Software Design Document

Barber Theory

Arijit Chowdhury

January 29, 2024

Introduction

Barber Theory is the name of a group of new and upcoming barbers who wanted to incorporate more theory driven work into the art of cutting hair. All current marketing being done to increase the client base was done through the word of mouth, thus client outreach was very limited to a social network bounded by geographical area. Over the past few months, a better attempt to garner client attention was in development, with a dynamic web application being the primary tool.

The Barber Theory Barbershop web application has been designed and implemented through the help of a few developers, in an effort to increase client outreach to greater lengths, connecting artist to client.

Scope

This document aims to give a detailed technical description of the BT-APP software project alongside the rationale behind the implementation and design choices. Not only does this document describes the software already in place, it is also intended for to enforce compatibility of future additions and modifications.

System Overview

The professional barbershop system is designed around four essential modules: data collection, configuration, processing, and user interface.

The data collection module gathers user and barber specific information, including service preferences, barber availability, and booking details. The configuration module, supported by MongoDB's flexible schema, manages data storage, ensuring customer records, booking histories, and stylist profiles are easily accessible and adaptable as the barbershop's data needs evolve.

The processing module employs React for frontend interactions and Django for server-side logic, to validate and handle user requests, and payment transactions securely. Lastly,the user interface, primarily built using React, provides a dynamic and responsive platform, enabling customers to intuitively browse nearby barbershops, select stylists, and plan appointments using interactive maps.

Design Considerations

This section is concerned with the overall technology stack (set of technologies) used.

- MongoDB -> Database Management
- Django -> Back End Framework
- React.js -> Front End Framework
- Node.js -> Runtime Environment and Package Management

Rationale

1.1 Database Management: MongoDB

MongoDB was selected as the database management system due to its specific advantages in handling booking data and location-based services, aligning perfectly with the barbershop's dynamic and evolving requirements.

MongoDB is a non-relational database, allowing for the storage of various types of data without the constraints of a fixed schema. This flexibility is crucial for a barbershop business, which needs to accommodate diverse data such as barber profile data, availability's specific to each barber, and payment information.

With future growth of the application and buisness in mind, the flexibility of MongoDB was an excellent choice as data requirements may change and new features might introduce new data such as customer profiles, preferences, and locations of multiple shops. The schema-less design can adapt to these changes without the need for extensive database schema modifications.

1.2 Back End Framework: Django

Django was chosen due to its robustness, and its exceptional compatibility with MongoDB, making it an ideal choice for managing the dynamic data needs of BT-APP.

Given that a barbershop website handles sensitive customer data and payment transactions, data security is paramount. Django's built-in security features, including protection against common web vulnerabilities like SQL injection and cross-site scripting (XSS), ensure the safety of customer data and payment information.

MongoDB's flexible schema can present challenges in ensuring data consistency. Django's ORM (Object-Relational Mapping) is invaluable in this context. It simplifies database operations, reducing the risk of data inconsistencies or errors in bookings, customer records, and barber information.

1.3 Front End Framework: React.js

React was chosen as the primary choice of Front End Development as it caters to the dynamic and interactive nature of modern day web applications. The main goal was to replace the nerve-racking experience of going to a barbershop with a more robust and simplistic style.

Just like the art of hair dressing, the business thrives on penalization. React's component-based architecture allows us to create a highly customized user interface, enabling customers to easily select their preferred services, stylists, and appointment times encapsulated by a seamless booking experience.

The diverse range of customers, from busy professionals to students, necessitates a responsive design. React ensures that the webpage is accessible and user-friendly on various devices.

1.4 Runtime Environment and Package Management: Node.js

Node.js was selected as the runtime environment and package management tool for its compatibility with the chosen Front End Framework, React.js.

Node.js provides a unified runtime environment that can run JavaScript on both the client (React) and server (Django), simplifying development and ensuring consistent behavior across the entire application. Node.js's event-driven, non-blocking architecture allows the BT-APP to handle multiple user-interactions simultaneously.

System Architecture

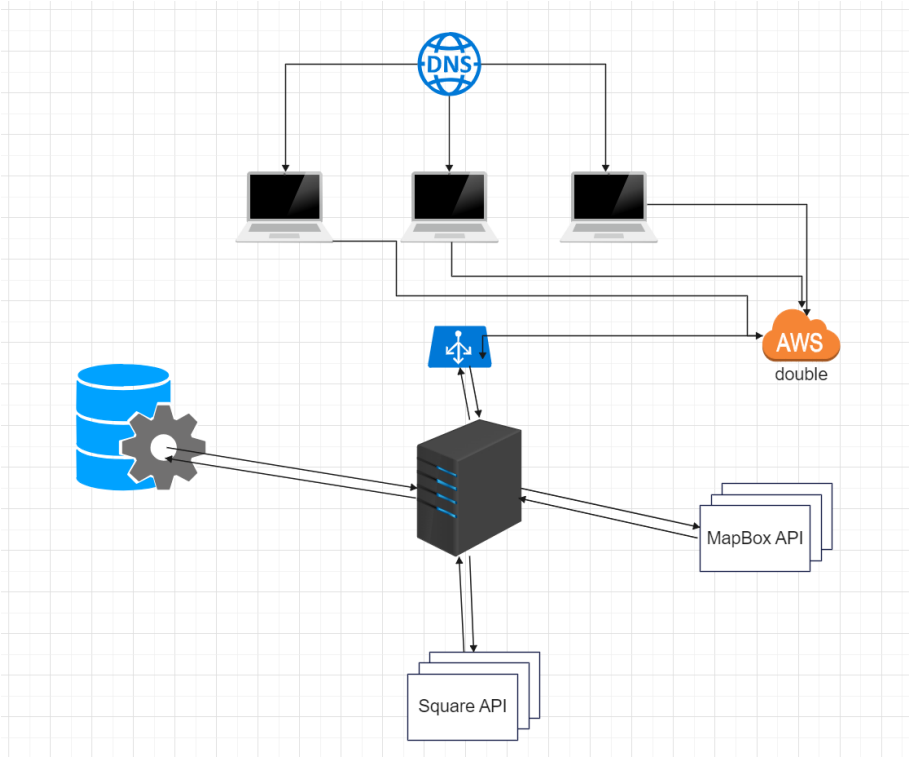

Figure 1: System Architecture Diagram

The system is comprised of a client, server, cloud services, a database, and external services. The client is made through React which handles the user interface and communicates with Django for server-side logic, which, in turn, interacts with MongoDB for data storage and retrieval. The application is hosted using Amazon Web Services (AWS) in which we used Elastic Cloud Compute (EC2) to host our application, with the addition of Elastic Load Balancers (ELB). Node.js serves as a unifying runtime and package management tool, facilitating smooth integration between all technologies. Square API handles secure payments, while Mapbox provides interactive maps on our UI. This architecture ensures a dynamic, secure, and responsive web-application catering to diverse customer needs.

Data Flow

User flow often dictates what data flows in and out of our databases. When a user is directed to the booking page of our application, data regarding our barbers are fetched from the database through Django. The availability dates, prices, services, and names are fetched, in which the React components update to display the information.

Information regarding prices, availability dates, and names are passed down from component to component. When the customer has finally paid for the service through the Square API, barber availability is updated.

Interface Design

The User Interface was designed with the following concept of edge and sharpness, an idea often associated to hair cutting. Monotone colours (black, white, grey) were used as an artistic way of conveying this, alongside the usage of four sided shapes. Shadows were also used.

The UI closely resembles the simple, yet elegant designs you see in MacOS, which was a primary goal.

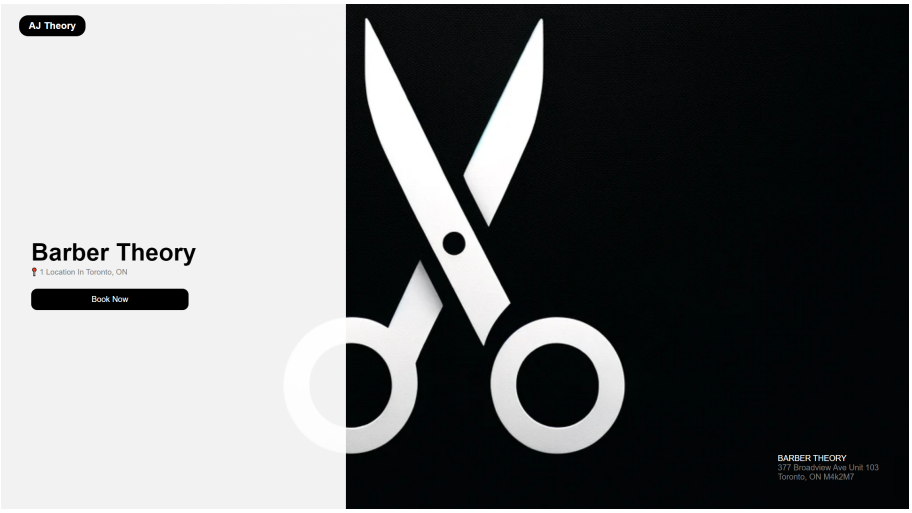

Figure 2: Landing Page Demo

Testing Strategy

We tested the application, focusing more on performance under heavy loads, compatibility, and integration.

For testing heavy loads, the goal was to simulate heavy traffic coming toward the application and observing the behaviour of it. Since we used AWS for hosting and load balancing, we tested our application using AWS Elastic Load Balance Testing.

To test browser compatibility, we simulated user work flows on our web application on Chrome, Firefox, and Safari. For integration testing, we tested to see how our MapBox API and more importantly, Square API worked with our application. We continuously tested these services via sandbox testing given to us from our Square Developer accounts.