интеграция Jinja2 для создания динамических HTML-страниц.
* Передача данных из представлений в шаблоны.

**4.3.2. Условия и циклы в шаблонах**

* Применение условий и циклов для динамического отображения данных в шаблонах.

4.4. Работа с формами

**4.4.1. Создание HTML-форм**

* Определение форм в HTML для ввода данных.

**4.4.2. Обработка форм в Flask**

* Получение данных из форм в представлениях Flask.
* Валидация ввода и обработка ошибок.

**4.4.3. Защита от CSRF-атак**

* Реализация защиты от атак, связанных с подделкой межсайтовых запросов (CSRF).

**Выводы**

Освоив основы микрофреймворка Flask, мы создадим прочную основу для разработки веб-приложения "Визитка". Установка, настройка, роутинг, представления, шаблонизация и работа с формами - все эти компоненты Flask будут внедрены в наш проект, обеспечивая эффективность и гибкость при разработке функциональности приложения.

**5. Базы данных и Flask-SQLAlchemy**

5.1. Введение в базы данных

Flask-SQLAlchemy - это расширение Flask, которое интегрирует SQLAlchemy, мощный инструмент для работы с базами данных, с фреймворком Flask. SQLAlchemy предоставляет ORM (Object-Relational Mapping) для работы с базами данных, что делает взаимодействие с ними более удобным и абстрагирует разработчика от прямых SQL-запросов.

**5.1.1. Роль баз данных в веб-приложениях**

* Обзор роли баз данных в хранении и организации данных приложения.
* Преимущества использования баз данных в веб-разработке.

**5.1.2. Типы баз данных**

* Обзор различных типов баз данных, таких как реляционные и NoSQL базы данных.

**5.1.3. Введение в язык SQL**

* Основы языка SQL для работы с реляционными базами данных.

5.2. Интеграция Flask и SQLAlchemy

**5.2.1. Установка и настройка Flask-SQLAlchemy**

* Установка Flask-SQLAlchemy и настройка подключения к базе данных.

pip install Flask-SQLAlchemy

**5.2.2. Создание экземпляра SQLAlchemy**

* Инициализация объекта SQLAlchemy для взаимодействия с базой данных.

**5.2.3. Конфигурация базы данных в Flask-приложении**

* Установка параметров подключения к базе данных в конфигурационных файлах Flask.

5.3. Модели данных

**5.3.1. Определение моделей SQLAlchemy**

* Создание классов моделей для представления таблиц в базе данных.
* Определение полей и их типов.

**5.3.2. Отношения между моделями**

* Определение связей и отношений между таблицами в базе данных.

**5.3.3. Запросы к базе данных**

* Использование SQLAlchemy для выполнения запросов к базе данных.

5.4. Миграции данных

**5.4.1. Введение в миграции данных**

* Обзор концепции миграций данных и их необходимость.

**5.4.2. Инструмент для миграций Flask-Migrate**

* Установка и настройка Flask-Migrate для управления миграциями данных.

**5.4.3. Создание и применение миграций**

* Создание первой миграции для инициализации базы данных.
* Применение миграций для обновления схемы базы данных.

**Пример использования Flask-SQLAlchemy:**

1. **Настройка:**

Для начала нужно настроить подключение к базе данных в вашем Flask-приложении. Например, для SQLite базы данных:

pythonCopy code

from flask import Flask from flask\_sqlalchemy import SQLAlchemy app = Flask(\_\_name\_\_) app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///site.db' db = SQLAlchemy(app)

Здесь мы создаем объект Flask и настраиваем его для использования базы данных SQLite, указывая путь к файлу базы данных.

1. **Определение модели:**

Затем нужно определить модели - классы, которые будут отображены на таблицы в базе данных. Пример модели пользователя:

pythonCopy code

class User(db.Model): id = db.Column(db.Integer, primary\_key=True) username = db.Column(db.String(20), unique=True, nullable=False) email = db.Column(db.String(120), unique=True, nullable=False) def \_\_repr\_\_(self): return f"User('{self.username}', '{self.email}')"

В данном примере, **User** - это модель пользователя, которая будет отображаться на таблицу в базе данных.

1. **Создание и миграция базы данных:**

После определения моделей, нужно создать базу данных и выполнить миграции (если используется SQLite):

pythonCopy code

# Создание базы данных db.create\_all() # Пример миграции (если используется SQLite) # flask db init # flask db migrate -m "initial migration" # flask db upgrade

Команды **flask db init**, **flask db migrate**, и **flask db upgrade** используются для создания структуры базы данных и применения изменений.

1. **Использование в приложении:**

Теперь можно использовать базу данных внутри Flask-приложения. Например, добавление нового пользователя:

pythonCopy code

from your\_app import User new\_user = User(username='john\_doe', email='john@example.com') db.session.add(new\_user) db.session.commit()

Здесь **db.session** - это сессия базы данных, которая отслеживает изменения и применяет их.

**Дополнительные возможности Flask-SQLAlchemy:**

* **Отношения между таблицами:**

pythonCopy code

class Post(db.Model): id = db.Column(db.Integer, primary\_key=True) title = db.Column(db.String(100), nullable=False) content = db.Column(db.Text, nullable=False) user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False) # Отношение "один ко многим" между User и Post User.posts = db.relationship('Post', backref='author', lazy=True)

* **Запросы к базе данных:**

pythonCopy code

# Пример запроса всех пользователей users = User.query.all() # Пример запроса пользователя по имени user = User.query.filter\_by(username='john\_doe').first()

* **Использование Flask-Migrate:**

Flask-Migrate - это дополнение к Flask-SQLAlchemy, которое облегчает управление миграциями базы данных:

bashCopy code

pip install Flask-Migrate

Использование:

bashCopy code

flask db init flask db migrate -m "initial migration" flask db upgrade

Эти команды помогут создавать и применять миграции при изменении структуры базы данных.

Flask-SQLAlchemy облегчает работу с базой данных в ваших Flask-приложениях, предоставляя удобные инструменты для определения моделей, создания и миграции баз данных, а также выполнения запросов

**Выводы**

Использование баз данных в сочетании с Flask-SQLAlchemy позволяет эффективно управлять данными в веб-приложении "Визитка". Разработка моделей данных и применение миграций обеспечат стабильность и масштабируемость системы, а интеграция SQLAlchemy позволит проводить эффективные запросы к базе данных.

**Разработка веб-приложения "Визитка"**

6.1. Создание структуры проекта

**6.1.1. Организация файлов и каталогов**

* Создание корректной структуры каталогов для проекта "Визитка" в соответствии с принципами организации Flask-проекта.

**6.1.2. Разделение кода на модули**

* Реализация модульной структуры проекта для легкости сопровождения и расширения.

**6.1.3. Использование шаблонов**

* Интеграция шаблонов для HTML-страниц с использованием Jinja2 для отделения представления от логики.

6.2. Реализация основных функций

**6.2.1. Создание и редактирование визитки**

* Разработка механизма создания и редактирования визиток для зарегистрированных пользователей.

**6.2.2. Просмотр и поиск визиток**

* Реализация функционала просмотра всех визиток и поиска по различным критериям.

**6.2.3. Авторизация и приватность**

* Обеспечение доступа только к собственным визиткам пользователя.
* Разработка механизма аутентификации для зарегистрированных пользователей.

6.3. Обработка форм и валидация данных

**6.3.1. Создание HTML-форм для ввода данных**

* Разработка форм для ввода информации о пользователе и его визитке.

**6.3.2. Обработка форм в Flask**

* Реализация механизма обработки данных, полученных из форм, в представлениях Flask.

**6.3.3. Валидация ввода и обработка ошибок**

* Интеграция механизмов валидации для корректного ввода данных пользователем.
* Обработка ошибок и предоставление информативных сообщений.

6.4. Аутентификация и авторизация пользователей

**6.4.1. Регистрация и вход в систему**

* Разработка механизма регистрации новых пользователей и их аутентификации.

**6.4.2. Разграничение прав доступа**

* Определение ролей пользователей и разграничение доступа в зависимости от их статуса.

**6.4.3. Использование сессий и токенов**

* Реализация безопасной аутентификации с использованием сессий и токенов.

**Выводы**

Разработка веб-приложения "Визитка" с использованием Flask включает в себя создание структуры проекта, реализацию основных функций, обработку форм и валидацию данных, а также аутентификацию и авторизацию пользователей. Развертывание этих компонентов обеспечит полноценную функциональность приложения, позволяя пользователям эффективно управлять своей визиткой в онлайн-среде.

**7. Тестирование веб-приложения**

7.1. Модульное тестирование

**7.1.1. Значение модульного тестирования**

Модульное тестирование — это процесс тестирования отдельных модулей или компонентов программы для проверки их корректности. В контексте веб-приложения "Визитка", модули могут включать в себя отдельные функции, классы, а также роуты и представления Flask.

**7.1.2. Инструменты для модульного тестирования**

Выбор и использование библиотек для тестирования, таких как **unittest** или **pytest**. Создание тестовых случаев для проверки функциональности каждого модуля.

**7.1.3. Мокирование и фикстуры**

Применение мокирования для изоляции тестируемых модулей от зависимостей. Использование фикстур для обеспечения начального состояния перед запуском тестов.

7.2. Интеграционное тестирование

**7.2.1. Цели интеграционного тестирования**

Проверка взаимодействия между различными модулями и компонентами приложения. Гарантия правильной работы веб-приложения в целом, а не только его отдельных частей.

Интеграционное тестирование - это фаза тестирования программного обеспечения, направленная на проверку взаимодействия между различными компонентами или модулями системы. Цели интеграционного тестирования могут варьироваться в зависимости от конкретных требований проекта, но обычно они включают в себя следующие аспекты:

Проверка взаимодействия между модулями:

Цель: Убедиться, что различные модули, компоненты или сервисы взаимодействуют друг с другом корректно и эффективно.

Пример: Проверка передачи данных между слоями приложения, обмен данными между микросервисами.

Выявление ошибок при интеграции:

Цель: Обнаружить ошибки, которые могут возникнуть в результате интеграции различных компонентов или изменений в коде.

Пример: Ошибки в интерфейсах между модулями, неправильное формирование данных при передаче между системами.

Подтверждение соответствия требованиям:

Цель: Проверить, что интегрированные компоненты соответствуют требованиям и спецификациям проекта.

Пример: Удостоверение, что данные, полученные от одного модуля, правильно обрабатываются и используются другим модулем.

Проверка стабильности системы:

Цель: Оценить стабильность системы при работе с интегрированными компонентами.

Пример: Мониторинг ресурсов и производительности при выполнении тестов с интегрированными компонентами.

Тестирование взаимодействия с внешними системами:

Цель: Проверка взаимодействия с внешними системами или сервисами, такими как базы данных, API, веб-сервисы.

Пример: Интеграция с базой данных для проверки правильности запросов и обработки данных.

Тестирование обработки ошибок и исключительных ситуаций:

Цель: Убедиться, что система может обрабатывать ошибки и исключительные ситуации при взаимодействии между компонентами.

Пример: Обработка сетевых ошибок, неправильных запросов, недоступности внешних сервисов.

Проверка сценариев использования в реальной среде:

Цель: Оценить, как система ведет себя в реальных условиях использования.

Пример: Сценарии использования, которые включают в себя взаимодействие нескольких компонентов для выполнения конкретных задач.

Проверка производительности и масштабируемости:

Цель: Оценить производительность и масштабируемость системы при работе с интегрированными компонентами.

Пример: Тестирование производительности при высоких нагрузках или проверка масштабируемости при увеличении числа пользователей.

Уверенность в качестве перед выпуском:

Цель: Предоставить уверенность в том, что система работает стабильно и корректно перед выпуском в продакшн.

Пример: Проведение комплексных интеграционных тестов перед каждым релизом для выявления потенциальных проблем.

Цели интеграционного тестирования направлены на обеспечение надежности и эффективности взаимодействия между компонентами системы, что важно для обеспечения качественного функционирования веб-приложений и служб.

**7.2.2. Инструменты для интеграционного тестирования**

Использование фреймворков для интеграционного тестирования, таких как **Selenium** или **Cypress**. Создание сценариев тестирования, которые моделируют взаимодействие пользователя с веб-приложением.

**7.2.3. Тестирование API**

Проверка корректности работы API веб-приложения. Интеграционное тестирование запросов и ответов API.

7.3. Тестирование безопасности

**7.3.1. Значение тестирования безопасности**

Гарантирование, что веб-приложение обладает необходимыми мерами безопасности. Предотвращение возможных атак, таких как инъекции, подделка запросов, перехват данных и другие.

**7.3.2. Инструменты для тестирования безопасности**

Использование специализированных инструментов, таких как **OWASP ZAP** или **Nmap**. Проведение анализа на уязвимости и тестирование на проникновение.

**7.3.3. Проверка правильности настроек аутентификации и авторизации**

Тестирование на наличие уязвимостей в механизмах аутентификации и авторизации. Проверка, что только аутентифицированным пользователям предоставляется доступ к соответствующей функциональности.

Проверка правильности настроек аутентификации и авторизации - это важный этап обеспечения безопасности информационных систем. Аутентификация и авторизация - это два ключевых процесса, которые гарантируют, что пользователь имеет доступ только к тем ресурсам и функциям, которые ему разрешены.

Вот более подробное объяснение каждого из этих понятий:

1. **Аутентификация:**
   * *Что это:* Аутентификация - это процесс проверки подлинности пользователя, подтверждение того, что он является тем, за кого себя выдаёт.
   * *Методы:* Обычно включают в себя использование паролей, биометрических данных (отпечатков пальцев, сканов сетчатки глаза), аутентификации по двухфакторному методу (2FA) и других методов.
2. **Авторизация:**
   * *Что это:* Авторизация - это процесс определения прав доступа пользователя к ресурсам или функциям системы после успешной аутентификации.
   * *Уровни доступа:* Включает в себя определение того, какие действия или ресурсы пользователь может использовать на основе его роли, группы или индивидуальных настроек.

Теперь, когда мы понимаем основные концепции, давайте рассмотрим, как проводится проверка настроек аутентификации и авторизации:

1. **Проверка аутентификации:**
   * Проверьте корректность настроек паролей: требования к сложности, срок действия, исключение использования слабых паролей.
   * Проверьте использование двухфакторной аутентификации (2FA) и его правильную настройку.
   * Проверьте правильность настроек биометрической аутентификации, если она используется.
   * Мониторинг неудачных попыток входа для выявления подозрительной активности.
2. **Проверка авторизации:**
   * Убедитесь, что каждый пользователь имеет уникальный идентификатор и правильно настроенные роли.
   * Проверьте, что разрешения на доступ к ресурсам или функциям соответствуют ролям пользователя.
   * Проведите регулярную рецензию и обновление прав доступа для актуализации с изменениями в организационной структуре или политиках безопасности.
   * Внедрите механизмы мониторинга для обнаружения необычной активности, такой как попытки несанкционированного доступа.

Эти шаги помогут обеспечить, что система аутентификации и авторизации работает правильно, что уменьшает риски нарушений безопасности и сохраняет конфиденциальность данных. Постоянный мониторинг и обновление политик безопасности помогут поддерживать высокий уровень защиты.

**Выводы**

Тестирование веб-приложения "Визитка" является ключевым этапом в его разработке. Модульное тестирование позволяет проверить корректность отдельных компонентов, интеграционное тестирование — взаимодействие между ними, а тестирование безопасности — обеспечивает защиту от потенциальных атак. Этот комплексный подход к тестированию гарантирует, что приложение работает стабильно, эффективно и безопасно.

**Оптимизация и масштабирование**

8.1. Оптимизация кода

**8.1.1. Использование инструментов профилирования**

* Применение инструментов, таких как **cProfile** в Python, для анализа производительности кода.
* Определение узких мест и оптимизация вычислительных задач.

**8.1.2. Кеширование результатов**

* Внедрение механизмов кеширования для ускорения доступа к часто используемым данным.
* Использование кэшей для хранения результатов запросов к базе данных или вычислений.

**8.1.3. Асинхронное программирование**

* Реализация асинхронных задач для обработки нескольких операций параллельно.
* Использование библиотек, таких как **asyncio**, для улучшения отзывчивости приложения.

8.2. Масштабирование приложения

**8.2.1. Вертикальное масштабирование**

* Увеличение мощности сервера или добавление ресурсов для обработки большего числа запросов.
* Использование балансировщиков нагрузки для равномерного распределения запросов.

Вертикальное масштабирование, также известное как "масштабирование вверх" или "увеличение производительности", представляет собой метод расширения ресурсов сервера путем увеличения мощности вычислительного оборудования. Этот подход используется для улучшения производительности системы путем увеличения ее вычислительных, памятных или хранилищеспособностей. Вот более подробное описание вертикального масштабирования:

Принципы вертикального масштабирования:

Увеличение мощности сервера:

Цель: Увеличить общую производительность путем увеличения ресурсов на одном сервере.

Пример: Установка более мощного процессора, увеличение объема оперативной памяти, подключение более быстрых накопителей.

Особенности архитектуры:

Монолитная архитектура: Вертикальное масштабирование часто используется в монолитных приложениях, где все компоненты работают внутри одного монолитного приложения.

Однопоточные приложения: Вертикальное масштабирование особенно эффективно для однопоточных приложений, где увеличение мощности процессора может прямо влиять на производительность.

Применение в виртуализированных средах:

Виртуализация: Вертикальное масштабирование легче реализовать в виртуализированных окружениях, где можно легко изменять конфигурацию виртуальных машин.

Облачные решения: В облачных средах вертикальное масштабирование может быть достигнуто с использованием облачных сервисов, позволяющих быстро масштабировать ресурсы.

Преимущества вертикального масштабирования:

Простота управления:

Управление ресурсами происходит на уровне одного сервера, что может быть проще, чем управление кластером серверов.

Эффективность для небольших проектов:

Вертикальное масштабирование может быть более эффективным для небольших проектов, где недостаточно высоких требований к масштабируемости.

Более легкая настройка:

Увеличение мощности сервера может быть проще в настройке и выполнении по сравнению с управлением кластером серверов.

Недостатки вертикального масштабирования:

Ограничения по производительности:

Вертикальное масштабирование имеет свои ограничения, и, в конечном итоге, вы можете достичь предела по мощности для конкретного сервера.

Остановка работы при масштабировании:

Обычно требуется перезагрузка сервера или отключение для применения изменений в его конфигурации, что может привести к простою системы.

Сложности масштабирования в горизонтальном направлении:

Если требуется еще больше производительности, масштабирование может стать сложным, и более эффективным решением может быть горизонтальное масштабирование.

Пример использования:

Предположим, у вас есть веб-приложение, и при увеличении числа пользователей уровень нагрузки на сервер становится критическим. Вместо добавления новых серверов и распределения нагрузки между ними, вы можете выбрать вертикальное масштабирование, увеличив ресурсы (например, процессор, оперативная память) на существующем сервере, чтобы обеспечить более высокую производительность.

Выбор между вертикальным и горизонтальным масштабированием зависит от требований вашего проекта, бюджета, и ожиданий по масштабируемости.

**8.2.2. Горизонтальное масштабирование**

* Разделение приложения на независимые компоненты и их размещение на нескольких серверах.
* Использование технологий контейнеризации, таких как Docker, для упрощения масштабирования.

8.3. Обработка ошибок и логирование

**8.3.1. Структурированное логирование**

* Внедрение структурированных журналов для улучшения читаемости и анализа логов.
* Использование библиотек, таких как **loguru** или **logging** в Python.

**8.3.2. Обработка и отображение ошибок**

* Разработка механизма обработки и отображения ошибок для уведомления администраторов и улучшения пользовательского опыта.
* Реализация страниц с информативными сообщениями о возникших проблемах.

**8.3.3. Мониторинг и трассировка**

* Внедрение инструментов мониторинга для отслеживания состояния приложения.
* Использование трассировки запросов для определения узких мест и улучшения производительности.

**Выводы**

Оптимизация и масштабирование важны для обеспечения эффективности и стабильности веб-приложения "Визитка". Оптимизация кода, масштабирование приложения и обработка ошибок с логированием позволяют создать устойчивое и отзывчивое приложение, способное обрабатывать увеличивающуюся нагрузку.

**Внедрение и деплой веб-приложения**

9.1. Выбор хостинга

**9.1.1. Критерии выбора хостинга**

* Анализ требований проекта и определение критериев выбора хостинга.
* Рассмотрение факторов, таких как масштабируемость, стоимость, поддержка языков программирования.

**9.1.2. Обзор популярных хостинг-провайдеров**

* Исследование предложений от провайдеров, таких как AWS, Heroku, DigitalOcean, и выбор наилучшего варианта для проекта.

На момент последнего обновления моих знаний в январе 2022 года существует множество хостинг-провайдеров, предоставляющих услуги размещения веб-сайтов и приложений. Каждый провайдер имеет свои особенности, преимущества и недостатки. Ниже приведен краткий обзор нескольких популярных хостинг-провайдеров:

Amazon Web Services (AWS):

Преимущества: Масштабируемость, широкий спектр услуг, включая вычислительные мощности, базы данных, хранилище и многие другие. Глобальная инфраструктура с множеством центров обработки данных.

Недостатки: Сложное ценообразование, начальная кривая обучения.

Microsoft Azure:

Преимущества: Интеграция с другими продуктами Microsoft, обширный выбор служб, поддержка разных операционных систем.

Недостатки: Также, как и AWS, ценообразование может быть сложным.

Google Cloud Platform (GCP):

Преимущества: Мощные инструменты для анализа данных и машинного обучения, хорошая масштабируемость, конкурентоспособные цены.

Недостатки: Уровень распространенности может быть ниже, чем у AWS и Azure.

DigitalOcean:

Преимущества: Простота использования, прозрачное ценообразование, быстрое развертывание виртуальных машин (Droplets).

Недостатки: Меньший выбор услуг по сравнению с крупными облачными провайдерами.

Bluehost:

Преимущества: Доступные тарифы, простота использования, бесплатный домен при регистрации.

Недостатки: Ограниченные опции на более дешевых тарифах, особенно для крупных проектов.

SiteGround:

Преимущества: Высокая производительность, обширные функции безопасности, поддержка технологии SSD.

Недостатки: Некоторые дополнительные услуги могут быть дороже.

HostGator:

Преимущества: Доступные тарифы, множество функций, бесплатный SSL-сертификат.

Недостатки: Некоторые отзывы указывают на проблемы с поддержкой.

A2 Hosting:

Преимущества: Высокая скорость хостинга, турбированные серверы, тарифы для разработчиков.

Недостатки: Не самый дешевый вариант, особенно для более мощных серверов.

Hostinger:

Преимущества: Очень низкие цены, бесплатный SSL, опция облачного хостинга.

Недостатки: Ограниченные дополнительные функции на более дешевых тарифах.

InMotion Hosting:

Преимущества: Бесплатные секционированные SSL-сертификаты, высокая производительность, бесплатный домен.

Недостатки: Цены могут быть выше в сравнении с некоторыми конкурентами.

При выборе хостинг-провайдера важно учитывать требования вашего проекта, бюджет, ожидания по производительности и поддержке. Перед принятием решения рекомендуется изучить отзывы пользователей, особенности тарифов и предоставляемые услуги каждого конкретного провайдера.

9.2. Настройка сервера

Настройка сервера - это важный этап в обеспечении стабильной и безопасной работы веб-приложений или других служб. Ниже представлено более подробное описание основных аспектов настройки сервера:

Выбор операционной системы:

Linux: Многие веб-серверы используют Linux, такие как Ubuntu, CentOS, или Debian. Linux часто предпочтителен из-за своей стабильности, открытого исходного кода и богатых возможностей для автоматизации.

Windows: Если вы разрабатываете приложение, зависящее от технологий Microsoft, то выбор Windows Server может быть предпочтительным.

Установка программного обеспечения:

Веб-сервер: Установите веб-сервер, такой как Apache, Nginx или Microsoft IIS, в зависимости от ваших потребностей и предпочтений.

База данных: Если приложение использует базу данных, установите и настройте соответствующую систему управления базами данных (например, MySQL, PostgreSQL, MongoDB).

Среда выполнения приложения: Установите и настройте необходимую среду выполнения (например, Node.js, Python, PHP) и другие зависимости.

Конфигурация веб-сервера:

Настройка виртуальных хостов: Если вы размещаете несколько сайтов, настройте виртуальные хосты для разделения их конфигураций.

SSL/TLS: Включите и настройте SSL/TLS для обеспечения безопасной передачи данных по протоколу HTTPS.

Настройка безопасности:

Брандмауэр: Настройте брандмауэр для фильтрации сетевого трафика и защиты сервера от несанкционированного доступа.

Обновления: Регулярно обновляйте операционную систему, веб-сервер и другие компоненты, чтобы устранять уязвимости безопасности.

Управление пользователями и правами доступа:

Добавление пользователей: Создайте отдельные пользовательские учетные записи для разных служб или приложений.

Настройка прав доступа: Ограничьте доступ к файлам и каталогам на сервере, установив правильные разрешения.

Настройка мониторинга и журналирования:

Настройка системного мониторинга: Используйте инструменты мониторинга, такие как Prometheus, Grafana или Nagios, для отслеживания ресурсов и обнаружения проблем.

Журналирование: Включите и настройте системное журналирование для отслеживания событий и проблем на сервере.

Резервное копирование и восстановление:

План резервного копирования: Разработайте и регулярно тестируйте план резервного копирования для обеспечения безопасности данных.

Автоматизация резервного копирования: Используйте автоматизированные средства для выполнения регулярных резервных копий.

Настройка сетевых параметров:

Настройка сетевых интерфейсов: Настройте параметры сетевых интерфейсов для обеспечения стабильной связи.

DNS: Укажите правильные настройки DNS для разрешения доменных имен.

Обеспечение масштабируемости и производительности:

Настройка кэширования: Включите и настройте кэширование для улучшения производительности.

Горизонтальное масштабирование: Рассмотрите варианты горизонтального масштабирования, если требуется обработка больших нагрузок.

Документация:

Документация сервера: Ведите документацию, описывающую конфигурацию сервера, используемые версии программного обеспечения и другие важные настройки.

Тестирование:

Тестирование производительности: Проводите тестирование производительности для оценки реакции сервера на различные условия и нагрузки.

Каждый сервер уникален, и настройка зависит от конкретных требований проекта. Важно регулярно аудитировать и обновлять настройки в соответствии с изменениями в требованиях и условиях окружающей среды.

**9.2.1. Выбор операционной системы и серверного ПО**

* Определение подходящей операционной системы (Linux, Ubuntu, CentOS) и серверного ПО (Nginx, Apache).

Выбор операционной системы (ОС) и серверного программного обеспечения (ПО) является критическим этапом при настройке сервера, поскольку от этого зависят производительность, безопасность и совместимость вашего веб-приложения или службы. Вот более подробное описание этого процесса:

Выбор операционной системы (ОС):

Linux:

Преимущества:

Открытый исходный код: Многие дистрибутивы Linux являются свободными и открытыми для общественности.

Стабильность: Linux обычно считается стабильной и надежной операционной системой.

Масштабируемость: Хорошо масштабируется и может быть использован на серверах различных масштабов.

Недостатки:

Сложность для новичков: Интерфейс командной строки может быть сложным для тех, кто только начинает работать с Linux.

Windows Server:

Преимущества:

Интеграция с технологиями Microsoft: Если ваш стек технологий ориентирован на Microsoft (например, .NET, SQL Server), Windows Server может быть логичным выбором.

Простота использования для пользователей Windows: Имеет графический интерфейс пользователя, что делает его более доступным для тех, кто привык к Windows.

Недостатки:

Лицензионные расходы: Операционная система Windows Server может быть дорогой, особенно для больших проектов.

Выбор серверного программного обеспечения (ПО):

Веб-сервер:

Apache:

Преимущества: Один из самых популярных веб-серверов, стабильный, обширно используемый в сообществе.

Недостатки: Некоторые пользователи предпочитают более легкие альтернативы. Nginx

Преимущества: Высокая производительность, эффективная обработка статических файлов и большое количество одновременных подключений.

Недостатки: Могут возникнуть сложности при работе с динамическими контентом.

Microsoft IIS:

Преимущества: Хорошо интегрируется с Windows-технологиями, удобен для тех, кто работает с .NET.

Недостатки: Ограничен в своей поддержке для технологий, не разработанных Microsoft.

База данных:

MySQL

Преимущества: Легковесная, быстрая и обширно используемая система управления базами данных (СУБД).

Недостатки: Могут возникнуть проблемы с производительностью при больших объемах данных.

PostgreSQL:

Преимущества: Мощная, поддерживает сложные запросы, обеспечивает высокую надежность.

Недостатки: Немного более сложная в использовании, чем MySQL.

MongoDB:

Преимущества: База данных NoSQL, хранящая данные в формате JSON, что удобно для работы с документами.

Недостатки: Не подходит для всех типов данных, требует особого подхода к проектированию.

Среда выполнения приложения:

Node.js:

Преимущества: Легкость, возможность использования JavaScript для разработки как на стороне сервера, так и на стороне клиента.

Недостатки: Относительная новизна, требует поддержки асинхронного программирования.

Python (Django, Flask):

Преимущества: Удобный синтаксис, множество библиотек, поддержка фреймворков.

Недостатки: Некоторые фреймворки могут быть не такими производительными, как их аналоги.

Java (Spring):

Преимущества: Кроссплатформенность, высокая производительность, масштабируемость.

Недостатки: Более тяжеловесен по сравнению с некоторыми другими средами выполнения.

Дополнительное программное обеспечение:

Системы управления версиями (Git): Настройте систему управления версиями для контроля кода и совместной разработки.

Языки программирования и библиотеки: Установите необходимые языки программирования и библиотеки для вашего приложения.

Общие рекомендации:

Совместимость: Убедитесь, что выбранная комбинация ОС и серверного ПО совместима с вашими приложениями и технологиями.

Безопасность: Всегда следите за актуальностью безопасности вашей ОС и программного обеспечения, устанавливайте обновления и следите за рекомендациями по безопасности.

Эффективность использования ресурсов: Обратите внимание на требования к ресурсам вашего приложения и выбирайте серверное ПО, которое оптимально использовать эти ресурсы.

Сообщество и поддержка: Рассмотрите, какая поддержка и сообщество доступны для выбранных вами технологий, так как это может существенно облегчить разработку и устранение неполадок.

Правильный выбор операционной системы и серверного программного обеспечения является ключевым фактором для успешного функционирования вашего сервера и приложения.

**9.2.2. Установка зависимостей**

* Установка и настройка необходимых зависимостей, таких как Python, база данных, библиотеки и веб-сервер.

**9.2.3. Настройка сетевых параметров**

* Конфигурация сетевых параметров, открытие портов, установка файрволла для обеспечения безопасности.

Настройка сетевых параметров играет важную роль в обеспечении правильной работы приложений и сервисов в сетевой среде. Это включает в себя ряд шагов и параметров, которые могут варьироваться в зависимости от конкретных требований и особенностей проекта. Вот более подробное описание основных аспектов настройки сетевых параметров:

IP-адреса и подсети:

Статические и динамические адреса: Определите, будете ли использовать статические или динамические IP-адреса. Статические адреса обеспечивают постоянство, тогда как динамические можно настраивать автоматически через DHCP.

Настройка подсетей: Разделите сеть на подсети для логической организации трафика и улучшения безопасности.

Протоколы сетевого уровня:

IPv4 и IPv6: Учитывать возможность использования IPv6 вместе с IPv4 для обеспечения будущей совместимости.

DNS (Domain Name System): Настройте DNS-серверы для преобразования доменных имен в IP-адреса.

Настройка сетевых устройств:

Маршрутизаторы и коммутаторы: Настройте маршрутизаторы для передачи данных между подсетями и коммутаторы для обеспечения связности внутри подсетей.

Файрволлы: Конфигурируйте файрволлы для фильтрации трафика, обеспечивая безопасность сети.

Настройка безопасности:

VPN (Virtual Private Network): Если требуется, настройте VPN для обеспечения безопасного удаленного доступа к сети.

SSL/TLS: Используйте шифрование SSL/TLS для защиты передаваемых данных между клиентами и серверами.

Качество обслуживания (QoS):

Настройка QoS: Определите приоритеты для различных видов трафика (голосовой, видео, данные) для обеспечения качественного обслуживания.

Настройка сетевых служб:

DHCP (Dynamic Host Configuration Protocol): Если используется динамическая настройка IP-адресов, настройте DHCP-сервер.

NTP (Network Time Protocol): Синхронизируйте часы на всех устройствах в сети с использованием NTP для обеспечения правильного времени.

Сетевые протоколы:

TCP/IP, UDP: Подберите подходящие параметры и настройки для протоколов передачи данных, в зависимости от требований приложения.

ICMP (Internet Control Message Protocol): Контролируйте и настраивайте обработку ICMP-сообщений в соответствии с политиками безопасности.

Мониторинг и журналирование:

Системы мониторинга: Настройте системы мониторинга для отслеживания состояния сети и выявления проблем.

Журналирование: Включите журналирование сетевых событий для анализа и выявления возможных угроз безопасности.

Сетевые настройки операционных систем:

Настройки брандмауэра: Убедитесь, что брандмауэры на серверах настроены для обеспечения безопасности и контроля трафика.

TCP/IP стек: Проверьте и настройте параметры TCP/IP стека операционной системы, такие как размеры буферов и тайм-ауты.

Резервное копирование и восстановление:

Настройка резервного копирования: Настройте регулярные резервные копии на случай потери данных или сбоев в сети.

План восстановления: Разработайте план восстановления после сбоя, чтобы быстро восстановить работоспособность сети.

Сетевая документация:

Документация сети: Ведите актуальную документацию сети, включая схемы, конфигурации и руководства по настройке.

Правильная настройка сетевых параметров обеспечивает стабильность и безопасность функционирования приложений в сети. Важно регулярно аудитировать и обновлять настройки в соответствии с изменениями в требованиях и условиях окружающей среды.

9.3. Деплой приложения

**9.3.1. Подготовка приложения к деплою**

* Создание производственной конфигурации приложения, включая установку параметров безопасности и оптимизации.

Подготовка приложения к деплою - это важный этап в разработке программного обеспечения, который включает в себя ряд шагов для обеспечения успешного развёртывания приложения в рабочей среде. Вот более подробное объяснение этого процесса:

Конфигурация окружения:

Настройка окружения разработки: Убедитесь, что все зависимости, необходимые для работы приложения (библиотеки, фреймворки, инструменты), установлены и правильно настроены в вашей среде разработки.

Определение окружения развертывания: Уточните, где будет развёртываться ваше приложение (локально, на сервере в облаке и т. д.) и учтите особенности этой среды.

Обработка зависимостей:

Управление зависимостями: Используйте инструменты управления зависимостями (например, npm, pip, Maven) для того, чтобы убедиться, что все необходимые библиотеки и пакеты правильно установлены.

Версионирование зависимостей: Поддерживайте актуальные версии зависимостей и избегайте использования устаревших или уязвимых версий.

Настройка базы данных:

Создание схемы базы данных: Если ваше приложение использует базу данных, убедитесь, что схема базы данных правильно настроена, и выполните все необходимые миграции данных.

Подключение к базе данных: Проверьте, что параметры подключения к базе данных (URL, учетные данные) настроены верно.

Обеспечение безопасности:

Настройка параметров безопасности: Убедитесь, что все параметры безопасности (например, ключи, секреты) хранятся в безопасном месте, и не передаются в открытом виде.

Проведение аудита безопасности: Проверьте код приложения на наличие уязвимостей безопасности, таких как инъекции SQL, кросс-сайтовые сценарии (XSS), и примените необходимые меры для их устранения.

Оптимизация ресурсов:

Сжатие ресурсов: Минимизируйте размер статических файлов (CSS, JavaScript), используйте сжатие и кэширование для улучшения производительности приложения.

Оптимизация изображений: Уменьшайте размер изображений и используйте современные форматы, чтобы ускорить загрузку страниц.

Настройка окружения развертывания:

Конфигурация переменных окружения: Используйте переменные окружения для хранения конфиденциальных данных и параметров, которые могут изменяться в различных средах (разработка, тестирование, продакшн).

Настройка серверных параметров: Подготовьте серверные настройки, такие как параметры запуска, настройки балансировки нагрузки и прокси.

Тестирование:

Автоматизированные тесты: Запустите все автоматизированные тесты, чтобы убедиться в корректности работы приложения после внесения изменений.

Тестирование совместимости: Проверьте совместимость приложения с различными браузерами и устройствами.

Документация:

Обновление документации: Обновите документацию по развёртыванию, включая шаги установки, требования к системе и необходимые конфигурации.

Подготовка приложения к деплою - это ответственный процесс, требующий внимания к деталям и системному подходу. Этот процесс поможет уменьшить возможные проблемы при развертывании и обеспечит стабильную работу приложения в целевой среде.

**9.3.2. Применение механизмов деплоя**

* Использование инструментов деплоя, таких как Docker, Fabric, или CI/CD пайплайны для автоматизации процесса выкладки.

**9.3.3. Мониторинг деплоя**

* Внедрение механизмов мониторинга для отслеживания процесса деплоя и обнаружения возможных проблем.

9.4. Обеспечение безопасности

**9.4.1. Применение SSL-шифрования**

* Настройка и использование SSL-сертификата для обеспечения безопасности передаваемых данных.

**9.4.2. Защита от атак**

* Внедрение средств защиты, таких как брандмауэры, для предотвращения атак сетевого уровня.
* Применение механизмов защиты от инъекций, CSRF и других уязвимостей.

**9.4.3. Регулярное обновление**

* Регулярное обновление операционной системы, серверного ПО и зависимостей для закрытия уязвимостей.

**Выводы**

Внедрение и деплой веб-приложения - это сложный процесс, требующий внимательной настройки, обеспечения безопасности и выбора подходящего хостинга. Правильное выполнение этих шагов гарантирует стабильность и безопасность работы веб-приложения "Визитка" в продакшн-среде.

**10. Заключение**

10.1. Выводы по результатам исследования

В ходе исследования и разработки веб-приложения "Визитка" с использованием технологий Python и Flask были достигнуты важные результаты. Мы рассмотрели актуальность создания такого приложения в контексте современных требований к визуальной репрезентации личной информации. Разработали структуру проекта, реализовали основные функции, провели тестирование и оптимизацию, что позволило создать стабильное и эффективное веб-приложение.

10.2. Перспективы развития проекта

Проект "Визитка" имеет потенциал для дальнейшего развития. Возможные направления включают в себя:

* Расширение функциональности: Добавление новых возможностей, таких как поддержка мультимедийных элементов, интеграция с социальными сетями и дополнительные опции настройки визиток.
* Улучшение интерфейса: Работа над дизайном и улучшение пользовательского интерфейса для более удобного взаимодействия.
* Масштабирование и оптимизация: Увеличение масштабов приложения для обслуживания большего числа пользователей и постоянная оптимизация кода для повышения производительности.
* Интеграция с облачными сервисами: Исследование возможности интеграции с облачными хранилищами и сервисами для улучшения доступности данных.

10.3. Резюме

В заключение можно сказать, что разработка веб-приложения "Визитка" стала важным этапом в исследовании возможностей языка программирования Python и микрофреймворка Flask. Создание структуры проекта, реализация функциональности, тестирование, оптимизация и успешный деплой позволили создать полноценное приложение, соответствующее поставленным целям.

Этот проект не только предоставляет удобное средство для создания визиток, но и служит отличным примером применения современных технологий в веб-разработке. Надеемся, что "Визитка" найдет своего пользователя и будет успешно использоваться, а дальнейшее развитие проекта принесет новые интересные возможности и улучшения.

**Список использованных источников**

1. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
2. Pallets Projects. (2022). Flask Documentation. https://flask.palletsprojects.com/
3. SQLAlchemy. (2022). SQLAlchemy Documentation. https://docs.sqlalchemy.org/
4. Python Software Foundation. (2022). Python 3 Documentation. https://docs.python.org/3/
5. Mitchell, R. (2021). Test-Driven Development with Python. O'Reilly Media.
6. Docker. (2022). Docker Documentation. https://docs.docker.com/
7. Nginx. (2022). Nginx Documentation. https://nginx.org/en/docs/
8. Ubuntu. (2022). Ubuntu Documentation. https://help.ubuntu.com/
9. DigitalOcean. (2022). DigitalOcean Documentation. https://www.digitalocean.com/docs/
10. AWS Documentation. (2022). Amazon Web Services Documentation. https://docs.aws.amazon.com/

**Приложения**

Приложение А: Код веб-приложения "Визитка"

[Ссылка на репозиторий на GitHub]

Приложение Б: Скриншоты приложения

[Ссылки на скриншоты интерфейса приложения]