# ТЗ: Сервис проверки ZIP-архивов

## 1. Цель разработки

Создать микросервис для проверки содержимого ZIP-архивов с использованием запросов к сторонним системам и получения набора характеристик файла.

## 2. Функциональные требования

### 2.1 Основные функции:

- Прием ZIP-архива от пользователя
- Проверка целостности архива
- Выполнение запросов к сторонним системам для получения характеристик (условно считаем что у нас таких систем 3)
- Агрегация и возврат результатов проверки

### 2.2 API endpoints:

- POST /upload - загрузка ZIP-архива
- GET /report/{id} - получение результатов проверки

*Примеры контрактов в приложении 1

## 3. Технические требования

### 3.1 Технологический стек:

- Python 3.12+
- Sanic или FastAPI
- Docker (+ docker-compose)
- ORM SQLAlchemy
- Pytest (tests outside application code)
- Alembic
- PostgreSQL

### 3.2 Интеграции:

- Аутентификация через внешний сервис учетных записей (Keycloak)
- Взаимодействие со сторонними API (Достаточно зафиксировать контракт, но можно и dummy сервис)
- Хранение данных во внешнем хранилище (MinIO)

## 4. Нефункциональные требования

### 4.1 Производительность:

- Обработка архива до 100 МБ

- Время обработки: не более 60 секунд

### 4.2 Надежность:

- Обработка ошибок при загрузке
- Валидация входных данных
- Логирование операций

## 5. Требования к разработке

### 5.1 Документация:

- README.md с инструкциями по запуску
- Автодокументация API (OpenAPI/Swagger)

### 5.2 Тестирование:

- Unit-тесты
- Интеграционные тесты
- Тесты производительности (желательно)

### 5.3 Деплой:

- Dockerfile
- docker-compose.yml
- Инструкции по развертыванию

## 6. Критерии приемки

- Наличие документации
- Возможность запуска в Docker
- Успешное прохождение тестов
- Корректная работа API
- Обработка краевых(крайних) случаев
- Соответствие кода стандартам PEP8
- Наличие TypeHint

## 7. Дополнительные рекомендации

- Использовать асинхронные запросы к сторонним системам
- Использовать асинхроные запросы к БД
- Реализовать кэширование результатов
- Добавить логирование всех операций
- Предусмотреть обработку больших архивов
- Проверять контрольные суммы при загрузке в хранилище
- Использовать глобальные обработчики ошибок

## 8. Приоритеты

1. Базовая функциональность

2. Тестирование
3. Документация
4. Оптимизация производительности

## 9. Дополнительные требования (не обязательно)

- Предусмотреть возможность работы сервиса в доверенной среде (без авторизации, но с учётом пользователя по его ID)
- Предусмотреть возможность загрузки данных из разных источников, например по ссылке на git репозиторий

## 10. Сроки реализации

8 календарных дней.

* Это тестовое задание и нет надобности доводить всё до идеала/полной готовности, достаточно показать концепцию и основные технические решения. Оценивается в т.ч. умение использовать готовые решения.

# Контракты API

## 1. POST /upload

### 1.1 Описание

Отправка ZIP-архива на проверку

### 1.2 Request

```
requestBody:
  content:
    multipart/form-data:
      schema:
        type: object
        properties:
          file:
            type: string
            format: binary
            description: ZIP архив для проверки
```

### 1.3 Response (201 Created)

```
content:
  application/json:
    schema:
      type: object
      properties:
        task_id:
          type: string
          description: ID задачи для отслеживания результатов
          example: "5f8a3b7e-1234-11ec-b909-0242ac130002"
```

# 2. GET /results/{task_id}

## 2.1 Описание

Получение результатов проверки

## 2.1 Path parameters

```
parameters:
  - name: task_id
    in: path
    description: ID задачи
    required: true
    schema:
      type: string
      example: "5f8a3b7e-1234-11ec-b909-0242ac130002"
```

## 2.3 Response (200 OK)

```
content:
  application/json:
    schema:
      type: object
      properties:
        status:
          type: string
          enum: ["PENDING", "IN_PROGRESS", "SUCCESS", "FAILED"]
          example: "SUCCESS"
        results:
          type: object
          properties:
            sonarqube:
              type: object
```

```
properties:
  overall_coverage:
    type: number
    description: Общий процент покрытия кода
    example: 85.5
  bugs:
    type: object
    properties:
      total:
        type: integer
        description: Общее количество багов
        example: 12
      critical:
        type: integer
        description: Критические баги
        example: 2
      major:
        type: integer
        description: Серьезные баги
        example: 5
      minor:
        type: integer
        description: Незначительные баги
        example: 5
  code_smells:
    type: object
    properties:
      total:
        type: integer
        description: Общее количество запахов кода
        example: 20
      critical:
```

```
        type: integer
        description: Критические запахи
        example: 3
      major:
        type: integer
        description: Серьезные запахи
        example: 10
      minor:
        type: integer
        description: Незначительные запахи
        example: 7
  vulnerabilities:
    type: object
    properties:
      total:
        type: integer
        description: Общее количество уязвимостей
        example: 4
      critical:
        type: integer
        description: Критические уязвимости
        example: 1
      major:
        type: integer
        description: Серьезные уязвимости
        example: 2
      minor:
        type: integer
        description: Незначительные уязвимости
        example: 1
```

## 2.4 Response (404 Not Found)

```
content:
  application/json:
    schema:
      type: object
      properties:
        error:
          type: string
          description: Сообщение об ошибке
          example: "Задача не найдена"
```