# User Requirements Document – Price Prediction Website

## 1. Project Overview

This website provides a platform for merchants and consumers to view and predict product prices. Merchants can upload historical price data for products, and consumers can view product price trends and system-generated price forecasts based on historical data.

## 2. User Roles

- Merchant Staff: Responsible for uploading historical product price data to support the system in generating price forecasts.

- Consumers: Can search for products of interest, view their historical price trends, and system-generated future price predictions.

## 3. Functional Requirements

### 3.1 Introductory View

Objective: Inform users about the platform’s main functions and provide registration/login access.

Features:

• Display the platform’s goals and features.  
• Provide user registration and login access.  
• Allow users to select an account type (Merchant or Consumer).

### 3.2 Upload Data View

Objective: Allow merchant users to upload historical product price data.

Features:

• Support batch file uploads (CSV format containing product ID, date, price, etc.).  
• Validate uploaded file format and display data preview.  
• Show upload progress and completion notification.  
• Provide a data management interface where merchants can view, update, or delete uploaded data.

### 3.3 Visualize Data View

Objective: Allow both merchants and consumers to view historical price charts and price forecasts.

Features:

• Search bar: Consumers can search by product name or ID.  
• Charts:  
 - Display historical price line charts.  
 - Display forecasted price charts.  
• Interactive chart features:  
 - Hover to see specific data point details.  
 - Zoom and pan.  
• Offer forecasts for different time ranges (e.g., 7 days, 30 days).

### 3.4 Share Data View

Objective: Allow users to share their data or forecast results with other users.

Features:

• Share charts of certain product via link or social platforms.  
• Display shared data records and allow cancellation.

## 4. Non-functional Requirements

### 4.1 Performance Requirements

• The system should be capable of handling large files (e.g., CSV files with more than 1000 rows) during data upload and provide quick feedback (or other input methods???).  
• Page load time within 3 seconds; chart generation within 5 seconds.

### 4.2 Security Requirements

• Passwords correctly stored as salted hashes using strong algorithms.  
• Use of CSRF tokens to prevent Cross-Site Request Forgery attacks on form submissions.  
• All sensitive configuration data, such as database passwords and API keys, should be correctly stored in configuration files and managed using environment variables. Hardcoding sensitive data in code is prohibited.

### 4.3 Usability Requirements

• The website should support mainstream browsers (Chrome, Firefox, Safari, Edge).  
• Accessible UI for users with visual impairments (optional).

### 4.4 Maintainability Requirements (NEED YOUR OPINIONS)

• Modular front-end and back-end design.  
• User activity and error logging.

## 5. User Interface Design (UI Design)

A few ideas for your reference:

* The registration and login buttons should be prominent, with clear guidance for users.
* Provide an intuitive upload area (e.g., drag-and-drop) and display file format requirements.
* After upload, the system should show a successful upload message and a preview of the data.
* Provide a clear entry for users to share their data.
* Users should be able to choose what content to share and with whom (e.g., via social media or user accounts).

## 6. Technology Stack

| Technology | Description | Rubric |
| --- | --- | --- |
| HTML |  | Valid HTML code, using a wide range of elements, clearly organised with appropriate use of Joomla templates. |
| CSS |  | Valid, maintainable code, using a wide range of custom selectors and classes, web page is reactive to screen size. |
| Javascript |  | Valid, well formatted code, including validation and DOM manipulation/AJAX that uses JavaScript best practices. |
| Bootstrap/Tailwind/  SemanticUI/Foundation | Choose one CSS framework to build responsive layouts and UI components. |  |
| JQuery | Simplifies JavaScript operations and handles DOM events. |  |
| Flask | Backend web framework for handling API requests, user management, data processing, etc. | Formatted, commented and well organised code that responds to requests by the client by performing non-trivial data manipulation and page generation operations. |
| AJAX/Websockets | Used for asynchronous communication and real-time data updates between the front-end and back-end. |  |
| SQLite | Used for data storage, storing historical price data, user information, etc. Accessed via SQLAlchemy. | Well considered database schema, good authentication, and maintainable models. Some evidence of DB migrations. |
| SQLAlchemy | Python ORM framework for interacting with the database. |  |

## 7. Limits

• Frameworks like React/Angular, MySQL, SASS are not allowed.  
• Allowed to use non-core JS/Python libraries (e.g., for charts, ChatGPT bindings).

## 8. GitHub Repository & README Requirements

The private GitHub repository must include a README.md file containing:

• Purpose Description: A clear explanation of the purpose of the application, its intended users, and the design philosophy (engaging, effective, intuitive).

• Team Member Table:  
 | UWA ID | Name | GitHub Username |  
 |--------|------|------------------|  
 | (example) 12345678 | Jane Doe | janedoe |

**9. Test Instructions:**

Comprehensive test suite, containing 5+ unit tests and 5+ selenium tests. The latter should run with a live version of the server.

## 10. Other Requirements

• Collect user feedback regularly (optional).  
• Provide basic technical support (optional).