Цифриум

Технологическая карта

Умный код на Python. Базовый уровень

Модуль 3 | Тема 1 | Урок 4

Авторизация в API

Уважаемый коллега, обратите внимание!

Учёт присутствия ребёнка на уроке ведётся только по цифровому следу. Каждый ученик из группы на уроке должен передать цифровой след на нашей платформе.

Вы отвечаете за передачу следа учеником на нашей платформе во время урока.

Передавать цифровой след нужно в личном кабинете ученика.

Цифровой след засчитывается по итогу выполнения учеником трёх задач на платформе в ходе урока.

Обращаем внимание — в задачах с отправкой ответа в виде файла наша платформа поддерживает следующие форматы: `.pdf, .doc, .png, .jpeg, .xlsx, .mp3, .py, .ipynb, .txt, .csv, .json, .xml, .sb3, .ino, .hex`.

Контролируйте время урока, чтобы не пропустить момент, когда ученики должны залогиниться в личном кабинете и перейти к решению задач на платформе.

Проследите, чтобы каждый ученик залогинился в личном кабинете и выполнил 3 задачи на платформе. В противном случае, Вам не будет засчитано, что вы провели урок, занятие не будет оплачено, а ученику не будет выставлена отметка о присутствии на уроке.

Просим вас в ходе 45 минут урока проконтролировать, чтобы:

1. Каждый из учеников залогинился в личном кабинете.
2. Каждый ученик в разделе **«Решить задания»** отправил решения по 3 задачам — «Практическое задание 1», «Практическое задание 2» и «Практическое задание 3».

О занятии

Краткое содержание занятия:

На этом занятии мы изучим методы авторизации при работе с API. Вы узнаете, зачем нужна авторизация, какие существуют способы её реализации и как безопасно хранить ключи доступа. Мы научимся работать с API ключами, передавать их в запросах и обрабатывать ошибки авторизации.

Цель занятия:

Освоить принципы авторизации в API и научиться безопасно работать с защищенными сервисами.

Задачи занятия:

- Понять необходимость авторизации при работе с API.
- Изучить основные методы авторизации.
- Научиться работать с API ключами.
- Освоить безопасное хранение секретных данных.
- Попрактиковаться в отправке авторизованных запросов.
- Научиться обрабатывать ошибки авторизации.

Необходимые материалы:

Для проведения занятия понадобятся: рабочая тетрадь, ваш документ/файл для записи терминов и выводов, ваш файл с расширением .py для написания скриптов, доступ к платформе Цифриум для самопроверки и ДЗ.

Темы и время

ЭТАП	ВРЕМЯ
Приветствие	5 мин.
Теория	25 мин.
Итоги занятия	5 мин.
Получение фидбека. Рефлексия. Проверка знаний. Домашнее задание	5 мин.
Вопрос-ответ	5 мин.
Итого	45 мин.

Ход занятия

Номер слайда	Пояснение к слайду
0	**Титульный слайд занятия**
1	**Повторение прошлого занятия**
2	**Анонс текущего занятия**
3	**Постановка цели и задач**
4	Авторизация в API — это механизм контроля доступа к защищённым ресурсам. Её основная задача — проверять, имеет ли конкретный клиент (пользователь или приложение) право на выполнение запроса. Во-первых, авторизация защищает конфиденциальные данные. Она предотвращает доступ к персональной информации, админ-панелям или платным функциям для тех, у кого нет соответствующих прав. Во-вторых, она позволяет идентифицировать, кто именно делает запрос. Это необходимо для персонализации ответов (например, показа данных конкретного пользователя) и для учёта действий. Наконец, авторизация — основа для управления нагрузкой. Зная, какое приложение отправляет запросы, сервис может применять к нему индивидуальные лимиты (rate limiting), распределять квоты или тарифицировать использование API.
5	Авторизация — это не просто техническая необходимость, а критически важный элемент для функционирования любого серьёзного API. Её основная цель — обеспечение безопасности. Она защищает персональные данные пользователей от утечек и

Ход занятия

	несанкционированного доступа. Авторизация позволяет реализовать бизнес-модели, основанные на подписке или оплате за использование, предоставляя доступ к расширенному функционалу только платным клиентам. Она также даёт возможность анализировать активность: какие методы API используются чаще, откуда приходят запросы, что помогает в развитии сервиса. С её помощью API может предоставлять персонализированные ответы, например, показывать пользователю только его заказы или настройки. Наконец, авторизация помогает соблюдать законодательные требования о защите данных (такие как GDPR), регулируя доступ к информации.
6	Существует несколько стандартных методов авторизации в API, различающихся по сложности и сценариям применения. Самый простой метод — API-ключи. Это уникальные строки, которые клиент передаёт в каждом запросе, обычно в заголовке или параметре URL. Подходят для идентификации приложения. Basic Authentication — это передача логина и пароля в заголовке запроса в закодированном виде. Просто в реализации, но менее безопасно, так как требует постоянной передачи учётных данных. OAuth 2.0 — промышленный стандарт для делегированного доступа. Позволяет приложению получать ограниченный доступ к данным пользователя на другом сервисе без передачи ему логина и пароля. Использует токены доступа (Access Tokens), которые являются временными ключами для доступа. JWT (JSON Web Tokens) — это самодостаточные токены, которые содержат в себе закодированные данные

Ход занятия

	(например, ID пользователя и срок действия) и могут быть проверены сервером без обращения к базе данных. Часто используются вместе с OAuth 2.0.
7-8	API-ключи — это самый простой и распространённый метод авторизации для доступа к публичным API, особенно когда нужно идентифицировать приложение, а не конкретного пользователя. Ключ — это длинная уникальная строка, которую сервис генерирует для вас после регистрации. Ваша задача — включить этот ключ в каждый запрос к API. Способов передачи ключа обычно два. Первый — через параметр URL, например: https://api.example.com/data?api_key=ваш_ключ. Второй, более предпочтительный и безопасный способ — через заголовок HTTP-запроса, например: X-API-Key: ваш_ключ. Сервер, получив запрос, проверяет переданный ключ на валидность и наличие необходимых прав, после чего либо выполняет запрос, либо возвращает ошибку 403 Forbidden. Ключи легко отозвать и перевыпустить в случае компрометации.
9-10	Передача API-ключа через заголовки HTTP-запроса считается более правильным и безопасным подходом по сравнению с передачей в URL. Причина в том, что URL часто логируются на промежуточных серверах и могут остаться в истории браузера, что подвергает ключ риску утечки. Заголовки же, как правило, не попадают в логи так легко. В библиотеке requests это делается просто. Вы создаёте словарь headers, где ключ — это название заголовка, требуемое API (например, X-API-Key или Authorization), а значение — сам ваш ключ. Затем этот словарь передаётся в параметр headers функции запроса (requests.get(), requests.post() и т.д.). Библиотека

Ход занятия

	автоматически добавит этот заголовок к вашему HTTP-запросу. Сервер на другой стороне извлечёт ключ из заголовка и проверит его.
11-13	Рассмотрим практический пример с использованием API погоды OpenWeatherMap. Этот сервис требует обязательной авторизации через API-ключ. В начале кода мы определяем две переменные: api_key — сюда нужно подставить ключ, полученный при регистрации на сайте сервиса, и city — город, для которого мы хотим получить погоду. Далее нам потребуется сформировать корректный URL для запроса. Обычно документация API указывает, в каком виде нужно передавать ключ. В случае OpenWeatherMap ключ чаще всего передаётся как параметр в строке запроса, например, appid={api_key}. Следующим шагом будет выполнение GET-запроса с этим URL и обработка ответа, который вернёт JSON с данными о погоде в указанном городе.
14	Bearer-токены — это распространённый стандарт передачи токенов доступа (чаще всего полученных через OAuth 2.0 или JWT) в HTTP-запросах. Название «Bearer» (предъявитель) означает, что любой, кто предъявит этот токен, получит доступ. Поэтому его нужно хранить и передавать с особой осторожностью. Формат передачи строго определён. Токен помещается в заголовок Authorization. После типа авторизации Bearer через пробел указывается сам токен: Bearer eyJhbGciOiJIUzI1NiIs.... В коде на Python с библиотекой requests это выглядит так: вы создаёте словарь headers с ключом "Authorization" и значением, начинающимся со слова Bearer. Этот словарь затем передаётся в параметр headers функции запроса.

Ход занятия

	Сервер, получив такой заголовок, извлечёт токен, проверит его подпись (если это JWT) и срок действия, а затем определит права доступа для связанного с токеном пользователя или приложения.
15	Безопасное хранение API-ключей и токенов — это критически важная практика. Попавший в чужие руки ключ может привести к утечке данных, несанкционированному использованию сервиса от вашего имени и финансовым потерям. Главное правило: ключи никогда не должны попадать в систему контроля версий (например, Git). Их нельзя жёстко прописывать в исходном коде. Стандартный и рекомендуемый способ — использование переменных окружения. Ключ хранится в настройках операционной системы, а ваш код считывает его оттуда с помощью os.getenv('API_KEY'). Альтернатива — хранение ключей в отдельном конфигурационном файле (например, config.py или secrets.json), который добавляется в .gitignore и не коммитится. Такой файл можно раздать вручную только тем, у кого должен быть доступ. Таким образом, в публичном репозитории остаётся только шаблон конфигурации без реальных секретов.
16	Использование переменных окружения — это стандартный и безопасный способ работы с конфиденциальными данными, такими как API-ключи. В Python для доступа к ним используется модуль os и его метод os.environ.get(). Строка api_key = os.environ.get("API_KEY") пытается получить значение переменной окружения с именем API_KEY. Если такая переменная не установлена, метод вернёт None, что позволит вашему коду корректно обработать эту ситуацию, например, вывести понятную ошибку.

Ход занятия

	Перед запуском программы вы должны установить эту переменную в системе. В Linux/macOS это делается в терминале командой export API_KEY='ваш_ключ'. В Windows — set API_KEY=ваш_ключ в командной строке или через настройки системы. Это гарантирует, что секретный ключ существует отдельно от кода и никогда не попадёт в репозиторий.
17	Файл .env (dotenv) — это удобное решение для локальной разработки, которое упрощает управление переменными окружения. Вместо того чтобы устанавливать их в системе вручную перед каждым запуском, вы храните все ключи в одном текстовом файле. Файл имеет простой формат: каждая строка содержит пару КЛЮЧ=значение. В примере это API_KEY=your_secret_key_here. Самое важное: файл .env содержит секреты и обязательно должен быть внесён в .gitignore, чтобы никогда не попасть в публичный репозиторий.
18	В этом примере показано, как использовать библиотеку python-dotenv для загрузки переменных из файла .env. Первая строка импортирует функцию load_dotenv из установленной библиотеки. Вызов load_dotenv() — это ключевое действие. Функция находит файл .env в текущей директории (или по указанному пути) и загружает все определённые в нём переменные в окружение операционной системы, как если бы они были установлены вручную. После этого вы можете получать значения как обычные переменные окружения с помощью os.getenv("API_KEY"). Если файл .env существует и содержит ключ API_KEY, то в переменную api_key будет записано его значение.

Ход занятия

	Этот подход удобен тем, что при развёртывании приложения в продакшене (например, на хостинге) вы просто устанавливаете переменные окружения напрямую в настройках сервера, а код при этом остаётся неизменным.
19-20	Вопросы для обсуждения
21–22	При работе с авторизацией в API обязательна обработка соответствующих ошибок. Сервер сообщает о проблемах с доступом через стандартные коды состояния HTTP. Код 401 Unauthorized означает, что запрос не был выполнен из-за отсутствия или недействительности учётных данных. Чаще всего это указывает на проблему с API-ключом или токеном: он не передан, просрочен, отозван или просто неверен. Код 403 Forbidden указывает на другую ситуацию: сервер понял запрос (учётные данные могут быть верны), но намеренно отказывается его авторизовать. Такое может быть из-за недостаточных прав у пользователя (например, попытка доступа к админ-функциям), блокировки по IP или другим политикам безопасности. В вашем коде после каждого запроса следует проверять статус ответа и корректно информировать пользователя или логировать проблему для дальнейшего анализа, как показано в примере.
23	Лимиты использования (rate limiting) — это важный аспект работы с любым публичным или платным API. Сервисы вводят их для защиты от перегрузки и злоупотреблений. Лимит обычно определяется как количество запросов, которое можно сделать за определённый интервал времени (например, 100 запросов в час). Эта информация всегда указана в документации API. Часто сервер сообщает в заголовках ответа (X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset)

Ход занятия

	о вашем текущем лимите и оставшихся запросах. Их стоит анализировать, чтобы не превысить квоту. При превышении лимита сервер возвращает ошибку 429 Too Many Requests. Ваш код должен корректно обрабатывать её, например, приостанавливая выполнение (time.sleep()) перед повторной попыткой. Для стабильной работы важно проектировать приложение с учётом этих лимитов, реализуя паузы между запросами или кэширование данных.

Представьте, что API — это библиотека с ценными книгами. Некоторые книги доступны всем (открытые API), но для доступа к особо ценным экземплярам нужен читательский билет. Авторизация в API — это и есть проверка вашего «читательского билета», который подтверждает ваше право на доступ к данным.

Авторизация — это процесс проверки прав доступа к ресурсам API. Она отличается от аутентификации (проверки личности): аутентификация отвечает на вопрос «Кто вы?», а авторизация — «Что вам разрешено делать?».

Зачем нужна авторизация в API?

- Безопасность данных — защита личной информации пользователей.
- Контроль доступа — разные уровни доступа для разных пользователей.
- Мониторинг использования — отслеживание кто и как использует API.
- Ограничение нагрузки — лимиты на количество запросов.
- Монетизация — платный доступ к премиум-функциям.

Основные методы авторизации:

1. API ключи (API Keys)

Самый простой и распространённый метод. API ключ — это уникальная строка символов, которая идентифицирует ваше приложение.

Ход занятия

```python
# Передача ключа в URL
api_key = "abc123xyz"
url = f"https://api.example.com/data?api_key={api_key}"

# Передача ключа в заголовках (более безопасно)
headers = {"X-API-Key": "abc123xyz"}
response = requests.get(url, headers=headers)
```

2. Bearer токены

Токен передается в заголовке Authorization с префиксом "Bearer":

```python
headers = {
    "Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
response = requests.get(url, headers=headers)
```

3. Basic Authentication

Передача логина и пароля в закодированном виде:

```python
from requests.auth import HTTPBasicAuth
response = requests.get(url,
auth=HTTPBasicAuth('username', 'password'))
```

Ход занятия

Безопасное хранение ключей

Никогда не храните ключи прямо в коде! Это основное правило безопасности. Вместо этого используйте:

1. **Переменные окружения**

```python
import os
api_key = os.environ.get("MY_API_KEY")
```

2. **Файлы .env с библиотекой python-dotenv**

```python
# Файл .env
API_KEY=your_secret_key_here
WEATHER_API_KEY=another_secret_key

# В коде Python
from dotenv import load_dotenv
import os

load_dotenv()  # Загружает переменные из .env
api_key = os.getenv("API_KEY")
```

3. **Добавьте .env в .gitignore, чтобы не загрузить ключи в репозиторий:**

.gitignore

.env

Ход занятия

*.key

secrets/

Обработка ошибок авторизации

```
Python
При работе с API важно правильно обрабатывать ошибки
авторизации:

response = requests.get(url, headers=headers)

if response.status_code == 200:
    print("Успешно!")
    data = response.json()
elif response.status_code == 401:
    print("Ошибка 401: Неверный ключ или токен")
elif response.status_code == 403:
    print("Ошибка 403: Доступ запрещён")
elif response.status_code == 429:
    print("Ошибка 429: Превышен лимит запросов")
else:
    print(f"Ошибка {response.status_code}:
{response.text}")
```

Практический пример: OpenWeatherMap API

Рассмотрим реальный пример работы с погодным API:

```
Python
import requests
from dotenv import load_dotenv
import os
```

Ход занятия

```python
# Загружаем переменные окружения
load_dotenv()

# Получаем ключ из переменной окружения
api_key = os.getenv("OPENWEATHER_API_KEY")

# Формируем запрос
city = "Moscow"
url = f"https://api.openweathermap.org/data/2.5/weather"
params = {
    "q": city,
    "appid": api_key,
    "units": "metric",
    "lang": "ru"
}

# Отправляем запрос
response = requests.get(url, params=params)

# Обрабатываем ответ
if response.status_code == 200:
    data = response.json()
    temp = data["main"]["temp"]
    description = data["weather"][0]["description"]
    print(f"Погода в {city}: {temp}°C, {description}")
else:
    print(f"Ошибка: {response.status_code}")
```

Помните: каждый API имеет свою документацию, где описано, как именно передавать авторизационные данные. Всегда внимательно изучайте документацию перед началом работы!

Практика

Вопросы для самопроверки

1. В чём разница между аутентификацией и авторизацией? Приведите примеры из реальной жизни.
2. Почему передача API ключа в заголовках запроса считается более безопасной, чем в URL параметрах?
3. Какие HTTP коды ошибок связаны с проблемами авторизации и что они означают?

Практическое задание

Задача 1:

Создайте функцию safe_api_request(url, api_key), которая отправляет GET-запрос с API ключом в заголовке "X-API-Key". Функция должна возвращать JSON-данные при успешном запросе или None при ошибке. Обработайте коды ошибок 401, 403 и 429 с выводом соответствующих сообщений.

Задача 2:

Напишите программу для работы с API курсов валют. Создайте файл .env с переменной EXCHANGE_API_KEY. Реализуйте функцию get_exchange_rate(from_currency, to_currency), которая получает курс обмена между двумя валютами. Используйте API exchangerate-api.com или аналогичный. Программа должна безопасно загружать ключ из файла .env.

Задача 3:

Создайте класс APIClient с методами для работы с защищённым API. Класс должен:

- Хранить API ключ (передаётся при создании объекта).
- Иметь метод get(endpoint) для GET-запросов.
- Автоматически добавлять заголовки авторизации.
- Вести лог всех запросов с их статусами.
- Реализовать метод get_remaining_requests() для проверки оставшихся запросов (если API возвращает эту информацию в заголовках).

Практика

Проверь себя по следующим критериям:

- В задаче 1: функция корректно передаёт ключ в заголовках, обрабатывает все указанные коды ошибок, возвращает данные или None.
- В задаче 2: ключ загружается из .env файла, не хранится в коде, программа успешно получает курсы валют.
- В задаче 3: класс инкапсулирует логику работы с API, автоматически добавляет авторизацию, ведёт лог запросов.

Примеры выполнения заданий

Задача 1:

Python

```python
import requests

def safe_api_request(url, api_key):
    headers = {"X-API-Key": api_key}

    try:
        response = requests.get(url, headers=headers)

        if response.status_code == 200:
            return response.json()
        elif response.status_code == 401:
            print("Ошибка 401: Неверный API ключ")
        elif response.status_code == 403:
            print("Ошибка 403: Доступ к ресурсу
запрещён")
        elif response.status_code == 429:
            print("Ошибка 429: Превышен лимит запросов")
        else:
            print(f"Ошибка {response.status_code}:
{response.text}")

        return None

    except requests.exceptions.RequestException as e:
        print(f"Ошибка при выполнении запроса: {e}")
        return None

# Пример использования
data = safe_api_request("https://api.example.com/data",
"your_api_key")
if data:
    print("Данные получены:", data)
```

Примеры выполнения заданий

Задача 2:

Python

```
import requests
from dotenv import load_dotenv
import os

# Загружаем переменные из .env
load_dotenv()

def get_exchange_rate(from_currency, to_currency):
    api_key = os.getenv("EXCHANGE_API_KEY")

    if not api_key:
        print("Ошибка: API ключ не найден в .env файле")
        return None

    url =
f"https://v6.exchangerate-api.com/v6/{api_key}/pair/{fro
m_currency}/{to_currency}"

    try:
        response = requests.get(url)

        if response.status_code == 200:
            data = response.json()
            if data["result"] == "success":
                return data["conversion_rate"]
            else:
                print(f"Ошибка API:
{data.get('error-type', 'Unknown')}")
        else:
            print(f"Ошибка HTTP:
{response.status_code}")

    except Exception as e:
```

Примеры выполнения заданий

```
        print(f"Ошибка: {e}")

    return None

# Пример использования
rate = get_exchange_rate("USD", "EUR")
if rate:
    print(f"1 USD = {rate} EUR")
```

Задача 3:

Python

```
import requests
from datetime import datetime

class APIClient:
    def __init__(self, api_key,
base_url="https://api.example.com"):
        self.api_key = api_key
        self.base_url = base_url
        self.request_log = []

    def get(self, endpoint):
        url = f"{self.base_url}/{endpoint}"
        headers = {"Authorization": f"Bearer
{self.api_key}"}

        # Записываем время начала запроса
        start_time = datetime.now()

        try:
```

Примеры выполнения заданий

```
            response = requests.get(url,
headers=headers)

            # Логируем запрос
            log_entry = {
                "timestamp": start_time,
                "endpoint": endpoint,
                "status_code": response.status_code,
                "success": response.status_code == 200
            }
            self.request_log.append(log_entry)

            if response.status_code == 200:
                return response.json()
            else:
                print(f"Ошибка {response.status_code}:
{response.text}")
                return None

        except Exception as e:
            log_entry = {
                "timestamp": start_time,
                "endpoint": endpoint,
                "status_code": None,
                "success": False,
                "error": str(e)
            }
            self.request_log.append(log_entry)
            print(f"Ошибка запроса: {e}")
            return None

    def get_remaining_requests(self):
        """Получает количество оставшихся запросов из
последнего ответа"""
        if self.request_log:
```

Примеры выполнения заданий

```
            # Делаем тестовый запрос для получения
заголовков
            response = requests.get(
                f"{self.base_url}/status",
                headers={"Authorization": f"Bearer
{self.api_key}"}
            )

            # Многие API возвращают лимиты в заголовках
            remaining =
response.headers.get("X-RateLimit-Remaining")
            limit =
response.headers.get("X-RateLimit-Limit")

            if remaining and limit:
                return f"Осталось {remaining} из {limit}
запросов"
            else:
                return "Информация о лимитах недоступна"

        return "Запросы ещё не выполнялись"

    def print_log(self):
        """Выводит лог всех запросов"""
        print("\n=== Лог запросов ===")
```

Сегодня на занятии вы:

- Познакомились с различными способами авторизации в API (ключи, токены, OAuth).
- Научились передавать токены и заголовки авторизации в запросах с requests.
- Разобрались с форматами передачи ключей API.
- Практиковались в обращении к ограниченным (требующим авторизации) API.
- Узнали о типичных ошибках авторизации и способах их обработки.

На следующем занятии вы:

- Научитесь работать с CSV-файлами через модуль csv.
- Освоите сохранение JSON-данных в файлы.
- Изучите работу с базами данных SQLite.
- Создадите программу для автоматического сбора и сохранения данных.

Получение фидбека. Рефлексия. Проверка знаний

Попросите ребят решить интерактивные задачи по теме урока.

Ответы к интерактивным задачам

Практическое задание 1

Что такое авторизация в API?

А) Процесс преобразования данных в JSON-формат.

Б) Процесс проверки прав доступа к ресурсам.

В) Метод увеличения скорости ответа от сервера.

Г) Способ кэширования данных для быстрого доступа.

Практическое задание 2

Какой HTTP-код статуса означает ошибку авторизации (неверный ключ)?

А) 200 (OK)

Б) 403 (Forbidden)

В) 429 (Too Many Requests)

Г) 401 (Unauthorized)

Практическое задание 3

В файлах какого формата рекомендуется хранить секретные ключи, чтобы не публиковать их в открытом репозитории?

env

Ответы к интерактивным задачам

Вопрос-ответ

Если во время подготовки или проведения урока возникают вопросы, их необходимо адресовать в методический чат.