# Web Scraper Implementation Guide

Andrew Medrano

November 20, 2024

## Contents

**1 Project Structure and Setup** **2**
- 1.1 Directory Structure . . . 2
- 1.2 Environment Setup . . . 2
  - 1.2.1 Prerequisites . . . 2
  - 1.2.2 Setup Instructions . . . 2

**2 Contributing** **3**
- 2.1 Prerequisites . . . 3
- 2.2 Development Workflow . . . 3

**3 Overview** **4**

**4 Data Requirements** **4**

**5 Base Class Structure** **4**

**6 Key Components to Implement** **4**
- 6.1 Class Definition . . . 4
- 6.2 Required Methods . . . 5
  - 6.2.1 `get_page_soup` . . . 5
  - 6.2.2 `get_items_from_page` . . . 5
  - 6.2.3 `get_item_details` . . . 6

**7 Adding New Scrapers** **6**

**8 Running Scrapers** **7**

**9 Best Practices and Tips** **7**
- 9.1 HTML Inspection . . . 7
- 9.2 Rate Limiting . . . 7
- 9.3 Error Handling and Logging . . . 7
- 9.4 Data Storage . . . 7
- 9.5 BeautifulSoup Quick Reference . . . 8
- 9.6 Final Checklist . . . 8

**10 Git Workflow and Pull Requests** **8**
- 10.1 Branching Strategy . . . 8
- 10.2 Pull Request Process . . . 9

# 1 Project Structure and Setup

## 1.1 Directory Structure

The project follows this directory structure for organization and consistency:

```
Incepta_backend/
|-- README.md           # Project overview and setup instructions
|-- requirements.txt    # Python package dependencies
|-- main/
|   |-- scrapers/     # All web scrapers go here
|   |   |-- base_scraper.py
|   |   |-- your_scraper.py
|   |-- static/
|   |   |-- images/   # Static assets
|   |-- templates/    # Web interface templates
|   |-- embeddings_generator.py
|   |-- semantic_search_app.py
|   |-- semantic_llm_search.py
|-- data/
    |-- tech/        # Scraped technology data goes here
        |-- stanford_2024_11_20.csv
    |-- grants/
        |-- grants_sbir_2024_11_20.csv
```

## 1.2 Environment Setup

### 1.2.1 Prerequisites

- Python 3.8 or higher
- pip (Python package installer)
- Git

### 1.2.2 Setup Instructions

1. Create and activate a virtual environment:

```
# On Windows
python -m venv venv
.\venv\Scripts\activate

# On macOS/Linux
python3 -m venv venv
source venv/bin/activate
```

2. Install required packages:

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

## 2 Contributing

### 2.1 Prerequisites

1. Create a GitHub account if you don't have one.
2. Fork the repository by clicking the "Fork" button at https://github.com/andrew-medrano/Incepta_backend.
3. Clone your fork (not the original repository):

```
git clone
  ↪ https://github.com/YOUR-USERNAME/Incepta_backend.git
cd Incepta_backend
```

4. Add the original repository as a remote named `upstream`:

```
git remote add upstream
  ↪ https://github.com/andrew-medrano/Incepta_backend.git
```

### 2.2 Development Workflow

1. Keep your fork up to date:

```
git checkout develop
git fetch upstream
git merge upstream/develop
git push origin develop
```

2. Create your feature branch:

```
git checkout -b feature/your-university-scraper
```

3. Make your changes and commit them:

```
git add .
git commit -m "feat: Add scraper for University Name"
```

4. Push to your fork:

```
git push origin feature/your-university-scraper
```

5. Create a Pull Request:
   - Go to https://github.com/andrew-medrano/Incepta_backend.
   - Click "New Pull Request".
   - Click "compare across forks".
   - Select your fork and branch.
   - Fill in the PR template with required information.

## 3 Overview

This guide explains how to create new web scrapers that inherit from the updated `BaseScraper` class. The goal is to collect technology listings from university technology transfer offices and compile them into a structured dataset.

## 4 Data Requirements

The data collected by the scraper should be assembled into a DataFrame with the following columns:

- **university**: Name of the university
- **title**: Title of the technology listing
- **link**: URL to the detailed technology page
- **description**: Detailed description of the technology, including application areas, benefits, and other relevant information
- **development_stage**: stage of development - patent, proof of concept, prototype, etc (if available)
- **patent**: patent number and date published (if available)

## 5 Base Class Structure

The `BaseScraper` class provides a robust foundation for implementing website-specific scrapers. It includes:

- Session management with customizable headers
- Abstract methods that must be implemented in subclasses
- Flexible data field configuration
- Error handling and logging
- Optional retry logic for network requests
- Data collection and assembly into a DataFrame

## 6 Key Components to Implement

### 6.1 Class Definition

Your scraper should inherit from `BaseScraper` and initialize with the target website's URL, university name, and required data fields.

```python
from base_scraper import BaseScraper
from bs4 import BeautifulSoup
from typing import List, Dict
import time
import logging

class YourWebsiteScraper(BaseScraper):
    def __init__(self):
        fieldnames = ["university", "title", "link",
            ↪ "description"]
        super().__init__(
            base_url="https://your-website-url.com/",
            fieldnames=fieldnames,
            headers={
                'User-Agent': 'YourScraperName/1.0
                    ↪ (contact@example.com)'
            }
        )
        self.university = "Your University Name"
```

## 6.2 Required Methods

### 6.2.1 get_page_soup

```python
@retry(stop=stop_after_attempt(3), wait=wait_fixed(2))
def get_page_soup(self, page_number: int) -> BeautifulSoup:
    url = f"{self.base_url}/listings?page={page_number}"
    response = self.session.get(url)
    response.raise_for_status()
    soup = BeautifulSoup(response.text, 'html.parser')
    time.sleep(1)
    return soup
```

### 6.2.2 get_items_from_page

```python
def get_items_from_page(self, soup: BeautifulSoup) ->
    ↪ List[Dict[str, str]]:
    """
    Extract items from a page.

    Args:
        soup (BeautifulSoup): Parsed HTML content of a page.

    Returns:
        List[Dict[str, str]]: List of items with their titles
            ↪ and links.
    """
    items = []
    listings = soup.find_all('div', class_='listing-item')

    for listing in listings:
        title_element = listing.find('h3', class_='title')
        link_element = listing.find('a', class_='details-link')

        if title_element and link_element:
            title = title_element.get_text(strip=True)
            link = self.make_absolute_url(link_element['href'])
            items.append({
                'university': self.university,
                'title': title,
                'link': link
            })
        else:
            logging.warning(f"Missing title or link in listing: {listing}")

    return items
```

### 6.2.3 get_item_details

```python
def get_item_details(self, link: str) -> Dict[str, str]:
    """
    Get detailed information for a single item.

    Args:
        link (str): URL of the item's page.

    Returns:
        Dict[str, str]: Dictionary containing item details.
    """
    response = self.session.get(link)
    response.raise_for_status()
    soup = BeautifulSoup(response.text, 'html.parser')
    time.sleep(0.5)

    description_element = soup.find('div', class_='description')
    description = description_element.get_text(strip=True) if description_element else ''
    description = self.clean_text(description)

    development_stage = soup.find('div', class_='development stage')
    patent = soup.find('div', class_='patent')

    return {'description': description, 'development_stage': development_stage, 'patent': patent}
```

## 7 Adding New Scrapers

When adding a new scraper:

1. Create a new file in `main/scrapers/` with your scraper class.

2. Follow the naming convention: `university_name_scraper.py`.
3. Ensure scraped data is saved to `data/tech/`.
4. Update `requirements.txt` if new dependencies are needed.

## 8 Running Scrapers

1. Ensure you're in the project root directory.
2. Activate the virtual environment.
3. Run your scraper:

```
python -m main.scrapers.your_scraper \
--output data/tech/university_name_$(date +%Y_%m_%d).csv
```

## 9 Best Practices and Tips

### 9.1 HTML Inspection

1. Use browser developer tools (F12) to inspect website HTML.
2. Look for unique class names or IDs.
3. Test selectors in the browser console first.

### 9.2 Rate Limiting

1. Include `time.sleep()` delays between requests.
2. Start with conservative delays (e.g., 1 second).
3. Check the website's `robots.txt` for guidance.

### 9.3 Error Handling and Logging

1. Use `try/except` blocks for critical sections.
2. Use `response.raise_for_status()` to handle HTTP errors.
3. Provide fallback values when data extraction fails.
4. Use the `logging` module for appropriate severity levels.

### 9.4 Data Storage

- All scraped data should be saved in the `data/tech/` directory.
- Use consistent naming: `university_name_YYYY_MM_DD.csv`.
- Include metadata in the CSV header when possible.
- Ensure proper encoding (UTF-8) for all saved files.

### 9.5 BeautifulSoup Quick Reference

Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Common selectors:

```
soup.find('tag_name', class_='class_name')
soup.find_all('tag_name', class_='class_name')
element.get_text(strip=True)
element['attribute_name']
element.find_next('tag_name')
element.find_parent('tag_name')
```

### 9.6 Final Checklist

1. Included `university` Field.
2. Verified Data Fields.
3. Ensured Proper Error Handling.
4. Tested Scraper with Multiple Pages.
5. Output Data Matches Specified Format.
6. Complied with Ethical Scraping Practices.
7. Handled Missing Data.
8. Used Absolute URLs.
9. Set Custom Headers.
10. Validated Data.
11. Documented Code.

## 10 Git Workflow and Pull Requests

### 10.1 Branching Strategy

- Main Branches:
  - `main`: Production-ready code.
  - `develop`: Integration branch for features.
- Feature Branches:
  - Name format: `feature/university-name-scraper`.
  - Branch from: `develop`.
  - Merge to: `develop`.
- Hotfix Branches:
  - Name format: `hotfix/brief-description`.
  - Branch from: `main`.
  - Merge to: both `main` and `develop`.

## 10.2 Pull Request Process

1. Create a new branch:

```
git checkout develop
git pull origin develop
git checkout -b feature/your-university-scraper
```

2. Commit your changes:

```
git add .
git commit -m "feat: Add scraper for University Name"
```

3. Push and create PR:

```
git push origin feature/your-university-scraper
```

4. PR Requirements:
   - Title follows format: "feat: Add scraper for University Name".
   - Description includes:
     - Brief overview of changes.
     - Testing methodology.
     - Sample of scraped data.
   - All tests passing.
   - Code follows style guide.
   - Documentation updated.

5. Review Process:
   - At least one approval required.
   - All comments addressed.