

# Практикум по разработке ML

Анисимов Я.О и ко

ИТМО

07 ноября 2023

№0

№0

- 1 Общее сведения о курсе
- 2 Brief: Разработка ML-сервиса с подсистемой биллинга
- 3 Задача курса
- 4 Основы дизайна API
- 5 Паттерны проектирования stateful API
- 6 Примеры использования cookie и JWT для RESTful API по заказу айтемов

## 7 Пример простого API для регистрации пользователя и работой с айтемом

## О чем курс

- Разработка бекенд сервиса в котором есть ML модельки
- Отработка конкретной бизнес-логики приложения
- Знакомство с технологиями: FastAPI, SQL, Docker
- Общие сведения о Clean Architecture

# Структура курса

- Лекционный блок(ноябрь)
- Серия ментор сессий/семинаров(декабрь)
- Консультации(январь)
- Зачет(январь)

# Темы лекционного блока



# Результат курса

- Работающий сервис

# Команда курса курса

- Анисимов Ян
- др



## Описание проекта

Цель проекта - разработать ML-сервис с подсистемой биллинга, который будет осуществлять предсказания на основе ML-моделей и списывать кредиты с личного счета пользователя за успешное выполнение предсказания. Сервис должен быть надежным и готовым для использования в продакшн-окружении.

# Основные требования

- 1 Возможность загрузки и использования ML-моделей: сервис должен иметь возможность загружать и использовать различные ML-модели для выполнения предсказаний. Входные данные для моделей должны подаваться в сервис с использованием удобного API (Application Programming Interface).
- 2 Биллинговая подсистема: сервис должен поддерживать функциональность биллинга, где пользователь хранит определенное количество кредитов на своем личном счете. При успешном выполнении предсказания, счет пользователя должен быть списан за использованные кредиты.
- 3 Пользовательская система: сервис должен иметь пользовательский интерфейс, позволяющий пользователям регистрироваться, входить в систему и управлять своим личным счетом.

- 4 Мониторинг и аналитика: сервис должен предоставлять возможность

# Технические требования

- 1 Язык программирования: разработка сервиса должна быть выполнена с использованием языка программирования, который наилучшим образом соответствует требованиям проекта (например, Python).
- 2 ML-фреймворк: для загрузки и использования ML-моделей рекомендуется использовать Scikit-learn.
- 3 База данных: для хранения пользовательских данных, моделей и биллинговой информации можно использовать реляционную базу данных (например, PostgreSQL или Sqlite).
- 4 API: сервис должен предоставлять удобное и документированное API для загрузки моделей, выполнения предсказаний и управления пользовательскими данными.
- 5 Инфраструктура: необходимо использовать технологии контейнеризации.

№

# План работы

- 1 Анализ требований: уточнение и детализация требований проекта, создание документации.
- 2 Проектирование архитектуры: разработка общей архитектуры сервиса, определение компонентов и API.
- 3 Разработка ML-функциональности: загрузка и использование ML-моделей, реализация функций предсказания.
- 4 Разработка биллинговой подсистемы: создание механизма учета кредитов и списывания с личного счета пользователя.
- 5 Разработка пользовательской системы: регистрация, аутентификация и управление личным счетом пользователей.
- 6 Внедрение и документация: установка сервиса в продакшн-окружение, создание документации для пользователей и администраторов.

№<sub>0</sub>

# Ожидаемые результаты

- Функционирующий ML-сервис с подсистемой биллинга, способный загружать и использовать ML-модели для выполнения предсказаний.
- Биллинговая система, позволяющая управлять пользовательскими счетами и списывать кредиты за успешное выполнение предсказания.
- Пользовательская система, позволяющая пользователям регистрироваться, входить в систему и управлять своим личным счетом.
- Масштабируемая инфраструктура, способная обрабатывать большое количество запросов и обеспечивать высокую доступность.
- Документация, описывающая работу сервиса, API и рекомендации по развертыванию и использованию.

# Формальное описание задачи

# ML задача курса

TBD

Ассистент

# Протокол HTTP

- Протокол передачи данных, используемый веб-серверами и клиентами.
- Основные методы HTTP:
  - GET: получение данных
  - POST: отправка данных на сервер
  - PUT: обновление данных на сервере
  - DELETE: удаление данных на сервере
- Коды состояния HTTP (status codes):
  - 200 OK: успешный запрос
  - 400 Bad Request: некорректный запрос
  - 404 Not Found: запрошенный ресурс не найден



# Паттерны проектирования API

## ❶ RESTful API:

- Основан на принципах REST.
- Ресурсы представлены в формате URL.
- Использует верблужью нотацию для именования ресурсов.

## ❷ GraphQL API:

- Модернизированный подход к созданию API.
- Клиенты выбирают, какие данные им нужны.
- Единый запрос для получения нескольких ресурсов.

# Инструменты проектирования API

## ❶ Swagger:

- Фреймворк для разработки, проектирования и документирования API.
- Позволяет создавать спецификацию API в формате JSON или YAML.
- Генерирует интерактивную документацию и клиентские библиотеки.

## ❷ API Blueprint:

- Язык для описания API в формате Markdown.
- Позволяет создавать простую и читабельную документацию.
- Поддерживает генерацию кода и автоматическую проверку API.

## ❸ RAML:

- YAML-ориентированный язык описания API.
- Позволяет задавать макет данных и примеры.
- Поддерживает генерацию кода для различных языков.

Дизайн API играет важную роль в успешном взаимодействии между клиентами и серверами. Корректно выбранный протокол HTTP, паттерн проектирования и инструменты, такие как Swagger, помогут создать эффективное и легко

# Принципы проектирования API

## ❶ Единообразие:

- Устанавливайте согласованные стандарты и используйте их повсюду.
- Имена ресурсов, методы HTTP и параметры запросов должны быть последовательными и понятными.

## ❷ Понятность:

- Легко понять, как использовать API и что ожидать в ответе.
- Правильно документируйте API, предоставляя примеры запросов и ответов.

## ❸ Безопасность:

- Используйте соответствующие механизмы аутентификации и авторизации.
- Защитите свои эндпоинты от нежелательного доступа и злоумышленников.

# Типичные ошибки в проектировании API и способы их исправления

## ❶ Нестабильность API:

- Избегайте изменений внутренней реализации, которые приводят к частым изменениям в API.
- Создайте стабильные версии API и поддерживайте их долгое время.

## ❷ Неправильная обработка ошибок:

- Возвращайте адекватные коды состояния и сообщения об ошибках.
- Предлагайте разработчикам способы понять и исправить ошибки.

## ❸ Неподходящая структура данных:

- Определите наиболее подходящую структуру в соответствии с потребностями клиентов.
- Используйте запросы с параметрами, чтобы фильтровать и сортировать данные.

## ❹ Недостаточная документация:

- Создайте полную и понятную документацию для вашего API.
- Обновляйте документацию с каждым изменением API.

# Пример хорошего RESTful API

## GET /items

- Запрос на получение списка всех айтемов.
- Бизнес-значимость: клиенты могут получить полный список доступных айтемов.

## GET /items/{id}

- Запрос на получение конкретного айтема по его идентификатору.
- Бизнес-значимость: клиенты могут получить информацию о конкретном айтеме, используя его идентификатор.

## POST /items

- Запрос на создание нового айтема.
- Бизнес-значимость: клиенты могут добавлять новые айтемы в систему.

 $\mathbb{N}_0$

# Пример плохого RESTful API

## GET /getAllItems

- Запрос на получение списка всех айтемов.
- Бизнес-значимость: в названии эндпоинта повторяется “все”, что является лишним, так как нет другой альтернативы.

## GET /getItemById/{id}

- Запрос на получение конкретного айтема по его идентификатору.
- Бизнес-значимость: параметр “ById” в названии эндпоинта излишен, так как уже понятно, что идентификатор используется.

## POST /addItemToInventory

- Запрос на создание нового айтема в инвентаре.
- Бизнес-значимость: в названии эндпоинта присутствует уточнение о добавлении айтема в инвентарь, что не является необходимым.

# Stateful API

- Сохранение состояния сервера между запросами клиента.
- Использует токены или данные в cookie для идентификации и аутентификации клиента.

# Cookie-based подход

- Идентификатор сессии хранится в cookie.
- Сервер проверяет и обновляет сессию при каждом запросе клиента.

## Пример

- 1 Клиент отправляет запрос на аутентификацию с логином и паролем.
- 2 Сервер проверяет и создает уникальный идентификатор сессии.
- 3 Сервер возвращает идентификатор сессии в виде cookie.
- 4 Клиент отправляет запросы с cookie в каждом последующем запросе.
- 5 Сервер считывает идентификатор сессии из cookie и обрабатывает запрос.

## Плюсы и минусы

### Плюсы

- Простая реализация.
- Сервер может хранить дополнительную информацию о сессии.

 $\mathbb{N}_0$



# JWT (JSON Web Token) подход

- Токен JWT содержит информацию о клиенте и подписывается сервером.
- Токен передается через заголовок Authorization или параметр запроса.

## Пример

- 1 Клиент отправляет запрос на аутентификацию с логином и паролем.
- 2 Сервер создает JWT с информацией о клиенте и подписывает его секретным ключом.
- 3 Сервер возвращает JWT клиенту.
- 4 Клиент отправляет JWT в заголовке Authorization или параметре запроса.
- 5 Сервер проверяет подпись и расшифровывает JWT для аутентификации и авторизации.

## Плюсы и минусы

### Плюсы

# Cookie

## Структура запроса

Пример запроса с использованием cookie:

GET /items HTTP/1.1

Host: example.com

Cookie: sessionId=abcd1234

## Данные аутентификации

Cookie может содержать данные аутентификации, такие как токен доступа или идентификатор сессии. В примере выше, sessionId является идентификатором сессии.

## Содержимое сессии

Cookie может использоваться для хранения информации о сессии пользователя.

 $\mathbb{N}_0$

# JWT (JSON Web Token)

## Структура запроса

Пример запроса с использованием JWT:

GET /items HTTP/1.1

Host: example.com

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOi

## Данные аутентификации

JWT представляет собой токен, который содержит информацию о пользователе или сессии и подписывается с помощью секретного ключа. В примере выше, `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9` представляет собой заголовок токена, а `SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c` представляет собой подпись.

# Слайд 1

## Что такое JWT?

- JWT (JSON Web Token) - это формат токена, который используется для представления информации между двумя сторонами в компактном и безопасном способе.
- JWT состоит из трех частей: заголовка, полезной нагрузки и подписи.
- Заголовок содержит информацию о типе токена и используемом алгоритме шифрования.
- Полезная нагрузка (payload) содержит данные, которые нужно передать.
- Подпись используется для проверки подлинности токена.

## Слайд 2

## Пример создания JWT токена на Python

- ❶ Установите библиотеку PyJWT: `pip install PyJWT`
- ❷ Импортируйте библиотеку и укажите секретный ключ:

```
import jwt
```

```
secret_key = "my_secret_key"
```

## Слайд 3

## Пример создания JWT токена на Python (продолжение)

- 1 Создайте функцию для создания токена с полезной нагрузкой:

```
def create_token(payload):  
    token = jwt.encode(payload, secret_key, algorithm="HS256")  
    return token
```

- 1 Пример использования функции:

```
user_id = 123  
username = "john_doe"  
  
payload = {"user_id": user_id, "username": username}  
token = create_token(payload)  
print(token)
```

## Слайд 4

## Пример создания JWT токена на Python (продолжение)

## ❶ Расшифровка токена:

```
def decode_token(token):  
    decoded = jwt.decode(token, secret_key, algorithms=["HS256"])  
    return decoded
```

## ❷ Пример использования функции:

```
decoded_token = decode_token(token)  
print(decoded_token)
```

## ● Результат:

```
{  
    "user_id": 123,  
    "username": "john doe"
```

## Слайд 5

## Важно!

- Обязательно храните секретный ключ в безопасном месте, чтобы посторонние лица не могли его получить.
- Проверяйте подпись токена для уверенности в его подлинности.
- Токен может содержать любые данные, но не храните в нем конфиденциальную информацию без необходимости.



Конец



# Слайд 1

## Введение

- API (Application Programming Interface) предоставляет набор функций и возможностей для взаимодействия с программным обеспечением
- Пример API будет представлен для регистрации пользователя и работы с айтемом

## Слайд 2

## Методы API

- POST /users - создание нового пользователя
- GET /users/{userId} - получение информации о пользователе по идентификатору
- PUT /users/{userId} - обновление информации о пользователе
- DELETE /users/{userId} - удаление пользователя по идентификатору
- GET /items/{itemId} - получение информации об айтеме по идентификатору
- POST /items - создание нового айтема
- PUT /items/{itemId} - обновление информации об айтеме
- DELETE /items/{itemId} - удаление айтема по идентификатору

## Слайд 3

## Пример описания API на Swagger

```
swagger: "2.0"
info:
  title: API
  version: 1.0.0
paths:
  /users:
    post:
      summary:
      parameters:
        - in: body
          name: user
          required: true
          schema:
```

## Слайд 4

## Основные принципы документирования API на Swagger

- **Читаемость и понятность:** Описание API должно быть легко читаемым и понятным для разработчиков, чтобы они могли быстро понять, как использовать API.
- **Описательность:** В Swagger должно быть полное описание всех доступных методов, их параметров, кодов ответа и схем данных. Это помогает разработчикам детально изучить функциональность API.
- **Поддержка семантики:** Swagger должен предоставлять возможность использования семантических конструкций для описания связей между различными элементами API, например, связь между идентификаторами пользователей и айтемов.
- **Валидация данных:** Swagger может использоваться для валидации данных входящих запросов и исходящих ответов, что помогает обеспечить целостность данных и предотвращает ошибки.

## Слайд 5

## Полезьа от документирования API на Swagger

- Ясность и простота использования для разработчиков, которые используют API
- Быстрая интеграция между различными системами, так как Swagger предоставляет полное описание API и его возможностей
- Улучшение коммуникации между разработчиками, тестировщиками и другими участниками проекта
- Уменьшение затрат на разработку и поддержку API, так как Swagger предоставляет шаблоны и инструкции для генерации документации и кода
- Повышение безопасности путем использования встроенной валидации данных в Swagger

# Слайд 6

## Заключение

- Пример простого API для регистрации пользователя и работы с айтемом был представлен
- Swagger позволяет документировать API, описывая его методы, параметры, коды ответа и схемы данных
- Важными принципами документирования API на Swagger являются читаемость, описательность и поддержка семантики
- Документирование API на Swagger обладает рядом преимуществ, таких как ясность использования, быстрая интеграция и улучшение коммуникации

# Пример простого API для регистрации пользователя и работой с айтемом

## Slide 1

### Описание проекта и использование Swagger

- Проект представляет собой простое API для регистрации пользователя и работы с айтемом.
- Swagger используется для документирования и предоставления интерактивной документации API.

## Slide 2

### URL и методы API

- URL базового пути API: /api/v1
- Методы API: POST, GET, PUT, DELETE

## Slide 3

