

## Documentation

Prepared by: *Karishma Mohammed*

### My Approach

For this task, I built a **Python program** to collect match and odds data for basketball, baseball, and football from the **HotStreak** website. Since the website loads data dynamically through JavaScript and APIs, normal HTTP requests didn't work. To handle this, I used **Playwright**, which lets the program interact with the website like a real browser and safely fetch the data. Inside this browser session, I made **GraphQL API calls** to retrieve real-time match details and odds. The data was then structured using two Python models — **Match** and **Odd** — and exported to a clean, formatted **JSON file** for easy access and analysis.

I focused on keeping the project **organized, modular, and user-friendly**. Each folder serves a clear purpose:

- **config/** → Configuration files like config.yml for flexible setup
- **src/** → Core logic, split into subfolders
  - **classes/** → Contains reusable models (Match, Odd) and utility functions
  - **connections/** → Manages API and browser connections
  - **connectors/** → Handles different connection types (API, browser, websocket)
- **data/** → Contains logs and output files such as sample\_output.json

This setup makes the project easy to understand, maintain, and scale in the future.

### Challenges Faced

- **Website Blocking (426 Error):**  
The API initially rejected direct requests with a “426 Upgrade Required” error. I resolved this by executing the API calls inside the Playwright browser session, allowing it to behave like a genuine user.
- **Dynamic Data Loading:**  
The HotStreak platform loads content dynamically, so I added short wait times and validation logic to ensure data was fully available before extraction.
- **Incomplete Data:**  
Some matches were missing team names or odds. I used safe defaults and logging to handle these cases gracefully instead of stopping the program.

### Solutions Implemented

- Used **Playwright** to handle JavaScript-heavy pages.
- Added **robust error handling** and **logging** throughout the process.
- Configurations stored in a **YAML file** for flexibility and clarity.
- Adopted a **modular folder structure** for maintainability.
- Exported structured, easy-to-read **JSON outputs**.

### Summary

My goal was to make this program **dependable, clean, and easy to run**. It handles changing data, connection issues, and missing information smoothly, while producing consistent, well-structured results.