Дипломная работа на тему:

Анализ и сравнение написания web-приложений с использованием разных фреймворков: Разработать простые веб-приложения с использованием Django, Flask и FastAPI, провести их сравнение.

Выполнила работу:

Жохова Яна Шамхаловна

2025 г.

**Оглавление**

1. Введение
2. Теоретические аспекты разработки веб-приложений
3. Описание используемых технологий

3.1. Django

3.2. Flask

3.3. FastAPI

1. Практическое применение

4.1. Разработка приложения на Django

4.2. Разработка приложения на Flask

4.3. Разработка приложения на FastAPI

1. Анализ и сравнение фреймворков
2. Выводы
3. **Введение**

Современные технологии позволяют разработчикам выбирать из множества фреймворков для создания веб-приложений. Каждый из них имеет свои преимущества и недостатки, которые зависят от специфики проекта, опыта команды и требований заказчика. В данной работе проводится сравнительный анализ трех популярных фреймворков для разработки веб-приложений на языке Python: Django, Flask и FastAPI. На примере простых веб-приложений рассматриваются особенности их применения, сравниваются подходы к решению одних и тех же задач, а также оценивается их пригодность для различных уровней разработчиков.

Цель исследования заключается в определении наиболее подходящего фреймворка для разработки веб-приложений различного масштаба и сложности, исходя из анализа их характеристик, производительности и удобства использования.

1. **Теоретические аспекты разработки веб-приложений**

Разработка веб-приложений включает в себя несколько ключевых этапов: проектирование архитектуры, реализация бизнес-логики, взаимодействие с базой данных, обработка запросов и ответов, а также обеспечение безопасности и масштабируемости. Важными аспектами являются выбор языка программирования и соответствующего фреймворка, который поможет упростить процесс разработки и сократить время на реализацию основных функций.

Python является одним из самых популярных языков программирования для веб-разработки благодаря своей простоте, читаемости кода и большому количеству библиотек и фреймворков. Рассмотрим три популярных фреймворка на Python: Django, Flask и FastAPI.

1. **Описание используемых технологий**
   1. **Django**

Django — это полнофункциональный фреймворк, предназначенный для быстрой разработки сложных веб-приложений. Он предоставляет встроенную ORM (Object-Relational Mapping), систему аутентификации, админ-панель и множество других полезных инструментов. Django следует принципу DRY (Don't Repeat Yourself), что помогает избежать дублирования кода и повышает его качество.

Основные характеристики Django:

* Полнофункциональная среда разработки.
* Встроенная поддержка ORM.
* Система аутентификации и авторизации.
* Административная панель.
* Широкая экосистема плагинов и дополнений.

**3.2. Flask**

Flask — это микро-фреймворк, который предоставляет минимальный набор инструментов для создания веб-приложений. Он идеально подходит для небольших проектов, где важна гибкость и возможность тонкой настройки. Flask не навязывает строгую архитектуру, что дает разработчику свободу выбора в реализации различных аспектов приложения.

Основные характеристики Flask:

* Минимализм и простота.
* Возможность расширения функционала с помощью сторонних библиотек.
* Хорошая интеграция с различными системами баз данных.
* Удобство для быстрого прототипирования.

**3.3. FastAPI**

FastAPI — это современный асинхронный фреймворк, основанный на стандартах Python типа аннотаций и Pydantic. Он обеспечивает высокую производительность и простоту разработки API. FastAPI использует последние достижения в области асинхронного программирования и предоставляет удобные инструменты для валидации данных и автоматической генерации документации.

Основные характеристики FastAPI:

* Высокая производительность за счет асинхронного подхода.
* Использование современных стандартов Python.
* Автоматическая генерация документации с помощью OpenAPI.
* Простота интеграции с другими сервисами и библиотеками.

1. **Практическое применение**
   1. **Разработка приложения на Django**

Для демонстрации возможностей Django было создано простое веб-приложение, включающее форму для ввода текста и базу данных для его сохранения. Рассмотрим ключевые этапы разработки этого приложения.

**Установка и настройка**

Установка Django осуществляется с помощью pip:

pip install django

После установки необходимо создать проект и приложение:

django-admin startproject mysite

cd mysite

python manage.py startapp main\_app

Далее нужно добавить приложение в список установленных приложений в mysite/settings.py:

INSTALLED\_APPS = [

# ...

'main\_app',

]

**Модели и миграции**

Модель для хранения текста создается в main\_app/models.py:

from django.db import models

class Text(models.Model):

text = models.TextField()

def \_\_str\_\_(self):

return self.text[:50]

Затем выполняются миграции базы данных:

python manage.py makemigrations

python manage.py migrate

**Представления и формы**

Представления создаются в main\_app/views.py:

from django.shortcuts import render, redirect

from .models import Text

from .forms import TextForm

def index(request):

texts = Text.objects.all().order\_by('-id')

form = TextForm()

context = {'texts': texts, 'form': form}

return render(request, 'main\_app/index.html', context)

def save\_text(request):

if request.method == 'POST':

form = TextForm(request.POST)

if form.is\_valid():

form.save()

return redirect('index')

return redirect('index')

Форма для ввода текста определяется в main\_app/forms.py:

from django import forms

from .models import Text

class TextForm(forms.ModelForm):

class Meta:

model = Text

fields = ['text']

**Шаблоны и маршруты**

Шаблон главной страницы создается в main\_app/templates/main\_app/index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Django App</title>

</head>

<body>

<h1>Главная страница</h1>

<form action="{% url 'save\_text' %}" method="post">

{% csrf\_token %}

{{ form }}

<button type="submit">Сохранить</button>

</form>

<hr>

<h2>Список текстов</h2>

<ul>

{% for text in texts %}

<li>{{ text }}</li>

{% endfor %}

</ul>

</body>

</html>

Маршруты определяются в mysite/urls.py и main\_app/urls.py:

# mysite/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('main\_app.urls')),

]

# main\_app/urls.py

from django.urls import path

from .views import index, save\_text

urlpatterns = [

path('', index, name='index'),

path('save-text/', save\_text, name='save\_text'),

]

После завершения разработки приложение можно запустить с помощью команды:

python manage.py runserver

**Так выглядит наше приложение во время работы:**

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

* 1. **Разработка приложения на Flask**

Для демонстрации возможностей Flask было создано аналогичное веб-приложение. Рассмотрим ключевые этапы его разработки.

**Установка и настройка**

Установка Flask осуществляется с помощью pip:

pip install flask

Создается файл app.py:

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

from flask\_sqlalchemy import SQLAlchemy

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

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///db.sqlite'

app.config['SECRET\_KEY'] = 'your\_secret\_key'

db = SQLAlchemy(app)

class Text(db.Model):

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

text = db.Column(db.String(255))

with app.app\_context():

db.create\_all()

@app.route('/')

def index():

texts = Text.query.order\_by(Text.id.desc()).all()

return render\_template('index.html', texts=texts)

@app.route('/add-text', methods=['POST'])

def add\_text():

new\_text = Text(text=request.form['text'])

db.session.add(new\_text)

db.session.commit()

return redirect(url\_for('index'))

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

app.run(debug=True)

**Шаблоны**

Шаблон главной страницы создается в templates/index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Flask App</title>

</head>

<body>

<h1>Главная страница</h1>

<form action="{{ url\_for('add\_text') }}" method="post">

<textarea name="text"></textarea><br>

<button type="submit">Сохранить</button>

</form>

<hr>

<h2>Список текстов</h2>

<ul>

{% for text in texts %}

<li>{{ text.text }}</li>

{% endfor %}

</ul>

</body>

</html>

Приложение запускается с помощью команды:

python app.py

**Так выглядит наше приложение во время работы:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**4.3. Разработка приложения на FastAPI**

Для демонстрации возможностей FastAPI было создано аналогичное веб-приложение. Рассмотрим ключевые этапы его разработки.

**Установка и настройка**

Установка FastAPI и необходимых библиотек осуществляется с помощью pip:

pip install fastapi uvicorn sqlalchemy aioredis

Создается файл main.py:

from fastapi import FastAPI, Request, Form

from fastapi.responses import HTMLResponse

from fastapi.templating import Jinja2Templates

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

Base = declarative\_base()

class Text(Base):

\_\_tablename\_\_ = 'texts'

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

text = Column(String)

engine = create\_engine('sqlite:///db.sqlite')

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base.metadata.create\_all(bind=engine)

app = FastAPI()

templates = Jinja2Templates(directory="templates")

@app.get("/", response\_class=HTMLResponse)

async def read\_root(request: Request):

db = SessionLocal()

texts = db.query(Text).order\_by(Text.id.desc()).all()

db.close()

return templates.TemplateResponse("index.html", {"request": request, "texts": texts})

@app.post("/add-text/", response\_class=HTMLResponse)

async def add\_text(request: Request, text: str = Form(...)):

db = SessionLocal()

new\_text = Text(text=text)

db.add(new\_text)

db.commit()

db.close()

return templates.TemplateResponse("index.html", {"request": request}, status\_code=200)

**Шаблоны**

Шаблон главной страницы создается в templates/index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>FastAPI App</title>

</head>

<body>

<h1>Главная страница</h1>

<form action="/add-text/" method="post">

<textarea name="text"></textarea><br>

<button type="submit">Сохранить</button>

</form>

<hr>

<h2>Список текстов</h2>

<ul>

{% for text in texts %}

<li>{{ text.text }}</li>

{% endfor %}

</ul>

</body>

</html>

Приложение запускается с помощью команды:

uvicorn main:app –reload

**Так выглядит наше приложиние во время работы:**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**5. Анализ и сравнение фреймворков**

На основании разработанных приложений можно сделать следующий анализ и сравнение фреймворков Django, Flask и FastAPI.

**Простота использования**

* **Django**: Предоставляет готовую инфраструктуру и множество встроенных инструментов, что облегчает разработку крупных проектов. Однако для новичков может показаться сложной из-за большого количества концепций и структур.
* **Flask**: Минимальный набор инструментов, что делает его идеальным для быстрого прототипирования и небольших проектов. Подойдет для новичков, так как требует минимальных знаний для начала работы.
* **FastAPI**: Современный и производительный фреймворк, однако требует понимания асинхронного программирования и современных стандартов Python, что может затруднить его использование для новичков.

**Производительность**

* **Django**: Оптимизирован для работы с большими проектами, но может уступать в производительности другим фреймворкам из-за своей архитектуры.
* **Flask**: Обеспечивает хорошую производительность благодаря своей легкости и минимализму.
* **FastAPI**: Наиболее производительный из рассматриваемых фреймворков благодаря использованию асинхронного программирования и современных стандартов Python.

**Гибкость**

* **Django**: Менее гибок по сравнению с Flask, так как предлагает готовые решения, которые могут ограничивать возможности кастомизации.
* **Flask**: Очень гибкий, позволяя разработчику контролировать каждый аспект приложения.
* **FastAPI**: Также обладает высокой степенью гибкости, особенно в контексте разработки API.

**Поддержка сообщества**

* **Django**: Огромное сообщество разработчиков и множество ресурсов для обучения и поддержки.
* **Flask**: Активное сообщество, хотя оно меньше, чем у Django.
* **FastAPI**: Быстро растущее сообщество, привлекающее внимание своими инновациями и производительностью.

**6. Выводы**

Каждый из рассмотренных фреймворков имеет свои сильные стороны и подходит для определенных типов задач. Выбор зависит от конкретных требований проекта и предпочтений разработчика.

* **Новичкам** лучше всего начинать с Flask, так как он прост в освоении и позволяет быстро создавать рабочие прототипы.
* **Среднему уровню** подойдет Django, который предложит мощный набор инструментов для реализации сложных проектов.
* **Опытным разработчикам** будет интересен FastAPI, идеально подходящий для создания высокопроизводительных API и работы с современными технологиями.

Эта дипломная работа представляет собой всесторонний анализ и сравнение трех популярных фреймворков для разработки веб-приложений на языке Python: Django, Flask и FastAPI.

Проведено практическое исследование путем создания простых веб-приложений на каждом из фреймворков, а также выполнен детальный анализ их характеристик, производительности и удобства использования.

Результаты исследования помогут начинающим и опытным разработчикам выбрать наиболее подходящий инструмент для своих проектов.