# ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ FLASK И PYTHON, МИКРОСЕРВИСНАЯ И ОБЛАЧНАЯ АРХИТЕКТУРА

**А. Д. Гончаренко**

*fpm.gonchareAD@bsu.by;*
*Научный руководитель – М. И. Давидовская, старший преподаватель*

Статья посвящена разработке структуры и архитектуры веб-приложения с механизмом поиска статей, спроектированного на основе микросервисной архитектуры. Содержимое сайта пополняется по расписанию из материалов, опубликованных на новостных сайтах в интернет после их синтаксического разбора. Для решения проблемы поиска новостной статьи внедрен механизм поиска Elasticsearch. В качестве механизма синтаксического разбора текстов применяется библиотека beautifulsoup. Область применения — моделирование автоматизированного новостного источника с помощью средств разработки веб-приложений.

***Ключевые слова:*** Python; Flask; Django; MySQL; Terraform; Kubernetes; ORM; Elasticsearch.

Новостные ресурсы могут быть классифицированы по географическому признаку, по направлению деятельности и способу наполнения материалами. Ресурс, аккумулирующий статьи профильных новостных ресурсов, предоставляет интерфейс для доступа к новостям из разных источников за счет автоматизированного синтаксического разбора текста, получения обновления новостей популярных сайтов по расписанию и их публикации в базу.

Проектирование веб-приложения состоит из нескольких этапов. На первом этапе сформулирована постановка задачи, разработаны функциональные и нефункциональные требования и выполнен анализ решений для разработки приложения.

Согласно постановке задачи требуется разработать новостной сайт с механизмом поиска статей, автоматическим добавлением статей, который агрегирует материалы других ресурсов. Выделены следующие функциональные требования [1]:

- Сохранение входа в систему при обновлении страницы;
- Корректное отображение текста после синтаксического разбора;
- Просмотр информации о создателях и правах;
- Возможность полнотекстового поиска;
- Отсутствие регистрации через сайт, только получение учетных данных от администратора;
- Синтаксический разбор других сайтов;

- Нефункциональные требования описывают характер поведения системы. И к ним относятся [1]:
- Быстродействие сайта.
- Удобство использования интерфейса просмотра статей ;
- Интуитивно понятный интерфейс.

В качестве механизма, облегчающего процесс разработки веб-приложений, применяются готовые решения в виде библиотек (веб-фреймворков). Для реализации основного проекта сайта был выбран фреймворк Flask. Flask поддерживает гибкую настройку и не требователен к ресурсам.

Для хранения данных используем систему управления базами данных (СУБД) MySQL. В качестве архитектуры проекта выбрана микросервисная архитектура для развертывания приложения на нескольких вычислительных узлах в публичном облаке. В контексте микросервисной архитектуры выбранная СУБД может быть представлена в виде отдельного сервиса, упакованного в контейнер.

Важным фактором при разработке архитектуры является обеспечение ее безопасности. Чтобы избежать таких уязвимостей, как SQL-инъекции, XSS атаки, SSTI и другие, используется механизм шаблонизации Jinja2. Этот механизм поддерживает удобный язык для создания HTML-страниц на основе шаблона и для отображения динамических страниц с выбором источника данных.

Одной из наиболее распространенных уязвимостей, связанных с базами данных, является SQL-инъекция. Например, если используем обычные SQL-запросы для доступа к базе данных, то пользователь может ввести SQL-запрос вместо своих данных. В результате могут быть получены данные всех пользователей. Для исключения таких проблем используется механизм объектно-реляционного отображения (Object Relational Mapping — ORM), например SQLAlchemy. Для применения SQLAlchemy необходимо подключить библиотеку flask_sqlalchemy и настроить конфигурацию (см. рис. 1):

```
app = Flask(__name__, template_folder="templates", static_folder="static")
secret_key = secrets.token_urlsafe(32)
app.config.update(
    DEBUG=True,
    SECRET_KEY=secret_key,
    ELASTICSEARCH_URL=os.environ.get('ELASTICSEARCH_URL')
)
app.elasticsearch = Elasticsearch([app.config['ELASTICSEARCH_URL']]) \
    if app.config['ELASTICSEARCH_URL'] else None

cred = pars_cred.init_cred()

host = os.environ.get('MYSQL_HOST')
app.config['SQLALCHEMY_DATABASE_URI'] = f'mysql+pymysql://{cred[0]}:{cred[1]}@{host}:3306/blog'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

*Рис. 1.* Конфигурация приложения и подключение к базе данных

Для работы с базой данных используем классы Article и User (Рис. 2). В них описаны поля и их свойства. Для хранения изображений используется поле с типом данных Integer, но в базе данных оно хранится с типом Blob. Учетные данные пользователей хранятся в зашифрованном виде.

```
class Article(db.Model):
    __searchable__ = ['intro']
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    intro = db.Column(db.String(300), nullable=False)
    text = db.Column(db.Text, nullable=False)
    date = db.Column(db.DateTime, default=datetime.utcnow)
    ImageID = db.Column(db.Integer, nullable=True)

    def __repr__(self):
        return '<Article %r>' % self.id


class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    login = db.Column(db.String(128), nullable=False, unique=True)
    password = db.Column(db.String(255), nullable=False)
```

*Рис. 2.* Моделирование таблиц Article и User

Для полнотекстового поиска записей по сайту был выбран ElasticSearch. Elasticsearch индексирует быстро меняющиеся данные менее, чем за 1 секунду [2]. Так как в проекте база данных постоянно изменяется, то выбор такого механизма ускорит обработку поисковых запросов.

Одним из преимуществ микросервисной архитектуры является принцип, который говорит о том, что при остановки работы одного сервиса остальные продолжат свою работу корректно. Следовательно, для синтаксического разбора статей и последующей обработки было разработано отдельное приложение на Python, схема работы которого представлена ниже:

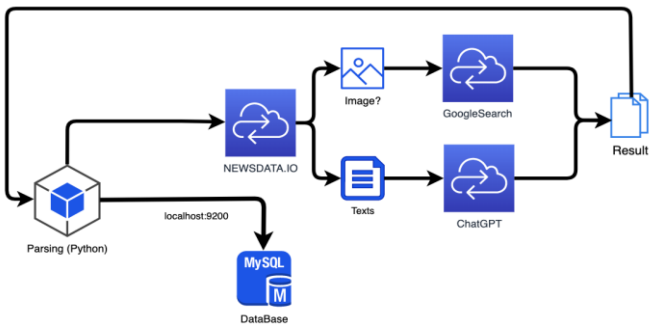


*Рис. 3.* Локальное представление работы сервиса

Чтобы желаемая архитектура развертывания в облаке была правильной и автоматизированной будем использовать Terraform для описания инфраструктуры, Kubernetes — для описания кластеров.

Перед тем, как развертывать инфраструктуру, необходимо создать конфигурационные файлы, описывающие Docker контейнеры. Такие файлы содержат правила создания образа будущего контейнера. Для хранения образов и дальнейшего взаимодействия с ним используем DockerHub.

Далее для описание инфраструктуры облачных вычислений применим методологию IaaC. Для ее реализации используются файлы конфигурации Terraform, в которых описывается сервисы: VPC (Virtual private cloud), EKS (Elastic kubernetes service), Compute Cloud и др.

После того, как была спроектирована облачная инфраструктура и созданы образы контейнеров, необходимо настроить сам оркестратор, который будет управлять работой контейнеров, взаимодействием с сервисами из вне и внутри кластера Kubernetes. Чтобы обеспечить защищенный доступ к сервисам, используем Hashicorp vault, который создает отдельные узлы с секретными данными. Авторизованные сервисы используют эти данные для взаимодействия с базой данных и внешними сервисами.

Согласно поставленной задаче были разработаны функциональные и нефункциональные требования. Спроектирована архитектура базы данных. Разработаны веб-приложение и схема его развертывания. Проект представляет собой приложение на основе микросервисной архитектуры и оптимизировано для развертывания в облачной инфраструктуре. Приложение адаптировано для мобильных устройств.

Данные технологии рекомендуется использовать для создания отказоустойчивых и масштабируемых сервисов за счет преимуществ развертывания в облаке. Облачная инфраструктура обладают большим количеством доступных ресурсов, выделением их по требованию и гарантией отказоустойчивости.

## Библиографические ссылки


1. *Гончаренко А. Д., Давидовская М. И.* ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ FLASK И PYTHON // Сетевое издание “Научные исследования XXI века”. 2023, Т. 21. № 1. С. 40–44. УДК 004.415.2.
2. Powering Uber Marketplace’s Real-Time Data Needs with Elasticsearch / ON-DEMAND WEBINAR [Electronic resource]. URL: https://www.elastic.co/elasticon/conf/2017/sf/powering-uber-marketplace-real-time-data-needs-with-elasticsearch (date of access: 08.10.2022).
3. *Rehan H.* Web API Development with Python. A begginer’s guide using Flask and FastAPI (First Edition). CloudBytes, 2021.