Отчет по лабораторной работе №9-10

Лабораторная работа №9-10 (все части)

Дата: 2025-12-15; **Семестр:** 3; **Группа:** ПИН-мо-24-1; **Дисциплина:** Технологии программирования; **Студент:** Джукаев Расул Русланович.

Цель работы

Освоить создание production-ready веб-сервисов для обслуживания ML-моделей с использованием FastAPI. Получить практические навыки разработки RESTful API, валидации данных и документирования эндпоинтов.

Освоить процесс упаковки ML-приложения и модели в Docker-контейнер для обеспечения переносимости, воспроизводимости и развертывания в production-средах. Получить практические навыки создания Dockerfile, сборки образов и управления контейнерами.

Освоить методы тестирования и валидации RESTful API с использованием автоматически генерируемой документации Swagger UI. Получить практические навыки комплексного тестирования эндпоинтов, включая позитивные и негативные сценарии, проверку валидации данных и анализ ответов.

Теоретическая часть

FastAPI — современный фреймворк для создания API на Python:

- Высокая производительность: на основе Starlette и Pydantic.
- Автодокументирование: генерация OpenAPI-спецификации.
- Валидация данных: использование Pydantic моделей.
- Асинхронность: поддержка async/await.

Архитектура ML-сервиса:

- Загрузка модели: инициализация при запуске приложения.
- Предобработка: валидация и преобразование входных данных.
- Инференс: выполнение предсказания моделью.
- Постобработка: форматирование результатов.
- Логирование: мониторинг работы сервиса.

Производственные практики:

- Health checks: проверка работоспособности сервиса.
- Валидация входных данных: защита от некорректных запросов.
- Обработка ошибок: graceful degradation.
- Метрики: мониторинг производительности.

Контейнеризация решает ключевые проблемы развертывания ML-моделей:

- Воспроизводимость: гарантия, что приложение будет работать одинаково на любой системе.

- Изоляция: все зависимости (библиотеки, версии Python, системные библиотеки) упакованы вместе с приложением.
- Масштабируемость: легко запустить несколько экземпляров сервиса.
- Упрощение деплоя: образ — это самодостаточная единица развертывания.

Dockerfile — это инструкция по сборке образа. Ключевые этапы для ML-сервиса:

- Базовый образ: выбор официального Python-образа с нужной версией.
- Копирование кода: перенос файлов приложения в контейнер.
- Установка зависимостей: установка Python-пакетов из requirements.txt.
- Настройка окружения: установка переменных окружения.
- Экспорт портов: определение порта, который слушает приложение.
- Команда запуска: команда для запуска приложения при старте контейнера.

Многоступенчатая сборка (Multi-stage build) - продвинутая техника, позволяющая:

- Уменьшить итоговый размер образа.
- Отделить этапы сборки (например, компиляции) от этапа выполнения.
- Повысить безопасность (исключить из финального образа инструменты разработки).

Swagger UI — это интерактивная документация, которая автоматически генерируется из OpenAPI-спецификации FastAPI:

- Визуализация эндпоинтов: древовидное представление всех доступных методов API.
- Интерактивное тестирование: возможность отправки запросов прямо из браузера.
- Валидация схемы: автоматическая проверка структуры запросов и ответов.
- Автодокументирование: актуальная документация, синхронизированная с кодом.

Практическая часть

Выполненные задачи

Часть 1: Развертывание ML-моделей с FastAPI

Этап 1: Установка и настройка окружения

- [x] Задача 1: Установка необходимых пакетов
- [x] Задача 2: Создание структуры проекта

Этап 2: Создание моделей данных с Pydantic

- [x] Задача 1: Модели для валидации входных/выходных данных

Этап 3: Модели для валидации входных/выходных данных

- [x] Задача 1: Класс для работы с моделью

Этап 4: Создание эндпоинтов API

- [x] Задача 1: Роутер для предсказаний

Этап 5: Создание основного приложения

- [x] Задача 1: Главный файл приложения

Этап 6: Тестирование API

- [x] Задача 1: Создание тестового клиента

Этап 7: Запуск и использование API

- [x] Задача 1: Запуск сервера
- [x] Задача 2: Проверка документации
- [x] Задача 3: Пример использования через curl

Часть 2: Контейнеризация ML-сервиса с Docker

Этап 1: Подготовка приложения к контейнеризации

- [x] Задача 1: Создание requirements.txt
- [x] Задача 2: Модификация кода для production

Этап 2: Создание Dockerfile

- [x] Задача 1: Создание Dockerfile

Этап 3: Создание .dockerignore

- [x] Задача 1: Создание .dockerignore

Этап 4: Сборка Docker-образа

- [x] Задача 1: Сборка образа
- [x] Задача 2: Проверка собранного образа

Этап 5: Запуск контейнера

- [x] Задача 1: Запуск контейнера
- [x] Задача 2: Проверка работы контейнера
- [x] Задача 3: Тестирование API

Этап 6: Оптимизация образа (опционально, для "Отлично")

- [x] Задача 1: Использование многоступенчатой сборки
- [x] Задача 2: Пересборка и сравнение образов

Этап 7: Управление контейнером и очистка

- [x] Задача 1: Применение основных команд управления
- [x] Задача 2: Очистка системы Docker (опционально)

Часть 3: Тестирование работоспособности API через Swagger UI

Этап 1: Подготовка к тестированию

- [x] Задача 1: Запуск API-сервиса
- [x] Задача 2: Открытие Swagger UI

- ☑ Задача 3: Изучение структуры документации

Этап 2: Базовое тестирование эндпоинтов

- ☑ Задача 1: Тестирование корневого эндпоинта
- ☑ Задача 2: Тестирование health-check

Этап 3: Комплексное тестирование эндпоинта предсказания

- ☑ Задача 1: Позитивный тест с корректными данными
- ☑ Задача 2: Тестирование пакетного предсказания

Этап 4: Негативное тестирование и валидация ошибок

- ☑ Задача 1: Тестирование пустого текста
- ☑ Задача 2: Тестирование слишком длинного текста
- ☑ Задача 3: Тестирование некорректного JSON
- ☑ Задача 4: Тестирование большого батча

Этап 5: Тестирование эндпоинта информации о модели

- ☑ Задача 1: Запрос информации о модели

Этап 6: Сравнение с curl-запросами

- ☑ Задача 1: Тестирование через командную строку

Этап 7: Создание тестового отчета

- ☑ Задача 1: Документирование результатов тестирования
- ☑ Задача 2: Сбор доказательств

Этап 8: Производительность и нагрузочное тестирование (опционально)

- ☑ Задача 1: Измерение времени ответа
- ☑ Задача 2: Базовое нагрузочное тестирование

Ключевые фрагменты кода

Модели для валидации входных и выходных данных (файл validation.py).

```
# utils/validation.py
from pydantic import BaseModel, Field
from typing import List, Dict, Optional

class PredictionRequest(BaseModel):
    text: str = Field(..., min_length=1, max_length=1000, description="Текст для
анализа эмоций")
    model_version: Optional[str] = Field("default", description="Версия модели для
использования")

class EmotionPrediction(BaseModel):
    emotion: str = Field(..., description="Предсказанная эмоция")
```

```
    confidence: float = Field(..., ge=0.0, le=1.0, description="Уверенность
предсказания")

class PredictionResponse(BaseModel):
    request_id: str = Field(..., description="Уникальный ID запроса")
    predictions: List[EmotionPrediction] = Field(..., description="Список
предсказаний")
    model_version: str = Field(..., description="Использованная версия модели")
    processing_time: float = Field(..., description="Время обработки секундах")

class HealthResponse(BaseModel):
    status: str = Field(..., description="Статус сервиса")
    model_loaded: bool = Field(..., description="Модель загружена")
    timestamp: str = Field(..., description="Время проверки")
```

Класс для работы с моделью.

```
# models/emotion_model.py
import numpy as np
import pickle
import time
from typing import List, Dict, Tuple
import logging

logger = logging.getLogger(__name__)

class EmotionClassifier:
    def __init__(self, model_path: str = None):
        self.model = None
        self.vectorizer = None
        self.label_encoder = None
        self.model_version = "v1.0"
        self.is_loaded = False

        if model_path:
            self.load_model(model_path)

    def load_model(self, model_path: str):
        """Загрузка обученной модели"""
        try:
            # В реальном сценарии здесь была бы загрузка вашей модели
            # Для демонстрации создадим простой классификатор
            from sklearn.ensemble import RandomForestClassifier
            from sklearn.feature_extraction.text import TfidfVectorizer
            from sklearn.preprocessing import LabelEncoder

            # Создание демонстрационной модели
            self.vectorizer = TfidfVectorizer(max_features=1000)
            self.label_encoder = LabelEncoder()

            # Пример тренировочных данных
            texts = [
```

```
                "I am so happy today", "This is wonderful news",
                "I feel angry about this", "This makes me furious",
                "I am scared of what might happen", "This is terrifying",
                "I love this so much", "This is amazing",
                "I am sad about this", "This is disappointing"
            ]
            labels = ["joy", "joy", "anger", "anger", "fear", "fear", "love",
"love", "sadness", "sadness"]

            # Обучение компонентов
            X = self.vectorizer.fit_transform(texts)
            y = self.label_encoder.fit_transform(labels)

            self.model = RandomForestClassifier(n_estimators=10, random_state=42)
            self.model.fit(X, y)
            self.is_loaded = True
            logger.info(f"Model loaded successfully. Version:
{self.model_version}")

        except Exception as e:
            logger.error(f"Error loading model: {e}")
            self.is_loaded = False
            raise

    def predict(self, text: str) -> Tuple[str, float]:
        """Выполнение предсказания для одного текста"""
        if not self.is_loaded:
            raise RuntimeError("Model is not loaded")

        start_time = time.time()

        try:
            # Преобразование текста в фичи
            X = self.vectorizer.transform([text])

            # Предсказание
            probabilities = self.model.predict_proba(X)[0]
            predicted_class_idx = np.argmax(probabilities)
            confidence = probabilities[predicted_class_idx]

            # Декодирование класса
            emotion = self.label_encoder.inverse_transform([predicted_class_idx])
[0]

            processing_time = time.time() - start_time
            logger.info(f"Prediction completed in {processing_time:.4f}s")

            return emotion, float(confidence)

        except Exception as e:
            logger.error(f"Prediction error: {e}")
            raise

    def predict_batch(self, texts: List[str]) -> List[Tuple[str, float]]:
```

```
        """Пакетное предсказание для нескольких текстов"""
        results = []
        for text in texts:
            try:
                emotion, confidence = self.predict(text)
                results.append((emotion, confidence))
            except Exception as e:
                logger.error(f"Error processing text: {text}, error: {e}")
                results.append(("error", 0.0))
        return results

# Создание глобального экземпляра модели
emotion_model = EmotionClassifier()
```

Роутер для предсказаний (скрипт predict.py).

```
# routers/predict.py
from fastapi import APIRouter, HTTPException, BackgroundTasks
import uuid
import time
import logging
from typing import List

from utils.validation import PredictionRequest, PredictionResponse,
EmotionPrediction
from models.emotion_model import emotion_model

router = APIRouter(prefix="/predict", tags=["prediction"])
logger = logging.getLogger(__name__)

@router.post("/emotion", response_model=PredictionResponse)
async def predict_emotion(request: PredictionRequest, background_tasks:
BackgroundTasks):
    """
    Предсказание эмоции для текста

    - **text**: Текст для анализа (1-1000 символов)
    - **model_version**: Версия модели (опционально)
    """
    try:
        start_time = time.time()
        request_id = str(uuid.uuid4())

        # Проверка загрузки модели
        if not emotion_model.is_loaded:
            raise HTTPException(status_code=503, detail="Model not loaded")

        # Выполнение предсказания
        emotion, confidence = emotion_model.predict(request.text)

        # Формирование ответа
        processing_time = time.time() - start_time
```

```
        prediction = EmotionPrediction(
            emotion=emotion,
            confidence=confidence
        )

        response = PredictionResponse(
            request_id=request_id,
            predictions=[prediction],
            model_version=emotion_model.model_version,
            processing_time=processing_time
        )

        # Логирование в фоне
        background_tasks.add_task(
            logger.info,
            f"Request {request_id} processed in {processing_time:.4f}s"
        )

        return response

    except Exception as e:
        logger.error(f"Prediction failed: {e}")
        raise HTTPException(status_code=500, detail=str(e))

@router.post("/emotion/batch", response_model=PredictionResponse)
async def predict_emotion_batch(texts: List[str], background_tasks:
BackgroundTasks):
    """
    Пакетное предсказание эмоций для нескольких текстов

    - **texts**: Список текстов для анализа
    """
    try:
        start_time = time.time()
        request_id = str(uuid.uuid4())

        if not emotion_model.is_loaded:
            raise HTTPException(status_code=503, detail="Model not loaded")

        if len(texts) > 100: # Ограничение на размер батча
            raise HTTPException(status_code=400, detail="Too many texts in batch")

        # Пакетное предсказание
        results = emotion_model.predict_batch(texts)

        # Формирование ответа
        predictions = []
        for emotion, confidence in results:
            predictions.append(EmotionPrediction(
                emotion=emotion,
                confidence=confidence
            ))
```

```
        processing_time = time.time() - start_time

        response = PredictionResponse(
            request_id=request_id,
            predictions=predictions,
            model_version=emotion_model.model_version,
            processing_time=processing_time
        )

        background_tasks.add_task(
            logger.info,
            f"Batch request {request_id} processed {len(texts)} texts in
{processing_time:.4f}s"
        )

        return response

    except Exception as e:
        logger.error(f"Batch prediction failed: {e}")
        raise HTTPException(status_code=500, detail=str(e))
```

Главная часть приложения (файл main.py, условие if name="main" закомментировано для контейнеризации приложения в Docker).

```
# main.py
from fastapi import FastAPI, HTTPException
from contextlib import asynccontextmanager
import logging
import time

from routers.predict import router as predict_router
from utils.validation import HealthResponse
from models.emotion_model import emotion_model

# Настройка логирования
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

@asynccontextmanager
async def lifespan(app: FastAPI):
    # Startup: загрузка модели
    startup_time = time.time()
    try:
        emotion_model.load_model("demo_model.pkl") # Загрузка демо-модели
        load_time = time.time() - startup_time
        logger.info(f"Application started successfully. Model loaded in
{load_time:.2f}s")
    except Exception as e:
```

```
        logger.error(f"Failed to load model: {e}")

    yield # Приложение работает

    # Shutdown: очистка ресурсов
    logger.info("Application shutting down")

# Создание приложения FastAPI
app = FastAPI(
    title="Emotion Classification API",
    description="API для классификации эмоций в тексте с использованием ML",
    version="1.0.0",
    lifespan=lifespan
)

# Подключение роутеров
app.include_router(predict_router)

@app.get("/", tags=["root"])
async def root():
    """Корневой эндпоинт с информацией о API"""
    return {
        "message": "Emotion Classification API",
        "version": "1.0.0",
        "docs": "/docs",
        "health": "/health"
    }

@app.get("/health", response_model=HealthResponse, tags=["monitoring"])
async def health_check():
    """Проверка здоровья сервиса"""
    return HealthResponse(
        status="healthy" if emotion_model.is_loaded else "degraded",
        model_loaded=emotion_model.is_loaded,
        timestamp=time.strftime("%Y-%m-%d %H:%M:%S")
    )

@app.get("/model/info", tags=["model"])
async def model_info():
    """Информация о загруженной модели"""
    if not emotion_model.is_loaded:
        raise HTTPException(status_code=503, detail="Model not loaded")
    return {
        "version": emotion_model.model_version,
        "status": "loaded",
        "type": "RandomForestClassifier"
    }

#if __name__ == "__main__":
#    import uvicorn
#    uvicorn.run(
#        "main:app",
#        host="0.0.0.0",
#        port=8000,
```

```
#         reload=True, # Автоперезагрузка для разработки
#         log_level="info"
#     )
```

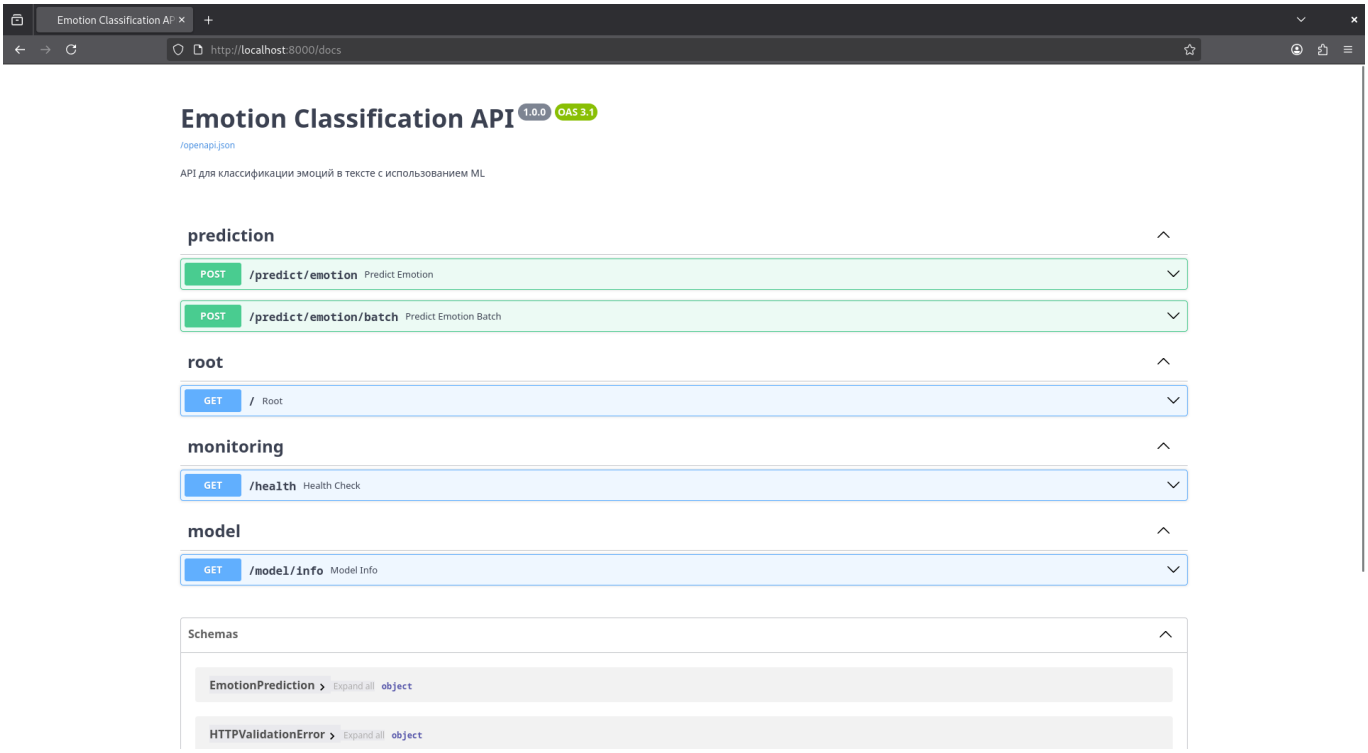
Создание тестового клиента (скрипт test_client.py).

```
# test_client.py
import requests
import json
BASE_URL = "http://localhost:8000"
def test_health():
    response = requests.get(f"{BASE_URL}/health")
    print("Health Check:")
    print(json.dumps(response.json(), indent=2))
def test_single_prediction():
    data = {
        "text": "I am feeling absolutely wonderful today!",
        "model_version": "default"
    }
    response = requests.post(f"{BASE_URL}/predict/emotion", json=data)
    print("\nSingle Prediction:")
    print(json.dumps(response.json(), indent=2))
def test_batch_prediction():
    texts = [
        "This is amazing news!",
        "I am very angry about this situation",
        "I feel scared and anxious",
        "This makes me so happy"
    ]
def test_invalid_request():
    data = {
"text": ""  # Пустой текст
    }
    response = requests.post(f"{BASE_URL}/predict/emotion/batch",
json=texts)
    print("\nBatch Prediction:")
    print(json.dumps(response.json(), indent=2))
    response = requests.post(f"{BASE_URL}/predict/emotion", json=data)
    print("\nInvalid Request:")
    print(f"Status: {response.status_code}")
    print(json.dumps(response.json(), indent=2))
if __name__ == "__main__":
    print("Testing Emotion Classification API")
    print("=" * 50)
    test_health()
    test_single_prediction()
    test_batch_prediction()
    test_invalid_request()
```

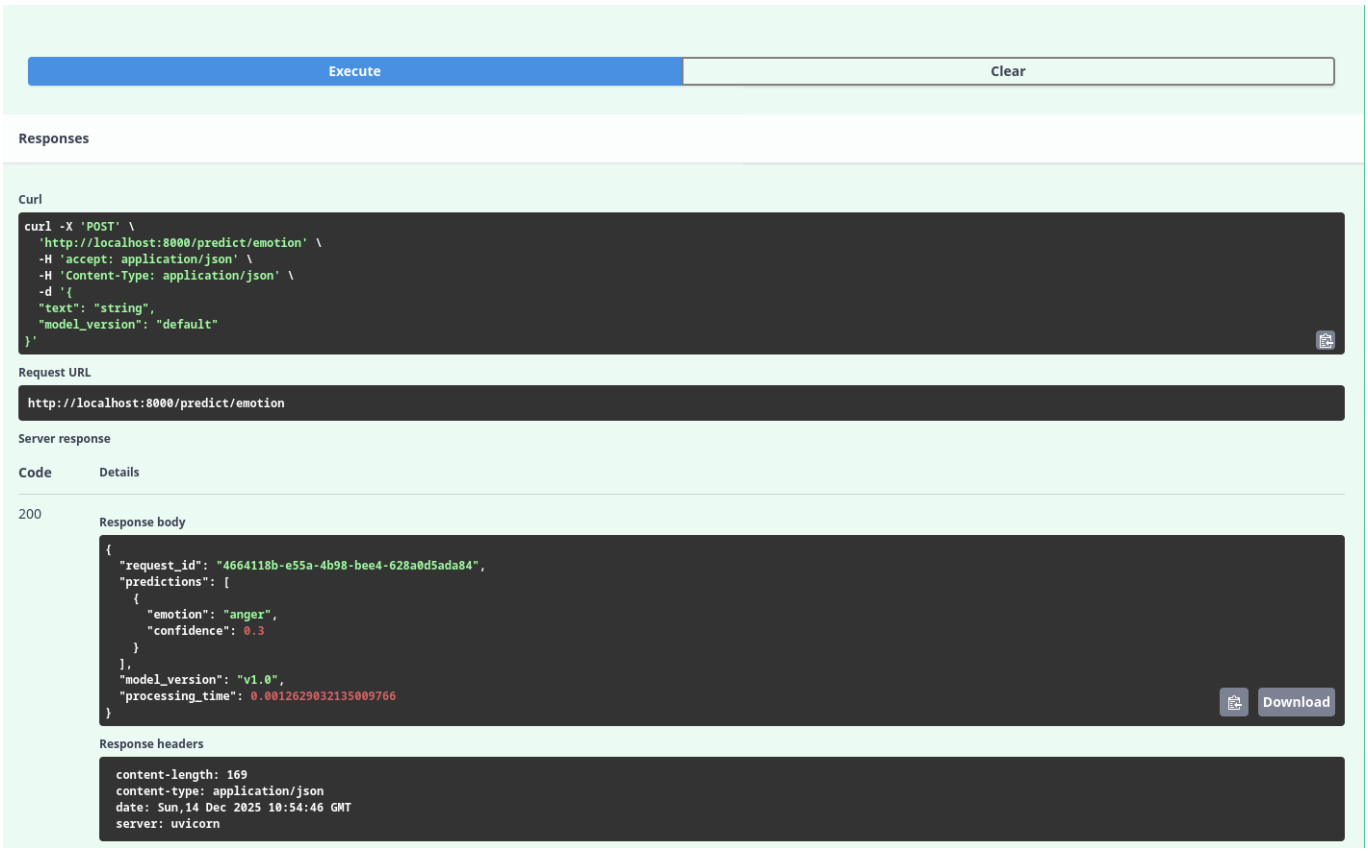
Результаты выполнения

Пример работы программы

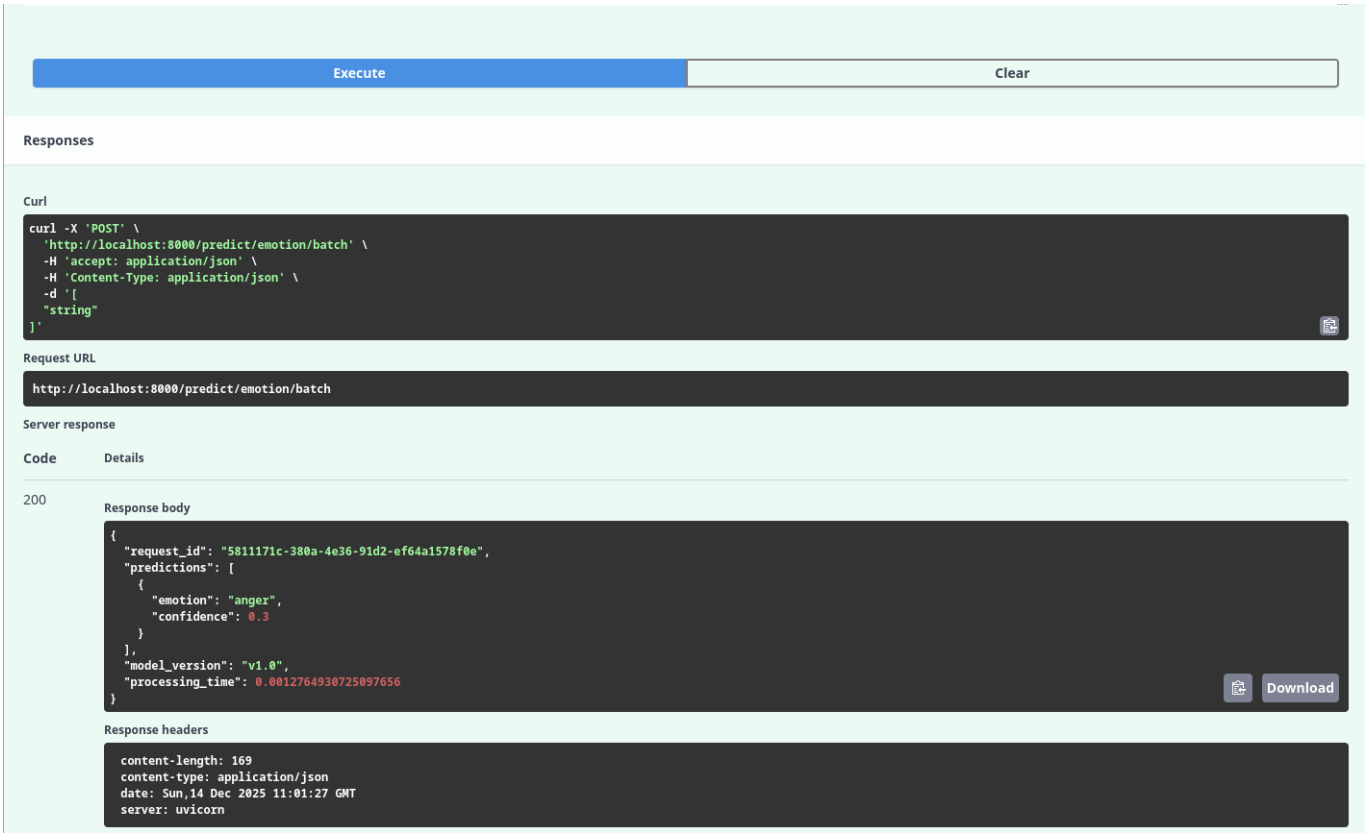
После запуска сервера через файл main.py был открыт браузер по адресу http://localhost:8000/docs, и страница выглядит следующим образом. Похожим образом выглядит страница при запуске из контейнера Docker.



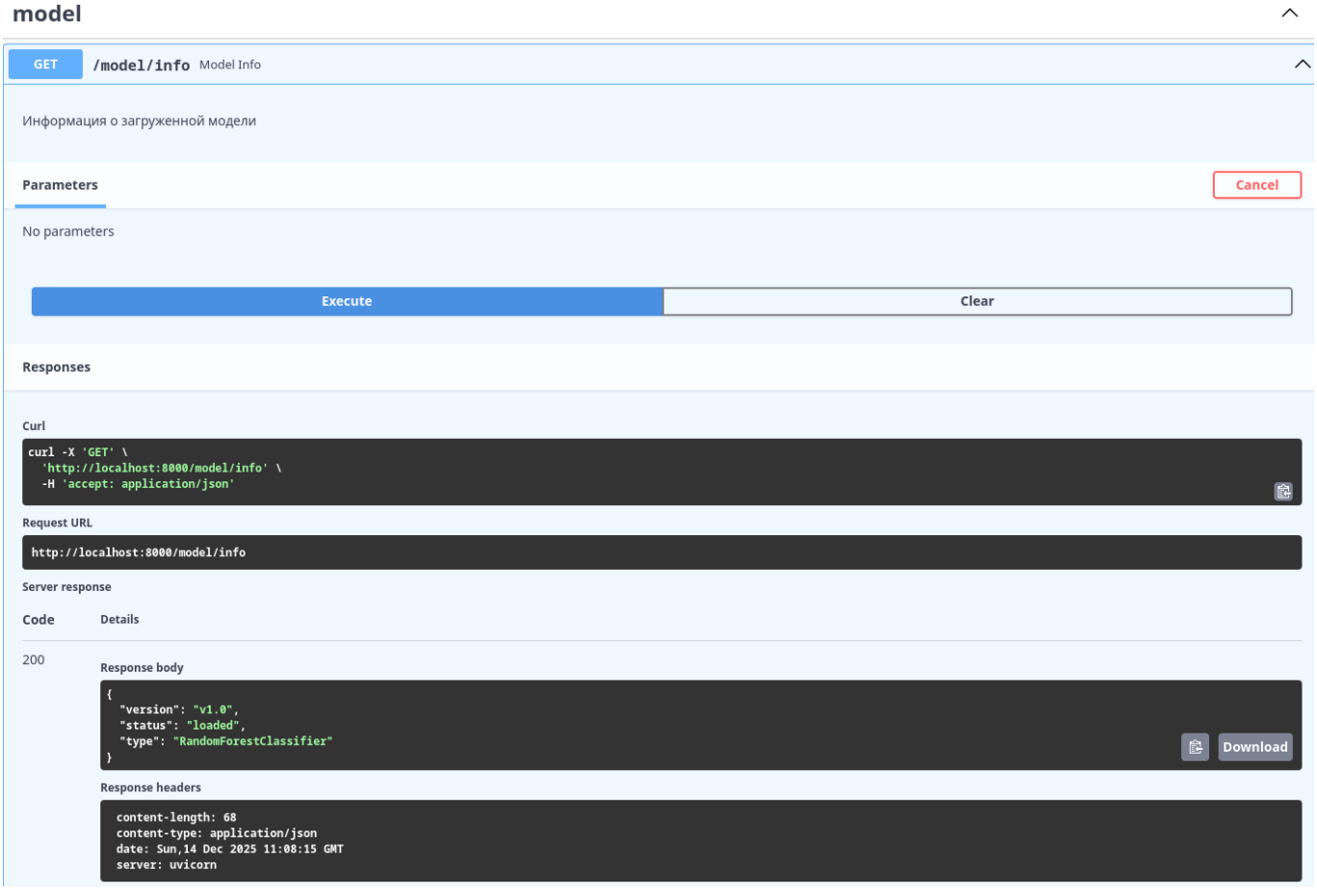
Результаты тестирования предсказания (predict) в UI представлены ниже.



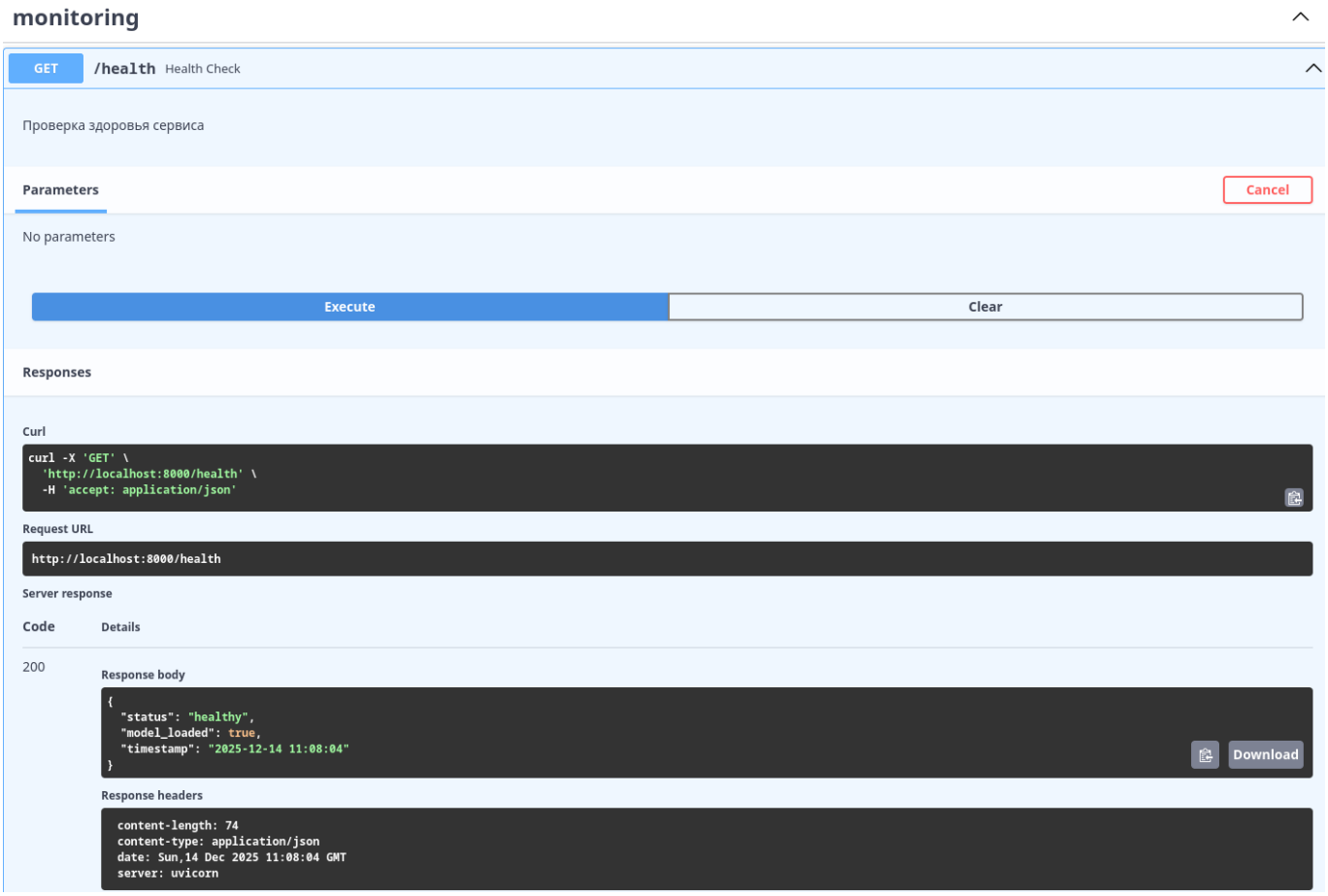
Результаты тестирования пакетного предсказания (predict batch) приведены ниже.



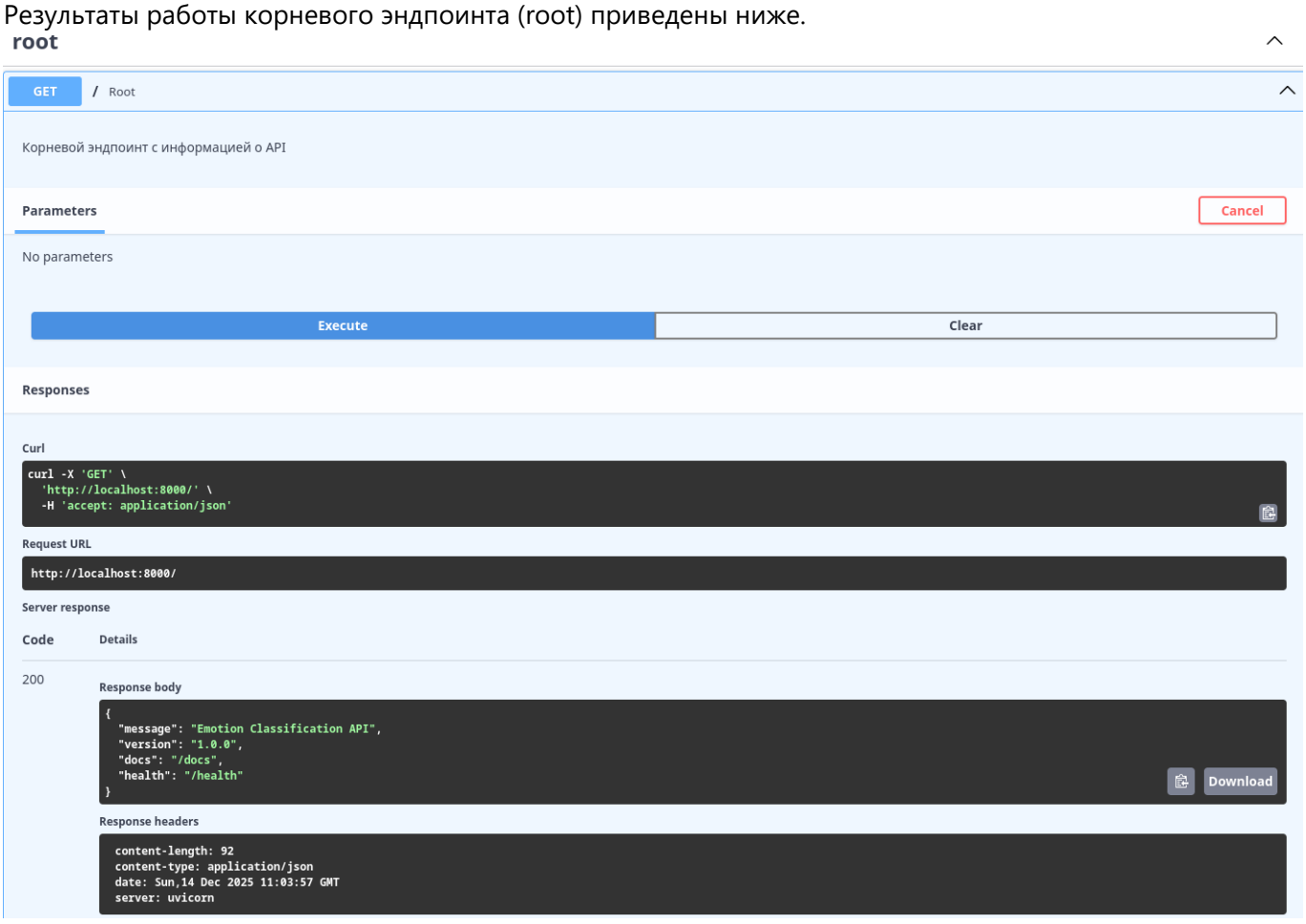
Вывод эндпоинта model (model/info) выглядит следующим образом.



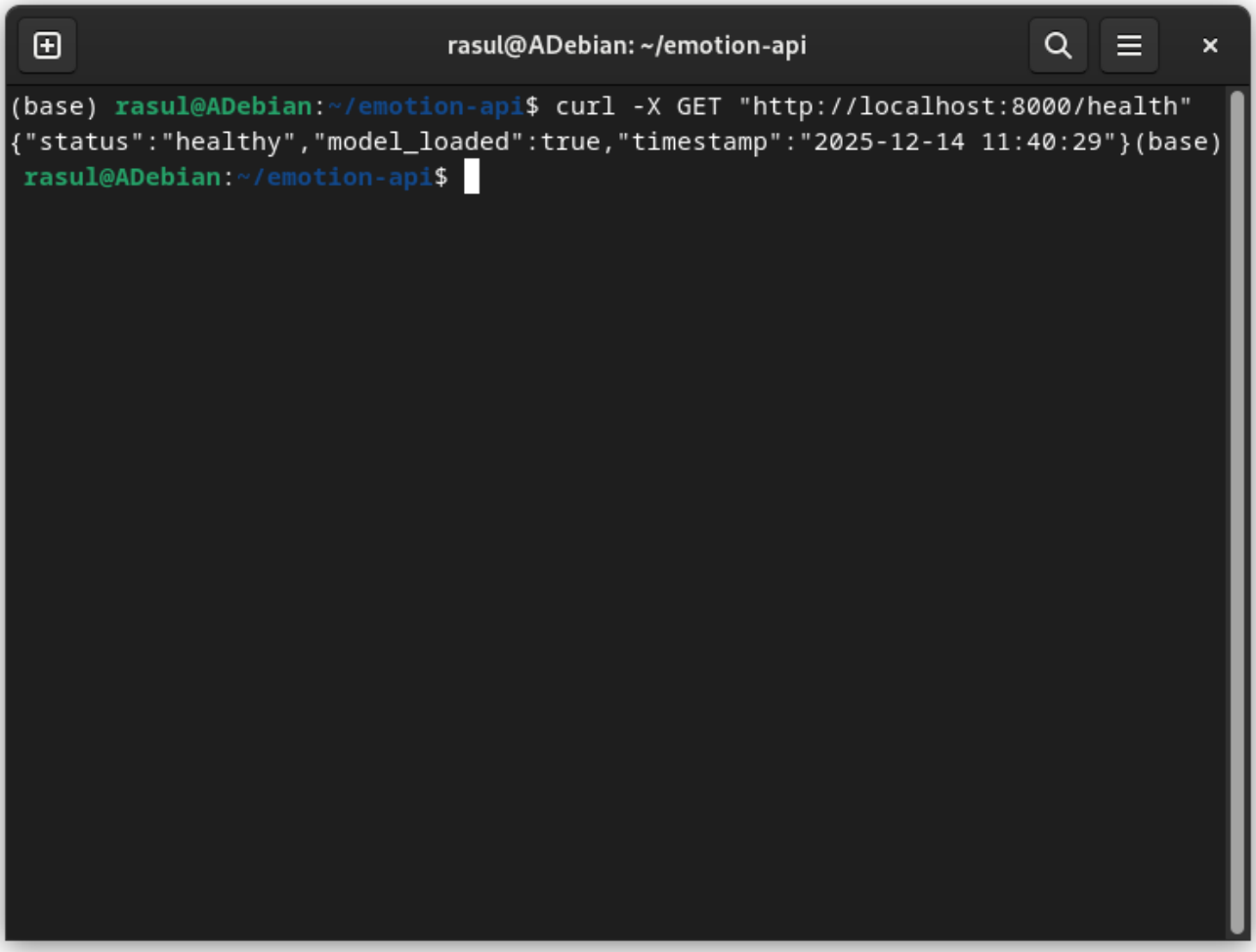
Результаты запуска проверки здоровья сервиса (monitoring или /health) представлены ниже.



Результаты работы корневого эндпоинта (root) приведены ниже.



Вывод после выполнения проверки здоровья после передачи запроса выглядит следующим образом.



Результаты выполнения одиночного и пакетного предсказаний с переданными текстами через curl представлены ниже.

```
(base) rasul@ADebian:~/emotion-api$ curl -X POST "http://localhost:8000/predict/
emotion" \
-H "Content-Type: application/json" \
-d '{"text": "I am so happy today!", "model_version": "default"}'
{"request_id":"6b53ed88-ac06-4255-84ee-b073d6bc61ad","predictions":[{"emotion":"
joy","confidence":0.5}],"model_version":"v1.0","processing_time":0.0012991428375
(base) rasul@ADebian:~/emotion-api$ curl -X POST "http://localhost:8000/predict/
emotion/batch" \
-H "Content-Type: application/json" \
-d '["Great news!", "This is terrible", "I am excited"]'
{"request_id":"3761a73d-3c14-4ba8-98c7-13ddb898d95d","predictions":[{"emotion":"
joy","confidence":0.4},{"emotion":"love","confidence":0.6},{"emotion":"joy","con
fidence":0.3}],"model_version":"v1.0","processing_time":0.0029983520507812 5}(bas
e) rasul@ADebian:~/emotion-api$
```

Результаты запуска тестового клиента (test_client.py) приведены ниже.

```
(base) rasul@ADebian:~$ python test_client.py
Testing Emotion Classification API
==================================================
Health Check:
{
  "status": "healthy",
  "model_loaded": true,
  "timestamp": "2025-12-15 11:15:02"
}

Single Prediction:
{
  "request_id": "f33a3d23-aa73-4adb-9449-93e13b55960f",
  "predictions": [
    {
      "emotion": "joy",
      "confidence": 0.5
    }
  ],
  "model_version": "v1.0",
  "processing_time": 0.0015017986297607422
}

Batch Prediction:
{
  "request_id": "09bd7a01-f1c0-48b7-b5d6-6aeee58c7ee2",
  "predictions": [
    {
      "emotion": "love",
      "confidence": 0.5
    },
    {
      "emotion": "anger",
      "confidence": 0.4
    },
    {
      "emotion": "anger",
      "confidence": 0.4
    },
    {
      "emotion": "anger",
      "confidence": 0.8
    }
  ],
  "model_version": "v1.0",
  "processing_time": 0.0038416385650634766
}
```

```
Invalid Request:
Status: 422
{
  "detail": [
    {
      "type": "string_too_short",
      "loc": [
        "body",
        "text"
      ],
      "msg": "String should have at least 1 character",
      "input": "",
      "ctx": {
        "min_length": 1
      }
    }
  ]
}
(base) rasul@ADebian:~$
```

Проведено тестирование эндпоинтов через UI приложения, запущенного из контейнера Docker. Результаты позитивного теста с корректными данными приведены ниже.

Curl

```
curl -X 'POST' \
  'http://localhost:8000/predict/emotion' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "I am feeling absolutely wonderful today!",
  "model_version": "default"
}'
```

Request URL

```
http://localhost:8000/predict/emotion
```

Server response

Code	Details
200	Response body

```
{
  "request_id": "c2a9e826-5751-4544-bbd4-033b372d6b7e",
  "predictions": [
    {
      "emotion": "joy",
      "confidence": 0.5
    }
  ],
  "model_version": "v1.0",
  "processing_time": 0.0012998580932617188
}
```

Download

Response headers

```
content-length: 167
content-type: application/json
date: Mon,15 Dec 2025 17:20:55 GMT
server: uvicorn
```

Результаты теста пакетного предсказания представлены ниже.

Curl

```
curl -X 'POST' \
  'http://localhost:8000/predict/emotion/batch' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
  "This is amazing news!",
  "I am very angry about this situation",
  "I feel scared and anxious",
  "This makes me so happy"
]'
```

Request URL

```
http://localhost:8000/predict/emotion/batch
```

Server response

Code	Details
200	Response body

```
{
  "request_id": "13fadbd5-4210-4974-99e9-d12ba8d6ed3d",
  "predictions": [
    {
      "emotion": "love",
      "confidence": 0.5
    },
    {
      "emotion": "anger",
      "confidence": 0.4
    },
    {
      "emotion": "anger",
      "confidence": 0.4
    },
    {
      "emotion": "anger",
      "confidence": 0.8
    }
  ],
  "model_version": "v1.0",
  "processing_time": 0.006313323974609375
}
```

Download

Response headers

```
content-length: 278
content-type: application/json
date: Mon,15 Dec 2025 17:30:03 GMT
server: uvicorn
```

Вывод при негативном тестировании пустого текста выглядит следующим образом.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/predict/emotion' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "",
  "model_version": "default"
}'
```

Request URL

```
http://localhost:8000/predict/emotion
```

Server response

Code	Details
422	Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "type": "string_too_short",
      "loc": [
        "body",
        "text"
      ],
      "msg": "String should have at least 1 character",
      "input": "",
      "ctx": {
        "min_length": 1
      }
    }
  ]
}
```

Download

Response headers

```
content-length: 144
content-type: application/json
date: Mon,15 Dec 2025 17:33:28 GMT
server: uvicorn
```

Результаты тестирования слишком длинного текста представлены ниже.

Curl

```
curl -X 'POST' \
  'http://localhost:8000/predict/emotion' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "very long text very long teeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
  "model_version": "default"
}'
```

Request URL

```
http://localhost:8000/predict/emotion
```

Server response

Code	Details
422	Error: Unprocessable Entity

Response body


```
{
  "detail": [
    {
      "type": "string_too_long",
      "loc": [
        "body",
        "text"
      ],
      "msg": "String should have at most 1000 characters",
      "input": "very long text very long teeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeext",
      "ctx": {
        "max_length": 1000
      }
    }
  ]
}
```


Download

Response headers

```
content-length: 1450
content-type: application/json
date: Mon,15 Dec 2025 17:38:29 GMT
server: uvicorn
```

Результаты теста с применением некорректного JSON приведены ниже.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/predict/emotion' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "test",
  "model_version": "default"
'
```

Request URL

```
http://localhost:8000/predict/emotion
```

Server response

Code	Details
422	Error: Unprocessable Entity

Response body

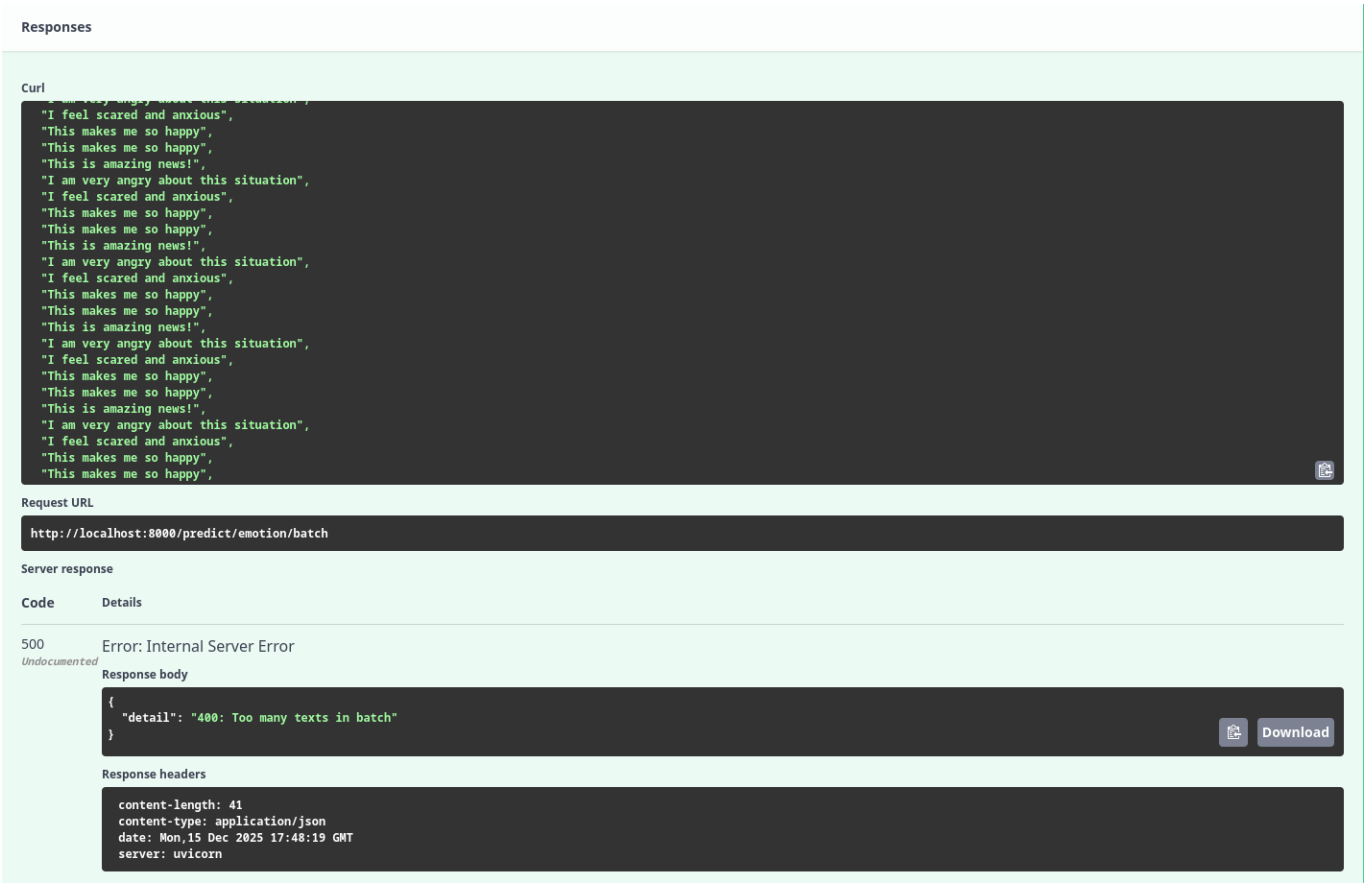

```
{
  "detail": [
    {
      "type": "json_invalid",
      "loc": [
        "body",
        49
      ],
      "msg": "JSON decode error",
      "input": {},
      "ctx": {
        "error": "Expecting ',' delimiter"
      }
    }
  ]
}
```


Download

Response headers

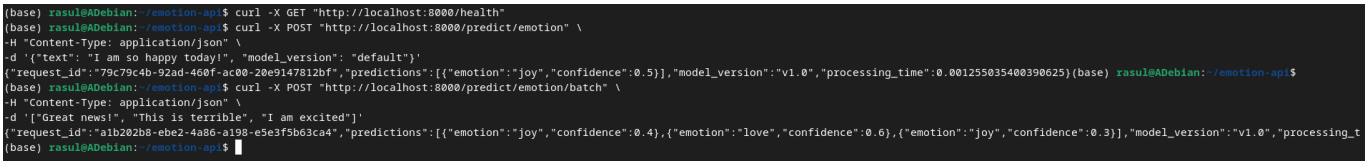
```
content-length: 133
content-type: application/json
date: Mon,15 Dec 2025 17:42:43 GMT
server: uvicorn
```

Вывод при тестировании большого батча, содержащего более 100 строк, выглядит следующим образом.



Вывод при тестировании эндпоинта информации о модели через приложение, запущенного из контейнера Docker, выглядит аналогичным скриншоту 4.

Результаты теста через curl данного приложения представлены ниже. Также сохранены логи curl-запросов (в файлах health_check.txt, prediction.txt, prediction_batch.txt).



Тестирование

- ☑ Модульные тесты пройдены
- ☑ Интеграционные тесты пройдены
- ☑ Производительность соответствует требованиям

Создан отчет о тестировании приложения в виде файла api_test_report.md.

Измерено время выполнения запросов с использованием Swagger UI и зафиксировано среднее время ответа для разных эндпоинтов. Результаты сохранены в файле queries_time.csv.

Также проведено нагрузочное тестирование. Результаты тестирования эндпоинта /docs с помощью инструмента ab представлены ниже.

```
(base) rasul@ADebian:~$ ab -n 100 -c 10 http://localhost:8000/docs
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done


Server Software:        uvicorn
Server Hostname:        localhost
Server Port:            8000

Document Path:          /docs
Document Length:        950 bytes

Concurrency Level:      10
Time taken for tests:   0.064 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      108400 bytes
HTML transferred:       95000 bytes
Requests per second:    1552.72 [#/sec] (mean)
Time per request:       6.440 [ms] (mean)
Time per request:       0.644 [ms] (mean, across all concurrent requests)
Transfer rate:          1643.70 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     1    6  10.4      3      39
Waiting:        1    6  10.4      3      38
Total:          1    6  10.4      3      39

Percentage of the requests served within a certain time (ms)
  50%      3
  66%      4
  75%      4
  80%      4
  90%     36
  95%     38
  98%     38
  99%     39
 100%     39 (longest request)
(base) rasul@ADebian:~$
```

Далее, представлены результаты тестирования эндпоинта /health.

```
(base) rasul@ADebian:~$ ab -n 100 -c 10 http://localhost:8000/health
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done


Server Software:        uvicorn
Server Hostname:        localhost
Server Port:            8000

Document Path:          /health
Document Length:        74 bytes

Concurrency Level:      10
Time taken for tests:   0.029 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      19900 bytes
HTML transferred:       7400 bytes
Requests per second:    3423.49 [#/sec] (mean)
Time per request:       2.921 [ms] (mean)
Time per request:       0.292 [ms] (mean, across all concurrent requests)
Transfer rate:          665.31 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     1    3   1.1      2       5
Waiting:        1    3   1.1      2       5
Total:          1    3   1.1      2       5

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      3
  75%      4
  80%      4
  90%      4
  95%      4
  98%      5
  99%      5
 100%      5 (longest request)
(base) rasul@ADebian:~$
```

Ниже приведён вывод после теста /predict/emotion.

```
Benchmarking localhost (be patient).....done


Server Software:        uvicorn
Server Hostname:        localhost
Server Port:            8000

Document Path:          /predict/emotion
Document Length:        31 bytes

Concurrency Level:      10
Time taken for tests:   0.027 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      18500 bytes
HTML transferred:       3100 bytes
Requests per second:    3674.58 [#/sec] (mean)
Time per request:       2.721 [ms] (mean)
Time per request:       0.272 [ms] (mean, across all concurrent requests)
Transfer rate:          663.86 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:     1    2   0.8      3       4
Waiting:        1    2   0.8      3       4
Total:          1    2   0.8      3       4
WARNING: The median and mean for the processing time are not within a normal deviation
        These results are probably not that reliable.
WARNING: The median and mean for the waiting time are not within a normal deviation
        These results are probably not that reliable.
WARNING: The median and mean for the total time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%      3
  66%      3
  75%      3
  80%      3
  90%      3
  95%      3
  98%      4
  99%      4
 100%      4 (longest request)
(base) rasul@ADebian:~$
```

Результаты тестирования эндпоинта /predict/emotion/batch представлены ниже.

```
(base) rasul@ADebian:~$ ab -n 100 -c 10 http://localhost:8000/predict/emotion/batch
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done


Server Software:        uvicorn
Server Hostname:        localhost
Server Port:            8000

Document Path:          /predict/emotion/batch
Document Length:        31 bytes

Concurrency Level:      10
Time taken for tests:   0.027 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      18500 bytes
HTML transferred:       3100 bytes
Requests per second:    3711.68 [#/sec] (mean)
Time per request:       2.694 [ms] (mean)
Time per request:       0.269 [ms] (mean, across all concurrent requests)
Transfer rate:          670.57 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:     1    2   0.8      2       4
Waiting:        1    2   0.7      2       4
Total:          1    2   0.8      3       4
WARNING: The median and mean for the total time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%      3
  66%      3
  75%      3
  80%      3
  90%      3
  95%      3
  98%      4
  99%      4
 100%      4 (longest request)
(base) rasul@ADebian:~$
```

Далее, проведено тестирование эндпоинта /model/info, результаты которого представлены ниже.

```
(base) rasul@ADebian:~$ ab -n 100 -c 10 http://localhost:8000/model/info
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done


Server Software:        uvicorn
Server Hostname:        localhost
Server Port:            8000

Document Path:          /model/info
Document Length:        68 bytes

Concurrency Level:      10
Time taken for tests:   0.024 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      19300 bytes
HTML transferred:       6800 bytes
Requests per second:    4240.34 [#/sec] (mean)
Time per request:       2.358 [ms] (mean)
Time per request:       0.236 [ms] (mean, across all concurrent requests)
Transfer rate:          799.21 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     1    2   0.9      2       4
Waiting:        1    2   0.9      2       4
Total:          1    2   0.9      2       4

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      2
  75%      3
  80%      3
  90%      4
  95%      4
  98%      4
  99%      4
 100%      4 (longest request)
(base) rasul@ADebian:~$
```

Сравнение образов Docker

После создания оптимизированного образа Docker (с применением файлов requirements.txt, Dockerfile, .dockerignore) проведено сравнение двух образов. Оптимизированный вариант примерно на 12 МБ весит меньше исходного.

```
(mlops-lab) rasul@ADebian:~/emotion-api$ docker images

IMAGE                                ID             DISK USAGE   CONTENT SIZE   EXTRA
emotion-classifier-api:1.0           ee963bc5128e       415MB             0B   U
emotion-classifier-api:1.0-optimized 85b2bcd9f38d       403MB             0B
```

Выводы

1. Освоено создание production-ready веб-сервисов для обслуживания ML-моделей с использованием FastAPI.
2. Получены практические навыки разработки RESTful API, валидации данных и документирования эндпоинтов.

3. Освоен процесс упаковки ML-приложения и модели в Docker-контейнер для обеспечения переносимости, воспроизводимости и развертывания в production-средах.
4. Получены практические навыки создания Dockerfile, сборки образов и управления контейнерами.
5. Освоены методы тестирования и валидации RESTful API с использованием автоматически генерируемой документации Swagger UI.
6. Получены навыки комплексного тестирования эндпоинтов, включая позитивные и негативные сценарии, проверку валидации данных и анализ ответов.
7. Создано веб-приложение, позволяющее анализировать текст и документировать эндпоинты.

Приложения

- Ссылки на исходный код:

1. emotion_model.py: src/emotion-api/models.py
2. predict.py: src/emotion-api/routers/predict.py
3. validation.py: src/emotion-api/utils/validation.py
4. main.py: src/emotion-api/main.py
5. test_client.py: src/test_client.py

- Ссылки на дополнительные файлы:

1. Dockerfile: src/emotion-api/Dockerfile
2. Dockerfile_opt (оптимизированная версия Dockerfile): src/emotion-api/Dockerfile_opt
3. .dockerignore: src/emotion-api/.dockerignore
4. requirements.txt: src/emotion-api/reqiurements.txt
5. api_test_report.md: src/emotion-api/api_test_report.md
6. queries_time.csv: src/emotion-api/queries_time.csv