Contents

[Zero-Maintenance E-Commerce Site Technical Guide 1](#_Toc153672469)

[Introduction 1](#_Toc153672470)

[PART 1.1 Backend Solution - Scrapping Logic 1](#_Toc153672471)

[Setup and Configuration 1](#_Toc153672472)

[Understanding the Scrapping Logic 2](#_Toc153672473)

[Running the Flask App 2](#_Toc153672474)

[Using the API 2](#_Toc153672475)

[PART 1.2 Backend Solution - Operational Logic (logic.py) 3](#_Toc153672476)

[Overview 3](#_Toc153672477)

[Key Features 3](#_Toc153672478)

[Setup and Configuration 3](#_Toc153672479)

[Detailed Functionality 3](#_Toc153672480)

[User Authentication 3](#_Toc153672481)

[User Registration 3](#_Toc153672482)

[Purchase Handling 4](#_Toc153672483)

[Username Availability 4](#_Toc153672484)

[Best Practices in logic.py 4](#_Toc153672485)

[PART 1.3 Backend Solution - Database Structure (Azure SQL DB) 4](#_Toc153672486)

[Database Tables and Relationships 5](#_Toc153672487)

[Relationships 6](#_Toc153672488)

[Schema 7](#_Toc153672489)

[Considerations 7](#_Toc153672490)

[PART 2.1 Frontend Solution - Index HTML Overview 8](#_Toc153672491)

[Structure of the Index HTML 8](#_Toc153672492)

[PART 2.2 Frontend Solution - Header HTML Overview 9](#_Toc153672493)

[Structure of the Header HTML 9](#_Toc153672494)

[PART 2.3 Frontend Solution - Register HTML Overview 11](#_Toc153672495)

[Structure of the Register HTML 11](#_Toc153672496)

[PART 2.4 Frontend Solution - Product Filter HTML Overview 12](#_Toc153672497)

[Structure of the Product Filter HTML 12](#_Toc153672498)

[PART 3.1 Frontend Solution - JavaScript Overview 14](#_Toc153672499)

[Closing Thoughts on the Zero-Maintenance E-Commerce Site Guide 16](#_Toc153672500)

[Embracing the Proof-of-Concept Stage 16](#_Toc153672501)

[Acknowledging the Gaps 16](#_Toc153672502)

[Future Directions 16](#_Toc153672503)

# Zero-Maintenance E-Commerce Site Technical Guide

# Introduction

Welcome to the technical guide for the "Zero-Maintenance E-Commerce Site." This project is a proof-of-concept e-commerce website that automates product scraping from major e-commerce platforms. It's designed to support dropshipping businesses by showcasing products they don't own, making it easier for them to expand their product catalog without manual effort.

This solution consists of two main components: **scrapping.py** and **logic.py**, both part of the backend system. The **scrapping.py** app handles product scraping, providing APIs for various product types, while **logic.py** manages user-related operations such as login/logout, registration, purchases, etc. All data is stored and managed in an SQL Azure Database.

# PART 1.1 Backend Solution - Scrapping Logic

## Setup and Configuration

Prerequisites:

* Python 3.x
* Flask
* BeautifulSoup4
* Requests
* Flask-CORS
* pyODBC

Installation:

1. Install Python 3.x from [the Python website](https://www.python.org/downloads/).
2. Install necessary libraries using pip:   
   

## Understanding the Scrapping Logic

The scrapping logic is implemented in **scrapping.py**. It's responsible for scraping product data from predefined e-commerce categories and serving this data through Flask APIs.

Key Components:

1. **Flask Setup**: The Flask app is set up with CORS (Cross-Origin Resource Sharing) enabled to allow requests from different origins.
2. **Scrape Category Function**: This function (**scrape\_category**) takes a category URL and its name, scrapes the webpage for product data, and returns a list of product dictionaries.
3. **API Endpoints**:
   * **/api/search**: Allows searching for products across all categories based on a query string.
   * **/api/<category\_name>**: Retrieves all products for a specific category.
   * **/api/all**: Merges all product data from other APIs into a single JSON response.
4. **Database Connection**: The **get\_db\_connection** function sets up the connection to the SQL Azure Database using environment variables for security.

## Running the Flask App

To run the Flask application:

1. Set the environment variables for the database connection.
2. Execute the script:

python scrapping.py

## Using the API

* **Search Products**: To search for products, send a GET request to **/api/search** with a query parameter, e.g., **http://localhost:5020/api/search?query=phone**.
* **Get Category Data**: To retrieve data for a specific category, send a GET request to **/api/<category\_name>**, e.g., **http://localhost:5020/api/phones%20&%20accessories**.
* **Get All Products**: To get all products from all categories, send a GET request to **/api/all**.

**Next Steps**

In the next sections, we'll dive into the **logic.py** script, which handles the operational logic of the website, including user authentication, order processing, and other functionalities.

# PART 1.2 Backend Solution - Operational Logic (logic.py)

## Overview

The **logic.py** script is a crucial part of the backend, managing user interactions, authentication, and purchase operations. It's built using Flask and integrates with an SQL Azure Database.

## Key Features

1. **User Authentication**: Includes endpoints for user login, logout, and registration.
2. **Admin Registration**: Special endpoint for registering admin users.
3. **Purchase Handling**: Endpoint for processing purchase requests.
4. **Username Availability Check**: Endpoint to check if a username is already taken.

## Setup and Configuration

Prerequisites:

* Similar to **scrapping.py**, ensure all required dependencies are installed.

Running the Script:

* Run **logic.py** as a Flask app:

python logic.py

## Detailed Functionality

### User Authentication

* **Login Endpoint** (**/login**):
  + Validates user credentials.
  + Sets up user session upon successful login.
* **Logout Endpoint** (**/logout**):
  + Removes user session details, effectively logging the user out.

### User Registration

* **Register Endpoint** (**/register**):
  + Handles new user registration.
  + Validates input and checks for existing usernames.
  + Passwords are hashed for security.
* **Register Admin Endpoint** (**/register\_admin**):
  + Specifically for registering admin users.
  + Similar validation as regular user registration.

### Purchase Handling

* **Purchase Endpoint** (**/api/purchases**):
  + Processes purchase requests.
  + Stores order details in the database.
  + Manages user information related to the purchase.

### Username Availability

* **Check Username Endpoint** (**/check-username**):
  + Checks if a provided username is already in use.
  + Returns a boolean indicating availability.

### Best Practices in logic.py

* **Security**: Passwords are hashed using Werkzeug's **generate\_password\_hash**.
* **Session Management**: Flask sessions are used to track logged-in users.
* **Error Handling**: Proper error handling and response formatting for API endpoints.
* **Database Connections**: Database connections are managed using context managers for safety.
* **Input Validation**: Basic validation checks for user input.
* **CORS Support**: CORS is configured for frontend integration.

**Next Steps**

We will dive into the DB structure and Schema.

# PART 1.3 Backend Solution - Database Structure (Azure SQL DB)

**Overview**

The Azure SQL Database is a fundamental part of the Zero-Maintenance E-Commerce site, storing all the necessary data for managing orders, products, and user information. Let's break down the structure of the database and its various tables.

## Database Tables and Relationships

1. dbo.order\_products

* **Description**: Links orders to products and stores quantity and price for each product in an order.
* **Fields**:
  + **order\_id** (int, Primary Key, Foreign Key): Links to **dbo.orders**.
  + **product\_id** (int, Primary Key, Foreign Key): Links to **dbo.products**.
  + **quantity** (int): The quantity of the product in the order.
  + **price** (decimal): The price of the product at the time of the order.

2. dbo.user\_info

* **Description**: Stores information about the users placing orders.
* **Fields**:
  + **user\_info\_id** (int, Primary Key): Unique identifier for user info.
  + **name** (nvarchar(255)): User's name.
  + **address** (nvarchar(500)): User's address.
  + **email** (nvarchar(255)): User's email.
  + **phone** (nvarchar(20)): User's phone number.

3. dbo.users

* **Description**: Manages user accounts, including authentication and admin status.
* **Fields**:
  + **user\_id** (int, Primary Key): Unique identifier for users.
  + **username** (nvarchar(50)): Username for login.
  + **password** (nvarchar(255)): Hashed password for user authentication.
  + **is\_admin** (bit): Flag to indicate if the user is an administrator.

4. dbo.products

* **Description**: Contains information about the products available for sale.
* **Fields**:
  + **product\_id** (int, Primary Key): Unique identifier for products.
  + **title** (nvarchar(255)): Product title.
  + **picture\_url** (nvarchar(500)): URL of the product image.
  + **price** (decimal): Product price.
  + **link** (nvarchar(500)): Link to the product page.

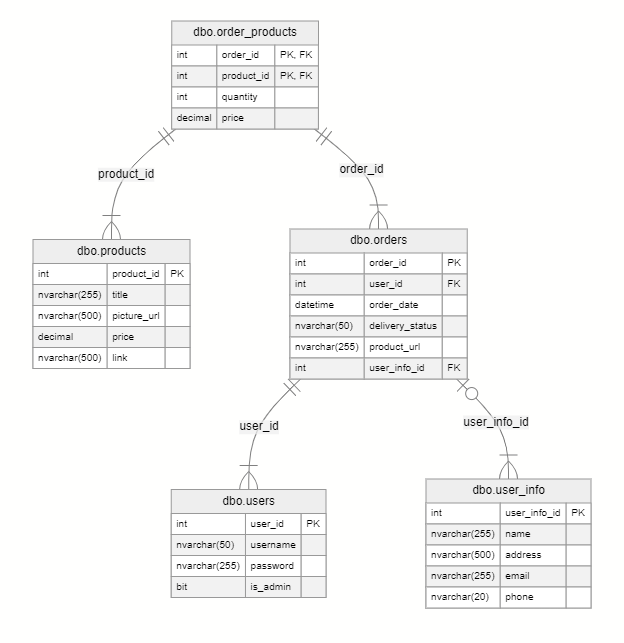
5. dbo.orders

* **Description**: Manages the orders placed by users.
* **Fields**:
  + **order\_id** (int, Primary Key): Unique identifier for orders.
  + **user\_id** (int, Foreign Key): Links to **dbo.users**.
  + **order\_date** (datetime): Date and time when the order was placed.
  + **delivery\_status** (nvarchar(50)): Current status of the order's delivery.
  + **product\_url** (nvarchar(255)): URL of the product.
  + **user\_info\_id** (int, Foreign Key): Links to **dbo.user\_info**.

### Relationships

* **dbo.order\_products** relates to **dbo.orders** and **dbo.products** via **order\_id** and **product\_id** respectively.
* **dbo.orders** has a foreign key relationship with **dbo.users** through **user\_id** and with **dbo.user\_info** through **user\_info\_id**.

### Schema



### Considerations

* **Normalization**: The database is designed to avoid redundancy and ensure data integrity.
* **Indexing**: Proper indexing on primary and foreign keys for performance optimization.
* **Security**: Sensitive data like passwords are stored in hashed form.
* **Scalability**: The structure is scalable and can handle increased data volume as the e-commerce site grows.

**Next Steps**

With the database structure defined, we can proceed to the Front-End speciation.

# PART 2.1 Frontend Solution - Index HTML Overview

## Structure of the Index HTML

The Index page serves as the homepage for your "Zero-Maintenance E-Commerce Site". It's designed with a focus on user experience and branding consistency.

Key Components:

1. **Header Section**: Reserved for navigation and branding elements. The header is loaded through a placeholder **<div id="header-placeholder"></div>**.
2. **Main Content Area**:
   * **Featured Products Section**: Highlights selected products. Products will be dynamically loaded into this section.
   * **Carousel Section**: Showcases products or promotions in a sliding carousel format. This is dynamically handled by JavaScript.
3. **Footer**: Contains copyright and any additional information relevant to your site.
4. **External Resources**:
   * **IndexStyles.css**: This CSS file ensures the page adheres to your cohesive branding guidelines.
   * **script.js**: Centralizes various scripts used across different pages for consistent functionality.

**Key HTML Elements**

* **<meta>** tags for character set and responsive design.
* **<link>** to attach the CSS file.
* **<script>** tags for JavaScript integration, both inline for specific actions and external for shared functionality.

**JavaScript Integration**

* An inline JavaScript snippet welcomes back users if their username is stored in **localStorage**.
* **script.js** is linked at the bottom of the body, ensuring all HTML elements are loaded before any JavaScript manipulation occurs.

**Styling and Responsiveness**

* CSS is used not only for branding but also to ensure the page is responsive and accessible on various devices.
* The carousel controls (**prev** and **next**) are integrated into the design for user interaction.

**Best Practices**

* **Separation of Concerns**: HTML for structure, CSS for styling, and JavaScript for interactivity.
* **Accessibility**: Proper use of HTML semantic elements and responsive design.
* **Performance Optimization**: Loading JavaScript at the end of the body for faster page rendering.
* **User Experience**: Engaging features like carousels and personalized greetings enhance the user experience.

**Next Steps**

We'll delve into other pages like the Register page, Admin Console, Product Filter pages, and their respective CSS styles to maintain cohesive branding across the site.

# PART 2.2 Frontend Solution - Header HTML Overview

## Structure of the Header HTML

The Header HTML is a crucial component of your "Zero-Maintenance E-Commerce Site", providing a consistent and user-friendly navigation experience across all pages.

Key Components:

1. **Logo**: A clickable image that redirects users to the homepage (**index.html**). It enhances brand recognition.
2. **Search Bar**: Allows users to search for products. This feature is essential for a seamless user experience in an e-commerce setting.
3. **Navigation Links**: Quick access to the Home page and the Product Filter page. These links ensure easy navigation through the site.
4. **Login/Logout Button**: A dynamic button for user authentication, which toggles between 'Login' and 'Logout' based on user state.
5. **Login Modal**:
   * **Tabbed Interface**: Separates 'User Login' and 'Admin Login' for clarity.
   * **User Login Form**: Collects username and password for user login.
   * **Admin Login Form**: Similar to the user login but for administrators.
   * **Register Link**: Redirects new users to the registration page.
6. **External Resources**:
   * **HeaderStyles.css**: Styles the header components for consistent design across the site.
   * **script.js**: Manages the dynamic functionalities like opening and closing modals, handling login, etc.

**HTML Elements and Attributes**

* **<meta>** tags for ensuring proper character encoding and responsiveness.
* **<link>** for attaching the CSS stylesheet.
* **<nav>** for semantic structuring of navigation links.
* **onclick** JavaScript event handlers for interactive elements like buttons and modal close icons.

**JavaScript and CSS Integration**

* The header relies on **script.js** for handling user interactions, particularly for the login modal and tabbed login interface.
* **HeaderStyles.css** ensures the header's look and feel aligns with the overall branding of the site.

**Best Practices**

* **Modular Design**: The header is separate, allowing for easy inclusion across different pages without redundancy.
* **Responsive Design**: Ensures the header is functional and visually appealing on different devices.
* **Accessibility**: Use of semantic HTML and alt tags for images.
* **User-Friendly Interface**: Intuitive navigation and clear distinction between user and admin login.

**Next Steps**

The next phase will involve detailing other pages like the Register page, Admin Console, and Product Filter page, ensuring they all integrate seamlessly with the header for a consistent user experience.

# PART 2.3 Frontend Solution - Register HTML Overview

## Structure of the Register HTML

The Register page is an essential component for user onboarding in your "Zero-Maintenance E-Commerce Site". It's designed to be user-friendly and secure, ensuring a smooth registration process for new users.

Key Components:

1. **Header Placeholder**: To include the common header across the site for consistent navigation.
2. **Main Content Area**:
   * **Sign Up Modal/Form**: The primary focus of this page where users can create their accounts.
   * Form fields for username and password, along with a password confirmation field.
3. **Terms and Conditions Checkbox**: A necessary component for legal compliance and user agreement.
4. **Registration Form Submission Handling**: A JavaScript snippet to manage the form submission process, including validation and interaction with the backend.
5. **External Resources**:
   * **RegisterStyle.css**: Provides styling specific to the registration page, ensuring it aligns with the site's overall design.
   * **script.js**: Includes shared JavaScript functionalities that might be used across different pages.

**HTML Elements and Attributes**

* **<form>** with **id="registration-form"** for easy JavaScript access.
* Input fields for username, password, and password confirmation.
* A checkbox for terms and conditions agreement.
* A submit button to send the form data.

**JavaScript Integration**

* Inline JavaScript for handling form submissions, including:
  + Preventing the default form submission action.
  + Validating password match.
  + Making a POST request to the backend's **/register** endpoint.
  + Handling the response and providing user feedback.
  + Redirecting the user upon successful registration.

**Styling and User Experience**

* The CSS file **RegisterStyle.css** ensures that the registration form is visually appealing and easy to use.
* Form fields and buttons are styled for better user experience and visual consistency with the rest of the site.

**Best Practices**

* **Client-Side Validation**: JavaScript is used for initial validation like password matching, enhancing user experience and reducing unnecessary server requests.
* **Security**: Passwords are handled securely, not exposed in the URL or any insecure manner.
* **User Feedback**: Clear messages are displayed to the user in case of successful registration or errors.
* **Responsive Design**: Ensures the registration page is accessible and functional on various devices.

**Next Steps**

With the registration page set up, the next step involves integrating it with the backend to handle user data securely and efficiently. Additionally, focusing on other pages like the Admin Console and Product Filter pages will be crucial for the complete functionality of the site.

# PART 2.4 Frontend Solution - Product Filter HTML Overview

## Structure of the Product Filter HTML

The Product Filter page is a key feature of your "Zero-Maintenance E-Commerce Site," enabling users to browse and filter products according to their preferences.

Key Components:

1. **Category Bar**:
   * Dynamically populated list of categories for users to choose from.
   * JavaScript event listeners to highlight the selected category.
2. **Product and Checkout Modals**:
   * **Product Modal**: Displays product details and a 'Buy Now' button.
   * **Checkout Modal**: Facilitates the order process, including a form for user details and a confirmation button.
3. **Filter Bar**:
   * A range slider and input for price filtering.
   * A dropdown for sorting products (e.g., by price).
4. **Product Grid**:
   * The main area where products will be dynamically loaded.
   * Includes a loading screen with an animation to enhance user experience during data fetching.
5. **External Resources**:
   * **ProductFilterStyles.css**: Custom CSS to style the product filter page.
   * **script.js**: Contains shared JavaScript functionalities across the site.

**HTML Elements and Attributes**

* Navigation bar (**<nav>**) with an unordered list (**<ul>**) for categories.
* Input elements for filtering and sorting products.
* Div containers for modals, including product details and checkout forms.
* Main content area for displaying the product grid.

**JavaScript Integration**

* Inline JavaScript scripts are used for:
  + Handling category selection and highlighting.
  + Opening and closing modals.
  + Redirecting to the registration page when necessary.
* Functions are defined for updating the price range and sorting products.

**Styling and User Experience**

* The CSS file ensures that the page is visually appealing and maintains consistency with the site's branding.
* The layout is designed to be intuitive and easy to navigate, enhancing the user shopping experience.

**Best Practices**

* **Modular Design**: Separates concerns by having distinct sections for categories, filters, and product display.
* **User Interaction**: Enhances engagement through interactive elements like modals, range sliders, and sorting options.
* **Performance Optimization**: Includes a loading screen to manage user expectations during data retrieval.
* **Accessibility and Responsiveness**: Ensures that the page is accessible and usable across different devices.

# PART 3.1 Frontend Solution - JavaScript Overview

The JavaScript code for your "Zero-Maintenance E-Commerce Site" plays a crucial role in enhancing the interactivity and functionality of the website. Let's dive into the key aspects of your script.

**Header Interaction**

* **Dynamic Header Loading**: The script fetches **header.html** and injects it into **#header-placeholder**. This approach ensures header consistency across different pages.
* **Search Button Listener**: The script attaches an event listener to the search button, triggering the **handleSearch** function upon a click event.

**Login Modal and Tab Interaction**

* **Tab Control in Modals**: Functions **openTab**, **openLoginModal**, and **closeLoginModal** manage the display of user and admin login forms within a modal.
* **Global Click Listener**: Closes modals when clicking outside, enhancing UX.

**Login and Logout Functionality**

* **Form Submission Handling**: **handleUserLogin** and **handleAdminLogin** prevent default form submission, gather form data, and call **sendLoginRequest**.
* **Login Request**: Sends AJAX requests to the backend's **/login** endpoint, handling user and admin logins separately.
* **Session Management**: Utilizes **sessionStorage** to maintain login states and user information.

**Product Filtering and Display**

* **Category Selection**: Populates categories and attaches event listeners for category selection.
* **Product Fetching and Display**: Functions **fetchAllProducts** and **fetchProducts** retrieve products from the backend and display them in the product grid.
* **Price Filtering and Sorting**: **updatePriceRange** and **sortProducts** allow users to filter and sort products based on price.

**Modals for Product Details and Checkout**

* **Product Detail Modal**: Displays product details in a modal upon clicking a product.
* **Checkout Modal**: Manages the checkout process, including user details form and order submission.

**Carousel Functionality**

* **Featured Products Carousel**: Dynamically loads featured products into a carousel.
* **Carousel Navigation**: Functions **showSlides** and **nextSlide** control the display and navigation of carousel items.

**Miscellaneous**

* **MutationObserver**: Monitors for changes in the DOM to update the login/logout button dynamically.
* **Event Listeners**: Various event listeners for modal interactions, form submissions, and user interactions.

**Best Practices**

* **Modular Functions**: Each functionality is encapsulated in specific functions, making the code organized and maintainable.
* **Asynchronous Operations**: Utilizes **async/await** for fetching data, improving readability and handling of asynchronous code.
* **Error Handling**: Includes proper error handling for network requests and DOM interactions.
* **User Experience**: Focuses on smooth and intuitive user interactions through modal controls, dynamic content loading, and responsive features like the carousel.

# Closing Thoughts on the Zero-Maintenance E-Commerce Site Guide

As we conclude this guide, it's essential to recognize that while the Zero-Maintenance E-Commerce Site is currently a proof-of-concept, it embodies a significant step towards realizing a nearly effortless dropshipping model through web scraping.

## Embracing the Proof-of-Concept Stage

The project, at this stage, is a testament to innovative thinking in the e-commerce space. The idea of automating product scraping to facilitate a dropshipping business with minimal effort is both ambitious and forward-thinking. This proof-of-concept lays the groundwork for a potentially transformative approach to online retail.

## Acknowledging the Gaps

Being a proof-of-concept, there are naturally gaps that need to be addressed before considering a full-scale production deployment. These include:

* **Legal and Ethical Considerations**: Ensuring compliance with data scraping laws and the terms of service of source websites.
* **Scalability and Reliability**: While the concept works on a smaller scale, it's crucial to test and ensure scalability and reliability as the user base and product listings grow.
* **User Experience Refinement**: Continued development and user testing are necessary to refine the UX, making it intuitive and seamless.
* **Integration with Payment Gateways**: For a fully functional e-commerce site, secure and reliable payment integration is a must.
* **Robust Error Handling and Logging**: Essential for diagnosing issues in a live environment.

## Future Directions

Moving forward, some considerations for evolving this project include:

* **Exploring Advanced Scraping Techniques**: Adapting to dynamic content and implementing more sophisticated scraping algorithms.
* **Developing a Robust Backend Infrastructure**: Ensuring the backend can handle increased loads and complex data processing.
* **Enhancing Security Measures**: Particularly around user data and transactions.
* **Automating Content Updates**: Implementing mechanisms to keep product listings current and accurate.