**МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ**
**Заключительный этап**
**Практика «Информационные Технологии»**
**Командно-практическое задание**

## Условие

В 2226 году Земля установила контакт с множеством инопланетных цивилизаций. Учёный Михаил занимается приёмом, анализом и классификацией (определением цивилизации-отправителя) радиосигналов, посылаемых братьями по разуму по всей галактики. Но вот беда: различных видов инопланетян оказалось поразительно много, а звуки их голосов не всегда различимы человеческим ухом. Михаил решил прибегнуть к помощи сверхточных датчиков для улавливания сигналов и искусственного интеллекта для их классификации. С датчиками Михаил в состоянии разобраться сам, а вот инженер-программист из учёного посредственный. Помогите Михаилу написать нейронную сеть для классификации сигналов и разработать приложение для удобного взаимодействия с обученной моделью.

Для реализации этой задачи коллеги Михаила записали довольно большое количество радиосигналов и с помощью группы практикантов вручную классифицировали их, потратив на это более двух лет. Такой набор данных позволит обучить нейросеть, которая в свою очередь, даст возможность не прибегать к помощи практикантов в будущем.

К сожалению, практиканты при классификации звуков повредили обозначения классов в наборе данных. И вместо целочисленных значений в нем представлены слабочитаемые строки. Перед тем как приступить к обучению нейросети, вам необходимо восстановить обозначения классов в наборе данных. Известно только то, что классы были обозначены порядковыми целыми числами, начиная с нуля.

## Техническое задание

Необходимо разработать программный продукт, позволяющий определять инопланетную цивилизацию по записи отправленных ими радиосигналов. Определение цивилизации должно происходить с помощью обученной модели нейросети. **Запрещается использовать готовые модели**. Программный продукт должен иметь графический интерфейс и возможность авторизации.

### Обучение модели

Обучение модели следует проводить в сервисе Яндекс DataSphere. Для доступа к сервису используйте учетные данные, выданные организаторами на площадке проведения, логин в домене `ppo.idp.yandexcloud.net` и пароль. Инструкция по работе с сервисом доступна по ссылке.

В наборе данных содержится 1600 записей, из них 1200 для обучения и 400 для валидации.

**Авторизация**

Необходимо предусмотреть 2 роли пользователей:

- администратор;
- пользователь.

*Администратор*

В функциональные возможности администратора должно входить только создание новых пользователей, с внесением информации о пользователе  не менее чем в два текстовых полях (Имя, Фамилия).

*Пользователь*

В функциональные возможности пользователя должно входить:

- загрузка набора данных для проверки работы модели;
- просмотр аналитики.

Данные для входа пользователей должны храниться в СУБД.

**Аналитика**

В интерфейсе пользователя необходимо реализовать следующую функциональность:

- просмотр графика зависимости точности на валидационных данных от количества эпох обучения;
- просмотр диаграммы с отображением количества записей в наборе данных для обучения, относящихся к каждой цивилизации (классу);
- диаграмму, демонстрирующую точность определения каждой записи из тестового набора данных;
- диаграмму, демонстрирующую топ - 5 наиболее часто встречающихся классов записей в валидационном наборе данных.

**Пользовательский интерфейс**

Интерфейс пользователя должен предоставлять следующие возможности:

- авторизация пользователей;

- экранная форма/веб-страница с информацией о пользователе;
- экранная форма/веб-страница с графиками и диаграммами;
- масштабирование построенных графиков и диаграмм;
- загрузка “тестового” набора данных через форму загрузки файла;
- отображение точности и потерь на тестовом наборе данных.

## Входные данные

Набор данных для обучения и валидации хранится в архиве в формате `.npz`. и доступен по ссылке: https://disk.yandex.ru/d/BA4oJb0BwaABxg.

В массиве `train_x` находятся wav-файлы посланий инопланетян, в `train_y` — соответствующие им поврежденные номера (классы) цивилизаций для обучения.

В массиве `valid_x` находятся wav-файлы посланий инопланетян, в `valid_y` - соответствующие им номера (классы) цивилизаций для валидации.

Набор данных для проверки работы модели нейросети хранится в архиве в формате `.npz`. и доступен по ссылке: https://disk.yandex.ru/d/39RhMM5NiJvgcg. *Пароль от архива будет выдан экспертной комиссией при проверке работы.*

В массиве `test_x` находятся wav-файлы посланий инопланетян, в `test_y` — соответствующие им номера (классы) цивилизаций.

## Рекомендации к выполнению

Программный продукт должен иметь графический пользовательский интерфейс и быть реализован на любой платформе.

Рекомендуется разделить программный продукт на модули: back-end и front-end.

Разработку рекомендуется вести с помощью системы контроля версий git.

Рекомендуется использовать unit-тестирование при разработке продукта.

## Регламент испытаний

1. Демонстрация лога обучения и методов сохранения модели
2. Запуск программного продукта
3. Авторизация администратора
4. Создание пользователя по данным от жюри
5. Авторизация пользователя

6. Загрузка тестового набора данных
7. Демонстрация информации о тестовом наборе данных (точность и потери)
8. Демонстрация графиков и диаграмм
9. Демонстрация работы СУБД, в том числе, включающая в себя проверку персистентного хранения данных
10. Демонстрация работы unit-тестов
11. Загрузка материалов в облачную папку

## Загрузка решений

По окончании защиты необходимо загрузить все исходные файлы проекта:

- архив с репозиторием/программный код;
- обученную модель в формате .h5;
- файлы БД (дамп БД);
- снимки экрана пользовательского интерфейса -

в облачную папку в соответствии с наименованием вашей команды.

**МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ**
**Заключительный этап**
**Практика «Информационные Технологии»**
**Командно-практическое задание**

## Оценка

### Программный код (суммарно)

| Уровень | Критерии | Максимальный балл 30 | Комментарий |
|---|---|---|---|
| 1 | Код написан без соблюдения стилистики, имена переменных не несут смысловой нагрузки, код в целом трудно читаем | 0 | |
| 2 | Код читаем, разработчики в целом придерживаются одного стиля | 10 | |
| 3 | Разработка велась с помощью git | 10 | |
| 4 | Предоставлены unit-тесты | 10 | |

### Алгоритм (суммарно)

| Уровень | Критерии | Максимальный балл 150 | Комментарий |
|---|---|---|---|
| 1 | Данные не восстановлены | 0 | |
| 2 | Данные восстановлены | 20 | |
| 3 | Подготовлено окружение для обучения | 10 | |
| 4 | Запущен процесс обучения | 20 | |
| 5 | Процент точности на тестовом наборе данных | 100 * процент точности | |

**МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ**
**Заключительный этап**
**Практика «Информационные Технологии»**
**Командно-практическое задание**

**Работа с данными (аналитика) (суммарно)**

| **Уровень** | **Критерии** | **Максимальный балл 70** | **Комментарий** |
|---|---|---|---|
| 1 | Построен график точности | 20 | |
| 2 | Построена диаграмма | 10 | |
| 3 | Построен график на тестовых данных | 20 | |
| 4 | Для хранения данных, включая пользователей, используется СУБД | 20 | 10 если все хранится в файлах |

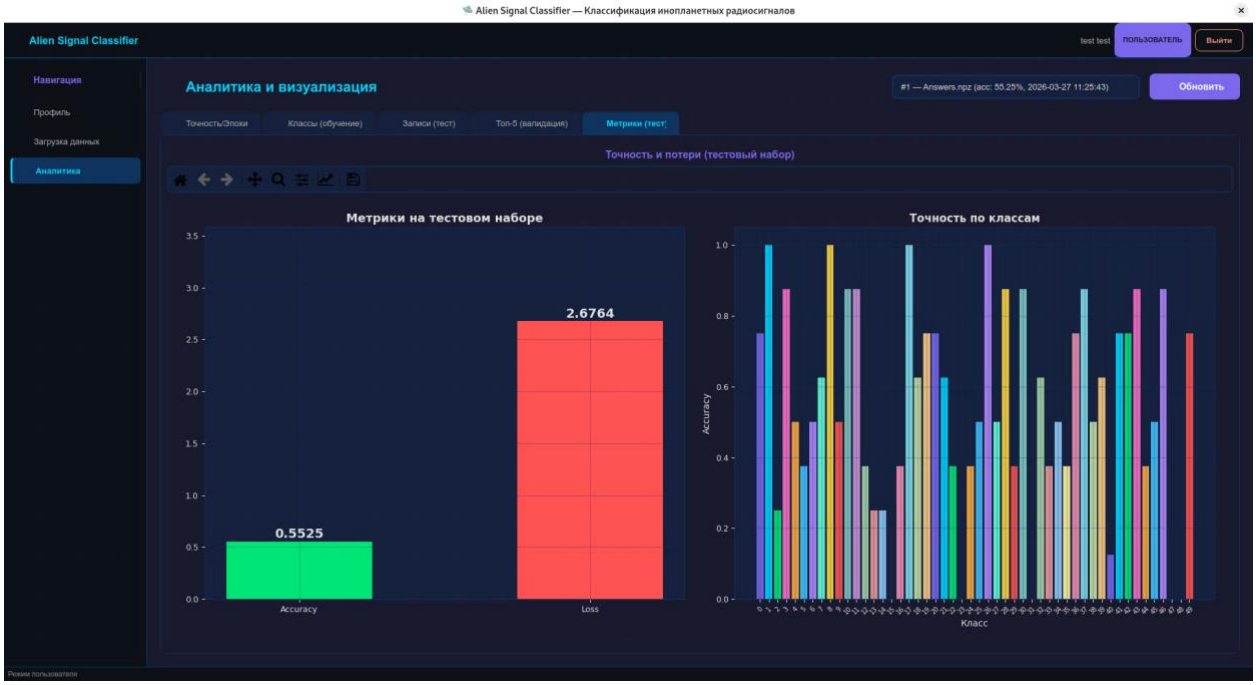
**Пользовательский интерфейс (накопительно)**

| **Уровень** | **Критерии** | **Максимальный балл 50** | **Комментарий** |
|---|---|---|---|
| 1 | Интерфейс пользователя отсутствует, в том числе реализован в командной строке с неполным соответствием ТЗ к пользовательскому интерфейсу | 0 | |
| 2 | Интерфейс реализован в командной строке (для инициализации, демонстрации карт и данных на них используются отдельные исполняемые файлы) Присутствуют все возможности (кроме масштабирования) | 10 | |
| 3 | Разработан графический интерфейс на выбранной платформе. Но нет возможности масштабировать графики и | 30 | |

| | | | |
|---|---|---|---|
| | диаграммы. | | |
| 4 | Разработан графический интерфейс на выбранной платформе. Все требования соблюдены. | 50 | |

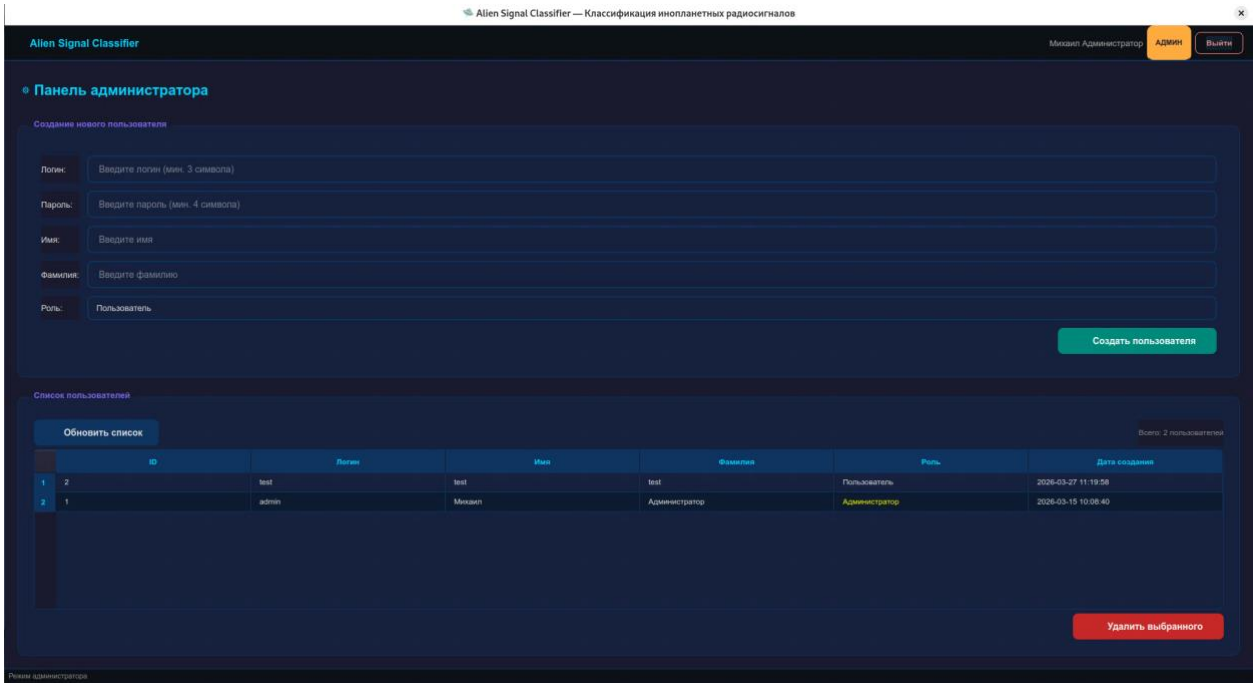
# МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
## Заключительный этап
## Практика «Информационные Технологии»
## Командно-практическое задание

## Пример решения



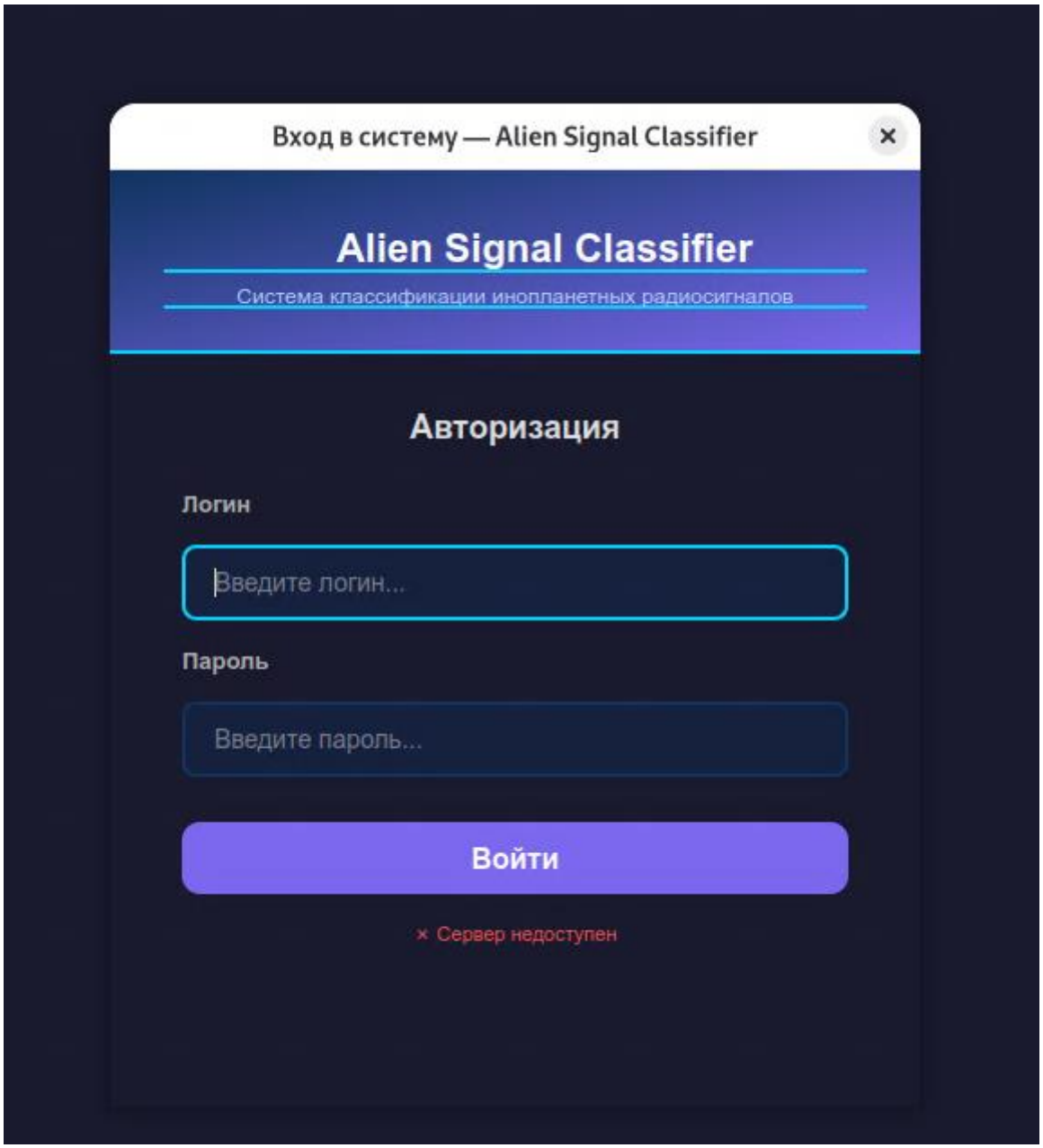

**Москва**
**2025/2026 уч. г.**

# МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
## Заключительный этап
## Практика «Информационные Технологии»
## Командно-практическое задание



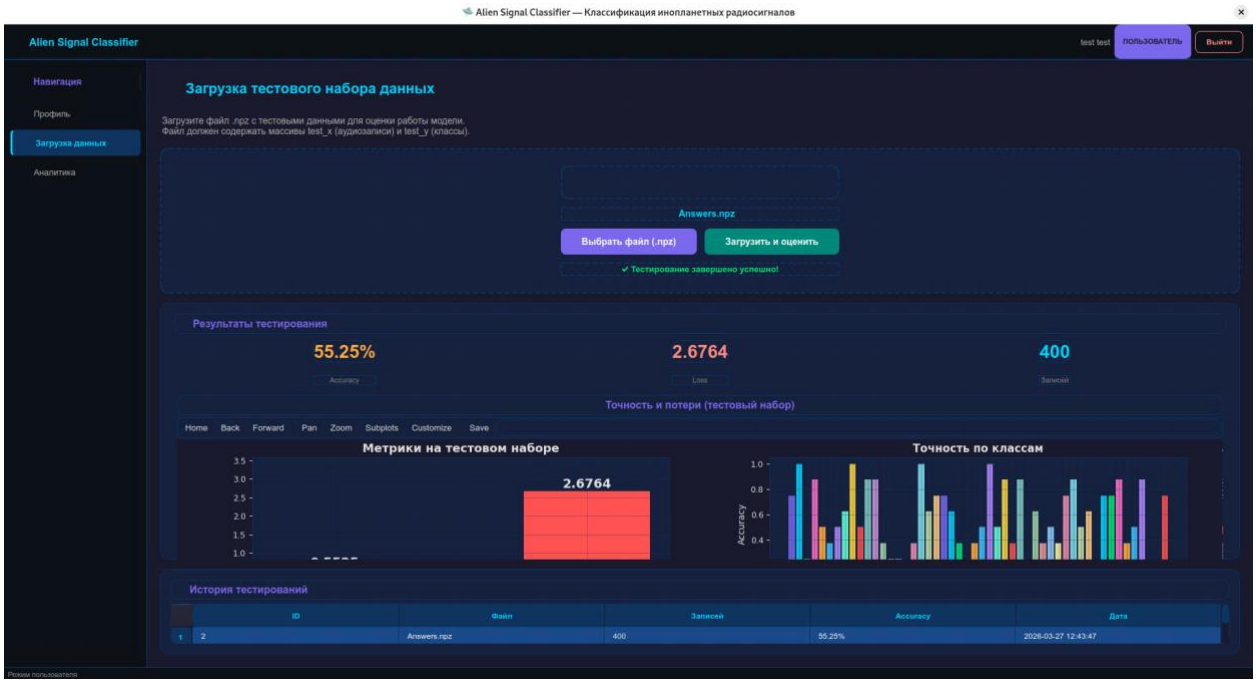

**Москва**
**2025/2026 уч. г.**

**МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ**
**Заключительный этап**
**Практика «Информационные Технологии»**
**Командно-практическое задание**




**Москва**
**2025/2026 уч. г.**

**МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ**
**Заключительный этап**
**Практика «Информационные Технологии»**
**Командно-практическое задание**




**Серверная часть**

database.py

```
import sqlite3
import os

DB_NAME = "back//database//database.db"
SCHEMA_FILE = "back//database//database.sql"


class Database:

    def __init__(self, db_name=DB_NAME):
        self.db_name = db_name
        self.conn = None

    def connect(self):
        self.conn = sqlite3.connect(self.db_name)
        self.conn.execute("PRAGMA foreign_keys = ON")
        self.conn.row_factory = sqlite3.Row
        return self.conn
```

```
    def close(self):
        if self.conn:
            self.conn.close()
            self.conn = None

    # Инициализация БД из SQL-файла
    def init_db(self):
        conn = self.connect()
        cursor = conn.cursor()

        with open(SCHEMA_FILE, 'r', encoding='utf-8') as file:
            sql_schema = file.read()

        cursor.executescript(sql_schema)
        conn.commit()
        self.close()
        print(f"[DB] База данных '{self.db_name}'
инициализирована из '{SCHEMA_FILE}'")

    # Пользователи
    def authenticate(self, login: str, password: str) -> dict |
None:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute(
            "SELECT id, role, first_name, last_name FROM users
WHERE login=? AND password=?",
            (login, password)
        )
        row = cursor.fetchone()
        self.close()

        if row:
            self.log_session(row["id"])
            return {
                "id": row["id"],
                "role": row["role"],
```

```
                "first_name": row["first_name"],
                "last_name": row["last_name"]
            }
        return None

    def create_user(self, login: str, password: str, role: str,
                    first_name: str, last_name: str) -> bool:
        try:
            conn = self.connect()
            cursor = conn.cursor()
            cursor.execute(
                "INSERT INTO users (login, password, role,
first_name, last_name) VALUES (?, ?, ?, ?, ?)",
                (login, password, role, first_name, last_name)
            )
            conn.commit()
            self.close()
            return True
        except sqlite3.IntegrityError:
            self.close()
            return False

    def get_all_users(self) -> list:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute("SELECT id, login, role, first_name,
last_name, created_at FROM users")
        rows = cursor.fetchall()
        self.close()
        return [dict(row) for row in rows]

    # Сессии
    def log_session(self, user_id: int):
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute("INSERT INTO sessions (user_id) VALUES
(?)", (user_id,))
```

```
        conn.commit()
        self.close()

    def get_sessions(self, user_id: int = None) -> list:
        conn = self.connect()
        cursor = conn.cursor()
        if user_id:
            cursor.execute(
                "SELECT s.id, u.login, u.first_name,
u.last_name, s.login_time "
                "FROM sessions s JOIN users u ON s.user_id =
u.id "
                "WHERE s.user_id = ? ORDER BY s.login_time
DESC",
                (user_id,)
            )
        else:
            cursor.execute(
                "SELECT s.id, u.login, u.first_name,
u.last_name, s.login_time "
                "FROM sessions s JOIN users u ON s.user_id =
u.id "
                "ORDER BY s.login_time DESC"
            )
        rows = cursor.fetchall()
        self.close()
        return [dict(row) for row in rows]

    # История обучения
    def save_training_history(self, history: list[dict]):
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute("DELETE FROM training_history")
        for entry in history:
            cursor.execute(
                "INSERT INTO training_history (epoch,
train_accuracy, val_accuracy, train_loss, val_loss) "
```

```
                "VALUES (?, ?, ?, ?, ?)",
                (entry["epoch"], entry["train_accuracy"],
entry["val_accuracy"],
                 entry["train_loss"], entry["val_loss"])
            )
        conn.commit()
        self.close()

    def get_training_history(self) -> list:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM training_history ORDER BY
epoch")
        rows = cursor.fetchall()
        self.close()
        return [dict(row) for row in rows]

    # Распределение классов
    def save_class_distribution(self, dataset_type: str,
distribution: dict):
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute("DELETE FROM class_distribution WHERE
dataset_type = ?", (dataset_type,))
        for class_id, count in distribution.items():
            cursor.execute(
                "INSERT INTO class_distribution (dataset_type,
class_id, record_count) VALUES (?, ?, ?)",
                (dataset_type, int(class_id), int(count))
            )
        conn.commit()
        self.close()

    def get_class_distribution(self, dataset_type: str) -> list:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute(
```

```
            "SELECT class_id, record_count FROM
class_distribution "
            "WHERE dataset_type = ? ORDER BY class_id",
            (dataset_type,)
        )
        rows = cursor.fetchall()
        self.close()
        return [dict(row) for row in rows]

    def get_top_classes(self, dataset_type: str, top_n: int = 5)
-> list:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute(
            "SELECT class_id, record_count FROM
class_distribution "
            "WHERE dataset_type = ? ORDER BY record_count DESC
LIMIT ?",
            (dataset_type, top_n)
        )
        rows = cursor.fetchall()
        self.close()
        return [dict(row) for row in rows]

    # Датасеты и предсказания
    def save_dataset_info(self, user_id: int, file_name: str,
file_path: str,
                          total_records: int, accuracy: float,
loss: float) -> int:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO datasets (user_id, file_name,
file_path, total_records, overall_accuracy, overall_loss) "
            "VALUES (?, ?, ?, ?, ?, ?)",
            (user_id, file_name, file_path, total_records,
accuracy, loss)
```

```
        )
        dataset_id = cursor.lastrowid
        conn.commit()
        self.close()
        return dataset_id

    def save_predictions(self, dataset_id: int, predictions:
list[dict]):
        conn = self.connect()
        cursor = conn.cursor()
        for pred in predictions:
            is_correct = 1 if pred["true_class"] ==
pred["predicted_class"] else 0
            cursor.execute(
                "INSERT INTO test_predictions "
                "(dataset_id, record_index, true_class,
predicted_class, confidence, is_correct) "
                "VALUES (?, ?, ?, ?, ?, ?)",
                (dataset_id, pred["record_index"],
pred["true_class"],
                 pred["predicted_class"], pred["confidence"],
is_correct)
            )
        conn.commit()
        self.close()

    def get_predictions(self, dataset_id: int) -> list:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute(
            "SELECT * FROM test_predictions WHERE dataset_id = ?
ORDER BY record_index",
            (dataset_id,)
        )
        rows = cursor.fetchall()
        self.close()
        return [dict(row) for row in rows]
```

```
    def get_latest_dataset(self) -> dict | None:
        conn = self.connect()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM datasets ORDER BY
uploaded_at DESC LIMIT 1")
        row = cursor.fetchone()
        self.close()
        return dict(row) if row else None

    # Дамп БД
    def dump_to_sql(self, output_file: str =
"back//database//dump.sql"):
        conn = self.connect()
        with open(output_file, 'w', encoding='utf-8') as f:
            for line in conn.iterdump():
                f.write(f"{line}\n")
        self.close()
        print(f"[DB] Дамп сохранён в '{output_file}'")


if __name__ == "__main__":
    db = Database()
    db.init_db()

    success = db.create_user("user1", "pass123", "user", "Иван",
"Петров")
    print(f"Создание пользователя: {'OK' if success else 'Уже
существует'}")

    user = db.authenticate("admin", "admin")
    print(f"Авторизация admin: {user}")

    user = db.authenticate("user1", "pass123")
    print(f"Авторизация user1: {user}")

    user = db.authenticate("hacker", "12345")
```

```
    print(f"Авторизация hacker: {user}")

    fake_history = []
    for e in range(1, 21):
        fake_history.append({
            "epoch": e,
            "train_accuracy": min(0.3 + e * 0.035, 0.99),
            "val_accuracy": min(0.25 + e * 0.03, 0.95),
            "train_loss": max(2.0 - e * 0.08, 0.1),
            "val_loss": max(2.2 - e * 0.07, 0.2),
        })
    db.save_training_history(fake_history)
    print(f"История обучения: {len(fake_history)} эпох")

    train_dist = {i: int(50 + 30 * (i % 3)) for i in range(10)}
    db.save_class_distribution("train", train_dist)

    valid_dist = {i: int(20 + 10 * (i % 4)) for i in range(10)}
    db.save_class_distribution("valid", valid_dist)

    top5 = db.get_top_classes("valid", top_n=5)
    print(f"Топ-5 классов valid: {top5}")

    print(f"Все пользователи: {db.get_all_users()}")
    print(f"Сессии: {db.get_sessions()}")

    db.dump_to_sql("dump.sql")
    print("[OK] Все тесты пройдены!")
```

routes.py

```
import os
import uuid
from flask import Blueprint, request, jsonify, g

from config import Config
from auth import (
    login_required, admin_required, user_required,
```

```
    login_user, logout_user
)
import database as db
import model_service
import analytics

api = Blueprint('api', __name__, url_prefix='/api')



@api.route('/auth/login', methods=['POST'])
def api_login():
    """Вход в систему"""
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Отсутствуют данные'}), 400

    login = data.get('login', '').strip()
    password = data.get('password', '')

    if not login or not password:
        return jsonify({'error': 'Логин и пароль обязательны'}),
400

    result = login_user(login, password)

    if 'error' in result:
        return jsonify(result), 401

    return jsonify(result), 200


@api.route('/auth/logout', methods=['POST'])
@login_required
def api_logout():
    auth_header = request.headers.get('Authorization', '')
    token = auth_header.split(' ', 1)[1] if
```

```
auth_header.startswith('Bearer ') else ''
    logout_user(token)
    return jsonify({'message': 'Выход выполнен успешно'}), 200


@api.route('/auth/me', methods=['GET'])
@login_required
def api_get_current_user():
    user = g.current_user
    return jsonify({
        'id': user['id'],
        'login': user['login'],
        'first_name': user['first_name'],
        'last_name': user['last_name'],
        'role': user['role'],
        'created_at': user.get('created_at')
    }), 200



@api.route('/admin/users', methods=['POST'])
@admin_required
def api_create_user():
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Отсутствуют данные'}), 400

    login = data.get('login', '').strip()
    password = data.get('password', '')
    first_name = data.get('first_name', '').strip()
    last_name = data.get('last_name', '').strip()
    role = data.get('role', 'user')

    errors = []
    if not login:
        errors.append('Логин обязателен')
    if not password:
```

```
        errors.append('Пароль обязателен')
    if not first_name:
        errors.append('Имя обязательно')
    if not last_name:
        errors.append('Фамилия обязательна')
    if len(login) < 3:
        errors.append('Логин должен содержать минимум 3
символа')
    if len(password) < 4:
        errors.append('Пароль должен содержать минимум 4
символа')
    if role not in ('user', 'admin'):
        errors.append('Роль должна быть "user" или "admin"')

    if errors:
        return jsonify({'error': '; '.join(errors)}), 400

    user = db.create_user(
        login=login,
        password=password,
        first_name=first_name,
        last_name=last_name,
        role=role,
        created_by=g.current_user['id']
    )

    if user is None:
        return jsonify({'error': f'Пользователь с логином
"{login}" уже существует'}), 409

    db.log_action(
        g.current_user['id'],
        'create_user',
        f'Создан пользователь: {login} ({first_name}
{last_name}), роль: {role}'
    )
```

```
    return jsonify({
        'message': 'Пользователь успешно создан',
        'user': user
    }), 201


@api.route('/admin/users', methods=['GET'])
@admin_required
def api_get_users():
    users = db.get_all_users()
    return jsonify({'users': users}), 200


@api.route('/admin/users/<int:user_id>', methods=['GET'])
@admin_required
def api_get_user(user_id):
    user = db.get_user_by_id(user_id)
    if user is None:
        return jsonify({'error': 'Пользователь не найден'}), 404
    user.pop('password_hash', None)
    return jsonify({'user': user}), 200


@api.route('/admin/users/<int:user_id>', methods=['DELETE'])
@admin_required
def api_delete_user(user_id):
    """Удаление пользователя (мягкое)"""
    if user_id == g.current_user['id']:
        return jsonify({'error': 'Нельзя удалить самого себя'}),
400

    success = db.delete_user(user_id)
    if not success:
        return jsonify({'error': 'Пользователь не найден'}), 404

    db.invalidate_user_sessions(user_id)
    db.log_action(g.current_user['id'], 'delete_user', f'Удалён
```

```
пользователь ID={user_id}')

    return jsonify({'message': 'Пользователь удалён'}), 200


@api.route('/admin/stats', methods=['GET'])
@admin_required
def api_get_stats():
    """Статистика системы"""
    stats = db.get_db_stats()
    return jsonify(stats), 200



@api.route('/test/upload', methods=['POST'])
@login_required
def api_upload_test_data():
    if 'file' not in request.files:
        return jsonify({'error': 'Файл не загружен'}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({'error': 'Файл не выбран'}), 400

    if not file.filename.lower().endswith('.npz'):
        return jsonify({'error': 'Допускаются только файлы
формата .npz'}), 400

    os.makedirs(Config.UPLOAD_FOLDER, exist_ok=True)
    unique_filename = f"{uuid.uuid4().hex}_{file.filename}"
    file_path = os.path.join(Config.UPLOAD_FOLDER,
unique_filename)
    file.save(file_path)

    try:
        result = model_service.evaluate_test_dataset(file_path)
```

```
        db_result = db.save_test_result(
            user_id=g.current_user['id'],
            filename=file.filename,
            total_samples=result['total_samples'],
            accuracy=result['accuracy'],
            loss=result['loss'],
            predictions=result['predicted_classes'],
            true_labels=result['true_classes'],
            per_class_accuracy=result['per_class_accuracy']
        )

        db.log_action(
            g.current_user['id'],
            'upload_test',
            f'Загружен тестовый набор: {file.filename}, '
            f'accuracy={result["accuracy"]:.4f},
loss={result["loss"]:.4f}'
        )

        return jsonify({
            'message': 'Тестовый набор загружен и обработан',
            'result_id': db_result['id'],
            'accuracy': result['accuracy'],
            'loss': result['loss'],
            'total_samples': result['total_samples'],
            'correct_predictions':
result['correct_predictions'],
            'per_class_accuracy': result['per_class_accuracy'],
            'per_sample_accuracy':
result.get('per_sample_accuracy', [])
        }), 200

    except Exception as e:
        return jsonify({'error': f'Ошибка обработки файла:
{str(e)}'}), 500
    finally:
```

```
        if os.path.exists(file_path):
            os.remove(file_path)


@api.route('/test/results', methods=['GET'])
@login_required
def api_get_test_results():
    results = db.get_test_results_by_user(g.current_user['id'])
    summary_results = []
    for r in results:
        summary_results.append({
            'id': r['id'],
            'filename': r['filename'],
            'total_samples': r['total_samples'],
            'accuracy': r['accuracy'],
            'loss': r['loss'],
            'created_at': r['created_at']
        })

    return jsonify({'results': summary_results}), 200


@api.route('/test/results/<int:result_id>', methods=['GET'])
@login_required
def api_get_test_result_detail(result_id):
    result = db.get_test_result_by_id(result_id)

    if result is None:
        return jsonify({'error': 'Результат не найден'}), 404

    if result['user_id'] != g.current_user['id'] and
g.current_user['role'] != 'admin':
        return jsonify({'error': 'Доступ запрещён'}), 403

    return jsonify({'result': result}), 200

@api.route('/analytics/accuracy-epochs', methods=['GET'])
```

```
@login_required
def api_accuracy_vs_epochs():
    data = analytics.get_accuracy_vs_epochs()
    return jsonify(data), 200


@api.route('/analytics/class-distribution', methods=['GET'])
@login_required
def api_class_distribution():
    data = analytics.get_class_distribution()
    return jsonify(data), 200


@api.route('/analytics/test-per-sample/<int:result_id>',
methods=['GET'])
@login_required
def api_test_per_sample(result_id):
    test_result = db.get_test_result_by_id(result_id)
    if test_result is None:
        return jsonify({'error': 'Результат не найден'}), 404

    if test_result['user_id'] != g.current_user['id'] and
g.current_user['role'] != 'admin':
        return jsonify({'error': 'Доступ запрещён'}), 403

    data = analytics.get_test_per_sample_accuracy(test_result)
    return jsonify(data), 200


@api.route('/analytics/top5-validation', methods=['GET'])
@login_required
def api_top5_validation():
    data = analytics.get_top5_validation_classes()
    return jsonify(data), 200


@api.route('/analytics/full', methods=['GET'])
```

```
@login_required
def api_full_analytics():
    result_id = request.args.get('result_id', type=int)
    data = analytics.get_full_analytics(result_id)
    return jsonify(data), 200


@api.route('/analytics/test-summary/<int:result_id>',
methods=['GET'])
@login_required
def api_test_summary(result_id):
    test_result = db.get_test_result_by_id(result_id)
    if test_result is None:
        return jsonify({'error': 'Результат не найден'}), 404

    if test_result['user_id'] != g.current_user['id'] and
g.current_user['role'] != 'admin':
        return jsonify({'error': 'Доступ запрещён'}), 403

    return jsonify({
        'accuracy': test_result['accuracy'],
        'loss': test_result['loss'],
        'total_samples': test_result['total_samples'],
        'filename': test_result['filename'],
        'created_at': test_result['created_at'],
        'per_class_accuracy':
test_result.get('per_class_accuracy', {})
    }), 200

@api.route('/model/info', methods=['GET'])
@login_required
def api_model_info():
    config = model_service.get_model_config()

    try:
        summary = model_service.get_model_summary()
    except Exception:
```

```
        summary = 'Модель не загружена'

    return jsonify({
        'config': config,
        'summary': summary
    }), 200


@api.route('/model/training-history', methods=['GET'])
@login_required
def api_training_history():
    history = model_service.get_training_history()
    return jsonify(history), 200


@api.route('/dataset/info', methods=['GET'])
@login_required
def api_dataset_info():
    info = model_service.get_training_dataset_info()
    return jsonify(info), 200



@api.route('/health', methods=['GET'])
def api_health():
    checks = {
        'server': 'ok',
        'database': 'ok' if db.check_db_exists() else 'error',
        'model': 'ok' if os.path.exists(Config.MODEL_PATH) else
'not_found',
        'dataset': 'ok' if os.path.exists(Config.DATA_PATH) else
'not_found'
    }

    status = 200 if all(v == 'ok' for v in checks.values()) else
503
    return jsonify(checks), status
```

model_service.py

```
import numpy as np
import json
import os
import io
from typing import Dict, List, Optional, Tuple

from config import Config

_model = None
_label_mapping = None
_model_config = None
_training_history = None


def _load_tensorflow():
    import tensorflow as tf
    return tf


def get_model():
    global _model
    if _model is None:
        if not os.path.exists(Config.MODEL_PATH):
            raise FileNotFoundError(f"Модель не найдена:
{Config.MODEL_PATH}")
        tf = _load_tensorflow()
        _model = tf.keras.models.load_model(Config.MODEL_PATH)
    return _model


def get_label_mapping() -> Dict:
    global _label_mapping
    if _label_mapping is None:
        if os.path.exists(Config.LABEL_MAPPING_PATH):
            with open(Config.LABEL_MAPPING_PATH, 'r') as f:
```

```
                _label_mapping = json.load(f)
        else:
            _label_mapping = {}
    return _label_mapping


def get_model_config() -> Dict:
    global _model_config
    if _model_config is None:
        if os.path.exists(Config.MODEL_CONFIG_PATH):
            with open(Config.MODEL_CONFIG_PATH, 'r') as f:
                _model_config = json.load(f)
        else:
            _model_config = {}
    return _model_config


def get_training_history() -> Dict:
    global _training_history
    if _training_history is None:
        if os.path.exists(Config.TRAINING_HISTORY_PATH):
            with open(Config.TRAINING_HISTORY_PATH, 'r') as f:
                _training_history = json.load(f)
        else:
            _training_history = {}
    return _training_history


def preprocess_audio(audio_data: np.ndarray) -> np.ndarray:
    audio = np.array(audio_data, dtype=np.float32)
    if len(audio.shape) == 1:
        audio = audio.reshape(-1, 1)
    max_val = np.max(np.abs(audio))
    if max_val > 0:
        audio = audio / max_val
    return audio
```

```
def predict_single(audio_data: np.ndarray) -> Dict:
    model = get_model()
    processed = preprocess_audio(audio_data)
    processed = np.expand_dims(processed, axis=0)  # batch dim

    predictions = model.predict(processed, verbose=0)
    predicted_class = int(np.argmax(predictions[0]))
    confidence = float(predictions[0][predicted_class])

    return {
        'predicted_class': predicted_class,
        'confidence': confidence,
        'probabilities': predictions[0].tolist()
    }


def evaluate_test_dataset(file_path: str) -> Dict:
    tf = _load_tensorflow()
    model = get_model()
    config = get_model_config()
    num_classes = config.get('num_classes', None)

    data = np.load(file_path, allow_pickle=True)

    test_x = data['test_x']
    test_y = data['test_y']

    X_test = np.array([preprocess_audio(x) for x in test_x])

    test_y_numeric = np.array(test_y, dtype=np.int32)

    if num_classes is None:
        num_classes = int(np.max(test_y_numeric)) + 1

    y_test = tf.keras.utils.to_categorical(test_y_numeric,
num_classes)
```

```
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)

    predictions_prob = model.predict(X_test, verbose=0)
    predicted_classes = np.argmax(predictions_prob,
axis=1).tolist()
    true_classes = test_y_numeric.tolist()

    confidences = [
        float(predictions_prob[i][predicted_classes[i]])
        for i in range(len(predicted_classes))
    ]

    per_class_accuracy = {}
    unique_classes = np.unique(test_y_numeric)
    for cls in unique_classes:
        cls = int(cls)
        mask = test_y_numeric == cls
        if np.sum(mask) > 0:
            cls_predictions = np.argmax(predictions_prob[mask],
axis=1)
            cls_accuracy = float(np.mean(cls_predictions ==
cls))
            per_class_accuracy[str(cls)] = {
                'accuracy': round(cls_accuracy, 4),
                'total': int(np.sum(mask)),
                'correct': int(np.sum(cls_predictions == cls))
            }

    per_sample_accuracy = []
    for i in range(len(predicted_classes)):
        per_sample_accuracy.append({
            'index': i,
            'true_class': true_classes[i],
            'predicted_class': predicted_classes[i],
            'confidence': round(confidences[i], 4),
            'correct': predicted_classes[i] == true_classes[i]
```

```
        })

    return {
        'accuracy': round(float(accuracy), 4),
        'loss': round(float(loss), 4),
        'total_samples': len(test_x),
        'correct_predictions': int(sum(
            1 for p, t in zip(predicted_classes, true_classes)
if p == t
        )),
        'predicted_classes': predicted_classes,
        'true_classes': true_classes,
        'confidences': confidences,
        'per_class_accuracy': per_class_accuracy,
        'per_sample_accuracy': per_sample_accuracy
    }


def get_training_dataset_info() -> Dict:
    if not os.path.exists(Config.DATA_PATH):
        return {'error': 'Датасет не найден'}

    data = np.load(Config.DATA_PATH, allow_pickle=True)

    result = {}

    if 'train_x' in data and 'train_y' in data:
        train_y = data['train_y']

        unique_labels = sorted(np.unique(train_y))
        label_map = {label: idx for idx, label in
enumerate(unique_labels)}
        train_y_numeric = np.array([label_map[l] for l in
train_y])

        class_distribution = {}
        for cls in range(len(unique_labels)):
```

```
            count = int(np.sum(train_y_numeric == cls))
            class_distribution[str(cls)] = count

        result['train'] = {
            'total_samples': len(data['train_x']),
            'num_classes': len(unique_labels),
            'class_distribution': class_distribution,
            'sample_shape': list(data['train_x'][0].shape) if
len(data['train_x']) > 0 else []
        }

    if 'valid_x' in data and 'valid_y' in data:
        valid_y = data['valid_y']

        unique_valid_labels = sorted(np.unique(valid_y))
        valid_label_map = {label: idx for idx, label in
enumerate(unique_valid_labels)}
        valid_y_numeric = np.array([valid_label_map[l] for l in
valid_y])

        valid_class_distribution = {}
        for cls in np.unique(valid_y_numeric):
            cls = int(cls)
            count = int(np.sum(valid_y_numeric == cls))
            valid_class_distribution[str(cls)] = count
        sorted_classes = sorted(
            valid_class_distribution.items(),
            key=lambda x: x[1],
            reverse=True
        )
        top5 = dict(sorted_classes[:5])

        result['valid'] = {
            'total_samples': len(data['valid_x']),
            'num_classes': len(unique_valid_labels),
            'class_distribution': valid_class_distribution,
            'top5_classes': top5
```

```
        }

    return result


def get_model_summary() -> str:
    model = get_model()
    stream = io.StringIO()
    model.summary(print_fn=lambda x: stream.write(x + '\n'))
    return stream.getvalue()
```

auth.py

```
import jwt
import datetime
import uuid
from functools import wraps
from flask import request, jsonify, g

from config import Config
import database as db


def generate_token(user_id: int, role: str) -> str:
    payload = {
        'user_id': user_id,
        'role': role,
        'jti': str(uuid.uuid4()),  # Уникальный ID токена
        'iat': datetime.datetime.utcnow(),
        'exp': datetime.datetime.utcnow() + datetime.timedelta(
            hours=Config.JWT_EXPIRATION_HOURS
        )
    }
    token = jwt.encode(payload, Config.SECRET_KEY,
algorithm='HS256')
    return token
```

```
def decode_token(token: str) -> dict:
    try:
        payload = jwt.decode(token, Config.SECRET_KEY,
algorithms=['HS256'])
        return payload
    except jwt.ExpiredSignatureError:
        return {'error': 'Токен истёк'}
    except jwt.InvalidTokenError:
        return {'error': 'Невалидный токен'}


def login_user(login: str, password: str) -> dict:
    user = db.authenticate_user(login, password)
    if user is None:
        return {'error': 'Неверный логин или пароль'}

    token = generate_token(user['id'], user['role'])

    db.create_session(
        user_id=user['id'],
        token=token,
        expiration_hours=Config.JWT_EXPIRATION_HOURS
    )

    db.log_action(user['id'], 'login', f'Вход в систему:
{login}')

    return {
        'token': token,
        'user': {
            'id': user['id'],
            'login': user['login'],
            'first_name': user['first_name'],
            'last_name': user['last_name'],
            'role': user['role']
        }
    }
```

```
def logout_user(token: str) -> bool:
    payload = decode_token(token)
    if 'error' not in payload:
        db.log_action(payload.get('user_id'), 'logout', 'Выход
из системы')
    db.invalidate_session(token)
    return True


def get_current_user_from_token():
    auth_header = request.headers.get('Authorization', '')

    if not auth_header.startswith('Bearer '):
        return None

    token = auth_header.split(' ', 1)[1]
    payload = decode_token(token)

    if 'error' in payload:
        return None

    session = db.get_session_by_token(token)
    if session is None:
        return None

    user = db.get_user_by_id(payload['user_id'])
    return user



def login_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        user = get_current_user_from_token()
        if user is None:
```

```
            return jsonify({'error': 'Требуется авторизация'}),
401
        g.current_user = user
        return f(*args, **kwargs)
    return decorated


def admin_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        user = get_current_user_from_token()
        if user is None:
            return jsonify({'error': 'Требуется авторизация'}),
401
        if user['role'] != 'admin':
            return jsonify({'error': 'Требуются права
администратора'}), 403
        g.current_user = user
        return f(*args, **kwargs)
    return decorated


def user_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        user = get_current_user_from_token()
        if user is None:
            return jsonify({'error': 'Требуется авторизация'}),
401
        g.current_user = user
        return f(*args, **kwargs)
    return decorated
```

app.py

```
import os
```

```
import sys

from flask import Flask, jsonify
from flask_cors import CORS

from config import Config
from routes import api
from init_db import init_database
import database as db


def create_app() -> Flask:

    app = Flask(__name__)
    app.config['SECRET_KEY'] = Config.SECRET_KEY
    app.config['MAX_CONTENT_LENGTH'] = Config.MAX_CONTENT_LENGTH

    CORS(app, resources={
        r"/api/*": {
            "origins": "*",
            "methods": ["GET", "POST", "PUT", "DELETE",
"OPTIONS"],
            "allow_headers": ["Content-Type", "Authorization"]
        }
    })

    app.register_blueprint(api)

    os.makedirs(Config.UPLOAD_FOLDER, exist_ok=True)
    os.makedirs(os.path.dirname(Config.DB_PATH), exist_ok=True)
    if not db.check_db_exists():
        print("БД не найдена, инициализация...")
        init_database()

    @app.errorhandler(404)
    def not_found(error):
        return jsonify({'error': 'Ресурс не найден'}), 404
```

```
    @app.errorhandler(405)
    def method_not_allowed(error):
        return jsonify({'error': 'Метод не разрешён'}), 405

    @app.errorhandler(413)
    def file_too_large(error):
        return jsonify({'error': 'Файл слишком большой (макс.
500 MB)'}), 413

    @app.errorhandler(500)
    def internal_error(error):
        return jsonify({'error': 'Внутренняя ошибка сервера'}),
500

    @app.route('/')
    def index():
        return jsonify({
            'name': 'Alien Signal Classifier API',
            'version': '1.0',
            'description': 'API для классификации инопланетных
радиосигналов',
            'endpoints': {
                'auth': {
                    'POST /api/auth/login': 'Вход в систему',
                    'POST /api/auth/logout': 'Выход из системы',
                    'GET /api/auth/me': 'Информация о текущем
пользователе'
                },
                'admin': {
                    'POST /api/admin/users': 'Создание
пользователя',
                    'GET /api/admin/users': 'Список
пользователей',
                    'GET /api/admin/users/<id>': 'Информация о
пользователе',
                    'DELETE /api/admin/users/<id>': 'Удаление
```

```
пользователя',
                    'GET /api/admin/stats': 'Статистика системы'
                },
                'test': {
                    'POST /api/test/upload': 'Загрузка тестового
набора данных',
                    'GET /api/test/results': 'Список результатов
тестирования',
                    'GET /api/test/results/<id>': 'Детали
результата'
                },
                'analytics': {
                    'GET /api/analytics/accuracy-epochs':
'Точность от эпох',
                    'GET /api/analytics/class-distribution':
'Распределение классов',
                    'GET /api/analytics/test-per-sample/<id>':
'Точность по записям',
                    'GET /api/analytics/top5-validation': 'Топ-5
классов валидации',
                    'GET /api/analytics/full': 'Полная
аналитика',
                    'GET /api/analytics/test-summary/<id>':
'Сводка теста'
                },
                'model': {
                    'GET /api/model/info': 'Информация о
модели',
                    'GET /api/model/training-history': 'История
обучения'
                },
                'other': {
                    'GET /api/dataset/info': 'Информация о
датасете',
                    'GET /api/health': 'Состояние сервера'
                }
            }
```

```
        }), 200

    return app


if __name__ == '__main__':
    app = create_app()
        app.run(
        host='0.0.0.0',
        port=5000,
        debug=True
    )
```

analytics.py

```
import json
import os
import numpy as np
from typing import Dict, List, Optional

from config import Config
import model_service


def get_accuracy_vs_epochs() -> Dict:
    history = model_service.get_training_history()

    result = {
        'epochs': list(range(1, len(history.get('accuracy', []))
+ 1)),
        'train_accuracy': history.get('accuracy', []),
        'train_loss': history.get('loss', []),
    }

    if 'val_accuracy' in history:
        result['val_accuracy'] = history['val_accuracy']
```

```
    if 'val_loss' in history:
        result['val_loss'] = history['val_loss']

    return result


def get_class_distribution() -> Dict:
    dataset_info = model_service.get_training_dataset_info()

    if 'train' not in dataset_info:
        return {'error': 'Данные обучения не найдены'}

    train_info = dataset_info['train']
    distribution = train_info.get('class_distribution', {})

    sorted_classes = sorted(distribution.items(), key=lambda x:
int(x[0]))

    return {
        'classes': [item[0] for item in sorted_classes],
        'counts': [item[1] for item in sorted_classes],
        'total_samples': train_info.get('total_samples', 0),
        'num_classes': train_info.get('num_classes', 0)
    }


def get_test_per_sample_accuracy(test_result: Dict) -> Dict:
        if not test_result:
        return {'error': 'Нет результатов тестирования'}

    per_sample = test_result.get('per_sample_accuracy', [])

    if not per_sample and 'confidences' in test_result:
        predicted = test_result.get('predicted_classes', [])
        true = test_result.get('true_classes', [])
        confidences = test_result.get('confidences', [])
```

```
        per_sample = []
        for i in range(len(predicted)):
            per_sample.append({
                'index': i,
                'true_class': true[i] if i < len(true) else
None,
                'predicted_class': predicted[i],
                'confidence': confidences[i] if i <
len(confidences) else 0,
                'correct': predicted[i] == true[i] if i <
len(true) else False
            })

    return {
        'samples': per_sample,
        'total': len(per_sample),
        'correct_count': sum(1 for s in per_sample if
s.get('correct', False)),
        'accuracy': test_result.get('accuracy', 0)
    }


def get_top5_validation_classes() -> Dict:
    dataset_info = model_service.get_training_dataset_info()

    if 'valid' not in dataset_info:
        return {'error': 'Валидационные данные не найдены'}

    valid_info = dataset_info['valid']
    distribution = valid_info.get('class_distribution', {})

    sorted_classes = sorted(
        distribution.items(),
        key=lambda x: x[1],
        reverse=True
    )[:5]
```

```
    return {
        'classes': [item[0] for item in sorted_classes],
        'counts': [item[1] for item in sorted_classes],
        'total_valid_samples': valid_info.get('total_samples',
0)
    }


def get_per_class_accuracy_from_test(test_result: Dict) -> Dict:
    if not test_result:
        return {'error': 'Нет результатов тестирования'}

    per_class = test_result.get('per_class_accuracy', {})

    sorted_classes = sorted(per_class.items(), key=lambda x:
int(x[0]))

    return {
        'classes': [item[0] for item in sorted_classes],
        'accuracies': [item[1]['accuracy'] for item in
sorted_classes],
        'totals': [item[1]['total'] for item in sorted_classes],
        'corrects': [item[1]['correct'] for item in
sorted_classes]
    }


def get_full_analytics(test_result_id: Optional[int] = None) ->
Dict:
    from server import database as db

    result = {
        'accuracy_vs_epochs': get_accuracy_vs_epochs(),
        'class_distribution': get_class_distribution(),
        'top5_validation': get_top5_validation_classes(),
    }
```

```
    if test_result_id:
        test_result = db.get_test_result_by_id(test_result_id)
        if test_result:
            result['test_per_sample'] =
get_test_per_sample_accuracy(test_result)
            result['test_per_class'] =
get_per_class_accuracy_from_test(test_result)
            result['test_summary'] = {
                'accuracy': test_result.get('accuracy'),
                'loss': test_result.get('loss'),
                'total_samples':
test_result.get('total_samples'),
                'filename': test_result.get('filename')
            }

    result['model_config'] = model_service.get_model_config()

    return result
```

**Клиентская часть**

styles.py

```
MAIN_STYLE = """
QMainWindow, QDialog, QWidget {
    background-color: #1a1a2e;
    color: #e0e0e0;
    font-family: 'Segoe UI', Arial, sans-serif;
    font-size: 13px;
}

QLabel {
    color: #e0e0e0;
    font-size: 13px;
}

QLabel#title {
```

```
    font-size: 22px;
    font-weight: bold;
    color: #00d4ff;
    padding: 10px 0;
}

QLabel#subtitle {
    font-size: 16px;
    font-weight: bold;
    color: #7b68ee;
    padding: 5px 0;
}

QLabel#info_label {
    font-size: 14px;
    color: #b0b0b0;
    padding: 3px 0;
}

QLabel#success_label {
    color: #00e676;
    font-size: 13px;
    font-weight: bold;
}

QLabel#error_label {
    color: #ff5252;
    font-size: 13px;
    font-weight: bold;
}

QLabel#stat_value {
    font-size: 28px;
    font-weight: bold;
    color: #00d4ff;
}
```

```
QLabel#stat_label {
    font-size: 11px;
    color: #888;
}

QLineEdit {
    background-color: #16213e;
    color: #e0e0e0;
    border: 2px solid #0f3460;
    border-radius: 8px;
    padding: 10px 14px;
    font-size: 14px;
    selection-background-color: #7b68ee;
}

QLineEdit:focus {
    border-color: #00d4ff;
}

QLineEdit:disabled {
    background-color: #0d1b30;
    color: #666;
}

QPushButton {
    background-color: #0f3460;
    color: #e0e0e0;
    border: none;
    border-radius: 8px;
    padding: 10px 24px;
    font-size: 14px;
    font-weight: bold;
    min-height: 20px;
}

QPushButton:hover {
    background-color: #1a4a8a;
```

```
}

QPushButton:pressed {
    background-color: #0a2540;
}

QPushButton:disabled {
    background-color: #1a1a2e;
    color: #555;
}

QPushButton#primary_btn {
    background-color: #7b68ee;
    color: white;
}

QPushButton#primary_btn:hover {
    background-color: #9580ff;
}

QPushButton#primary_btn:pressed {
    background-color: #5a4db8;
}

QPushButton#success_btn {
    background-color: #00897b;
    color: white;
}

QPushButton#success_btn:hover {
    background-color: #00a890;
}

QPushButton#danger_btn {
    background-color: #c62828;
    color: white;
}
```

```
QPushButton#danger_btn:hover {
    background-color: #e53935;
}

QPushButton#logout_btn {
    background-color: transparent;
    color: #ff8a80;
    border: 1px solid #ff8a80;
    padding: 6px 16px;
    font-size: 12px;
}

QPushButton#logout_btn:hover {
    background-color: #ff8a80;
    color: #1a1a2e;
}

QPushButton#nav_btn {
    background-color: transparent;
    color: #b0b0b0;
    border: none;
    border-radius: 6px;
    padding: 10px 16px;
    font-size: 13px;
    font-weight: normal;
    text-align: left;
}

QPushButton#nav_btn:hover {
    background-color: #16213e;
    color: #e0e0e0;
}

QPushButton#nav_btn_active {
    background-color: #0f3460;
    color: #00d4ff;
```

```
    border: none;
    border-radius: 6px;
    padding: 10px 16px;
    font-size: 13px;
    font-weight: bold;
    text-align: left;
    border-left: 3px solid #00d4ff;
}

QComboBox {
    background-color: #16213e;
    color: #e0e0e0;
    border: 2px solid #0f3460;
    border-radius: 8px;
    padding: 8px 12px;
    font-size: 13px;
}

QComboBox:hover {
    border-color: #00d4ff;
}

QComboBox::drop-down {
    border: none;
    width: 30px;
}

QComboBox QAbstractItemView {
    background-color: #16213e;
    color: #e0e0e0;
    selection-background-color: #0f3460;
    border: 1px solid #0f3460;
}

QTableWidget {
    background-color: #16213e;
    color: #e0e0e0;
```

```
    border: 1px solid #0f3460;
    border-radius: 8px;
    gridline-color: #0f3460;
    selection-background-color: #0f3460;
    font-size: 12px;
}

QTableWidget::item {
    padding: 6px 10px;
    border-bottom: 1px solid #0a1e3d;
}

QTableWidget::item:selected {
    background-color: #1a4a8a;
}

QHeaderView::section {
    background-color: #0f3460;
    color: #00d4ff;
    padding: 8px 10px;
    border: none;
    border-right: 1px solid #1a1a2e;
    font-weight: bold;
    font-size: 12px;
}

QScrollArea {
    border: none;
    background-color: transparent;
}

QScrollBar:vertical {
    background-color: #1a1a2e;
    width: 10px;
    border-radius: 5px;
}
```

```
QScrollBar::handle:vertical {
    background-color: #0f3460;
    border-radius: 5px;
    min-height: 30px;
}

QScrollBar::handle:vertical:hover {
    background-color: #1a4a8a;
}

QScrollBar::add-line:vertical, QScrollBar::sub-line:vertical {
    height: 0;
}

QScrollBar:horizontal {
    background-color: #1a1a2e;
    height: 10px;
    border-radius: 5px;
}

QScrollBar::handle:horizontal {
    background-color: #0f3460;
    border-radius: 5px;
    min-width: 30px;
}

QScrollBar::handle:horizontal:hover {
    background-color: #1a4a8a;
}

QScrollBar::add-line:horizontal, QScrollBar::sub-line:horizontal
{
    width: 0;
}

QGroupBox {
    background-color: #16213e;
```

```
    border: 1px solid #0f3460;
    border-radius: 10px;
    margin-top: 10px;
    padding: 15px;
    padding-top: 25px;
    font-size: 13px;
    font-weight: bold;
    color: #7b68ee;
}

QGroupBox::title {
    subcontrol-origin: margin;
    left: 15px;
    padding: 0 8px;
}

QTabWidget::pane {
    background-color: #1a1a2e;
    border: 1px solid #0f3460;
    border-radius: 8px;
}

QTabBar::tab {
    background-color: #16213e;
    color: #b0b0b0;
    padding: 10px 20px;
    margin-right: 2px;
    border-top-left-radius: 8px;
    border-top-right-radius: 8px;
    font-size: 13px;
}

QTabBar::tab:selected {
    background-color: #0f3460;
    color: #00d4ff;
    font-weight: bold;
}
```

```
QTabBar::tab:hover {
    background-color: #1a3a5c;
}

QProgressBar {
    background-color: #16213e;
    border: 1px solid #0f3460;
    border-radius: 8px;
    text-align: center;
    color: #e0e0e0;
    font-size: 12px;
    min-height: 22px;
}

QProgressBar::chunk {
    background-color: #7b68ee;
    border-radius: 7px;
}

QMessageBox {
    background-color: #1a1a2e;
    color: #e0e0e0;
}

QMessageBox QLabel {
    color: #e0e0e0;
    font-size: 13px;
}

QMessageBox QPushButton {
    min-width: 80px;
}

QStatusBar {
    background-color: #0d1117;
    color: #888;
```

```
    font-size: 11px;
    border-top: 1px solid #0f3460;
}

QToolTip {
    background-color: #16213e;
    color: #e0e0e0;
    border: 1px solid #0f3460;
    border-radius: 4px;
    padding: 5px;
    font-size: 12px;
}

QSplitter::handle {
    background-color: #0f3460;
    width: 2px;
}
"""

# Цвета для графиков matplotlib
CHART_COLORS = {
    'background': '#1a1a2e',
    'face': '#16213e',
    'text': '#e0e0e0',
    'grid': '#0f3460',
    'primary': '#7b68ee',
    'secondary': '#00d4ff',
    'success': '#00e676',
    'danger': '#ff5252',
    'warning': '#ffab40',
    'accent1': '#ff6ec7',
    'accent2': '#40c4ff',
    'accent3': '#b388ff',
    'accent4': '#64ffda',
    'accent5': '#ffd740',
    'bar_colors': [
        '#7b68ee', '#00d4ff', '#00e676', '#ff6ec7', '#ffab40',
```

```
        '#40c4ff', '#b388ff', '#64ffda', '#ffd740', '#ff5252',
        '#80cbc4', '#ce93d8', '#a5d6a7', '#ef9a9a', '#90caf9',
        '#fff59d', '#f48fb1', '#80deea', '#c5e1a5', '#ffcc80'
    ]
}
```

main.py

```
import sys
import os
sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from PyQt5.QtWidgets import QApplication
from PyQt5.QtCore import Qt
from client.styles import MAIN_STYLE
from client.widgets.main_window import MainWindow

def main():
    QApplication.setAttribute(Qt.AA_EnableHighDpiScaling, True)
    QApplication.setAttribute(Qt.AA_UseHighDpiPixmaps, True)
    app = QApplication(sys.argv)
    app.setApplicationName("Alien Signal Classifier")
    app.setOrganizationName("AlienLab2226")
    app.setStyleSheet(MAIN_STYLE)
    server_url = "http://localhost:5000"
    if len(sys.argv) > 1:
        server_url = sys.argv[1]
    window = MainWindow(server_url)
    window.show()
    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

api_client.py

```
import requests
from typing import Optional, Dict, Any, Tuple
class ApiClient:
    def __init__(self, base_url: str = "http://localhost:5001"):
        self.base_url = base_url.rstrip('/')
        self.token: Optional[str] = None
        self.current_user: Optional[Dict] = None
        self.session = requests.Session()
        self.timeout = 120  # секунд (модель может долго
обрабатывать)

    def _headers(self) -> Dict[str, str]:
        headers = {"Content-Type": "application/json"}
        if self.token:
            headers["Authorization"] = f"Bearer {self.token}"
        return headers

    def _get(self, endpoint: str, params: Optional[Dict] = None)
-> Tuple[int, Dict]:
        try:
            resp = self.session.get(
                f"{self.base_url}{endpoint}",
                headers=self._headers(),
                params=params,
                timeout=self.timeout
            )
            return resp.status_code, resp.json()
        except requests.ConnectionError:
            return 0, {"error": "Нет соединения с сервером"}
        except requests.Timeout:
            return 0, {"error": "Таймаут соединения"}
        except Exception as e:
            return 0, {"error": str(e)}

    def _post(self, endpoint: str, data: Optional[Dict] = None)
-> Tuple[int, Dict]:
        try:
```

```
            resp = self.session.post(
                f"{self.base_url}{endpoint}",
                headers=self._headers(),
                json=data,
                timeout=self.timeout
            )
            return resp.status_code, resp.json()
        except requests.ConnectionError:
            return 0, {"error": "Нет соединения с сервером"}
        except requests.Timeout:
            return 0, {"error": "Таймаут соединения"}
        except Exception as e:
            return 0, {"error": str(e)}
    def _post_file(self, endpoint: str, file_path: str) ->
Tuple[int, Dict]:
        try:
            headers = {}
            if self.token:
                headers["Authorization"] = f"Bearer
{self.token}"
            with open(file_path, 'rb') as f:
                files = {"file": (file_path.split('/')[-
1].split('\\')[-1], f)}
                resp = self.session.post(
                    f"{self.base_url}{endpoint}",
                    headers=headers,
                    files=files,
                    timeout=self.timeout
                )
            return resp.status_code, resp.json()
        except requests.ConnectionError:
            return 0, {"error": "Нет соединения с сервером"}
        except requests.Timeout:
            return 0, {"error": "Таймаут соединения"}
        except Exception as e:
            return 0, {"error": str(e)}
```

```
    def _delete(self, endpoint: str) -> Tuple[int, Dict]:
        try:
            resp = self.session.delete(
                f"{self.base_url}{endpoint}",
                headers=self._headers(),
                timeout=self.timeout
            )
            return resp.status_code, resp.json()
        except requests.ConnectionError:
            return 0, {"error": "Нет соединения с сервером"}
        except requests.Timeout:
            return 0, {"error": "Таймаут соединения"}
        except Exception as e:
            return 0, {"error": str(e)}

    def login(self, login: str, password: str) -> Tuple[bool,
Dict]:
        status, data = self._post("/api/auth/login", {
            "login": login,
            "password": password
        })
        if status == 200 and "token" in data:
            self.token = data["token"]
            self.current_user = data.get("user")
            return True, data
        return False, data

    def logout(self) -> Tuple[bool, Dict]:
        status, data = self._post("/api/auth/logout")
        self.token = None
        self.current_user = None
        return status == 200, data

    def get_current_user(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/auth/me")
        if status == 200:
            self.current_user = data
```

```
        return status == 200, data

    def is_authenticated(self) -> bool:
        return self.token is not None

    def is_admin(self) -> bool:
        if self.current_user:
            return self.current_user.get("role") == "admin"
        return False

    def create_user(self, login: str, password: str,
                    first_name: str, last_name: str,
                    role: str = "user") -> Tuple[bool, Dict]:
        status, data = self._post("/api/admin/users", {
            "login": login,
            "password": password,
            "first_name": first_name,
            "last_name": last_name,
            "role": role
        })
        return status == 201, data

    def get_users(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/admin/users")
        return status == 200, data

    def get_user(self, user_id: int) -> Tuple[bool, Dict]:
        status, data = self._get(f"/api/admin/users/{user_id}")
        return status == 200, data

    def delete_user(self, user_id: int) -> Tuple[bool, Dict]:
        status, data =
self._delete(f"/api/admin/users/{user_id}")
        return status == 200, data

    def get_admin_stats(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/admin/stats")
```

```
        return status == 200, data

    def upload_test_data(self, file_path: str) -> Tuple[bool,
Dict]:
        status, data = self._post_file("/api/test/upload",
file_path)
        return status == 200, data

    def get_test_results(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/test/results")
        return status == 200, data

    def get_test_result_detail(self, result_id: int) ->
Tuple[bool, Dict]:
        status, data =
self._get(f"/api/test/results/{result_id}")
        return status == 200, data

    def get_accuracy_vs_epochs(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/analytics/accuracy-
epochs")
        return status == 200, data

    def get_class_distribution(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/analytics/class-
distribution")
        return status == 200, data

    def get_test_per_sample(self, result_id: int) -> Tuple[bool,
Dict]:
        status, data = self._get(f"/api/analytics/test-per-
sample/{result_id}")
        return status == 200, data

    def get_top5_validation(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/analytics/top5-
validation")
```

```
        return status == 200, data

    def get_full_analytics(self, result_id: Optional[int] =
None) -> Tuple[bool, Dict]:
        params = {}
        if result_id:
            params["result_id"] = result_id
        status, data = self._get("/api/analytics/full",
params=params)
        return status == 200, data

    def get_test_summary(self, result_id: int) -> Tuple[bool,
Dict]:
        status, data = self._get(f"/api/analytics/test-
summary/{result_id}")
        return status == 200, data

    def get_model_info(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/model/info")
        return status == 200, data

    def get_training_history(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/model/training-history")
        return status == 200, data

    def get_dataset_info(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/dataset/info")
        return status == 200, data

    def health_check(self) -> Tuple[bool, Dict]:
        status, data = self._get("/api/health")
        return status == 200, data
```

**Модуль обучения нейросети**

decode.py

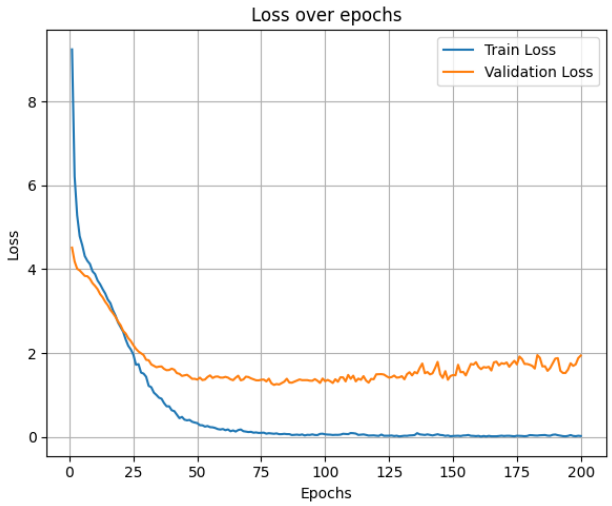
```
import sys
```

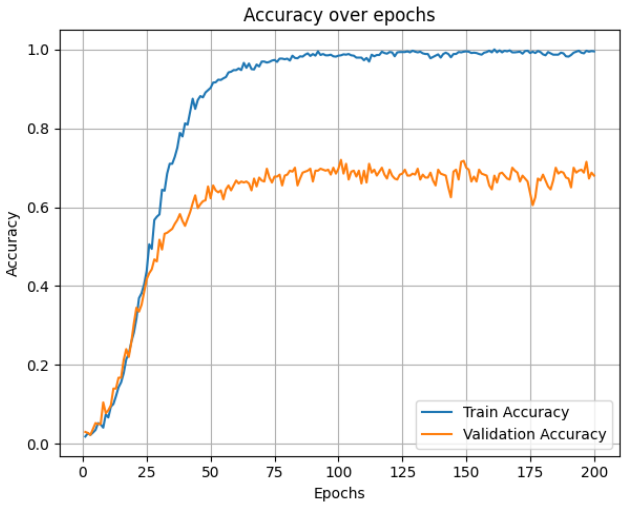
```
import hashlib


def decode(encode_class):
    _hash = encode_class[:32]  # первые два символа - hash
    salt = encode_class[32:]  # после 32 символом "соль",
которая использовалась при хэшировании
    for n in range(1400):
        if hashlib.md5((str(n) + salt).encode()).hexdigest() ==
_hash:
            return n
    return None


def main():
    encode_class = sys.argv[1]
    decode_class = decode(encode_class)
    print(decode_class)


if __name__ == '__main__':
    main()
```




learning.py

```
import os
```

```
os.environ["KERAS_BACKEND"] = "jax"

import keras
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.io.wavfile
from keras import layers
from scipy.signal import resample

keras.utils.set_random_seed(41)

BATCH_SIZE = 32
NUM_CLASSES = 50
EPOCHS = 200
SAMPLE_RATE = 16000
```

```
import numpy as np

data = np.load('Data.npz')


train_x1 = data['train_x']
train_y1=data['train_y']
valid_x1=data['valid_x']
valid_y1=data['valid_y']
```

```
input = layers.Input((None, 1))
spectrograms = [
    layers.STFTSpectrogram(
```

```
            mode="log",
            frame_length=SAMPLE_RATE * frame_size // 1000,
            frame_step=SAMPLE_RATE * 15 // 1000,
            fft_length=2048,
            padding="same",
            expand_dims=True,
        )(input)
        for frame_size in [30, 40, 50]
]

multi_spectrograms = layers.Concatenate(axis=-1)(spectrograms)

img_model = keras.applications.MobileNet(include_top=False,
pooling="max")
output = img_model(multi_spectrograms)

output = layers.Dropout(0.5)(output)
output = layers.Dense(256, activation="relu")(output)
output = layers.Dense(256, activation="relu")(output)
output = layers.Dense(NUM_CLASSES, activation="softmax")(output)
model2d = keras.Model(input, output,
name="model_2d_trainble_stft")

model2d.compile(
    optimizer=keras.optimizers.Adam(1e-4),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)
model2d.summary()




history_model2d = model2d.fit(
    train_x1,
    train_y1,
    batch_size=BATCH_SIZE,
```

```
    validation_data=(valid_x1, valid_y1),
    epochs=EPOCHS,
    callbacks=[
        keras.callbacks.EarlyStopping(
            monitor="val_loss",
            patience=EPOCHS,
            restore_best_weights=True,
        )
    ],
)



import matplotlib.pyplot as plt

train_loss = history_model2d.history['loss']
val_loss = history_model2d.history['val_loss']
train_acc = history_model2d.history.get('accuracy')
val_acc = history_model2d.history.get('val_accuracy')
epochs_range = range(1, len(train_loss) + 1)

plt.figure(figsize=(12, 5))


plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, label='Train Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss over epochs')
plt.legend()
plt.grid(True)


if train_acc and val_acc:
    plt.subplot(1, 2, 2)
```

```
    plt.plot(epochs_range, train_acc, label='Train Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Accuracy over epochs')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

Ссылка на модель