**Boba Boys – Full Stack E-Commerce Single-Page Application**

[**https://boba-boys.web.app/**](https://boba-boys.web.app/)

[**https://github.com/ahnafahsan/boba-boys**](https://github.com/ahnafahsan/boba-boys)

**Introduction**

The project is a full-stack single page web application designed to serve as a boba tea ecommerce website. The functionalities that will be found in this project include: browsing the various boba tea selections, user authentication, user checkout etc. The project aims to make use of various modern technologies whilst adhering to best industry practices to achieve a sleek and performant end product.

**Technologies**

**Svelte**

To build this project as a single page application an important decision had to be made behind choosing which front-end framework to use, to which Svelte was the framework that was chosen for this task. Unlike other popular frameworks/libraries such as React, Angular, & Vue which ship a JavaScript runtime to the browser, Svelte is a compiler which converts declarative code into imperative native JavaScript code which results in highly performant code using a small package. Svelte code is also often less verbose than its competitors and very closely resembles JavaScript meaning it’s easy for people with even the most basic of JavaScript knowledge to look into the Svelte source code and get some sort of idea of what’s happening as opposed to for example looking into Angular source and having no idea. To set up the Svelte development environment, **Vite**.js – a performant front-end build tool was used.

**TailwindCSS**

TailwindCSS is a utility-first CSS framework. It allows for easily and rapidly building custom user interfaces compared to standard CSS approaches. A disadvantage of TailwindCSS is that it often leads to less readable files due to the long class names, however, for a small individual project like this which is not built for scalability, TailwindCSS felt like an excellent tool to use. TailwindCSS also allows for plugins, this project uses **daisyUI** which is a component library tailored to TailwindCSS.

**Google Firebase**

Google firebase is a Backend-as-a-Service (BaaS) which offers a variety of tools. Out of these services the ones used in this project are hosting to deploy the project on the web, the real time database to store information, and firebase authentication to handle logins. As this project will only be processing light and simple data, the firebase real time database is a great tool for this use case.

**Yarn Package Manager**

Package managers are almost essential for reducing hassle and seamlessly managing dependencies. When starting the project, there was a choice to be made of which package manager to use. The two most popular package managers used for web development are Yarn and NPM. Yarn was chosen for this project as tests show that it is more performant than NPM, although the decision was mostly made based on preference.

**Github**

This project uses GitHub for version control and also as a means to share the source code in an organised and familiar manner.

**Pages & Components**

A component-style approach was taken when building this application. To increase the readability and organisation of the project, components are separated into a components folder.

This section will be covering the various pages and UI components that make up this application and their functionalities.

**Navigation Bar**

The navigation bar was the first component built and serves a few purposes: allow a user to logout, return to the homepage by clicking the logo, view the shopping basket, and view the total price.

**Shopping Menu**

This menu contains all the items on sale. Items are displayed in a card container which has a picture of the item, the name of the item, the price of the item, and a button to add the item to basket.

**Checkout Page**

This page contains an overview of the items the user has in their basket as well as a form to enter payment and shipping details.

**Orders Page**

This page contains the history of orders that the user has made.

**Login and Registration Page**

This page allows the user to login, or alternatively register an account in the case where a user does not have an account. For this project, users will be able to also use the site as guests without needing to create an account. The reason behind this is because forcing a user to have an account could have a negative influence in sales, especially in a takeaway/collection type business.

**Responsive Design**

In order to provide a more user-friendly experience, the application must be adjusted to accommodate for different screen sizes. Using a standard CSS technique would require the use of a lot of media queries which can result in increasingly verbose code and decrease readability. TailwindCSS makes this relatively straightforward using breakpoint classes which are what is used in this project. The application has been designed using a **mobile-first approach**, meaning elements have been styled to look good on mobile screens first and then responsiveness is added for bigger screen sizes afterwards.

**Page Routing**

To manage the routing for a Svelte single page application a few routes could’ve been taken to achieve this. The standard way to implement routing in Svelte projects is by using SvelteKit, however, for a small project like this which only has a few pages and does not necessarily need all the features that come with SvelteKit, Svelte SPA Router - a small library which utilizes hash routing - is used for this project.

**Setting up the Backend**

To connect this web application to the firebase backend, the firebase JavaScript library was needed and some configurations had to be set up after obtaining an application specific API key.

**Verification & Authentication**

After the creation of the login and registration frontend and handling the logic behind verifying that user input is correct, the next step was to integrate it with the firebase backend. To achieve this, the firebase authentication tool had to be enabled which would then allow the web application access to useful functions for signing up, logging in and signing out. Once a user is logged in, firebase returns an authentication key which the application would then store globally using Svelte Stores so that any component is able to access the authentication status.

**Basket Logic**

The frontend had already implemented solutions to adding & removing items from the basket, however, upon refreshing all these items would be lost. The solution used for this in the project is to make use of the browser local storage to store these items. Initially the solution was going to involve hardcoding items to the real time database, however, storing these items on the client side then only sending them to the database upon checkout effectively achieves the same goal but with less requests to the server.

**Checkout & Order Page Logic**

Due to being unable to set up a business account, the checkout system in this application is only designed as a simulation for an actual checkout. It prompts the user to enter their shipping and payment details, verifies the validity of the information, and then alerts the user of the success of their order. Furthermore, if the user is logged in, the order is saved into their account through the real time database where the user can then view their orders in the order page.

**Deploying the Application**

Firebase hosting was used to deploy the application live. This was done using the firebase CLI.

**Evaluation**

All the goals that were set out for this project were successfully met – the application is modern and blazing fast thanks to the single page application architecture, however, one thing does come to mind which could be argued is an essential part of any ecommerce application, and this is: Search Engine Optimization (SEO). SEO was not taken into consideration when building out this project, yet it is almost crucial for a business to have their websites highly ranked on search engines to allow increased traffic to the business. This project used hash routing and client side rendering, two features which are known to negatively affect SEO. Both of these can be solved by using SvelteKit which allows for standard routing as well as building server-side rendered applications.