УДК 004.4
DOI 10.58649/1694-8033-2025-1(121)-318-322

МУРАТАЛИЕВА В.Т.[1], ДЮШЕБЕКОВ А.Т.
Ж. Баласагын атындагы КУУ
МУРАТАЛИЕВА В.Т., ДЮШЕБЕКОВ А.Т.
КНУ имени Ж. Баласагына
MURATALIEVA V.T., DUSHEBEKOV A.T.
KNU named after J. Balasagyn

ORCID: 0000-0002-7246-7529[1]

PYTHON ТИЛИНДЕ ВЕБ-САЙТ ҮЧҮН БЭКЭНДДИ ИШТЕП ЧЫГУУ ЖАНА ИШКЕ АШЫРУУ

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ БЭКЭНДА ДЛЯ ВЕБ-САЙТА НА PYTHON

DESIGN AND IMPLEMENTATION OF A BACKEND FOR A WEBSITE IN PYTHON

Кыскача мүнөздөмө: Макала Python программалоо тилинде веб-сайт учун бэкенд иштеп чыгууга жана ишке ашырууга арналган. Өнүктүрүүнүн негизги этаптары, анын ичинде технологияларды тандоо, архитектуралык дизайн, RESTful API түзүү жана маалымат базасы менен интеграциялоо каралат. Колдонуучуларды аутентификациялоо жана уруксат берүү маселелерине, ошондой эле веб-тиркемелердин коопсуздук маселелерине өзгөчө көңүл бурулат. Макалада колдонгон Python кодунун мисалдары, ошондой эле оптималдаштыруу жана маалыматтарды коргоо боюнча сунуштар келтирилген.

Аннотация: Статья посвящена проектированию и реализации бэкенда для веб-сайта с использованием Python. Рассматриваются основные этапы разработки, включая выбор технологий, проектирование архитектуры, создание RESTful API и интеграцию с базой данных. Особое внимание уделено вопросам аутентификации и авторизации пользователей с использованием JWT, а также вопросам безопасности веб-приложений. В статье представлены примеры кода на Python с использованием Flask, а также рекомендации по оптимизации и защите данных.

Abstract: The article is devoted to the design and implementation of a backend for a website using Python. The main stages of development are considered, including the choice of technologies, architecture design, creation of a RESTful API and integration with a database. Special attention is paid to the issues of authentication and authorization of users using JWT, as well as security issues of web applications. The article provides examples of Python code using Flask, as well as recommendations for optimization and data protection.

Негизги сөздөр: веб-сайт; бэкенд; проектирлоо; маалымат базалары; маалымат коопсуздугу.
Ключевые слова: веб-сайт; бэкенд; проектирование; базы данных; безопасность данных.
Keywords: website; backend; design; databases; data security.

Разработка бэкенда для веб-сайта — ключевая часть создания современного веб-приложения. Бэкенд обеспечивает функциональность, отвечает за хранение и обработку данных, а также за взаимодействие с клиентом через API. Python, благодаря своей простоте и мощным библиотекам, является отличным выбором для реализации бэкенда. В данной статье будет рассмотрено, как спроектировать и реализовать бэкенд для веб-сайта на Python, начиная с проектирования архитектуры и заканчивая реализацией основных функций.

Бэкенд веб-сайта отвечает за такие важные задачи, как хранение и обработка данных, взаимодействие с базами данных, управление пользователями и их сессиями, а также обработку запросов от клиента. Python предоставляет множество инструментов для реализации этих функций, включая фреймворки, такие как Flask, Django и FastAPI. В этой статье будем использовать Flask — легкий и гибкий фреймворк для создания RESTful API, который подходит для большинства современных веб-приложений.[1]

Архитектура бэкенда

Проектирование бэкенда начинается с выбора архитектуры системы. В большинстве случаев для веб-сайтов используется архитектура **клиент-сервер**, где клиент — это веб-браузер, а сервер обрабатывает запросы и отвечает на них.

Для простоты рассмотрим базовую архитектуру:

- **API** — интерфейс между клиентом и сервером, реализованный через RESTful сервисы. API управляет обменом данных, передаваемых между клиентом и сервером.
- **База данных** — хранилище данных. Бэкенд взаимодействует с базой данных для получения и хранения информации о пользователях, постах, комментариях и других сущностях.
- **Система аутентификации и авторизации** — система, которая управляет доступом пользователей к различным частям сайта.
- **Механизм обработки ошибок** — для правильной обработки запросов и возврата ошибок в случае необходимости.

Основные компоненты

1. **Сервер**: Сервер отвечает за принятие запросов от клиентов и их обработку. Flask или Django позволяют организовать обработку HTTP-запросов, создание маршрутов и маршрутизацию данных.
2. **База данных**: База данных — это хранилище, в котором сохраняются все данные, включая информацию о пользователях, постах и комментариях. Для небольших проектов можно использовать SQLite, для более крупных — PostgreSQL или MySQL.
3. **API**: API отвечает за передачу данных между сервером и клиентом. В основном используется формат JSON для обмена данными.
4. **Аутентификация и авторизация**: Для обеспечения безопасности и разделения прав доступа между пользователями необходимо реализовать аутентификацию (проверку подлинности пользователя) и авторизацию (проверку прав доступа).
5. **Механизм обработки ошибок**: Важно правильно обрабатывать ошибки, чтобы предоставить пользователям понятные сообщения и логировать важные события для разработчиков.

Выбор технологий

Для создания бэкенда на Python можно использовать различные фреймворки и библиотеки. Рассмотрим несколько популярных вариантов:

Flask

Flask — это минималистичный веб-фреймворк, который позволяет быстро начать разработку. Он предоставляет необходимый функционал для создания REST API, работы с базами данных через SQLAlchemy и управления сессиями. [2]

Преимущества Flask:

- Легкость и гибкость.
- Хорошо подходит для небольших и средних проектов.
- Широкая документация и большое сообщество.

Django

Django — более «тяжелый» фреймворк, который подходит для крупных проектов, требующих комплексной структуры. Он предоставляет множество встроенных

решений, таких как системы аутентификации, административная панель, ORM и многое другое. [3]

Преимущества Django:

- Быстрая разработка с минимальными усилиями.
- Большая поддержка и множество готовых решений.
- Встроенная система аутентификации и авторизации.

FastAPI

FastAPI — новый фреймворк, ориентированный на создание API с высокой производительностью. Он предоставляет возможности для автоматической генерации документации и быстрого написания высокоскоростных API.

Преимущества FastAPI:

- Высокая производительность.
- Поддержка асинхронных операций.
- Автоматическая документация через OpenAPI.

В данной статье будем использовать **Flask**, так как он является хорошим выбором для небольших и средних проектов, где важна скорость разработки и простота использования.

Процесс проектирования

Проектирование базы данных

Для бэкенда веб-сайта нам нужно продумать структуру базы данных. База данных будет хранить информацию о пользователях, постах, комментариях и других сущностях. Для примера возьмем базу данных для сайта с блогом, где пользователи могут регистрироваться, входить в систему и оставлять комментарии под постами. [4]

Пример структуры базы данных:

- **Users**: таблица с информацией о пользователях (ID, имя, email, хэш пароля, дата регистрации).
- **Posts**: таблица с постами (ID, заголовок, содержание, дата публикации, ID пользователя).
- **Comments**: таблица с комментариями (ID, текст комментария, дата, ID поста, ID пользователя).

Пример структуры таблицы **Users**:

Поле	Тип данных	Описание
id	INT	Уникальный идентификатор
username	VARCHAR	Имя пользователя
email	VARCHAR	Электронная почта
password	VARCHAR	Хэш пароля
created_at	DATETIME	Дата регистрации

API

API будет основным интерфейсом для взаимодействия с клиентом. Веб-сайт будет взаимодействовать с клиентской частью через RESTful API. Примерный список эндпоинтов:

- **POST /api/register** — регистрация нового пользователя.
- **POST /api/login** — аутентификация пользователя.
- **GET /api/posts** — получение списка постов.
- **POST /api/posts** — создание нового поста.
- **GET /api/posts/<id>** — получение информации о конкретном посте.
- **POST /api/comments** — добавление комментария к посту.

Аутентификация и авторизация

Для аутентификации и авторизации пользователей можно использовать систему с JWT (JSON Web Tokens). JWT позволяет безопасно передавать данные между клиентом и сервером. Каждый раз, когда пользователь выполняет запрос, он передает свой токен, и сервер проверяет его подлинность. [5]

Реализация API с использованием Flask

Для реализации API мы используем Flask, который позволяет быстро создать маршруты и обработать запросы. Сначала установим Flask и другие необходимые библиотеки:

bash

```
pip install Flask Flask-JWT-Extended Flask-SQLAlchemy
```

Пример кода для создания простого API с Flask: Python

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_jwt_extended import JWTManager, create_access_token, jwt_required
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
```

```
app.config['JWT_SECRET_KEY'] = 'supersecretkey'
db = SQLAlchemy(app)
jwt = JWTManager(app)
# Модель пользователя
class User(db.Model):
 id = db.Column(db.Integer, primary_key=True)
 username = db.Column(db.String(120), unique=True, nullable=False)
 email = db.Column(db.String(120), unique=True, nullable=False)
 password = db.Column(db.String(60), nullable=False)
# Регистрация нового пользователя
@app.route('/api/register', methods=['POST'])
def register():
 data = request.get_json()
 username = data['username']
 email = data['email']
 password = data['password']
 new_user = User(username=username, email=email, password=password)
 db.session.add(new_user)
 db.session.commit()
 return jsonify(message="User created"), 201
# Вход в систему
@app.route('/api/login', methods=['POST'])
def login():
 data = request.get_json()
 username = data['username']
 password = data['password']
 user = User.query.filter_by(username=username).first()
 if user and user.password == password:
  token = create_access_token(identity=username)
  return jsonify(access_token=token), 200
 return jsonify(message="Invalid credentials"), 401
# Пример защищенного маршрута
@app.route('/api/protected', methods=['GET'])
@jwt_required()
def protected():
 return jsonify(message="This is a protected route"), 200
if __name__ == '__main__': app.run(debug=True)
```

Безопасность и защита данных

Безопасность данных — важная часть разработки бэкенда. [6] Для защиты данных пользователей и предотвращения несанкционированного доступа следует использовать:

- **Шифрование паролей**: Все пароли должны храниться в зашифрованном виде с использованием алгоритмов, таких как bcrypt.
- **JWT и безопасность токенов**: Токены должны быть защищены, а срок их действия ограничен.
- **SSL/TLS**: Для защиты данных в процессе передачи между клиентом и сервером рекомендуется использовать протокол HTTPS.
- **Защита от атак**: Использование техник защиты от SQL-инъекций, CSRF и других атак.

Заключение

Проектирование и реализация бэкенда для веб-сайта на Python с использованием Flask — это процесс, который требует внимательного подхода к архитектуре, безопасности и организации взаимодействия между компонентами. Flask предоставляет удобный инструмент для быстрого создания API, а также гибкость для добавления различных функций. Системы аутентификации, авторизации и безопасности являются неотъемлемой частью разработки, обеспечивающей защиту данных пользователей и корректную работу веб-сайта.

.

Список использованной литературы

1. Гринберг М. Flask Web Development. – 2-е изд. – О'Рейли Медиа, 2018, 500 с. ISBN 978-1-4919-5241-1.
2. Форсье Д. Python Web Development with Django / Д. Форсье, П. Биссекс, У. Чун. – Питер, 2008, 624 с. ISBN 978-0-321-53224-5.
3. Дуайер Г. Flask By Example. – Packt Publishing, 2018, 350 с. ISBN 978-1-78712-239-6.
4. Винсент У. Django for Beginners. – Self-published, 2020, 180 с. ISBN 978-1-68407-806-6.
5. Горелик М., Озсвалд И. High Performance Python. – O'Reilly Media, 2014, 450 с. ISBN 978-1-4493-7152-6.
6. Документация Flask. – Режим доступа: https://flask.palletsprojects.com/, свободный (Дата обращения: 10.12.2024).
7. Документация Django. – Режим доступа: https://docs.djangoproject.com/, свободный. (Дата обращения: 10.12.2024).
8. Real Python. Building RESTful Web APIs with Flask. – Режим доступа: https://realpython.com/flask-connexion-rest-api/, свободный. (Дата обращения: 10.12.2024).
9. Django REST Framework Documentation. – Режим доступа: https://www.django-rest-framework.org/, свободный. (Дата обращения: 10.12.2024).
10. DigitalOcean. Python HTTP Servers for Web Applications. – Режим доступа: https://www.digitalocean.com/community/tutorials, свободный. (Дата обращения: 10.12.2024).
11. Мигель Гринберг. The Flask Mega-Tutorial. – Режим доступа: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world, свободный. (Дата обращения: 10.12.2024).
12. OpenAPI Specification. – Режим доступа: https://swagger.io/specification/, свободный. (Дата обращения: 10.12.2024).
13. OAuth 2.0 Authorization Framework. – Режим доступа: https://tools.ietf.org/html/rfc6749, свободный. (Дата обращения: 10.12.2024).
14. JSON Web Tokens (JWT). – Режим доступа: https://jwt.io/introduction/, свободный. (Дата обращения: 10.12.2024).

Рецензент: к.п.н., доцент Нуржанова С.А.