Low Level Design

Shopping Cart Web Application

|  |  |
| --- | --- |
| Written By | Vishal Katkar |
| Last Revised Date | 20– July-2023 |

**Document Control**

### Change Record:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 0.1 | 20 – july -  2023 | Vishal Katkar | Introduction & Architecture defined, Discription, Unit test cases defined and appended |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Reviews:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Reviewer** | **Comments** |
| 0.1 | 20– july-  2023 | Vishal Katkar | Document Content , Version Control and Unit Test Cases to be added |

### Approval Status:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Review**  **Date** | **Reviewed By** | **Approved By** | **Comments** |
|  |  |  |  |  |

Contents

1. [Introduction 1](#_bookmark0)
   1. [What is Low-Level design document? 1](#_bookmark1)
   2. [Scope 1](#_bookmark2)
2. [Architecture 2](#_bookmark3)
3. [Architecture Description 3](#_bookmark4)
   1. [Database Design 3](#_bookmark6)

[a. Customer Registration Collection ("registers"): 3](#_bookmark7)

[b. Shops Collection ("shops"): 3](#_bookmark7)

* 1. [Authentication Mechanism 4](#_bookmark8)

[a. Password Hashing with Bcrypt.js: 4](#_bookmark7)

[b. Token-Based Authentication with JSON Web Tokens (JWT): 4](#_bookmark7)

1. [Unit Test Cases 5](#_bookmark20)

# Introduction

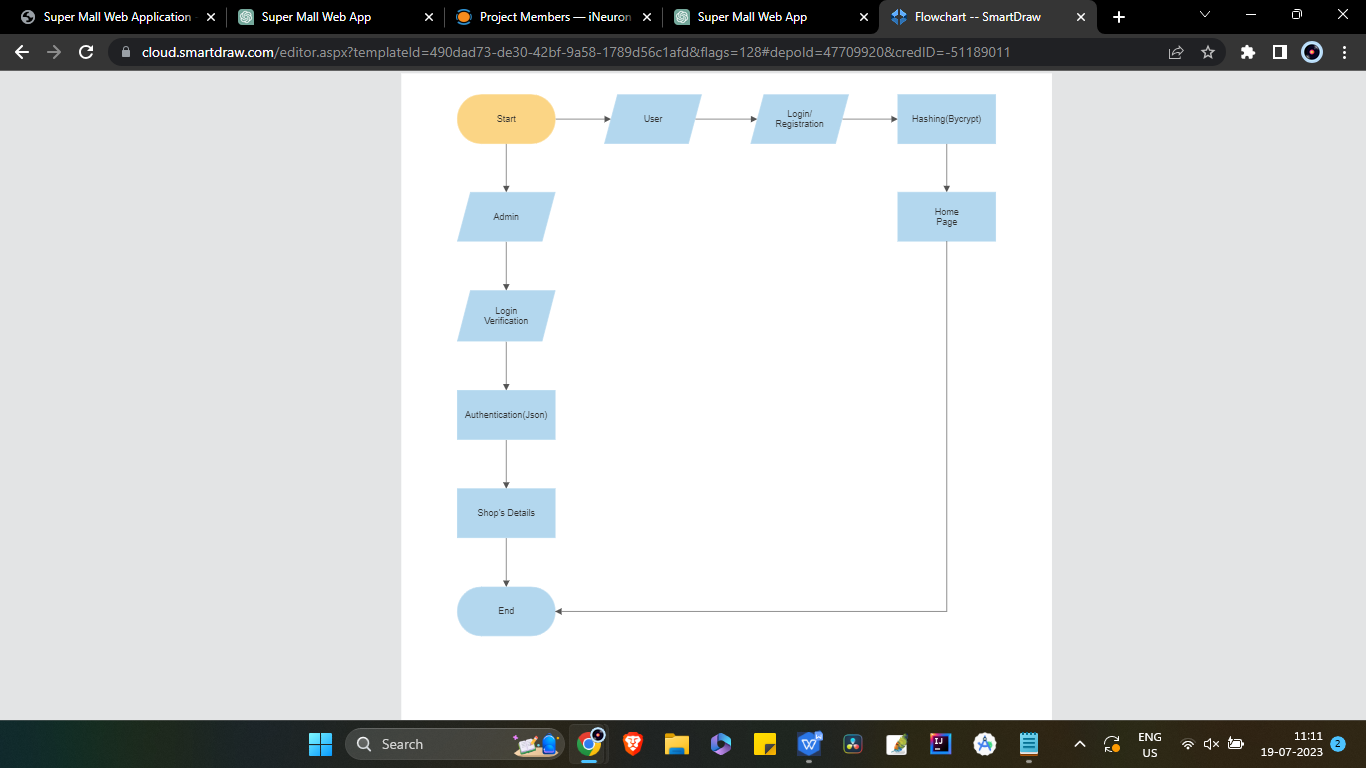
## What is Low-Level design document?

The Low-Level Design (LLD) document is a detailed technical document that describes the internal design of the Shopping Cart Web Application. It focuses on the implementation details of the system and provides a blueprint for developers to follow during the coding phase. The LLD document complements the high-level design and serves as a bridge between the system requirements and the actual code.

## Scope

The scope of this Low-Level Design document is to provide an in-depth understanding of the architecture, components, and implementation details of the Shopping Cart Web Application. It covers the design considerations and decisions made for each component of the system, including the React.js, Redux , Node.js , Html, css, JavaScript etc and unit testing.

# Architecture



# Architecture Description

* The architecture of the Shopping Cart Web Application can be described as a three-tier architecture. The presentation layer is built using handle bar templates, css, which handles the user interface and user interactions. The business logic layer is implemented using Express, which processes requests, interacts with the database, and enforces business rules.
* In addition, the application employs the Handlebars template engine for rendering dynamic views on the server-side.
* Ihe authentication mechanism includes hashing passwords using Bcrypt.js and token-based authentication using JSON Web Tokens (JWT).

## Database Design:

1. Customer Registration Collection ("registers"):

The "registers" collection holds customer registration data. It includes attributes such as customer name, email, password, and other relevant information. The schema for the "registers" collection defines the fields and their data types, ensuring consistent and structured data storage.

1. Shops Collection ("shops"):

The "shops" collection stores information about the different shops within the Super Mall. It includes attributes such as shop name, location, contact details, product offerings, and any special offers. The schema for the "shops" collection defines the fields and their data types, allowing for efficient retrieval and manipulation of shop data.

## Template Engine: Handlebars

* The Shopping Cart Web Application utilizes the Handlebars template engine for rendering dynamic views on the server-side. Handlebars allows for the creation of reusable templates that can be populated with data and rendered into HTML pages.
* The template engine is integrated into the Express application, allowing for the separation of the presentation logic from the business logic. Handlebars templates contain HTML structure with embedded expressions and placeholders for dynamic data. These templates can be extended, reused, and populated with data from the backend to generate dynamic content.
* Handlebars provides features such as conditionals, loops, partials, and helpers, which enhance the flexibility and functionality of the views. It enables the application to generate dynamic pages based on user input, data retrieved from the database, or any other relevant sources.

## Authentication Mechanism:

To secure the Shopping Cart Web Application, a robust authentication mechanism is implemented using Bcrypt.js for password hashing and JSON Web Tokens (JWT) for token-based authentication.

1. Password Hashing with Bcrypt.js:When a user registers or updates their password, Bcrypt.js is used to hash the password. The hashed password is then stored in the "registers" collection. Bcrypt.js utilizes a one-way hashing algorithm, making it computationally expensive to reverse-engineer the original password from the stored hash. This ensures the security of user passwords even if the database is compromised.
2. Token-Based Authentication with JSON Web Tokens (JWT):

Upon successful authentication, the application generates a JSON Web Token (JWT) and sends it to the client. The client includes this token in subsequent requests to authenticate and authorize access to protected routes. The server verifies the authenticity of the token, extracts the user's identity from it, and grants access to the requested resources if the token is valid. JWT provides a stateless and scalable authentication mechanism, eliminating the need for server-side session management.

Architecture Summery

The architecture design of the Shopping Cart Web Application combines the power of the utilization of the Handlebars template engine for server-side rendering. The Express-based backend, with the app.js file as the entry point, manages the server configuration, routing, and middleware. The MongoDB database stores customer registration and shop-related data in separate collections ("registers" and "shops"). The authentication mechanism employs Bcrypt.js for secure password hashing and JSON Web Tokens (JWT) for token-based authentication.

By leveraging these architectural components and design choices, the Super Mall Web Application can deliver a secure, scalable, and user-friendly experience. The separation of concerns between the backend and frontend, combined with the flexibility of Handlebars templates, ensures maintainability, code reusability, and the ability to handle dynamic data, presentation requirements, and secure authentication effectively.

The use of the Handlebars template engine enhances the development process by separating the presentation logic from the backend. This enables the reuse of templates, resulting in cleaner code and improved maintainability. Handlebars' features such as conditionals, loops, partials, and helpers enable the creation of dynamic and customized views.

To ensure the security of user data, the application employs Bcrypt.js for password hashing. This cryptographic hashing algorithm provides a high level of security, protecting user passwords even in the event of a data breach. Additionally, token-based authentication using JSON Web Tokens (JWT) adds an extra layer of security to protect sensitive routes and resources.

By incorporating these architectural components and design choices, the Super Mall Web Application can provide a seamless and secure shopping experience. The separation of concerns, scalability, and flexibility offered by the mongoDb, Node, express, html, css etc, along with the added security measures, ensure the application meets the requirements of the industry domain and delivers a reliable and user-friendly platform.

Overall, the chosen architecture and design considerations ensure the Super Mall Web Application is well-equipped to handle the complexities of managing shop details, offers, products, and user authentication. It sets the foundation for a successful and scalable web application that can cater to the needs of both shop owners and customers, providing a seamless and secure shopping experience.

# Unit Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case Description** | **Pre-Requisite** | **Expected Result** |
| Verify whether the Application URL is  accessible to the user | 1. Application URL  should be defined | Application URL should be  accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed | 1. Application URL is accessible 2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether the User is able to sign  up in the application | 1. Application is  accessible | The User should be able to sign up  in the application |
| Verify whether user is able to successfully login to the application | 1. Application is accessible 2. User is signed up to the application | User should be able to successfully login to the application |
| Verify whether user is able to see input fields on logging in | 1. Application is accessible 2. User is signed up to the application 3. User is logged in   to the application | User should be able to see input fields on logging in |
| Verify whether user is able to edit all input fields | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be able to edit all input fields |
| Verify whether user gets Submit button to submit the inputs | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should get Submit button to submit the inputs |
| Verify whether user is presented with recommended results on clicking  submit | 1. Application is accessible 2. User is signed up to the application 3. User is logged in   to the application | User should be presented with recommended results on clicking  submit |
| Verify whether the recommended results are in accordance to the selections user made | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | The recommended results should be in accordance to the selections user made |
| Verify whether user has options to filter the recommended results as well | 1. Application is accessible 2. User is signed up | User should have options to filter the recommended results as well |