Ice Cream Shop Advertisement Website

**Submitted**

By

S. T. Anbarasu (20cos104)

Under the Guidance of – Mr.C. B. Sudharsen, MCA 2016

Submitted to

-The Department of Computer Science (UG)

FOR THE PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF



THE AMERICAN COLLEGE

(An Autonomous Institution Affiliated to Madurai Kamaraj University)Re-accredited with (2 cycle ) by NAAC with Grade "A", CGPA-3.46 on a 4-Points scaleMADURAI-625002

BONAFIDE CERTIFICATE

This is to certify that the Project Report entitled "Website Development For Ice Cream Store Advertisement Website" is the

bona fide record work done by S. T. Anbarasu (20cos104) in partial fulfilment for the award of the

degree B .Sc (Computer Science), American College, Madurai.

Submitted for the viva-voce Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Internal Guide Head of the Department

DECLARATION

I hereby declare that the project work entitled "Website Development For Ice Cream Store Advertisement Website" is a project report of the original work done by myself. This project work is submitted to American College

(Affiliated to Madurai Kamaraj University) in partial fulfilment of the degree of Bachelor of Science

during the academic year 2020-2023.

I declare that this dissertation or any part thereof has not been submitted for getting any degree or diploma from any university or college.

Place: Signature Date:

# **ACKNOWLEDGEMENT**

# **I am pleased to convey my sincere thanks to our Principal DR.M.DAVAMANI CHRISTOBER. M.Sc., M.Phil., M.ED., PGDCA., Ph.D.,**

# **I am Grateful to Head of the department Mr.J. John Jeya kamaraj,M.Sc,PGDCA,B.Ed,MCA,M.Phil. Bachelor of Computer Science Department to his Energetic Encouragement.**

# **I express my deep sense of Gratitude and Indebtedness to**

# **Mr. Sudhaesan. MCA his inspiring guidance and continuous encouragement which has enabled me to complete this project Also I would like to thank my Teachers.**

# **Friends and my well wishers who have helped me during the course of completion of this project.**

# 

# 

# **Ice Cream Shop Advertisement Website Documentation**

# **Table of Contents**

# 1.Introduction

# 2. Technologies Used

# 3. Project Structure

# 4. Installation and Setup

# 5. Configuration

# 6. Routes

# 7. Controllers

# 8. Views

# 9. Public Assets

# 10. Environmental Variables

# 11. ER Diagram

# **Introduction**

# The Ice Cream Shop Advertisement Website is a web application that promotes an ice cream shop, allowing users to browse available flavors, place orders, and view information about the shop and its offerings. The website provides separate modules for admin login and user login. Admins have access to the database and can perform editing operations, while users can view shop details, products, and services.

# **Technologies Used**

# The Ice Cream Shop Advertisement Website is built using the following technologies:

# **Node.js:** A JavaScript runtime environment used for server-side development.

# **Express:** A web application framework for Node.js used for handling routes

# and middleware.

# **HTML:** Hypertext Markup Language for creating the structure and content

# of the website.

# **CSS:** Cascading Style Sheets for styling the website.

# **JavaScript:** A programming language used for client-side interactions and

# dynamic website behavior.

# **MySQL**: A relational database management system for storing shop and

# user-related data.

# **WHY NODE JS ?**

# **Node.js:**

# Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows you to run JavaScript on the server-side, making it a great choice for building scalable and high-performance web applications. Node.js provides an event-driven, non-blocking I/O model, which allows handling a large number of concurrent connections efficiently.

# **Asynchronous Programming:**

# Node.js uses asynchronous programming techniques, which allow for handling multiple requests without blocking the execution of other tasks. This makes it well-suited for building real-time applications and handling heavy loads.

# **Large and Active Community:**

# Node.js has a large and active community of developers, which means there are plenty of resources, tutorials, and libraries available. This makes it easier to find support and solutions to common problems while developing your application.

# **Express.js:**

# Express is a lightweight and flexible web application framework for Node.js. It provides a simple and intuitive API for building web applications and APIs. Express helps with routing, middleware handling, and request/response handling, making it easier to build the backend of your web application.

# **Middleware Support:**

# Express.js has a robust middleware system that allows you to extend the functionality of your application. You can easily add middleware for handling authentication, logging, error handling, and more. This modular approach simplifies the development process and promotes code reusability.

# 

# **Integration with other Modules and Libraries:**

# Express.js seamlessly integrates with other modules and libraries from the Node.js ecosystem. You can easily add database connectors, template engines, authentication libraries, and more to enhance the functionality of your web application.

# ***Overall, Node.js and Express provide a powerful combination for building scalable, efficient, and modular web applications. They offer a familiar and consistent development experience using JavaScript across the entire stack, making it easier to build full-stack applications.***

# **WHY HTML ?**

# **Structure:**

# HTML provides a structured way to organize and format content on a webpage. It uses tags to define elements such as headings, paragraphs, lists, tables, forms, and more. This structure helps browsers and search engines understand the content and display it correctly.

# **Accessibility:**

# HTML includes built-in features to make web content more accessible to people with disabilities. It supports the use of alt attributes for images, semantic markup for indicating the purpose of content (e.g., headings, lists), and ARIA (Accessible Rich Internet Applications) roles and attributes.

# **Compatibility:**

# HTML is supported by all major web browsers, making it a universal language for building websites that can be accessed by users across different platforms and devices.

# **Integration:**

# HTML seamlessly integrates with other technologies like CSS (Cascading Style Sheets) for styling and JavaScript for interactivity. It allows you to create dynamic and interactive web pages by incorporating multimedia elements, forms, scripts, and more.

# **SEO (Search Engine Optimization):**

# HTML mark up plays a crucial role in optimizing web pages for search engines. Using proper heading tags, semantic structure, and meaningful content organization helps search engines understand the page's context and improves its visibility in search results.

# **Flexibility:**

# HTML provides a wide range of elements and attributes to cater to different types of content and layout requirements. It allows you to create a variety of web page structures and adapt to evolving web design trends and standards.

# **WHY CSS ?**

# **Separation of Concerns:**

# CSS allows you to separate the style and layout of a webpage from its structure and content. This separation improves code organization, readability, and maintainability. With CSS, you can define styles in a separate file or within the HTML document's **<style>** tags, making it easier to update and modify the visual aspects of a website without affecting its underlying structure.

# **Consistency:**

# CSS enables consistent styling across multiple web pages. By defining styles in a central CSS file, you can apply the same styles to various elements throughout your website. This consistency helps establish a cohesive and professional look and feel across all pages, improving the user experience.

# **Flexibility:**

# CSS provides extensive control over the presentation of web content. You can change fonts, colors, backgrounds, margins, padding, positioning, and more. CSS allows you to create visually appealing designs, customize the layout, and adapt the presentation to different screen sizes and devices using responsive design techniques.

# **Efficiency:**

# CSS allows you to apply styles to multiple elements simultaneously using classes, IDs, or element selectors. This reduces code duplication and makes your stylesheets more efficient and easier to maintain. CSS also supports inheritance and cascading, which means styles can be inherited by child elements and overridden selectively, providing greater control and flexibility.

# **Browser Compatibility:**

# CSS is supported by all major web browsers, making it a widely adopted standard. It ensures consistent styling across different browsers, platforms, and devices, reducing the need for browser-specific hacks and workarounds.

# **Animations and Interactions:**

# CSS includes powerful features for creating animations, transitions, and interactive elements on web pages. Using CSS animations and transforms, you can add movement and visual effects without relying on JavaScript or external libraries.

# **WHY JAVASCRIPT ?**

# **Interactivity:**

# JavaScript enables dynamic and interactive features on websites. It allows you to create responsive user interfaces, handle user events (e.g., button clicks, form submissions), and update content without requiring a page refresh. With JavaScript, you can build engaging experiences that respond to user actions in real-time.

# **Client-Side Execution:**

# JavaScript runs directly in the user's web browser, enabling client-side processing. This reduces the need for server round-trips and improves the performance and responsiveness of web applications. It also offloads some of the processing tasks from the server, resulting in more scalable applications.

# **DOM Manipulation:**

# JavaScript provides powerful tools for manipulating the Document Object Model (DOM), which represents the structure of an HTML document. You can use JavaScript to dynamically modify the content, style, and behavior of web pages. This allows for creating interactive elements, adding or removing elements dynamically, and updating the visual representation of the page.

# **Browser Compatibility:**

# JavaScript is supported by all major web browsers, making it a universal language for web development. You can write JavaScript code that runs consistently across different browsers, ensuring a seamless user experience for your website visitors.

# **Extensibility:**

# JavaScript allows you to extend the functionality of web pages by integrating third-party libraries and frameworks. There is a vast ecosystem of JavaScript libraries and frameworks available, such as React, Angular, and Vue.js, which provide ready-to-use components and tools for building complex web applications.

# **Server-Side Development:**

# In addition to client-side scripting, JavaScript can also be used for server-side development. Node.js is a popular JavaScript runtime that allows you to build scalable and high-performance server applications. With JavaScript on both the client and server sides, you can share code and data structures, improving development efficiency.

# **Integration with APIs:**

# JavaScript enables seamless integration with various APIs (Application Programming Interfaces) and web services. It allows you to make HTTP requests, retrieve data from external sources, and update your web pages dynamically. This makes it easier to incorporate data from different sources and create rich, interactive experiences.

# **WHY MYSQL WORKBENCH ?**

# **Database Design:**

# MySQL Workbench allows you to visually design and model your database schema. It provides a graphical interface where you can create tables, define relationships between tables, set up constraints, and establish data types for columns. This visual approach simplifies the process of designing and structuring your database.

# **Data Modelling:**

# With MySQL Workbench, you can create entity-relationship (ER) diagrams to represent the logical structure of your database. ER diagrams help you understand the relationships between tables, identify primary and foreign keys, and visualize the overall data model. They serve as a blueprint for your database design and can be shared with team members and stakeholders for better collaboration.

# **Query Development:**

# MySQL Workbench includes a built-in SQL editor that allows you to write and execute SQL queries directly within the tool. This is especially useful for testing and debugging database queries during development. The SQL editor provides syntax highlighting, auto-completion, and query result visualization, making it easier to work with your database.

# **Database Administration:**

# MySQL Workbench provides features for managing and administering your database. You can perform tasks such as creating and managing database connections, creating and modifying database objects, importing and exporting data, and optimizing database performance. It offers a user-friendly interface for carrying out administrative tasks without needing to rely solely on command-line tools.

# **Collaboration and Documentation:**

# MySQL Workbench allows you to generate detailed documentation of your database schema, including tables, relationships, and constraints. This documentation can be valuable for team collaboration, sharing database designs with developers and stakeholders, and maintaining an up-to-date record of your database structure.

# **Visual Data Manipulation:**

# MySQL Workbench provides a visual interface for managing and manipulating data in your database. You can browse tables, edit records, insert new data, and delete existing data using an intuitive graphical interface. This can be particularly helpful for quick data updates and inspections during development or database administration tasks.

# **Cross-Platform Compatibility:**

# MySQL Workbench is available for multiple operating systems, including Windows, macOS, and Linux. This cross-platform compatibility allows developers to work with MySQL databases regardless of their preferred operating system.

# **Project Structure**

# The project follows a standard directory structure:

## - Ice Cream Store Advertisement Website

## **- controllers**

## - productcontroller.js

## - usercontroller.js

## **- routes**

## - product.js

## - user.js

## **- views**

## **- *partials***

## - header.hbs

## - footer.hbs

## ***-hbs***

## - home.hbs

## - adview.hbs

## - product.hbs

## - services.hbs

## - blog.hbs

## - addproduct.hbs

## - editproduct.hbs

## - about.hbs

## - admin.hbs

## - register.hbs

## - login.hbs

## - offers.hbs

## - offerview.hbs

## - offeradminview.hbs

## **- public**

## ***- images***

## ***- img***

## ***- icons***

## ***- fonts***

## ***- css***

## - offer.css

## - home.css

## - service.css

## - blog.css

## - adview.css

## - product.css

## - services.css

## - about.css

## - admin.css

## - register.css

## - login.css

## **- .env**

## **- app.js**

## **- package.json**

## **- README.md**

# **PACKAGE.JSON:**

# **This is file which act as a npm (node package manager)** **that will handles and contain all the package that are installed by npm an used in my code, which look like**

# The **package.json** file is an essential part of any Node.js project. It is used to manage the project's dependencies, define scripts, and store various project metadata. Here's a detailed explanation of the **package.json** file:

# **Dependency Management**:

# The **package.json** file lists all the dependencies required by the project. These dependencies are third-party modules or libraries that the project relies on to function properly.

# **Versioning:**

# The **package.json** file also records the specific version numbers of the installed dependencies, ensuring consistency across different development environments.

# **Configuration:**

# It allows you to define custom scripts, set project metadata, specify entry points, and configure various project-related settings.

# **Contents:**

# **Name:** The name of your project.

# **Version:** The version number of your project.

# **Description:** A brief description of your project.

# **Main:** The entry point of your application (typically the starting

# JavaScript file).

# **Scripts:**

# Custom scripts that can be executed using npm or yarn commands.

# **Dependencies:**

# A list of runtime dependencies required by your project.

# **DevDependencies:**

# Development dependencies required for tasks like testing, linting, and building.

# **Keywords:** Keywords or tags associated with your project.

# **Author:** The name of the project's author.

# **License:** The license under which the project is distributed.

# **Managing Dependencies:**

# Dependencies are listed under the **dependencies** field in the **package.json** file.

# Each dependency entry includes the package name and its version range.

# When you run **npm install** or **yarn install**, the package manager reads the **package.json** file and installs the specified dependencies.

# **Scripts:**

# The **scripts** section allows you to define custom commands to be executed using **npm run** or **yarn run**.

# Common scripts include **start** to run the application, **test** to execute test cases, and **build** to build production-ready assets.

# Custom scripts can be used to automate common development tasks like linting, formatting, database migration, etc.

# **Metadata and Configuration:**

# The **package.json** file can include additional fields for storing project metadata, such as the project's repository, contributors, homepage, etc.

# Configuration settings specific to tools or frameworks used in the project can also be stored in the **package.json** file.

# ***Overall, the package.json file serves as a central configuration file for Node.js projects. It ensures consistent dependencies, allows for custom script execution, and provides metadata about the project. It is crucial for managing and sharing Node.js projects effectively.***

# Top of Form

# -package.json

{

  "name": "trywithdb",

  "version": "1.0.0",

  "description": "Project With Database",

  "main": "app.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

  "start": "nodemon app.js"

  },

  "keywords": [],

  "author": "The Hero",

  "license": "ISC",

"dependencies": {

"alert": "^5.1.4",

    "bcryptjs": "^2.4.3",

    "body-parser": "^1.20.2",

"cookie-parser": "^1.4.6",

    "dotenv": "^16.0.3",

    "express": "^4.18.2",

"express-fileupload": "^1.4.0",

    "hbs": "^4.2.0",

    "jsonwebtoken": "^9.0.0",

"multer": "^1.4.5-lts.1",

    "mysql": "^2.18.1",

    "node-schedule": "^2.1.1",

    "nodemon": "^2.0.20"

  }

}

**-This will contain all the installed and those packages are installed and used**

## **- app.js**

# In a Node.js project, the **app.js** file is commonly used as the main entry point or the central file where the application is initialized and configured. It typically sets up the server, handles routing, and integrates various middleware and modules. Here's a detailed explanation of the **app.js** file:

# **Purpose:**

# **Server Initialization:**

# The **app.js** file is responsible for initializing the Node.js server using frameworks like Express.js. It creates an instance of the server and sets up the necessary configurations.

# **Routing:**

# It defines the routes for different URLs and maps them to appropriate handlers or controllers.

# **Middleware Integration:**

# The **app.js** file is used to integrate middleware functions that enhance the functionality of the server, such as logging, authentication, error handling, etc.

# **Application Configuration:**

# It can contain configuration settings for the application, including database connections, environment variables, template engines, static file serving, etc.

# **Contents:**

# Importing Dependencies: The **app.js** file begins with importing the required dependencies and modules, such as Express.js, other custom modules, middleware, and routes.

# **Server Initialization:**

# It creates an instance of the Express.js application and sets up the server by specifying the port number and any other server configurations.

# **Middleware Integration:**

# Middleware functions are integrated using **app.use()** to enhance the request/response handling. Common middleware includes body parser, cookie parser, session management, logging, authentication, etc.

# **Routing:**

# The **app.js** file defines the routes for various URLs by using the **app.get(), app.post(), app.put(), app.delete(),** or similar methods provided by Express.js. Each route is associated with a corresponding controller or handler function.

# **Static File Serving:**

# It configures the server to serve static files, such as HTML, CSS, JavaScript, images, etc., using the **express.static()** middleware.

# **Error Handling:**

# Error handling middleware can be included to handle and respond to errors that occur during request processing.

# **Database Connection:**

# If the application requires a database connection, the **app.js** file may contain code to establish the connection and configure the database settings.

# **Other Application Configurations:**

# Additional configurations specific to the application, such as template engines, session management, caching, etc., can be set up in the **app.js** file.

# **Usage:**

# The **app.js** file is typically run using the Node.js command (**node app.js**) to start the application.

# It serves as the central file where the application's server and middleware stack are configured and defined.

# 

# ***Overall, the app.js file plays a crucial role in setting up the server, handling routing, integrating middleware, and configuring the application in a Node.js project. It acts as the entry point for the application and brings together various components to create a functioning web server.***

const express = require("express");

const mysql = require("mysql")

const app = express();

const dotenv = require("dotenv")

const path = require("path")

const hbs = require("hbs");

const cookieParser = require("cookie-parser");

const bodyParser = require("body-parser")

const { config } = require("dotenv");

require("dotenv"),config()

app.use(bodyParser.json());

app.use(cookieParser());

app.use(express.urlencoded({extended:false}))

//for styling connection

const location = path.join(\_\_dirname,"./public");

app.use(express.static(location))

app.set("view engine", "hbs")

//for partials

const partialspath = path.join(\_\_dirname,"views", "partials")

hbs.registerPartials(partialspath)

//for other js file connection

app.use("/",require("./routes/pages"))

app.use("/authen",require("./routes/authen"))

app.use("/",require("./routes/product"))

app.listen(5000,()=>{

    console.log("server started at port 5000");

});

# **Explanation of the project structure**

# The **app** folder contains the main application logic.

# The **controllers** folder holds the Node.js controller files, including **productcontroller.js** and **usercontroller.js,** which handle the business logic for respective functionalities.

# The **routes** folder contains the route files responsible for mapping HTTP requests to the appropriate controller actions.

# The **views** folder contains the Handlebars (hbs) templates for rendering the different pages of the website. The **partials** folder contains reusable partial templates such as the header and footer.

# The **public** folder contains static assets such as images, icons, fonts, and CSS files.

# The **.env** file stores environment variables for your application.

# **app.js** is the main entry point of your Node.js application.

# **package.json** lists the project dependencies and configuration.

# **README.md** contains the documentation and instructions for setting up and running the application.

# **Installation and Setup**

# ***To install and set up the Ice Cream Shop Advertisement Website, follow these steps:***

# Clone the repository from GitHub: git clone https://github.com/your/repository.git

# Navigate to the project directory: cd icecream-shop-website

# Install dependencies: npm install

# Start the application: npm start

## Set up the MySQL database using **MySQL Workbench** or another MySQL client.

## Configure the database connection in the **.env** file.

# Open a web browser and visit http://localhost:3000 to access the website.

# **5.Configuration**

# The configuration files for the project are located in the config/ directory. They include:

# **config.js:**

# Contains general configuration settings such as the port number and database connection details.

# **database.js:**

# Contains configuration for the database connection.

# ***I modify these files according to your specific environment and requirements.***

# **Query Used For Databases:**

# **//login\_crud Database**

create database login\_crud;

use login\_crud;

show tables;

create table users

(

ID int auto\_increment primary key,

NAME varchar(100),

EMAIL varchar(100),

PASS varchar(200)

);

use login\_crud;

create table adminlogin(

ID int auto\_increment primary key,

name varchar(400),

password varchar(100)

);

select \* from adminlogin;

select \* from users;

# **//login\_crud Database**

create database product\_crud;

use product\_crud;

create table products(

ID int auto\_increment primary key,

NAME varchar(150),

IMAGE varchar(200),

DES varchar(200),

About varchar(200)

);

use  product\_crud;

create table OFFERS(

ID int auto\_increment primary key,

ICECREAM varchar(400),

SCOUP varchar(400),

PHOTO varchar(400),

IMAGE varchar(400),

QUANTITY varchar(50),

CUANTITY  varchar(50),

price varchar(400)

);

select \* from OFFERS

# **Routes**

# The routes of the application are defined in the **routes/** directory. The following routes are available:

# In a Node.js application, routes are used to define the endpoints or URLs that the server will respond to and the corresponding actions to be taken for each endpoint. Routes handle the incoming requests and determine how the server should respond. Here's an explanation of routes in Node.js:

# **Purpose:**

# **Request Handling:**

# Routes determine how the server should handle different types of requests, such as GET, POST, PUT, DELETE, etc., for different URLs.

# **Endpoint Definition:**

# They define the URLs or endpoints that clients can access to interact with the server and the resources they can manipulate.

# **Routing Logic**:

# Routes contain the logic to process requests, validate input, interact with databases or other services, and send responses back to the client.

# **Routing in Node.js:**

# Route Definition: Routes are defined using the routing methods provided by the Express.js framework, such as **app.get(), app.post(), app.put(), app.delete(),** etc. These methods specify the HTTP verb and the URL pattern for the route.

# **URL Patterns:**

# Routes can include URL patterns with parameters to capture dynamic values from the URL, such as **/users/:id** or **/products/:productId.**

# **Request Handlers:**

# Each route is associated with a request handler function that defines the actions to be taken when a request matches that specific route. The handler function receives the request (**req**) and response (**res**) objects and can perform operations like data retrieval, validation, processing, and sending a response back to the client.

# **Middleware Integration:**

# Routes can also include middleware functions to perform operations before or after the request handler function, such as authentication, input validation, logging, etc.

# **Route Examples:**

# **GET Route:**

# Handle a GET request to the "/products" URL to retrieve a list of products.

# app.get('/products', (req, res) => {

# // Retrieve products from the database

# // Send the products as a response

# });

# **POST Route:**

# Handle a POST request to the "/products" URL to create a new product.

# app.post('/products', (req, res) => {

# // Validate and process the request body

# // Create a new product in the database

# // Send a success response

# });

# **URL Parameters:**

# Handle a GET request to retrieve a specific product based on its ID.

# Home page route that displays the home view.

# app.get('/products/:id', (req, res) => {

# const productId = req.params.id;

# // Retrieve the product with the specified ID from the database

# // Send the product as a response

# });

# **Middleware:**

# Use middleware to authenticate requests before accessing certain routes.

# const authenticate = (req, res, next) => {

# 

# // Perform authentication logic

# // If authentication is successful, call next() to proceed to the route handler

# // If authentication fails, send an error response

# };

# app.get('/admin/dashboard', authenticate, (req, res) => {

# // Only authenticated users can access this route

# // Handle the request and send a response

# });

# **Organizing Routes:**

# Routes can be organized in separate files or modules to improve code maintainability. For larger applications, it is common to use a router module to define routes for specific functionalities or resources.

# Overall, routes in Node.js are used to define the URLs and associated actions that the server can handle. They provide a way to handle incoming requests, process data, and send appropriate responses back to the client. Routes play a vital role in defining the API endpoints and implementing the server-side logic of a web application

# **/adview:** Route for displaying the advertisements.

# **/products:** Route for displaying the available products.

# **/services:** Route for displaying the services offered by the shop.

# **/addproduct:** Route for adding a new product (admin only).

# **/editproduct/:id:** Route for editing a product (admin only).

# **/about:** Route for displaying information about the shop.

# **/admin:** Route for admin login and dashboard.

# **/register:** Route for user registration.

# **/login:** Route for user login.

# I modify or add additional routes as per My application's needs.

# **Controllers**

# The controllers in the **controllers/** directory handle the application logic for each route. They interact with the models and views to process and respond to user requests. Some example controllers

# **UserController.js:**

# Handles requests related to the home page.

# **ProductController.js:**

# Handles requests related to ice cream flavours.

# **Views**

# The **views/** directory contains HTML templates for rendering views. These templates are responsible for presenting the data to the user. Example view templates include:

# **Home:** Represents the homepage of the website.

# **Adview:** Displays advertisements for the ice cream shop.

# **Product**: Shows the details of the available products.

# **Services:** Provides information about the services offered by the ice cream shop.

# **Add Product:** Allows the admin to add new products.

# **Edit Product:** Enables the admin to modify existing product details.

# **About**: Presents information about the ice cream shop.

# **Admin:** Admin dashboard or control panel for managing the website.

# **Register:** User registration page.

# **Login:** User login page.

## **- views**

## 

## **- *partials***

## - header.hbs

## - footer.hbs

## 

## ***-hbs***

## - home.hbs

## - adview.hbs

## - product.hbs

## - services.hbs

## - blog.hbs

## - addproduct.hbs

## - editproduct.hbs

## - about.hbs

## - admin.hbs

## - register.hbs

## - login.hbs

## - offers.hbs

## - offerview.hbs

## - offeradminview.hbs

# **9. Public Assets**

# The **public/** directory contains static assets such as CSS, images, and JavaScript files. It includes the following directories:

# **css/:** Contains CSS files for styling the website.

# **img/:** Contains image files used in the website.

# **js/:** Contains JavaScript files for client-side interactions and

# dynamic behavior.

# **10. Environmental Variables**

# The **.env** file at the root of the project is used for configuring environmental variables. You can store sensitive information such as database credentials, API keys, or any other configuration values in this file. Make sure not to share this file publicly or include it in version control repositories.

DATABASE=login\_crud

DATABASE1=user\_comments

DB\_NAME = product\_crud

DATABASE\_HOST=localhost

DATABASE\_USER=root

DATABASE\_PASS=root123

JWT\_SECRET=1234

JWT\_EXPIRES\_IN="90d"

JWT\_COOKIE\_EXPIRES=90

# **11. ER Diagram**

# The Entity-Relationship (ER) diagram for the login\_crud and product\_crud databases is as follows:

# The **login\_crud** database contains a **User** table with columns **id,** **username**, **email,** and **password.**

# The **product\_crud** database contains a **Product** table with columns **id,** **name, about,** and **description.**

+-------------------------+

| **login\_crud** |

+-------------------------+

| **users**  |

+------------------------+

| ID (PK) |

| USERNAME |

| EMAIL |

| PASSWORD |

+-------------------------+

|

|

|

|

**Admin Access**

|

|

+--------------------------+

| **product\_crud** |

+--------------------------+

|  **admin** |

+---------------------------+

| ID (PK) |

| EMAIL |

| PASSWORD |

+---------------------------+

|

|

|

|

|

**Admin Access**

|

|

v

+---------------------------+

| **product**  |

+---------------------------+

| |

| ID (PK) |

| PRODUCTNAME |

| IMAGE |

| DES |

| ABOUT |

+-------------------------+

|

|

|

|

**Admin Access**

|

|

v

+------------------------+

|  **offer** |

+------------------------+

| |

| ID (PK) |

| PRODUCT1 |

| PRODUCT2 |

| QUANTITY1 |

| QUANTITY2 |

| IMAGE1 |

| IMAGE2 |

| PRICE |

+------------------------+

# 

## The system consists of two databases: **"login\_crud"** and **"product\_crud".**

## **"login\_crud"** database:

## Contains a **"users"** table with columns: ID, USERNAME, EMAIL, PASSWORD.

## **"product\_crud"** database:

## Contains an **"admin"** table with columns: ID, EMAIL, PASSWORD.

## Contains a **"product"** table with columns: ID, PRODUCTNAME, IMAGE, DES, ABOUT.

## Contains an **"offer"** table with columns: ID, PRODUCT1, PRODUCT2, QUANTITY1, QUANTITY2, IMAGE1, IMAGE2, PRICE.

# In this ER diagram, the **"login\_crud"** database contains a **"users"** table with columns for ID, USERNAME, EMAIL, and PASSWORD. The **"product\_crud"** database contains an **"admin"** table with columns for ID, EMAIL, and PASSWORD, as well as a **"product"** table with columns for ID, PRODUCTNAME, IMAGE, DES, and ABOUT. Additionally, there is an "offer" table with columns for ID, PRODUCT1, PRODUCT2, QUANTITY1, QUANTITY2, IMAGE1, IMAGE2, and PRICE.