**A**

**Project Report**

**On**

**Web Scraper**

**Abstract**

The proposed Web Scraper system is a powerful, user-centric tool designed to simplify the process of web data extraction for users with little to no programming experience. The platform enables users to visually select webpage elements through a point-and-click interface, eliminating the need for manual code writing. It supports both authenticated and non-authenticated website access, allowing for secure login when necessary.

With features such as automated scheduling, real-time scraping previews, and customizable export formats (CSV, JSON, XLSX, and database integration), the system is versatile and adaptable to different data collection needs. Users can manage their scraping tasks through a task dashboard, monitor scraping history, and replicate or modify existing tasks with ease.

The backend is built using Python, leveraging robust libraries like BeautifulSoup, Scrapy, and optionally Selenium for dynamic content handling. The frontend can be implemented using Flask or Django, providing a responsive and modern user experience. Celery and APScheduler are employed for task scheduling and automation, while Pandas handles data formatting and export.

Security is a core aspect of the system, incorporating user authentication, encrypted data storage, and support for proxies and rate-limiting to avoid detection or IP bans. The tool is ideal for businesses, analysts, academic researchers, and digital marketers who require structured web data for insights, reporting, or automation.

Overall, the Web Scraper provides an all-in-one, scalable, and cloud-ready solution to streamline the web scraping process, reduce human error, and enhance productivity.

**INDEX**

|  |  |  |
| --- | --- | --- |
| No. | Description | Page  No. |
| **1.** | **Company Profile** | **1-5** |
| **2.** | **Project Profile**   * 1. Project Definition   2. Project Description   3. Existing System / Work Environment   4. Problem Statements   5. Need for New System   6. Proposed System & Features   7. Scope   8. Outcomes   9. Tools & Technology used   10. Project Plan | **6-9** |
| **3.** | **Requirement Analysis**   * 1. Feasibility Study   2. Users of the System   3. Modules   4. Process Model   5. Hardware & Software Requirements   6. Use cases   7. Use case Diagram | **10-16** |
| **4.** | **Design**  4.1 Use Case Scenarios  4.2 Diagrams[Structured OR OOAD]  Structured:  4.2.1 Data Flow Diagram  4.3flow Chart  4.4 structure Diagram  OOAD  4.5 Class Diagram  4.6 System Architecture  4.7 Sequence Diagram  4.8 Activity Diagram  4.9 Deployment Diagram  4.10 ER Diagram  4.11 Data Dictionary | **17-28** |
| **5.** | **Implementation**  5.1 Form layout  5.2 page layout  5.3 Report layout | **29-31** |
| **6.** | **Testing**  6,1Testing strategy  6.2Test cases | **32-35** |
| **7.** | **Future Enhancement** | **36** |
| **8.** | **Bibliography** | **37** |

1. **Chapter 1**

**Company Profile**

**Company Profile:**

**Company Overview**

**Company Address:**

**Email:**

**Contact**:

1. **Project Profile**
   1. **Project Definition**

The project is a Web Scraper application that enables users to extract structured data from websites without requiring any coding knowledge. It provides an intuitive interface for selecting HTML elements, scheduling scraping tasks, and exporting results.

* 1. **Project Description**

The Web Scraper is designed to assist users in automating the process of data extraction from websites. Users can visually select elements they wish to scrape, configure export formats, and set schedules for regular data updates. It simplifies web data collection for research, business, or personal use.

The goal is to create a robust, intuitive, and feature-rich web scraping system that runs in the cloud. Users will be able to define scraping tasks using a browser-based UI, schedule scrapes at regular intervals, and export the scraped data to preferred formats like CSV, Excel, or JSON. Additionally, users can view the scraping history, receive logs, and be notified through email or WhatsApp when the data is ready. The backend will use Python and modern libraries like BeautifulSoup, Scrapy, or Selenium for scraping and Pandas for data processing.

* 1. **Existing System / Work Environment**

Traditional web scraping involves writing scripts in Python or JavaScript. Tools like Scrapy, Selenium, or Puppeteer require a programming background and are not suited for non-technical users. Furthermore, these tools lack user-friendly GUIs and scheduling features unless heavily customized. Manual scraping through copy-paste or browser extensions is tedious and prone to error.

|  |  |
| --- | --- |
| **Existing System** | **New System** |
| More manual hours are needed for Scraper the data. | Avoids more manual hours that need to Scraper the data. |
| Need lots of man power and time | Not need much man power and time |
| There is compulsory need in physical presence in the time to copy and paste data on File. | The User are allowed to scraper online at any time and they can even export the data in any type by sitting at home. |

* 1. **Problem Statements**
     + **Technical Challenges:**
       1. **Handling Dynamic Content** – Many websites load data dynamically using JavaScript, requiring tools like Selenium or Puppeteer.
       2. **CAPTCHA and Anti-Scraping Measur**es – Websites implement CAPTCHAs, IP blocking, and bot detection to prevent automated access.\
       3. **Website Structure Variabili**ty – Different sites have unique layouts, requiring customized scraping logic.
       4. **Rate Limits and Throttling** – Frequent requests can trigger IP bans, necessitating request delays or proxy rotation.
       5. **Data Cleaning and Processing** – Extracted data may be unstructured or contain inconsistencies that require preprocessing.
     + **Ethical & Legal Concerns:**
       1. **Data Ownership and Privacy** – Scraping personal or proprietary data without permission can raise ethical and legal issues.
       2. **Website Terms of Service Compliance** – Many sites prohibit scraping, making it essential to review their policies before proceeding.
       3. **Fair Use and Copyright Laws** – Some data may be protected by copyright or fair use restrictions.
       4. **Impact on Website Performance** – High-frequency scraping can strain servers, potentially disrupting a website’s normal operation.
  2. **Need for New System**

There is a strong demand for a tool that combines ease of use, automation, and robust export options. A modern system should allow both novice and advanced users to efficiently collect web data.

There is a critical need for an automated system that simplifies the entire data scraping workflow and provides real-time control over tasks without requiring a technical background. Users must be able to:

* + - 1. Schedule tasks remotely
      2. Extract specific HTML elements visually
      3. Receive data through WhatsApp or email
      4. View logs and previous scraping tasks
      5. Reuse or modify old scraping configuration
  1. **Proposed System & Features**
     + The proposed system offers:
     + Visual selection of elements
     + Scheduled scraping tasks
     + Multiple export formats (CSV, JSON, Excel)
     + History tracking and task management
     + Secure user authentication
     + Email notifications and real-time previews
  2. **Scope**
     + For General Users: Access and extract web data without coding.
     + For Developers: Extend the system via REST API and custom plugins.
     + For Businesses: Automate competitive research, price monitoring, lead collection, etc.
     + For Academic Use: Collect data for ML training, surveys, and research datasets.
  3. **Outcomes**
     + There is a critical need for an automated system that simplifies the entire data scraping workflow and provides real-time control over tasks without requiring a technical background. Users must be able to:
     + Schedule tasks remotely
     + Extract specific HTML elements visually
     + Receive data through WhatsApp or email
     + View logs and previous scraping tasks
     + Reuse or modify old scraping configuration
  4. **Tools & Technology Used**
* **Tools & Technology Used:**

|  |  |  |
| --- | --- | --- |
| Component | Technology | Justification |
| Frontend UI | Flask + Bootstrap | Quick deployment, responsive design |
| Backend | Python (Scrapy, BeautifulSoup, Selenium) | Rich scraping libraries, easy integration |
| Scheduling | Celery + Redis / APScheduler | Reliable task automation and background job management |
| Notification System | WhatsApp Cloud API / Pipedream | Send alerts directly to WhatsApp via Gmail or webhook |
| Export & Processing | Pandas | DataFrame support for export formats and data cleaning |
| Database | SQLite | Lightweight or scalable options depending on deployment |
| Cloud Platform | Pipedream / Render / PythonAnywhere | Affordable and serverless deployment choices |

* 1. **Project Plan**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Development phase** | ***75 Days*** | | | | | | **Duration**  **N**  **(days)** |
| 0to10 days | 11to20  days | 21to30 days | 31to40 days | 41to50 days | 51to75  days |
| **Requirement**  **Gathering** |  |  |  |  |  |  | 07 |
| **Analysis** |  |  |  |  |  |  | 09 |
| **Design** |  |  |  |  |  |  | 10 |
| **Development Phase 1** |  |  |  |  |  |  | 13 |
| **Development Phase 2** |  |  |  |  |  |  | 13 |
| **Development Phase 3** |  |  |  |  |  |  | 13 |
| **Documentation** |  |  |  |  |  |  | 10 |
| **Total time**  **(Days)** |  | | | | | | **75** |

Figure 2.10 Time Line Char

1. **Requirement Analysis**
   1. **What is Feasibility?**

A feasibility study is simply an assessment of the practicality of a proposed project plan or method. This is done by analysing technical, economic, legal, operational and time feasibility factors. Just as the name implies, you’re asking, “Is this feasible?” For example, do you have or can you create the technology to do what you propose? Do you have the people, tools and the resources necessary? And, will the project get you the ROI you expect?

* + - **Technical Feasibility**

The system is built on mature, open-source technologies (Python, Flask, BeautifulSoup, Scrapy) and incorporates modern features such as a visual element selector, automated scheduling, and proxy support to handle anti-scraping measures. These factors ensure a robust and scalable technical foundation.The project is technically feasible as it uses Python libraries (BeautifulSoup, Selenium) which are mature, well-documented, and widely used in web scraping. Flask or Django enables easy web development, while SQLite or PostgreSQL provides stable data management.

* + - **Economic Feasibility**

Leveraging open-source components and scalable cloud infrastructure keeps initial costs low. The automation of scraping and data export processes reduces labor expenses and increases productivity, making the project economically viable even when considering future legal and maintenance updates. The system uses open-source technologies, reducing software and licensing costs. Hosting can be done on free or low-cost cloud platforms.

* + - **Operational Feasibility**

With an intuitive interface, drag-and-click element selection, automated task management, and secure user authentication, the system is designed for ease of use by both beginners and professionals. This user-centric design supports efficient daily operations and minimizes manual intervention. The system is designed to be user-friendly and does not require coding skills. It is feasible for businesses and individuals alike.

* 1. **Users of the System**
* Admin
* User
  + - **Description of User Role**

1. **Admin**

Admin can manage user accounts, review websites, configure parameters, update system settings, and view system logs.

1. **User**

Users can register, login, submit URLs for web scraper, view Scraper results, update profiles, and logout.

* 1. **Modules**
* Admin Module
* User Module
* Scraper Module
* Export Module
* Scheduler Module
  + - **Modules Description**
* **Admin Module:**

This module enables system administrators to oversee and maintain the overall health and security of the Web Scraper system. Admins can manage users, review system activities, configure global settings, and monitor task performance.

* **Review Scraping Tasks:**

Allows the admin to review all scheduled, running, and completed scraping tasks. Admins can inspect task details, verify extraction accuracy, and address any errors or anomalies detected during scraping.

* **Manage User Accounts:**

Enables the admin to view, approve, update, or suspend user accounts. This submodule ensures that only authorized users can access the system and that account data is kept up to date.

* **Configure System Settings:**

Provides options to adjust global parameters such as scraping intervals, export formats, and proxy settings. Admins can update thresholds and rules to optimize performance and adapt to changing requirements.

* **Monitor System Logs:**

Offers a comprehensive view of system activities, including login attempts, task submissions, and error logs. This submodule helps ensure system security and facilitates performance tuning by providing detailed diagnostics.

* **User Module:**

This module empowers end users to interact with the Web Scraper for data extraction purposes without requiring coding expertise. Users can schedule tasks, visually select webpage elements, and access the extracted data.

* **User Registration:**

Allows new users to register and existing users to log in securely. This submodule ensures that access to the scraping tools is restricted to authorized individuals.

* **Create and Configure Scraping Tasks:**

Enables users to input target website URLs, use a drag-and-click interface to select webpage elements, and set scheduling options (e.g., one-time or recurring tasks).

* **View and Download Scraping Results:**

Provides a dashboard for users to review the results of their scraping tasks. Users can view task histories, inspect extracted data, and download results in multiple formats (CSV, JSON, Excel).

* **Scraper Module:**
* Execute scraping logic using selected parameters
* Support for dynamic/static pages
* **Export Module:**

Converts scraped data into CSV, Excel, or JSON format.

* **Scheduler Module:**

Allows users to automate scraping tasks at set intervals.

* 1. **Process Model**
     + **Process Model Used: Incremental Process Model**

This model allows the system to be built in phases, with each increment delivering a functioning component of the system:

* **Increment 1:** User Authentication and Manual Scraping
* **Increment 2**: Data Export and Report Generation
* **Increment 3:** Job Scheduling and Automation
* **Increment 4:** Admin Dashboard, Logging, and Notification

**Benefits of Incremental Model:**  
✅ Better risk management  
✅ Easier bug tracking and fixes  
✅ Early feedback from end-users  
✅ Faster deployment of core features

* 1. **Hardware & Software Requirements**
     + **Hardware Requirements**

|  |  |
| --- | --- |
| Component | Specification |
| Processor | Intel i5/i7 or AMD Ryzen 5 |
| RAM | Minimum 8 GB |
| HDD/SSD | Minimum 256 GB |
| Internet | Required for API and scraping access |

* + - **Software Requirements**

|  |  |
| --- | --- |
| Software/Tools | Purpose |
| Python 3.x | Core language |
| Flask | Backend framework |
| Selenium | For scraping JavaScript-heavy websites |
| BeautifulSoup | HTML parsing |
| SQLite/PostgreSQL | Database |
| CRON (or APScheduler) | Task automation |
| Pandas | Data processing |
| HTML/CSS + JS | Frontend |
| ChromeDriver | Selenium automation |
| Git | Version control |
| Render/Railway | Hosting |

* 1. **Use Cases**
     + **Actors:**
* **User** – The person initiating and managing scraping tasks.
* **System Admin** – Manages settings, users, and system parameters**.**
  + - **Use Cases:**

The Web Scraping System is designed for users to input website URLs, define which page elements to scrape, and retrieve/export data efficiently. Here's what each actor and use case does:

Actors:

User:

* + Can register/login to access the system.
  + Enter the target website URL and select HTML elements (like tables, text, images) for scraping.
  + Set scraping preferences, such as frequency, output format, and filters.
  + Authenticate if the target site requires login.
  + Execute scraping or schedule automated tasks.
  + View and download results after execution.
* System Admin:
  + Has access to user management, system logs, and detection rule configurations.
  + Can monitor scraping activity and fine-tune parameters for more accurate results.
  1. Use Case Diagram

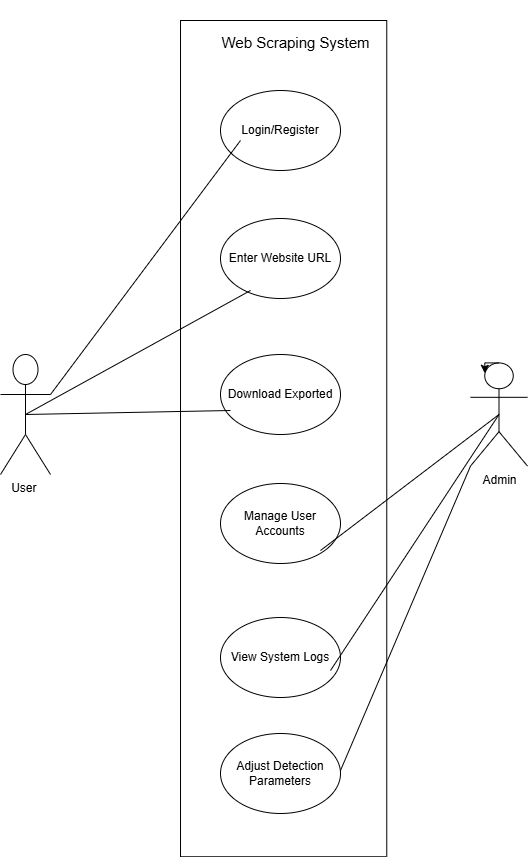


Figure 3.7. Use Case Diagram

1. **Design**
   1. **Use Case Scenarios**
      * Use Case Scenarios provide real-life examples of how users interact with the system. These scenarios explain the specific sequence of actions users and the system perform to achieve a particular goal.
        1. Example 1: User Submitting a URL for Scraping
      * Actor: Registered User
      * Precondition: User is logged in.
      * Steps:
        1. User navigates to the scraping interface.
        2. Enters a valid URL into the input field.
        3. Clicks on 'Scrape'.
        4. System fetches data, parses content, and stores result.
        5. System displays the extracted content.
        6. Postcondition: User receives structured data.
        7. Example 2: Admin Monitoring Scheduled Jobs
      * Actor: Admin
      * Precondition: Admin is authenticated.
      * Steps:
        1. Admin accesses the dashboard.
        2. Navigates to 'Scheduled Jobs'.
        3. Reviews upcoming jobs.
        4. Edits or deletes unnecessary jobs.
      * Postcondition: Jobs are managed effectively.
   2. **Data Flow Diagram**
      * 1. Context Level DFD’s

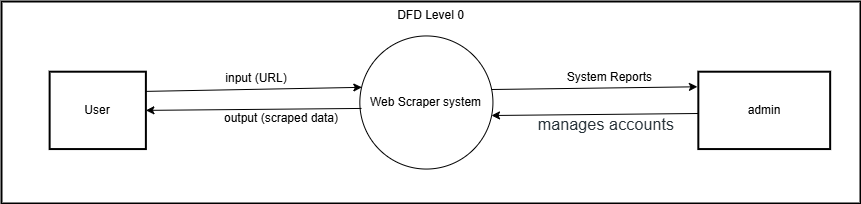


Figure 4.2.1. Context Level DFD: 0 Level

4.2.2. **Description of Context Level DFD: 0 Level -** At the highest level, the Web Scraper System interacts with two external entities: the User (who creates and monitors scraping tasks), the Admin (who manages accounts, settings, and logs).

* + - 1. Level 1 DFD’s:

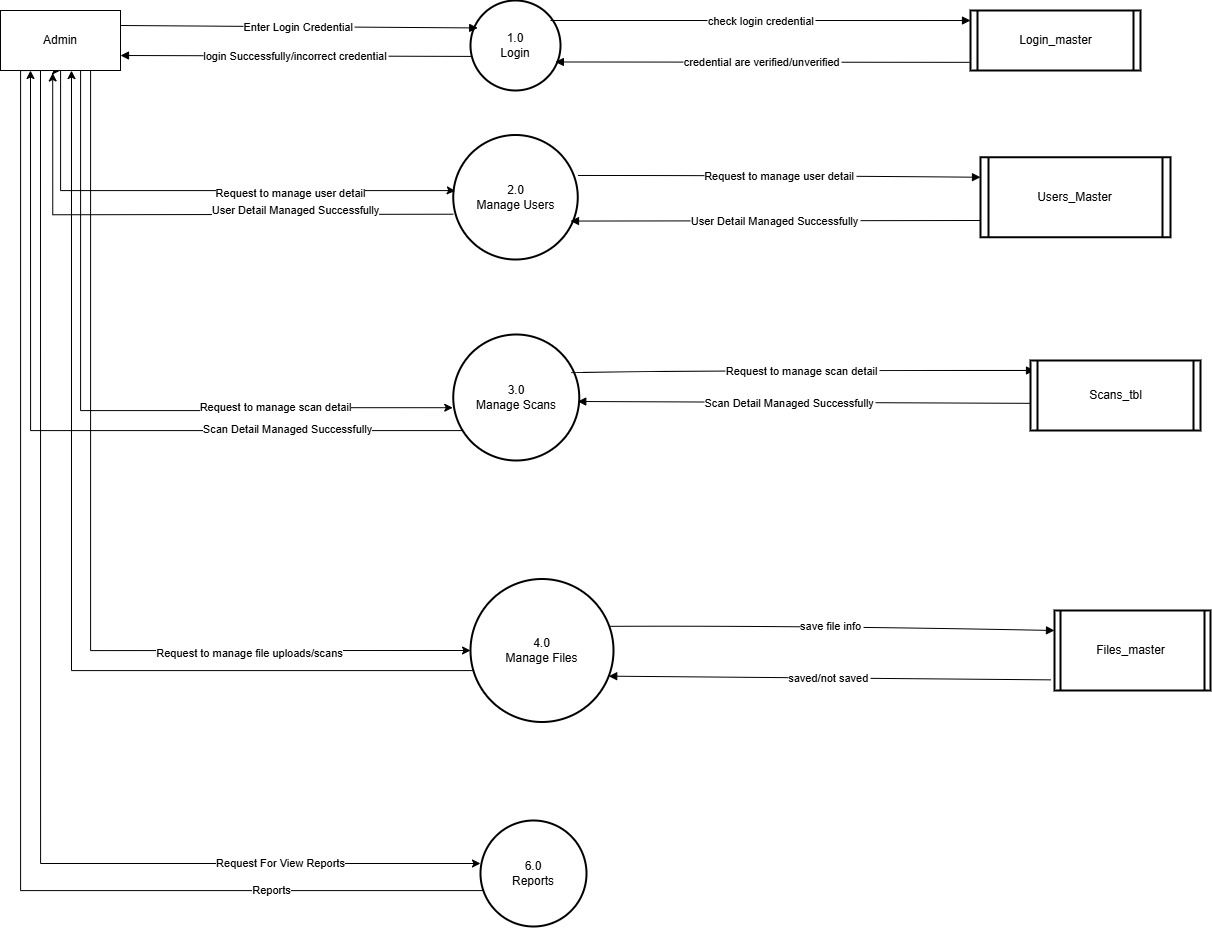


Figure 4.2.2 1st level DFD

**4.2.2 Description of Context Level DFD: 1 Level**

This diagram decomposes the system (Process 0) into three main sub-processes:

* + Manage User Accounts (handled primarily by Admins),
  + Scrape Task (core data extraction process),
  + Export Data (formatting and delivering results).  
    All these processes interact with the database.
* Figure 4.2.2 1st level DFD
  1. **Flow Chart**

Figure 4.3. Flow Chart



4.3.1. **Description of Flow Chart**

The flowchart visually represents the logical steps involved in the web scraping process. It provides a structured approach from input to output, ensuring smooth execution.

**Step-by-Step Description:**

1. **Start** – The process begins when the user initiates the scraping task.
2. **Enter Website URL** – The user inputs the target website URL.
3. **Select Page Elements** – The user selects the elements (text, tables, images) they want to scrape.
4. **Does Page Require Authentication?**
   * **Yes:** The user must enter login credentials.
   * **No:** The system skips authentication and proceeds.
5. **Configure Export & Scheduling** – The user sets preferences for exporting and scheduling the data extraction.
6. **Customize Scraping Preferences?**
   * **Yes:** The user selects detailed preferences (e.g., frequency, filters).
   * **No:** The system uses default settings.
7. **Execute Scraping** – The scraper starts extracting data from the selected elements.
8. **Validate Extracted Data** – The system checks for data accuracy and completeness.
9. **Data Valid?**
   * **Yes:** The extracted data is stored in the database.
   * **No:** The system reviews and adjusts the scraping logic.
10. **Process Data** – The scraped data is cleaned and prepared for export.
11. **Format Data for Export** – The system converts data into user-selected formats (CSV, JSON, Excel, etc.).
12. **Schedule Regular Exports?**

* **Yes:** The user sets up an automatic schedule for future exports.
* **No:** The system exports the data immediately.

1. **End** – The process is completed.
   1. **Structure Diagram**

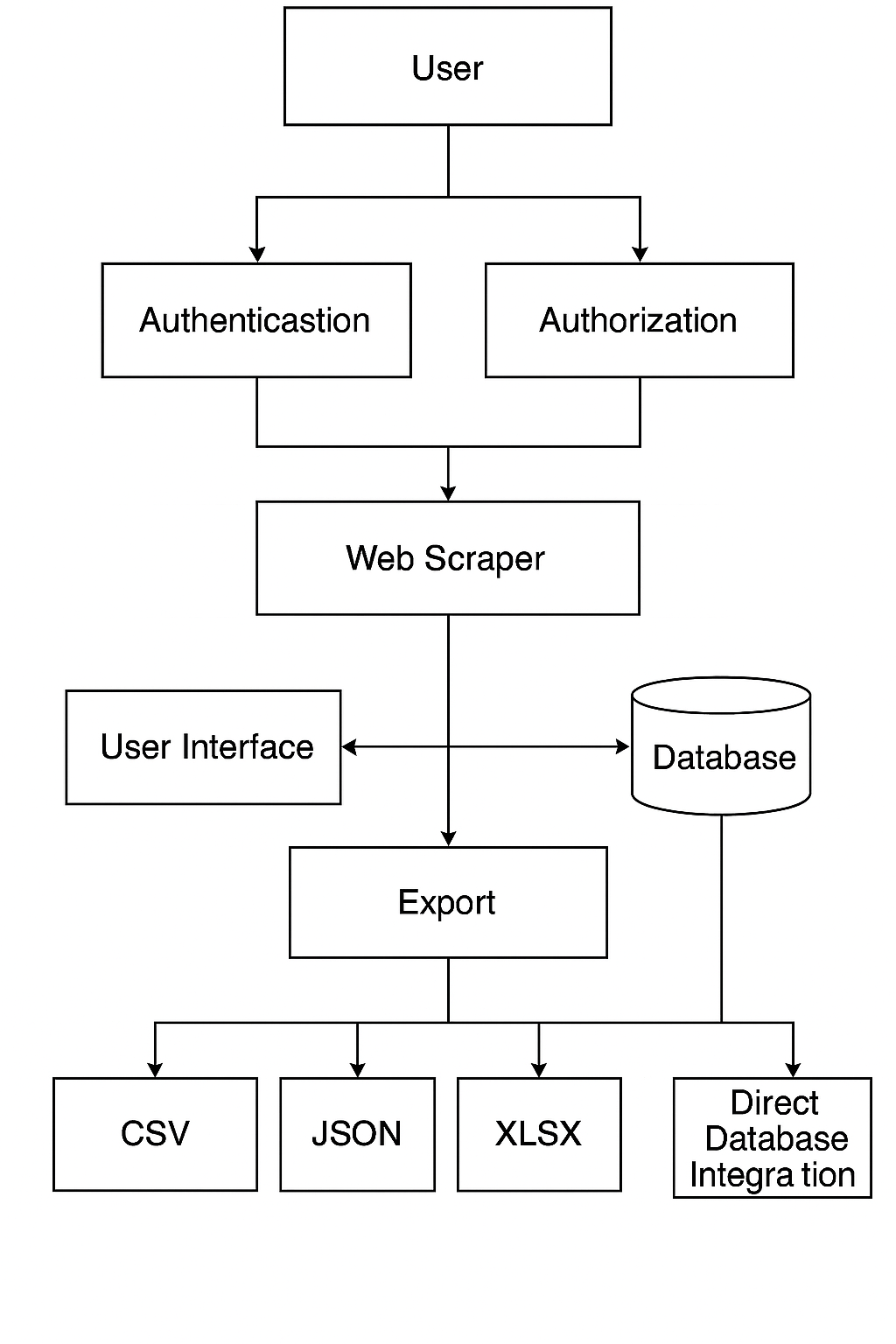


Figure 4.4. **Structure Diagram**

**4.4.1.Description of Structure Diagram**

**4.4.1.1.Represents hierarchical view:**

* + Web Scraper System
  + Authentication Module
  + Scraping Module
  + Scheduler
  + Exporter
  1. **Class Diagram**

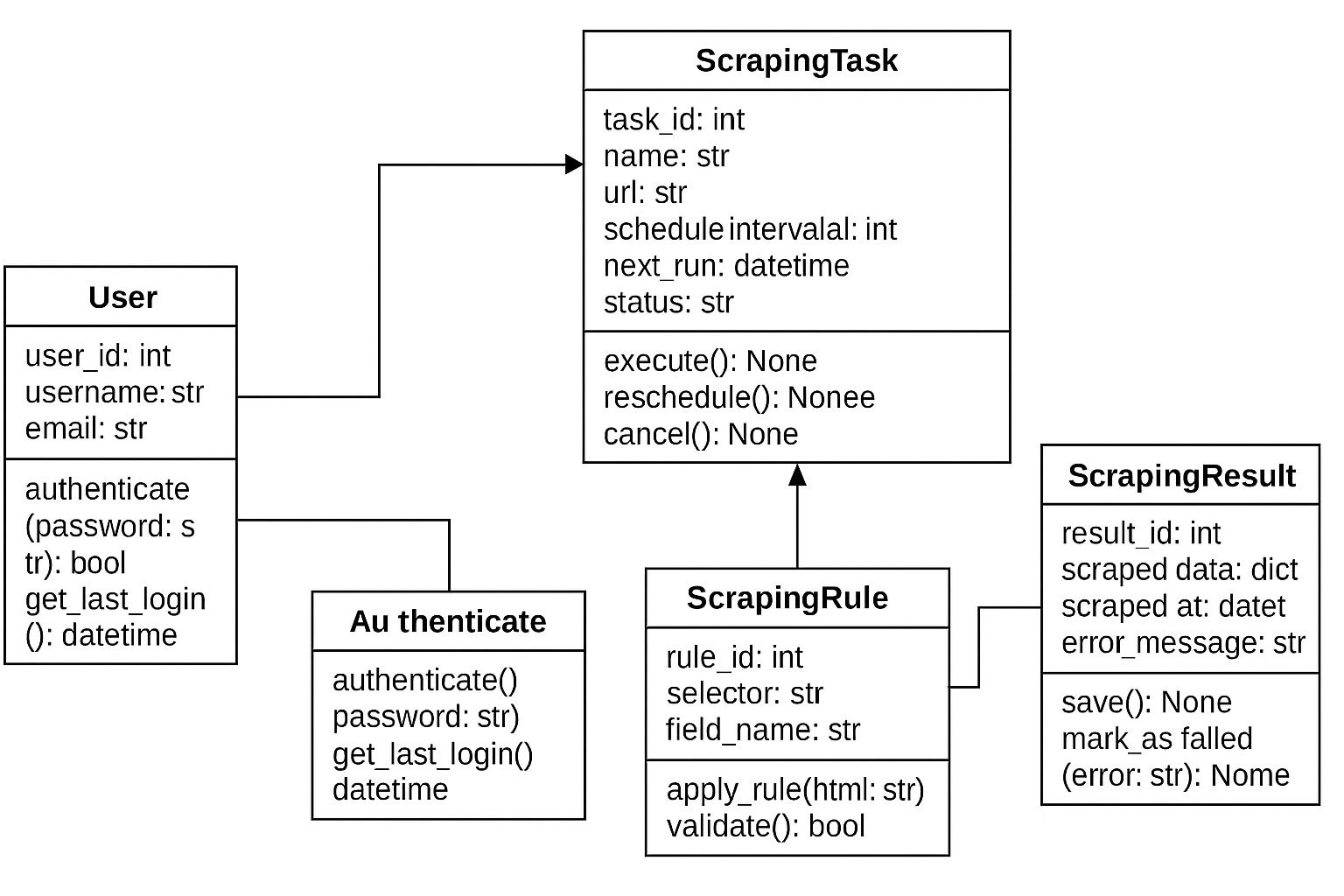


Figure 4.5**. Class Diagram**

**4.5.1.Description of Class Diagram**

**4.4.1.1.** Defines the main classes and their relationships:

* User: userID, email, password
* Scraper: method to fetch, parse, clean data
* JobScheduler: schedules scraping jobs
* Exporter: exports results to CSV/JSON
* Database: stores user, job, result info

Relationships: Association between User and Job, Aggregation of Results in Jobs.

* 1. **System Architecture**

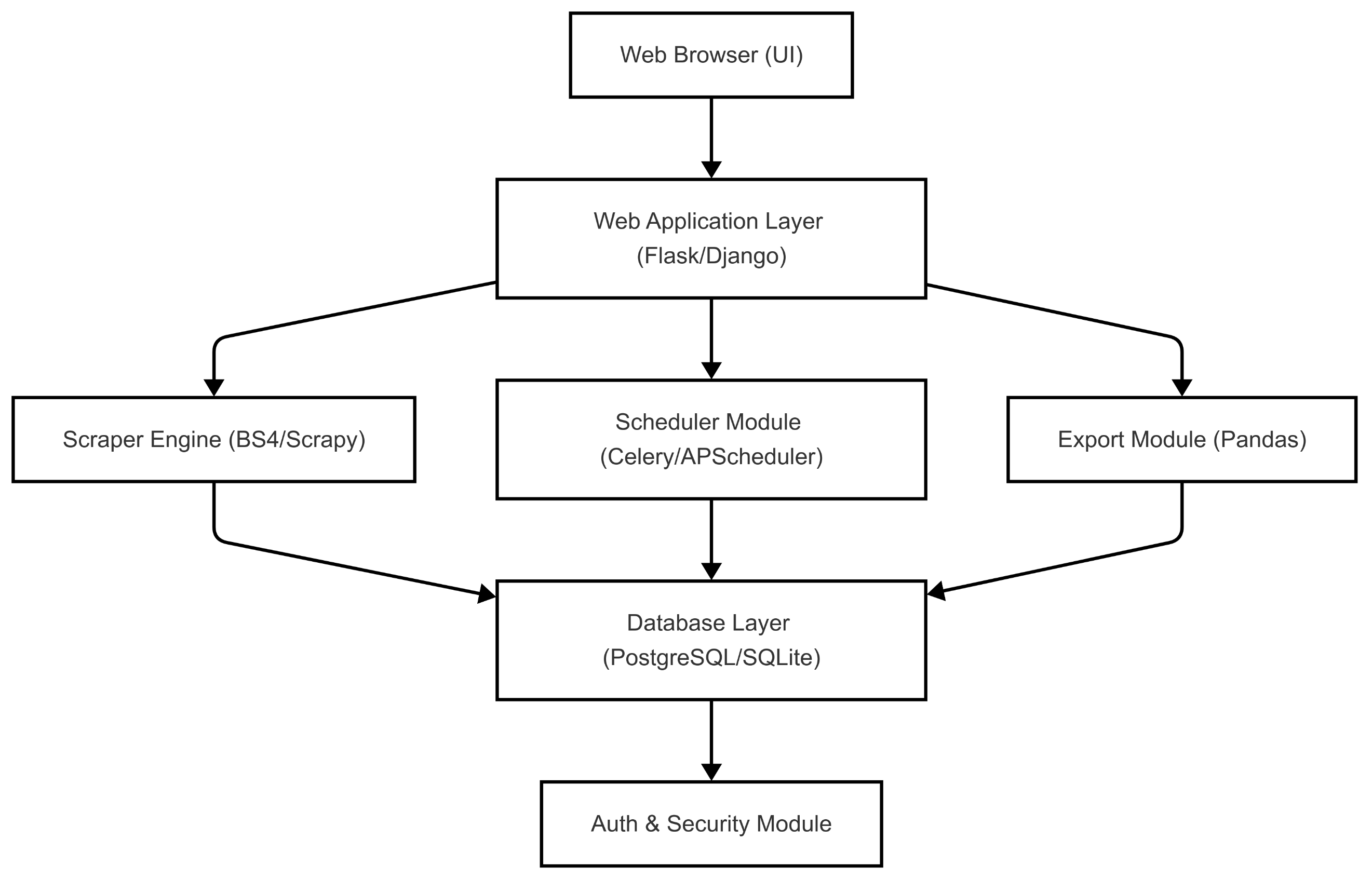


Figure 4.5. System Architecture

**4.5.1Description of System Architecture**

This layered architecture diagram outlines the major components:

* **User Interface:** Delivered via a web browser.
* **Web Application Layer:** Built on frameworks like Flask or Django.
* **Core Services:** Include the Scraper Engine, Scheduler, and Export Module.
* **Database:** Persistent storage for tasks and results.
* **Security:** Authentication and protection functions.
  1. **Sequence Diagram**

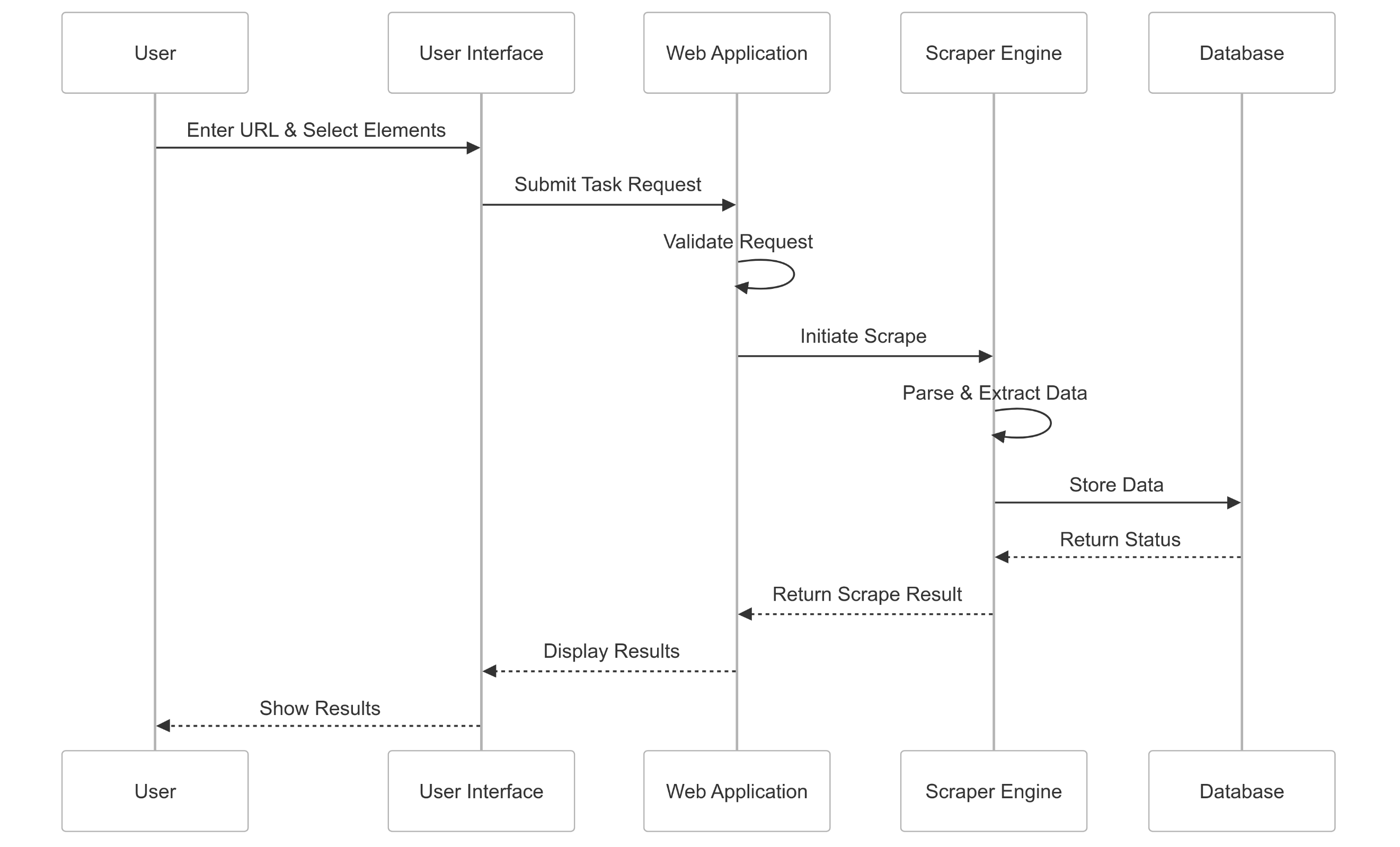


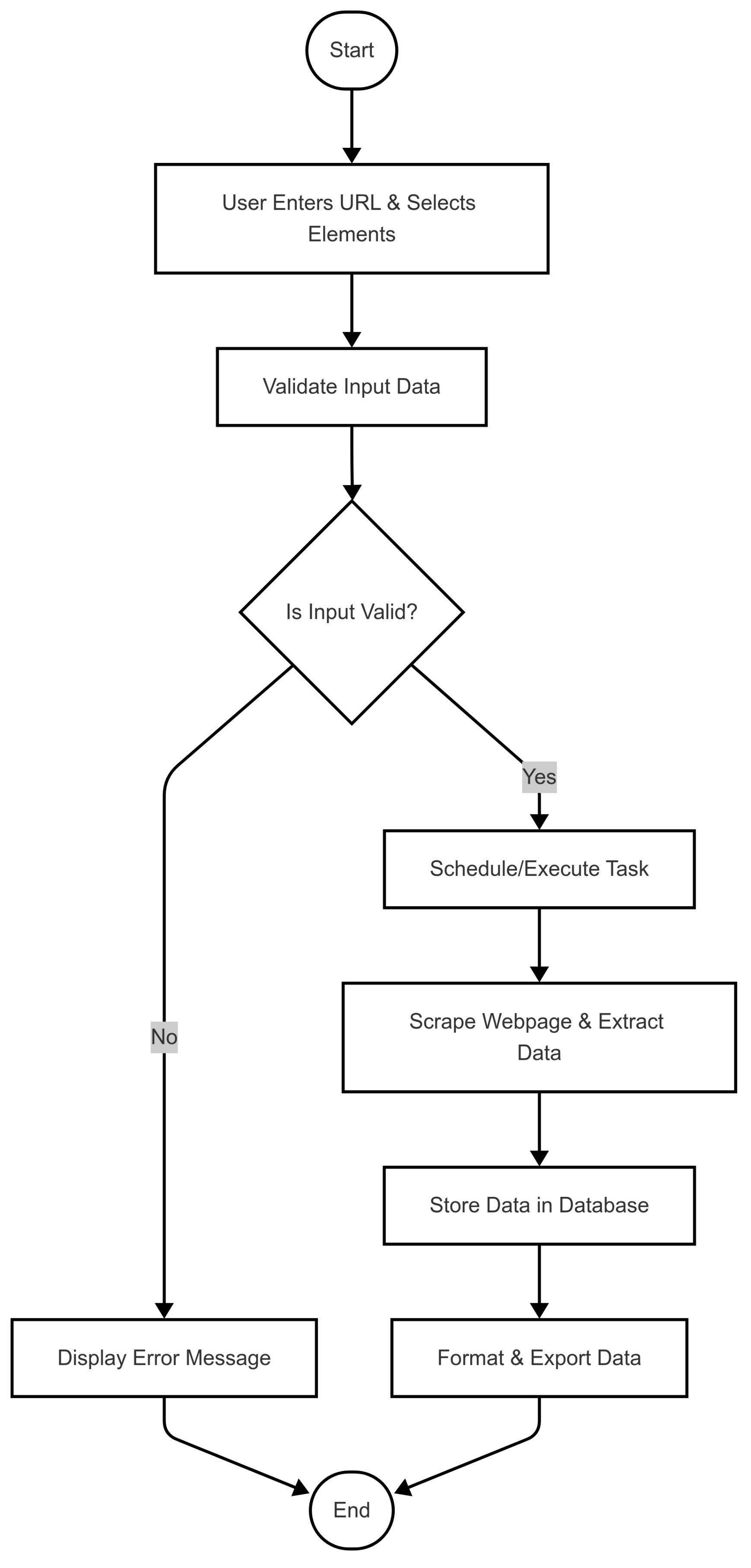
Figure 4.7Sequence Diagram

**4.7.1 Description of Sequence Diagram:**

Unified Modelling Language (UML) is a modeling language in the field of software engineering that aims to set standard ways to visualize the design of a system. UML guides the creation of multiple types of diagrams such as interaction, structure, and behavior diagrams. A sequence diagram is the most commonly used interaction diagram.

This sequence diagram details the interactions when a user initiates a scraping task:

1. The **User** enters the URL and selects elements.
2. The **UI** sends the task request to the **Web App**.
3. The **Web App** validates the request and triggers the **Scraper Engine**.
4. The **Scraper Engine** parses the webpage, extracts data, and stores results in the **Database**.
5. Finally, results are returned to and displayed by the UI.
   1. **Activity Diagram**



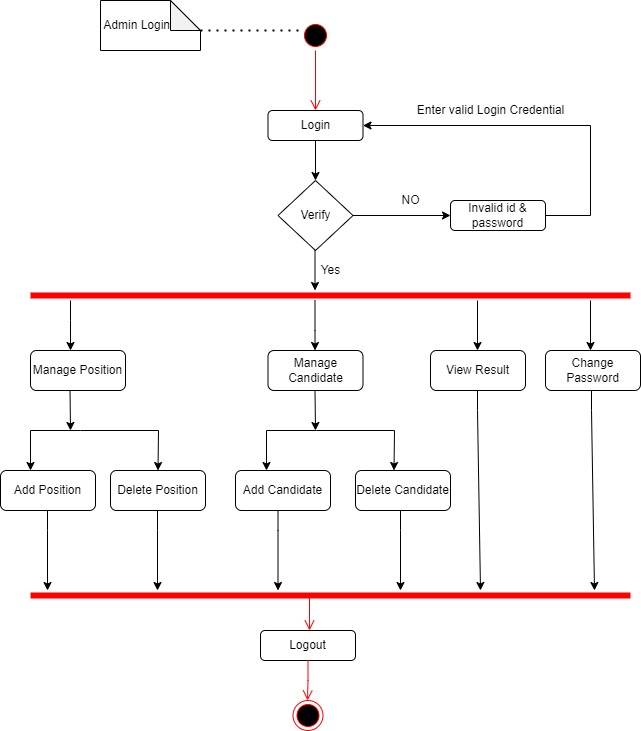


Figure 4.8 Activity Diagram

**4.8.1 Description of Activity Diagram**

This activity diagram captures the detailed flow for a scraping task—from user input and validation, through scraping, data storage, and export.

An activity diagram is a **behavioural diagram** i.e., it depicts the behaviour of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

* 1. **Deployment Diagram**

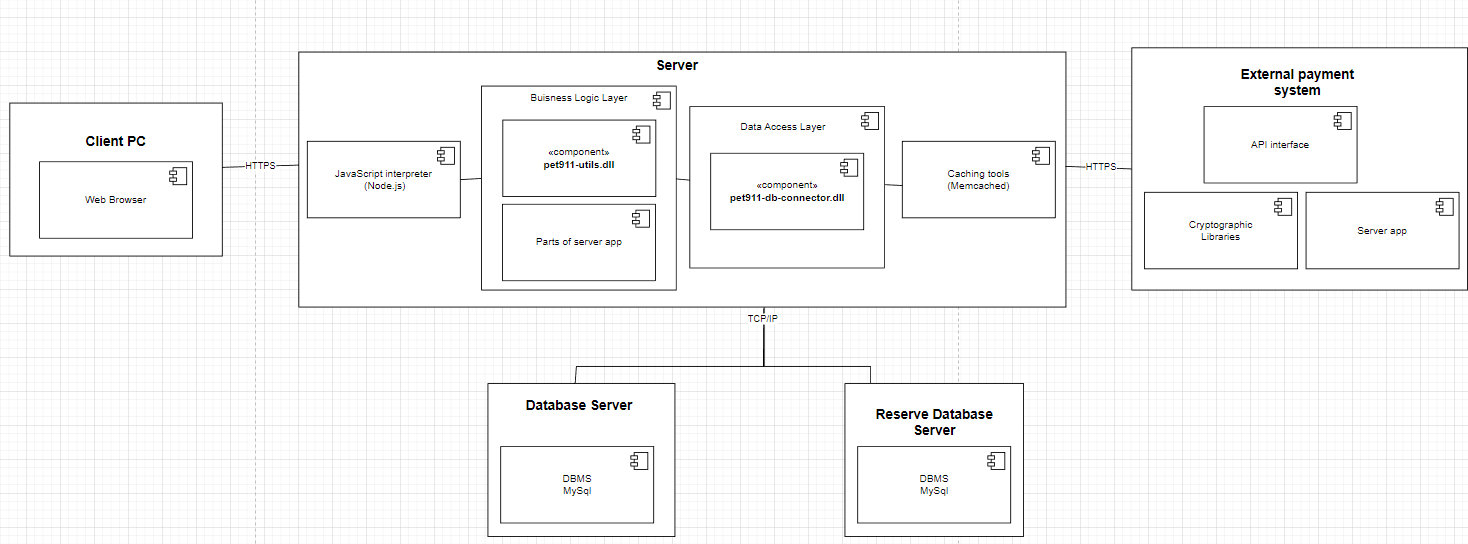


Figure 4.9 Deployment Diagram

* 1. **ER Diagram**

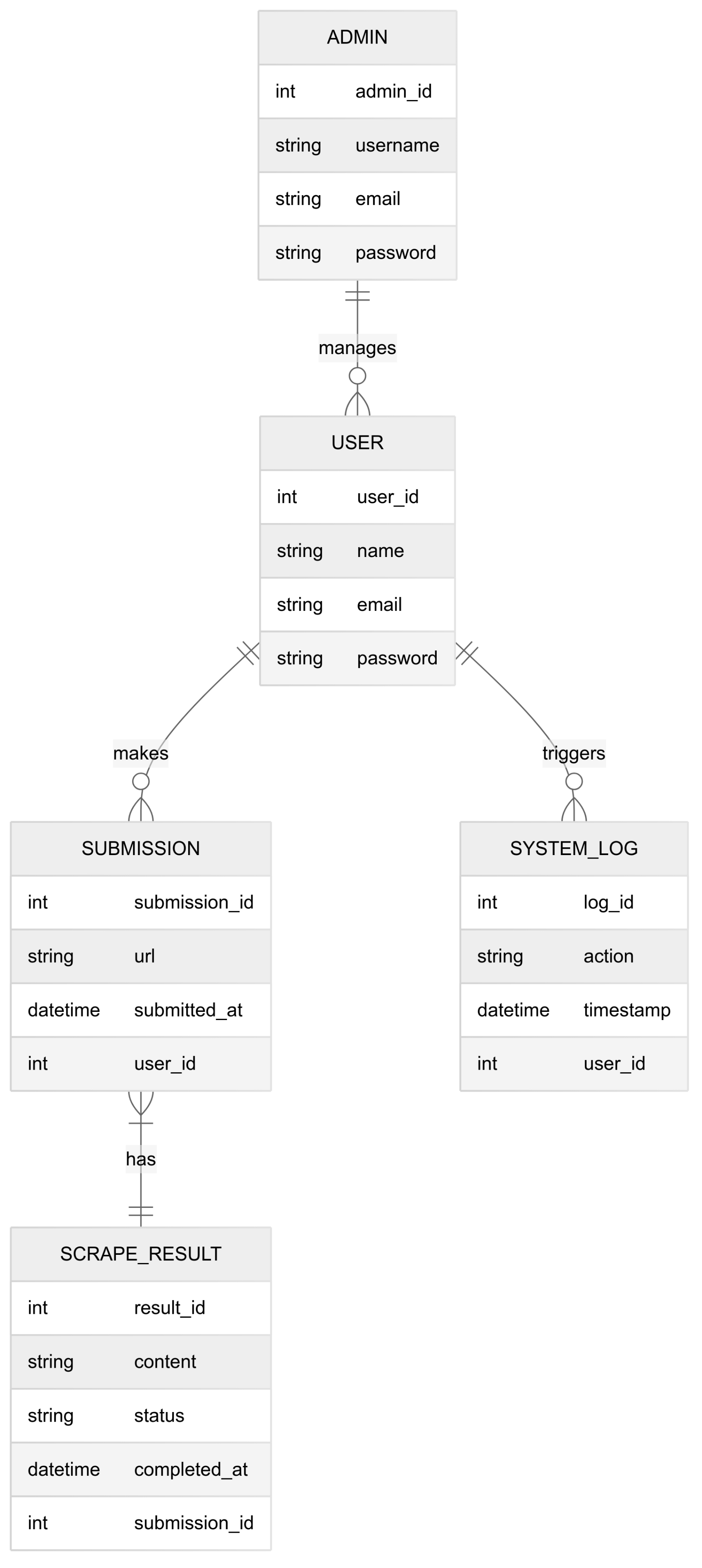


Figure 4.10ER Diagram

**4.9.1 Description of ER Diagram:**

Entities: User, URL, Job, Scrape\_Result, Export\_Log

Relationships like:

* A User can schedule many Jobs
* Each Job produces one or many Scrape\_
  1. **Data Dictionary**

**4.10.1. Admin Table:**

Below table show that the detail information about admin like admin name, admin id, password, etc. which contains different field name, data type, constrains and many more.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Field Name** | **Datatype** | **Size** | **Description** | **Constraint** | **Example** |
| 1 | Email | varchar | 45 | NOT NULL | Admin email address | admin@gmail.com |
| 2 | Password | varchar | 15 | NOT NULL | Admin password | Admin123 |

Table 1. Admin Table

**4.10.2 User Table:**

Below table show that the detail information about User like User name, User id, password, etc. which contains different field name, data type, constrains and many more.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Field Name** | **Data Type** | **Size** | **Constraint** | **Description** | **Example** |
| 1 | Email | varchar | 45 | NOT NULL | User email address | user@gmail.com |
| 2 | Password | varchar | 15 | NULL | User password | user123 |

Table 2. User Table

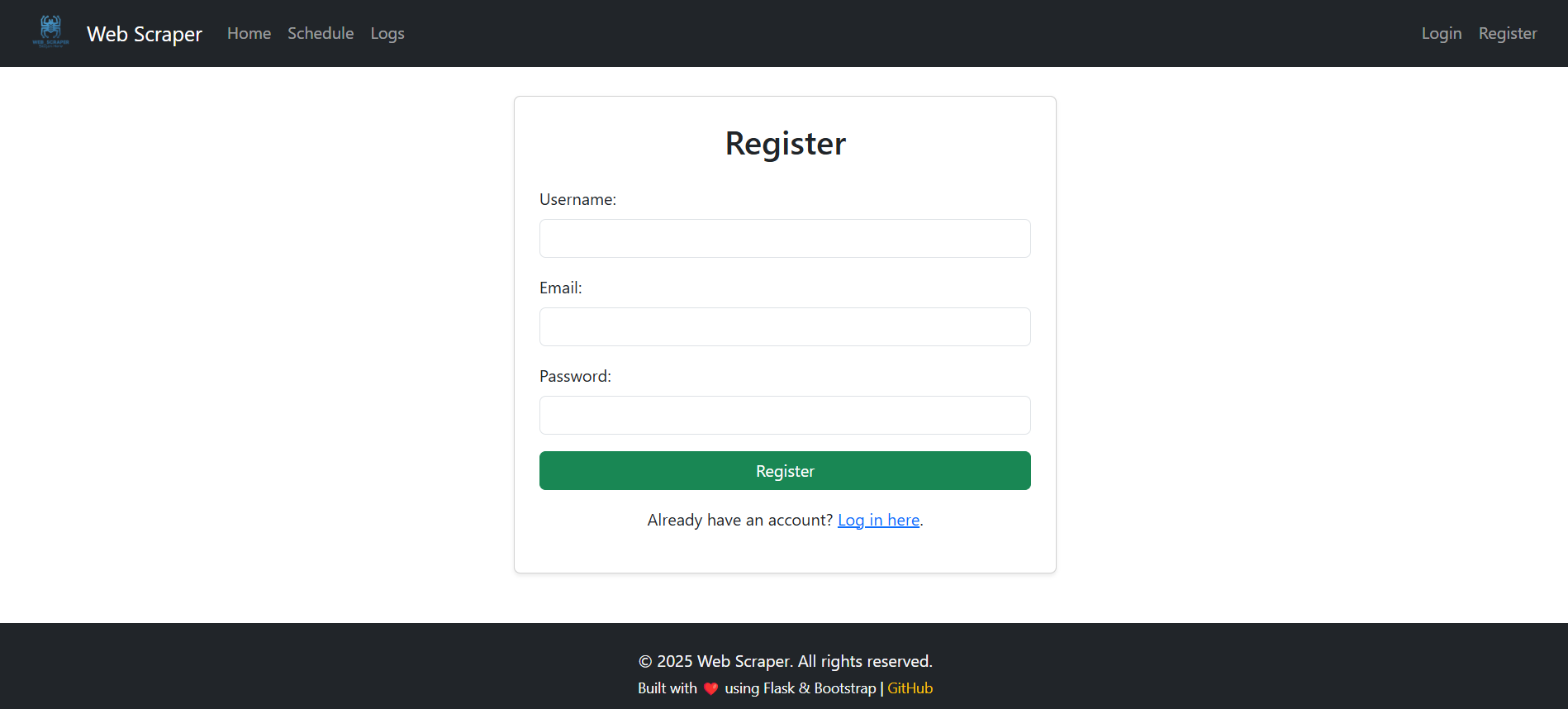
1. **Implementation**

**(Screenshots Phase 1, 2, 3)**

**5.1 Development Phase -1**

* **Form(Login) Page Design:-**

Below screenshot shows the login page which contain username/email, password and there is login button which perform after correctauthentication entered by User.



* **Code :-**

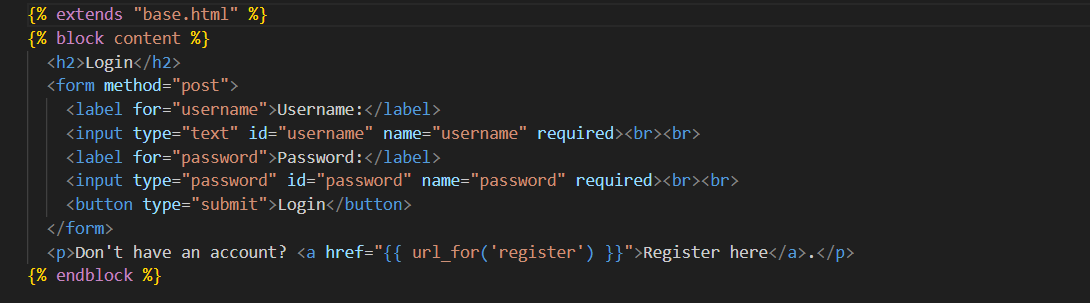
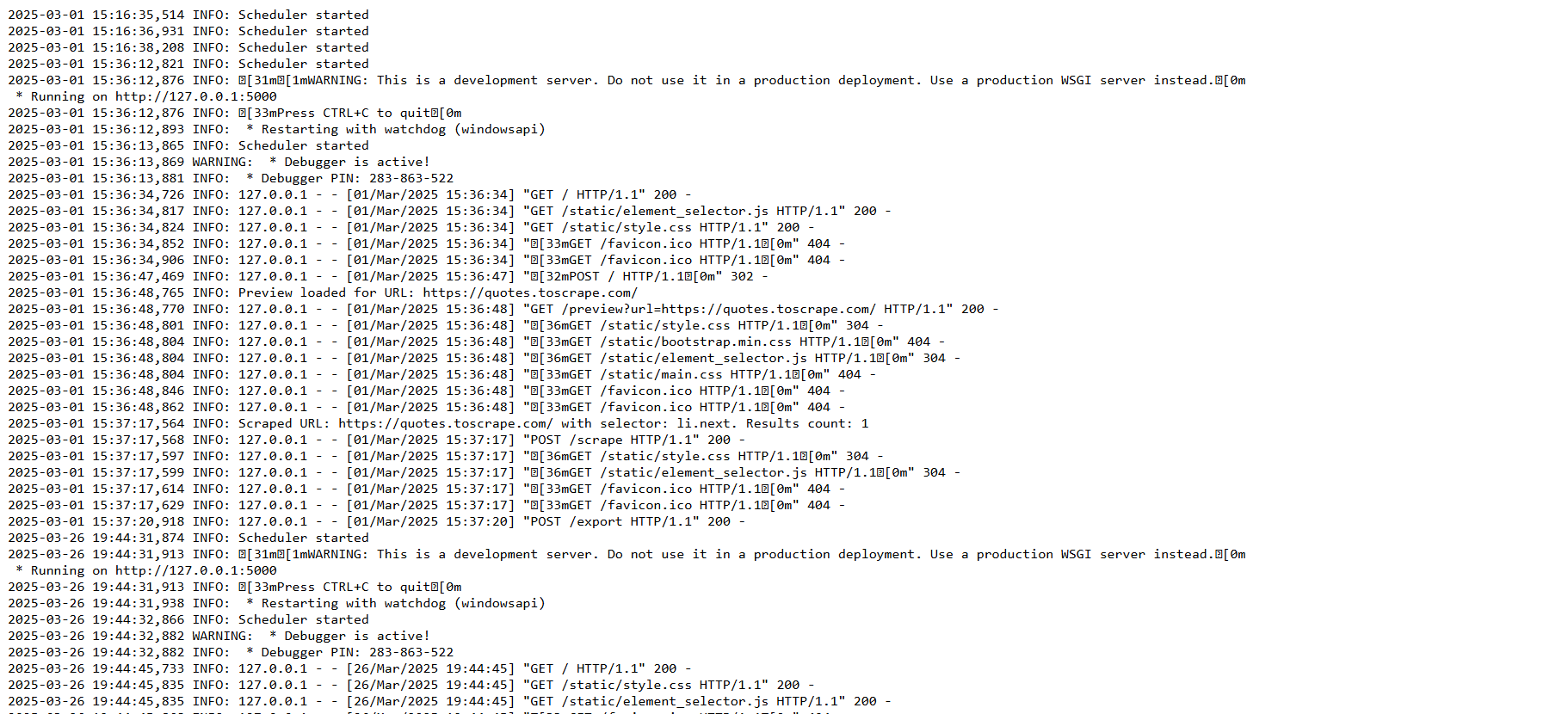


Figure 5.1 Login Page Design

**5.2 Development Phase -2**

* **Report (Logs) Design:-**

Below screenshot shows the logs page which contain username and the logs page to display log data directly in a timestamp format with **Username, Action, Timestamp, and Details**, along real-time updates. here is logs data entered by User.



* **Code :-**



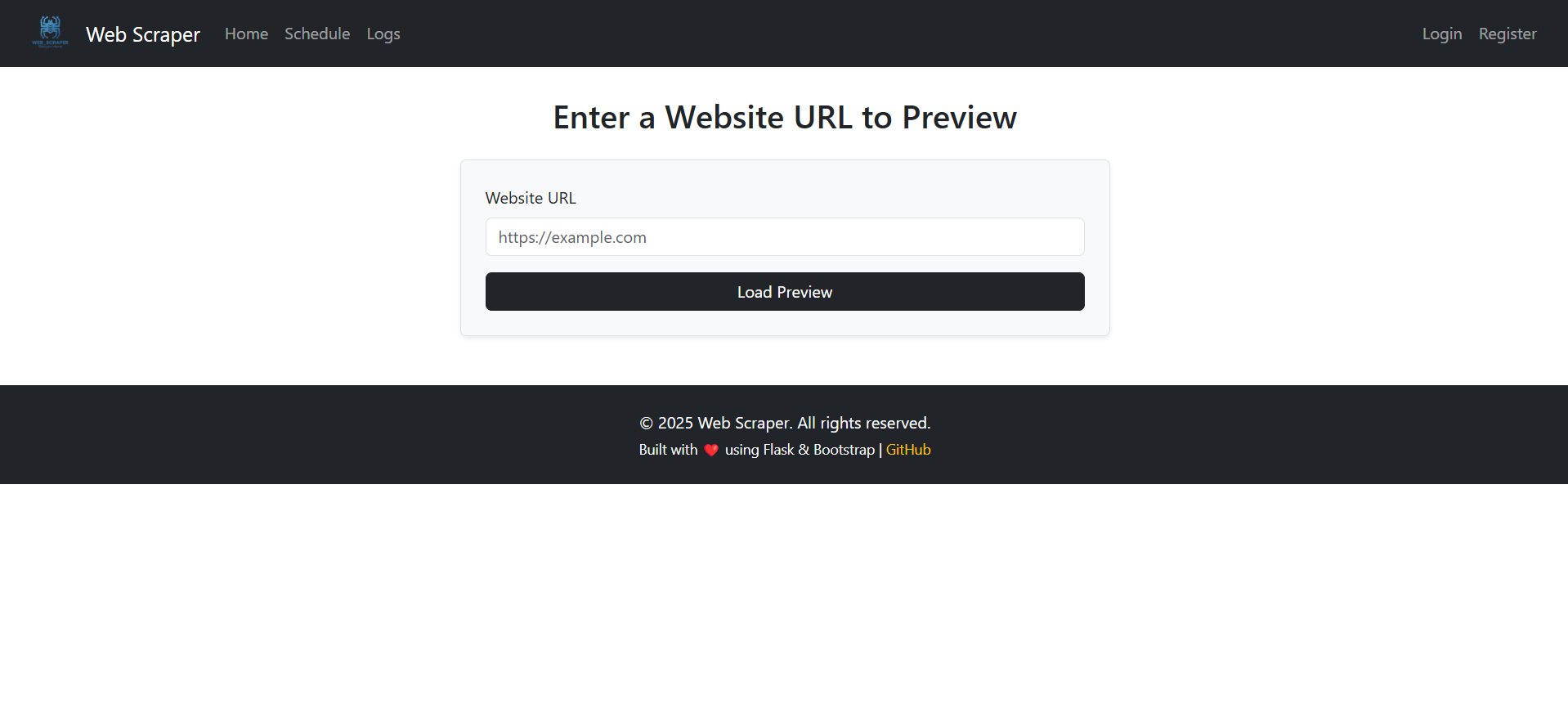
Figure 11.2

Figure 5.2 Logs Design

**11.3 Development Phase -3**

* **Page(Web Scraper) Design:-**

Below screenshot shows the Web Scraper page which contain input field where the user can enter data and there is button which perform after correctauthentication entered by User.



* **Code :-**

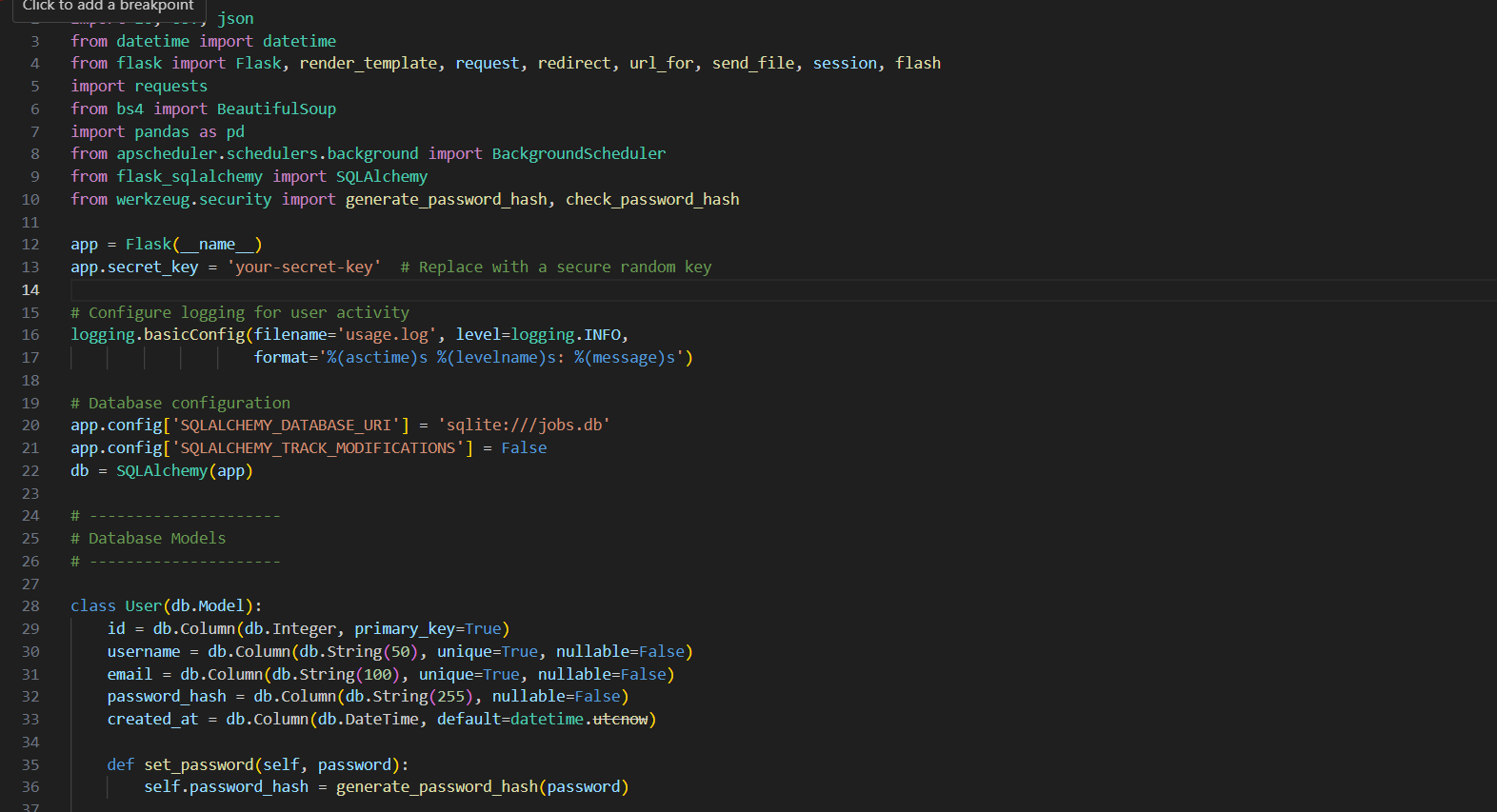


Figure 11.2

Figure 5.3 Web Scraper Design

**6. What is testing?**

**Web Testing,** or website testing is checking your web application or website for potential bugs before its made live and is accessible to general public. Web Testing checks for functionality, usability, security, compatibility, performance of the web application or website.

During this stage issues such as that of web application security, the functioning of the site, its access to handicapped as well as regular users and its ability to handle traffic is checked.

**1 Importance of testing**

* **Product quality** - Products always serve users, so it is very important that they deliver the values they promise. Hence, they should work properly to ensure a great customer experience. Following product requirements is imperative because it helps you get the wanted results.
* **Security** - there are a bunch of situations in which the information and details of the users are stolen. Therefore, people look for all-tested and reliable products. Testing will ensure that the user gets a trustworthy product, keep the user's personal information and data safe, and provide vulnerability-free products.
* **Customer satisfaction** - the first impression is important, and if you fail at it, users are going to find another product that will accomplish all their requirements. One reason why apps should be tested is to provide the best user experience possible. Being the best product in the market will help you gain trust-worthy clients, which will have great long-term effects.
* **Cost effectiveness** - including testing in your project can save money in the long run. Software development consists of many stages, and if bugs are caught in the earlier phases, it costs much less to fix them. That is why it is important to start with testing as soon as possible.

**. Types of testing**

* **Unit tests** - Unit tests are very low level and close to the source of an application. They consist in testing individual methods and functions of the classes, components, or modules used by your software. Unit tests are generally quite cheap to automate and can run very quickly by a continuous integration server.
* **Integration tests** - Integration tests verify that different modules or services used by your application work well together. For example, it can be testing the interaction with the database or making sure that microservices work together as expected.
* **Functional tests**- Functional tests focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action.
* **End to end tests**- End-to-end testing replicates a user behaviour with the software in a complete application environment.
* **Acceptance testing-** Acceptance tests are formal tests that verify if a system satisfies business requirements. They require the entire application to be running while testing and focus on replicating user behaviours.
* **Performance testing-** Performance tests evaluate how a system performs under a particular workload. These tests help to measure the reliability, speed, scalability, and responsiveness of an application.

**6.1 Testing Strategy**

The testing strategy for the web scraping system ensures the functionality, accuracy, performance, and reliability of the scraper across various websites and user interactions. The goal is to identify and fix bugs, validate the scraping logic, ensure compliance with website structures, and handle edge cases like CAPTCHA, redirections, or broken links.

✔️ Types of Testing Used:

* Unit Testing: Validates individual components such as HTML parsers, URL validators, and data extraction functions.
* Integration Testing: Tests the interaction between the scraping module, scheduler, and database.
* System Testing: Validates the entire flow—from user URL submission to data storage/export.
* User Acceptance Testing (UAT): Conducted with users to validate if the scraping meets their expectations and is easy to use.
* Error Handling Testing: Ensures the system handles invalid URLs, timeouts, and blocked pages gracefully.
* Performance Testing: Checks how the system performs under multiple scheduled jobs or large data sets.

**6.2 Test Cases**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Description | Input | Expected Output | Actual Output | Status |
| TC001 | **Submit valid URL for scraping** | [**https://movie96.com**](https://movie96.com) | **Extracted data displayed** | **Extracted data displayed correctly** | **✅ Passed** |
| TC002 | **Submit invalid URL** | [**https://invalid**](https://invalid) | **Show error "Invalid URL"** | **Error message shown** | **✅ Passed** |
| TC003 | **Scraping dynamic JS content** | [**https://jsheavy.com**](https://jsheavy.com) | **Load rendered content** | **Only static data scraped** | **❌ Failed (JS support missing)** |
| TC004 | **Schedule URL scraping job** | **Set scrape at 10:00 PM daily** | **Job created successfully** | **Job saved and appears in scheduler** | **✅ Passed** |
| TC005 | **Export scraped data to CSV** | **Click on “Export CSV”** | **CSV file downloaded** | **File downloaded successfully** | **✅ Passed** |
| TC006 | **Handle 404 pages** | **Submit a non-existing URL** | **Show “Page not found” or skip** | **Proper error message shown** | **✅ Passed** |
| TC007 | **Check for login-protected content** | **Submit login-required URL** | **Prompt for credentials or deny** | **"Access Denied" message** | **✅ Passed** |
| TC008 | **User views history of scraped URLs** | **Click “History”** | **Show list of previously scraped URLs** | **List displayed** | **✅ Passed** |

**7. Future Enhancement**

The Web Scraper can be enhanced in the future by implementing the following methods:

* **REST API Integration:**

Develop a comprehensive RESTful API to allow programmatic control of the scraper. This would enable third-party applications or developers to trigger scraping tasks, retrieve results, and integrate the scraper into larger workflows.

* **Cloud-Based Distributed Scraping:**

Transition to a cloud-based, distributed architecture to improve scalability and performance. This would involve leveraging cloud resources (e.g., AWS, Azure, or Google Cloud) to run concurrent scraping tasks across multiple nodes, thereby handling higher loads and reducing task execution times.

* **Advanced Analytics Dashboard:**

Build an interactive dashboard that presents detailed analytics on scraping performance, error rates, data trends, and user activity. This dashboard can help both administrators and users make data-driven decisions and monitor system health in real time.

* **Custom Scraping Rules Engine:**

Implement a flexible, rule-based engine that allows users to define custom rules and logic for scraping. This could include advanced filtering, conditional extraction based on dynamic criteria, and the ability to adapt to changes in website structures automatically**.**

* **Browser Extension Integration:**

Create a browser extension to facilitate an even more user-friendly experience. With a browser extension, users could select elements directly on any web page with a single click, streamlining the setup process for scraping tasks.

Enhanced Reporting and Analytics:

Introduce an advanced reporting module that provides detailed, real-time insights into the scraping operations. This module would feature interactive dashboards displaying key performance metrics, data trends, error rates, and historical comparisons. Users and administrators can leverage these insights to monitor system health, optimize scraping configurations, and make data-driven decisions for continuous improvement.

**8. References & Bibliography**

Book:

[1]. Grinberg, M. (2014) ‘Flask Web Development: Developing Web Applications with Python. Sebastopol’, CA: O'Reilly Media.

[2]. Grinberg, M. (2018) ‘Flask Web Development: Developing Web Applications with Python (2nd ed.). Sebastopol’, CA: O'Reilly Media.

[3]. Stouffer, J. (2015) ‘Mastering Flask. Birmingham’, UK: Packt Publishing.

[4]. Aggarwal, S. (2014) ‘Flask Framework Cookbook. Birmingham’, UK: Packt Publishing.

[5]. Dwyer, G. (2016) ‘Flask By Example. Birmingham’, UK: Packt Publishing.

[6]. Mitchell, R. (2015) ‘Web Scraping with Python: Collecting Data from the Modern Web. Sebastopol’, CA: O'Reilly Media.

[7]. Mitchell, R. (2018) ‘Web Scraping with Python: Collecting More Data from the Modern Web (2nd ed.). Sebastopol’, CA: O'Reilly Media.

[8]. Ronacher, A. (n.d.) 'Flask Documentation'. Retrieved [April 10, 2025] from https://flask.palletsprojects.com/en/latest/

[9]. Mozilla Developer Network. (n.d.) 'HTML: HyperText Markup Language'. Retrieved [April 10, 2025] from https://developer.mozilla.org/en-US/docs/Web/HTML

[10]. Mozilla Developer Network. (n.d.) 'CSS: Cascading Style Sheets'. Retrieved [April 10, 2025] from https://developer.mozilla.org/en-US/docs/Web/CSS

[11]. Nielsen, J. (1999) ‘Designing Web Usability. Indianapolis’, IN: New Riders Publishing.

[12]. Smith, A. (2020) ‘Modern Web Design and Development’. New York, NY: Web Publishers.

[13]. W3Schools. (n.d.) 'CSS Tutorial'. Retrieved [April 10, 2025] from <https://www.w3schools.com/css/>

Website:

1. <https://www.w3schools.com>
2. <https://www.tutorialspoint.com>
3. <https://www.geeksforgeeks.org>
4. https://pypi.org/project/beautifulsoup4