Group 6: Colton Tucker, Tyler Gerth, Rylan McAlister, Dalynn Hatch, Brayden Alvarez, Landry Ishimwe Karangwa

UNIVERSITY OF ARKANSAS
College of Engineering

Software Justification Document

ReZerve

4/29/2021

Introduction

Throughout the design of this project, we primarily used React as the framework for our application. This document will give an overview of React, as well as our reasons behind using it and the main supplemental libraries we used alongside it.

React

First of all, React is a JavaScript framework utilizing a syntax extension called JSX (which can be treated as a bundle of JavaScript, HTML, and CSS, just in a different syntax than normal). JSX allows for JavaScript and the component variables to directly affect what is being rendered and displayed. Developers create JSX, which is then rendered and displayed in a user interface in a typical software lifecycle (construction, initiation of class variables and initial methods, and then rendering). Throughout our project, we primarily used TSX – an extension of JSX for a Typescript environment – in order to take advantage of the variable typing and class structures of the Typescript language. We built our project by having each file be a different component (things like the CustomerBusinessSearch component, the EmployeeHome component, and the CustomerCheckout component), which were then combined together to display the UI to our users and respond to the way they interacted with the application.

The two main component variables to be aware of in React are state and props. State is essentially the instance variables for a component, while props are variables passed into a component from outside sources. This allows for React to maintain local state within components, as well as have different components interact with each other, whether it be from a parent component passing in the name of a business to be displayed in the BusinessInfo component, or a child component calling a parent confirmation function that had been passed into it, so that the action is executed within the parent.

Another reason to use React is due to its immense amount of documentation and ability to pair up with outside APIs such as Firebase (our database) and Stripe (our payment system). Since React is a large, mainstream user interface framework, many developers have contributed to it by building libraries or answering common questions on Stack Overflow. This support made it much easier to implement our designs and functionality throughout the semester.

React Redux

Soon after starting the development of our application, we realized that the basic state and props provided by React would not be good enough to manage all the passing of variables between different components and storage of user authentication information. Because of this, we decided to start using the React Redux store, which is essentially like an in-memory database that all components across the application, regardless of proximity to each other, can access. This greatly simplifies how parent and child components need to interact, since we no longer need to manage the thirty different props that a child component might need from its parent.

React Redux has three main parts – the store (the storage of data), the actions (the way that components ask to update the information in the store), and the reducers (the method in which actions actually update the store). In every React Redux project, the store is first initialized to a starting state based off of what information needs to be accessible across the application. Then, components can read the information contained in the store by a process called mapStateToProps, where different sections of the store state can be mapped to the props of a component both on initialization and whenever the store changes. This allows for real-time updates of data across components, since all one component needs to do is send an action asking to update a part of the store in order to have that change to be seen in another component. These actions might be anything from setting the information of the currently logged in user to an employee so they get showed the business flow of the application, or changing the status of an appointment from ‘requested’ to ‘accepted’ so that it will show up on an employee’s calendar. Once the action reaches the store, it goes through a reducer, which updates the store information with the newly requested action’s data (or payload). However, Redux always maintains past store information by never directly changing the current state. Instead, it makes a copy of the entire current store state, changes the requested attribute, and then saves that copy as the new current state. This makes it so that the only way the store can be changed is through the dispatching of actions, a way to force developers to understand what is going on through the whole process and not have unintended side effects.

In React Redux, store-dispatch actions, as well as store state, get mapped to the props of a component. Since the developer can define what actions get mapped, as well as how the store state gets mapped, this allows a sort of definition of what read/write access each component has, to make sure it only reads/updates the store state information within its scope.

Material UI

The third main software library we used through the development of this application was Material UI – a styling library. Since we were building out a user interface, it needed to look attractive and professional in order to entice people to continue using it. Material UI provides lots of different styling components, such as buttons, loading spinners, image carousels, etc., which can all be utilized throughout an application. Although all the components are different, they all still have the same look and feel, allowing the application as a whole to seem more cohesive