Table Of Contents

Introduction 2

Software Requirements Specifications (SRS) 2

Functional Requirements 2

Non Functional Requirements 3

Technologies 4

Front-end – React Native: 4

Back-end – NodeJS: 5

Database – MongoDB: 7

Conclusion 7

**INTRODUCTION**

Welcome to our birthday discount campaign architecture design document. In this document, we will discuss the proposed solution to extend our system for the campaign, which seeks to deliver personalised discounts and suggested products to our customers throughout the week around their birthday. Using modern technology and best practises, the solution is designed to be scalable, reliable, consistent, and secure. We will begin by describing the existing system and its constraints, followed by a discussion of the proposed design and its components.

**SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)**

1. **Functional Requirements**

**Actors:**

**Customers –** They are the users who will interact with your product. They can signup and create their accounts. Users can view products added from the admin side of the system.

1. **System Functional Requirements**

A functional requirement defines a system or its components. The functional requirements are as follows:

**Authentication –** Users can authenticate using valid credentials that include email and password. After authentication, the user will be redirected to the home screen where products are listed.

**Authorization –** User is authorized to perform limited functions. The user can not edit things.

**Products –** The home page of the application includes the listing personalized suggestion of products. You can filter the products by categories available. If there is no product in the selected category, “No products available” tag is placed.

**Add to Cart –** From the Home Screen the products can be added to the cart. Once the product is added to the cart, you can modify the number of pieces of that product according to your need. Also, the number of pieces can not be more than the quantity available in stock. You can also delete products from the cart if you want to do so.

**Placing Order –** After adding the products to the cart, users can checkout and can place orders. Initially the order status will be in the processing stage. Later on the status can be changed to shipping or delivered.

1. **Non Functional Requirements**

A non-functional requirement defines the quality attribute of a software system. The non-functional requirements are as follows:

* **Speed** – How fast the system performs certain activities.
* **Availability** – How much of the time the system is available e.g. does it operate overnight, or every day of the year, or not.
* **Scalability** – What the limits are of what the system is able to handle.
* **Reliability** – How dependable the system is. To achieve high reliability, all bugs that may influence the code safety and issues with system components must be removed.
* **Usability** – How easy the system is to use for the customer or end user. Usability is the degree of ease with which the user will interact with your products to achieve required goals effectively and efficiently.
* **Performance** – It describes how your solution behaves when users interact with it in various scenarios. Poor performance may lead to a negative user experience and jeopardize system safety.
* **Security -** To ensure the security of the system, its data, and its users

**TECHNOLOGIES**

**FRONTEND ARCHITECTURE**

**Technologies Used:** React Native, React, Expo GO, XCode

The frontend of the project is built on **React Native**. React Native lets you create truly native apps and doesn't compromise your users' experiences. It provides a core set of platform agnostic native components like View, Text, and Image that map directly to the platform’s native UI building blocks.

With React, you can make components using either classes or functions. Originally, class components were the only components that could have state. But since the introduction of React's Hooks API, you can add state and more to function components.

Hooks were introduced in React Native 0.59., and because Hooks are the future-facing way to write your React components, we wrote this app using function components.

**BACKEND ARCHITECTURE**

**Technologies Used:** NestJS, TyeScript, PostgreSQL database, nodemailer, TypeORM, JWT

The backend architecture is built using the **NestJS** framework written in **TypeScript**, which is based on the popular NodeJS framework. NestJS provides a modular, scalable, and maintainable architecture for building server-side applications. It consists of the following modules:

1. **Authentication Module:** This module handles authentication and authorization of users. It includes a secure login system using JWT tokens to ensure that only authenticated users can access the system.
2. **Customer Module:** This module handles all customer-related operations such as registration and updating customer information.
3. **Product Module:** This module handles all product-related operations such as adding products, deleting products, and updating products.
4. **Discount Module:** This module handles all discount-related operations such as generating discount codes, applying discounts to orders, and retrieving discount information.
5. **Email Service:** This service is built using nodemailer (which is an email service used in development). For production deployment, a more robust cloud email service like SendGrid or Amazon SES should be used. It will be responsible for sending out personalized emails to customers with their discount codes and suggested products.
6. **Scheduling Service:** A cron job was implemented to run periodically (e.g. every day at 6AM) to check if any customers have their birthday in the upcoming week. If there are, a personalized email will be generated for each of them, with a unique discount code and a list of personalized suggested products.
7. **Database Module:** This module handles all interactions with the database, including data storage and retrieval.

**DATABASE ARCHITECTURE**

The database architecture is built using a relational database management system (PostgreSQL). It is designed to support the following entities:

* **Customer:** This entity should store information about customers such as their name, email, phone number, and date of birth.

|  |  |  |  |
| --- | --- | --- | --- |
| **S/N** | **COLUMN** | **TYPE** | **DESCRIPTION** |
| 1 | id | bigint | This is the unique identifier for each customer. It's the primary key for the Customer table in the database. It's an auto-incrementing integer. |
| 2 | username | character varying | This is the username that the customer uses to log into their account |
| 3 | name | character varying | This is the customer's full name |
| 4 | email | character varying | This is the customer's email address |
| 5 | phoneNumber | character varying | This is the customer's phone number |
| 6 | dob | character varying | This is the customer's date of birth |
| 7 | password | character varying | This is the hashed password that the customer uses to log into their account |
| 8 | birthdayDiscountCode | character varying | This is the discount code that the customer will receive on their birthday |
| 9 | dateCreated | timestamp | This is the timestamp of when the customer's account was created |
| 10 | dateUpdated | timestamp | This is the timestamp of when the customer's account was last updated |
| 11 | lastLoginAt | timestamp | This is the timestamp of when the customer last logged into their account |
| 12 | token | character varying | This is a unique token that is associated with a user or session, and is used for authentication purposes. |
| 13 | favoriteCategory | integer | This is the category that the customer has selected as their favorite. |

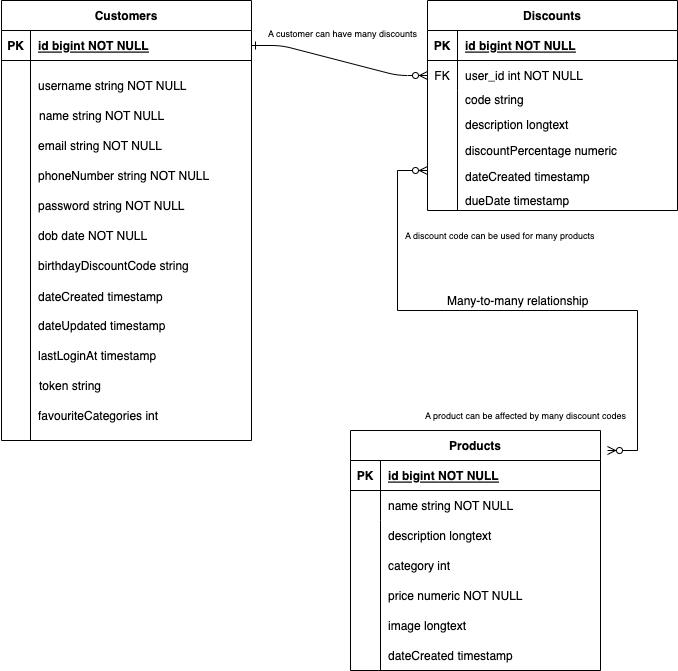
* **Product:** This entity should store information about products such as their name, description, price, and category.

|  |  |  |  |
| --- | --- | --- | --- |
| **S/N** | **COLUMN** | **TYPE** | **DESCRIPTION** |
| 1 | id | bigint | This is the unique identifier for each discount. It's the primary key for the Product table in the database. It's an auto-incrementing integer. |
| 2 | name | character varying | This is the name of the product |
| 3 | description | longtext | This is a string column that stores a description of the product |
| 4 | price | numeric | This is the price of the product in floating number |
| 5 | image | longtext | This hold the url of the product image |
| 6 | category | integer | This is the category of the product |
| 7 | dateCreated | timestamp | This is the timestamp of when the product was created |

* **Discount:** This entity should store information about discounts such as the discount code, discount percentage, and expiration date.

|  |  |  |  |
| --- | --- | --- | --- |
| **S/N** | **COLUMN** | **TYPE** | **DESCRIPTION** |
| 1 | id | bigint | This is the unique identifier for each discount. It's the primary key for the Discount table in the database. It's an auto-incrementing integer. |
| 2 | code | character varying | This holds the discount code for the discount |
| 3 | description | longtext | This is a string column that stores a description of the discount |
| 4 | dateCreated | timestamp | This is the timestamp of when the discount was created |
| 5 | dueDate | date | This is the timestamp of when the discount will expire |

The image below shows and entity relationship diagram showing all the different entities and their relationships.



**DEPLOYMENT ARCHITECTURE**

The system should be deployed to a cloud platform such as Amazon AWS, Google Cloud Platform or any virtual server. The frontend should be hosted on Apple App Store, Google Play Store, Huawei App Gallery, Xiaomi MI Store etc. The backend should be deployed on an auto-scaling group to handle fluctuations in traffic.

In summary, the software architecture for this project was designed using React Native for the frontend, NestJS and TypeScript for the backend, PostgreSQL for the database. This architecture will provide a scalable, maintainable, and secure system that can handle the needs of the project.