

PROJECT COURSE

# Crunchyroll Downloader - Practical Scripting

6 Chapters

Philip Osztromok

Generated with Claude

# Contents

**Chapter 1** JWT & API Setup ..... 2

**Chapter 2** Configuration Management ..... 5

**Chapter 3** JWT Login & Authentication ..... 6

**Chapter 4** Building Utilities ..... 8

**Chapter 5** Complete Workflow ..... 9

**Chapter 6** Generalization & Extension ..... 10

# 🔐 JWT & API Setup

> *This chapter teaches you about JWT (JSON Web Tokens), how they work, and how to set up your project to authenticate with the Crunchyroll API. JWT is used by many modern web services for secure, stateless authentication.*

# 🔑 What is JWT?

JWT stands for **JSON Web Token**. It's a secure way to prove your identity to an API without sending your password every time.

## How JWT Works

| Step | What Happens |
|---|---|
| 1. Login | Send username/password to API |
| 2. API validates | Server checks credentials |
| 3. Token issued | Server sends back JWT token |
| 4. Store token | You save the token locally |
| 5. Use token | Send token in header for requests: `Authorization: Bearer TOKEN` |
| 6. Token expires | Token has expiration time (usually hours) |

## JWT Structure

A JWT looks like:

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE`

It has 3 parts separated by dots:

1. **Header:** Type of token (JWT) and algorithm
2. **Payload:** Your actual data (user ID, username, etc.)
3. **Signature:** Server's signature to prove it's legitimate

# 📁 Project Structure

```
crunchy-downloader/
├── .env                    ← Username/password (kept secret!)
├── .env.example            ← Template for others
├── requirements.txt        ← Python dependencies
├── config.py               ← Configuration management
├── crunchyroll_utils.py    ← API utility functions
├── download.py             ← Main download script
```

```
├── token_cache.json          ← Cached JWT token (gitignore!)
└── .gitignore                ← Don't commit secrets!
```

## 📦 Dependencies

Create `requirements.txt`:

```
requests==2.31.0
python-dotenv==1.0.0
```

Install them:

```
pip install -r requirements.txt
```

## 🔒 Storing Credentials Safely

Create `.env.example` (template — share this):

```
# Crunchyroll credentials
CRUNCHYROLL_EMAIL=your-email@example.com
CRUNCHYROLL_PASSWORD=your-password

# API settings
API_BASE_URL=https://api.crunchyroll.com
TOKEN_CACHE_FILE=token_cache.json
```

Create `.env` (YOUR actual credentials — never commit!):

```
# Crunchyroll credentials
CRUNCHYROLL_EMAIL=emubantam@gmail.com
CRUNCHYROLL_PASSWORD=mypassword123

# API settings
API_BASE_URL=https://api.crunchyroll.com
TOKEN_CACHE_FILE=token_cache.json
```

Create `.gitignore`:

```
# Secrets
.env
token_cache.json
credentials.json

# Python
__pycache__/
*.pyc

# IDE
```

```
.vscode/
.idea/
```

## 🌐 Understanding the Crunchyroll API

Crunchyroll's API requires:

- Login with email/password → get JWT token
- Use token in request header: `Authorization: Bearer {token}`
- Fetch episodes from `/api/v2/episodes` endpoint
- Handle token expiration and refresh

## 💻 Coding Challenges

**Challenge 1: Project Setup**
Set up the complete project structure:

1. Create crunchy-downloader/ directory
2. Create .env and .env.example files
3. Create requirements.txt with dependencies
4. Create .gitignore with appropriate entries

**Goal:** Complete the project setup.

**Challenge 2: JWT Decoding**
Understand JWT structure:

1. Get a JWT token string
2. Decode the payload (base64 decode middle part)
3. Extract user information
4. Check expiration timestamp

**Goal:** Understand JWT internals.

**Challenge 3: HTTP Headers with JWT**
Make authenticated HTTP requests:

1. Create sample JWT token
2. Create HTTP headers with `Authorization: Bearer {token}`
3. Test request to sample API
4. Verify token is sent correctly

**Goal:** Learn JWT-based API authentication.

# ⚙️ Configuration Management

> *Learn to load and manage Crunchyroll credentials securely, validate configuration, and handle sensitive data properly. This is critical for keeping your login information safe while building reusable code.*

## 🔒 Building a Secure Config Class

```
import os
from pathlib import Path
from dotenv import load_dotenv

ENV_FILE = Path(__file__).parent / ".env"
load_dotenv(ENV_FILE)

class Config:
    """Crunchyroll configuration."""

    def __init__(self):
        self.email = os.environ.get('CRUNCHYROLL_EMAIL')
        self.password = os.environ.get('CRUNCHYROLL_PASSWORD')
        self.api_base = os.environ.get('API_BASE_URL', 'https://api.crunchyroll.com')
        self.token_cache = os.environ.get('TOKEN_CACHE_FILE', 'token_cache.json')

    def validate(self) -> bool:
        """Validate configuration."""
        if not self.email or not self.password:
            print("❌ Missing email or password in .env")
            return False
        if '@' not in self.email:
            print("❌ Invalid email format")
            return False
        return True
```

## 💻 Coding Challenges

**Challenge 1: Config Class**
Create config.py with Config class that loads and stores credentials.

**Challenge 2: Validation**
Extend Config to validate email format and required fields.

**Challenge 3: Error Handling**
Add error handling for missing .env file and type conversion errors.

# 🔐 JWT Login & Authentication

> *Learn how to authenticate with Crunchyroll using email/password, receive a JWT token, cache it for reuse, and handle token expiration. This is the core of JWT-based API authentication.*

## 🔑 The Login Flow

```
import requests
import json
from pathlib import Path
from datetime import datetime

def login(config) -> str:
    """Login to Crunchyroll and get JWT token."""

    url = f"{config.api_base}/auth/v1/authenticate"

    payload = {
        "username": config.email,
        "password": config.password,
        "grant_type": "password"
    }

    try:
        # POST request to login endpoint
        response = requests.post(url, json=payload, timeout=10)

        if response.status_code != 200:
            print(f"❌ Login failed: {response.status_code}")
            print(response.json())
            return None
```

## 💾 Caching the Token

```
def save_token(token: str, cache_file: str):
    """Save token to file for reuse."""
    data = {
        "token": token,
        "timestamp": datetime.now().isoformat()
    }
    with open(cache_file, 'w') as f:
        json.dump(data, f)
    print(f"✅ Token cached to {cache_file}")

def load_cached_token(cache_file: str) -> str:
    """Load cached token if it exists."""
    if not Path(cache_file).exists():
        return None
```

```
    try:
        with open(cache_file, 'r') as f:
            data = json.load(f)
            return data.get("token")
    except:
        return None
```

## 📝 Using the Token in Requests

```
def get_with_token(url: str, token: str) -> dict:
    """Make GET request with JWT token."""

    headers = {
        "Authorization": f"Bearer {token}",
        "User-Agent": "Mozilla/5.0"
    }

    try:
        response = requests.get(url, headers=headers, timeout=10)

        if response.status_code == 401:
            print("❌ Token expired or invalid")
            return None

        if response.status_code == 200:
            return response.json()

        print(f"❌ Request failed: {response.status_code}")
        return None

    except Exception as e:
        print(f"❌ Error: {e}")
        return None
```

## 💻 Coding Challenges

**Challenge 1: Implement Login**
Create login() function that authenticates with Crunchyroll and returns JWT token.

**Challenge 2: Token Caching**
Implement save_token() and load_cached_token() to avoid re-logging in.

**Challenge 3: Authenticated Requests**
Create get_with_token() that includes JWT in Authorization header.

# 🔧 Building Crunchyroll Utilities

> *Create reusable utility functions for fetching episode data, parsing responses, and handling errors. Build a CrunchyrollClient class that encapsulates all API interactions.*

## 📦 CrunchyrollClient Class

```
class CrunchyrollClient:
    """Reusable Crunchyroll API client."""

    def __init__(self, config, token: str):
        self.config = config
        self.token = token
        self.base_url = config.api_base
        self.episodes_fetched = 0

    def _get(self, endpoint: str) -> dict:
        """Make authenticated GET request."""
        url = f"{self.base_url}{endpoint}"
        headers = {
            "Authorization": f"Bearer {self.token}"
        }
        response = requests.get(url, headers=headers, timeout=10)
        if response.status_code == 200:
            return response.json()
        return None

    def get_episodes(self) -> list:
        """Fetch all watched episodes."""
        episodes = []
        try:
            data = self._get("/api/v2/episodes")
```

## 💻 Coding Challenges

**Challenge 1: Build CrunchyrollClient**
Create class with __init__, _get(), and get_episodes() methods.

**Challenge 2: Episode Parsing**
Parse episode data to extract show name, episode name, date watched.

**Challenge 3: Error Handling**
Handle API errors, token expiration, network issues gracefully.

# 🔄 Complete Workflow

> *Orchestrate the complete end-to-end workflow: load config, login with JWT, fetch episodes, parse data, and export to CSV. This brings all previous components together.*

# 📋 The Workflow

```
def main():
    print("🚀 CRUNCHYROLL DOWNLOADER")

    # Step 1: Load config
    config = Config()
    if not config.validate():
        return False

    # Step 2: Try cached token, or login
    token = load_cached_token(config.token_cache)
    if not token:
        print("🔐 Logging in...")
        token = login(config)
        if not token:
            return False
        save_token(token, config.token_cache)

    # Step 3: Create client and fetch episodes
    print("📡 Fetching episodes...")
    client = CrunchyrollClient(config, token)
    episodes = client.get_episodes()
    print(f"✅ Fetched {len(episodes)} episodes")

    # Step 4: Export to CSV
    print("💾 Exporting to CSV...")
```

# 💻 Coding Challenges

**Challenge 1: JWT Login Flow**
Implement complete login: config → login → cache token → return token.

**Challenge 2: Episode Export**
Export episode data to CSV with columns: show, episode, date watched.

**Challenge 3: Complete Script**
Build main() that orchestrates entire workflow with error handling.

# 🎯 Generalization & Extension

> *Apply JWT authentication patterns to other services. This pattern works for ANY REST API that uses JWT: Spotify, Discord, Twitch, and hundreds of others. Learn to generalize and extend your solution.*

## 🔄 JWT Pattern Across Services

| Service | Login Endpoint | Token Location | Header Format |
|---|---|---|---|
| Crunchyroll | /auth/v1/authenticate | response.access_token | Authorization: Bearer TOKEN |
| Spotify | /api/token | response.access_token | Authorization: Bearer TOKEN |
| Discord | /oauth2/token | response.access_token | Authorization: Bearer TOKEN |
| Twitch | /oauth2/token | response.access_token | Authorization: Bearer TOKEN |

**The pattern is IDENTICAL:** Authenticate → Get token → Use in headers

## 📊 Real-World Extensions

### 1. Spotify Listening History

- Fetch your saved tracks and playlists
- Analyze listening patterns
- Export to CSV or database

### 2. Twitch Streaming Data

- Fetch your stream history
- Get viewer statistics
- Track channel performance

### 3. Discord Message Archive

- Export server/DM messages
- Analyze conversation patterns
- Create searchable archive

## 💻 Coding Challenges

> **Challenge 1: Spotify Adapter**
> Adapt pattern to Spotify API:

1. Create SpotifyClient class (similar to CrunchyrollClient)
2. Implement login for Spotify OAuth
3. Fetch saved tracks
4. Export to CSV

**Challenge 2: Add Scheduling**
Schedule automatic downloads:

1. Use APScheduler to run daily
2. Add logging for audit trail
3. Send email notifications

**Challenge 3: Production-Ready Utility**
Build complete system with:

1. Multiple service support
2. Database storage
3. Web dashboard for viewing data
4. Error monitoring and alerts

# 🎓 Course Complete!

**✅ What You've Mastered**

- JWT authentication and token management
- REST API interaction with token headers
- Building reusable API client classes
- Complete end-to-end workflows
- Data export and reporting
- Applying patterns to multiple services

You now have the skills to work with ANY JWT-based API! The Crunchyroll pattern you learned applies to thousands of services. Keep building! 🚀