System Design - RecruitMe

Fall 2022 CSCC01 Project

Pritish Panda | Alton Liu | Tony Xia | Raymond Weng | Billy Zhou | Raisa Haque | Rakshit Patel

Table of Contents

Front-End CRC Cards

View_profile Parent class: none Subclasses: none Responsibilities: - Users can view their profile - A user can only edit his/her profile only - - Collaborations (models): - Users -	**Landing Page** Parent class: none Subclasses: none Responsibilities: - Redirects users to login/sign up. Collaborations: - Login - Sign up -	**Recruiter Profile:** Parent class: none Subclasses: none Responsibilities: - For registered recruiters to edit their profile such as name, phone, and which company they represent via the company page Collaborations (models): - Users
Jobseeker Profile: Parent class: none Subclasses: none Responsibilities: For a registered job seeker user to be able to edit his/her profile information such as name, phone number, bio, work experiences, upload profile picture and resume. - Collaborations (models): - User	**Signup Page** Parent class: none Subclasses: none Responsibilities: For users to select a job seeker or recruiter account type and register with their email and password Collaborations (models): - Users	**Login Page** Parent class: none Subclasses: none Responsibilities: For job seekers and recruiters to log in to their profile using their email and password Collaborations (models): - Users

Job Detail Page:	**Job Listings Page:**	**Dashboard:**
Parent class: none Subclasses: none Responsibilities: - Show job posting detail and a button to apply to the job posting as a job seeker Collaborations (models): - Job listings - Dashboard - Navbar	Parent class: none Subclasses: none Responsibilities: - Show all open job posts and a button for users to click into the individual job post - Allow users to filter jobs by role, description, and qualifications Collaborations (models): - Job detail page - Dashboard - Navbar	Parent class: none Subclasses: none Responsibilities: - Display a condensed view of information relevant to the user - Jobseekers can see their most recent job applications and a few job postings they may be interested in - Recruiters can use the dashboard to post jobs Collaborations (models): - Navbar - Job listings - Application Tracker - Job Posting Form

Application Tracker:	**Applicant Tracker:**	**Company Page:**
Parent class: none Subclasses: none Responsibilities: - Display job seeker's current job applications. Collaborations (models): - Dashboard	Parent class: none Subclasses: none Responsibilities: - Display job seekers that applied to a recruiter's job posting - Allow recruiters to move jobseekers to next stage or reject them Collaborations (models): - Dashboard	Parent class: none Subclasses: none Responsibilities: - Display company information, open job posts, and reviews - Allow recruiters of the company to edit the page - Allow jobseekers to leave reviews of the company Collaborations (models): - Recruiter Profile - Job Detail page

Interview Invite	**Interview Scheduler**	**Interview Calendar**
Parent class: none Subclasses: none Responsibilities: - Allow recruiters to select dates and time to send to applicants for interview scheduling Collaborations (models): - Dashboard - Applicant Tracker	Parent class: none Subclasses: none Responsibilities: - Allow jobseekers to pick dates given by recruiters to schedule interview Collaborations (models): - Dashboard - Application Tracker	Parent class: none Subclasses: none Responsibilities: - Display a user's upcoming interviews in a calendar Collaborations (models): - Dashboard

Back-End CRC Cards

User	**Login:**	**Sign up:**
Parent class: none Subclasses: none Responsibilities: - Save all user information including email, password, recruiter/job-seeker status and so on - Have abstract user information where built-in password verification can be used - Have the ability to create profile, register	Parent class: none Subclasses: none Responsibilities: - Authenticates users who have registered themselves to access the application - Display an error message if authentication fails - The user can browse and access the application so long as they are not logged	Parent class: none Subclasses: none Responsibilities: - Require user to fill the mandatory fields for registering: - email - Email Address - Password - User need to follow some rules while making a new password - Not similar with personal

and login - Able to authenticate if user is logged in or not Collaborations: - Sign up - Sign up for Recruiter Profiles - Sign up for Job-seeker Profiles - Login	out or the session has timed-out. - Allows the user to reset their password - Provides a link to users who sign-up if they haven't registered themselves before. Collaborations: - Users - Signup	information - Passwords should be at least 8 characters. - Password can't be entirely numeric - Display a Welcome message after signing up. Collaborations: - Users
Profile for Recruiter: Parent class: User Subclasses: none Responsibilities: - Save all user information including name, company, age,bio,user_id, workExp,jobPosts, current Status, and profile picture - Recruiter can view Job Seekers Profiles Collaborations: - Users - Post - Profile for Recruiter	**Profile for Job-seeker:** Parent class: User Subclasses: none Responsibilities: - Save all user information including name,user_id,phoneNumber,phoneNumber, age,bio,workExperience,education,appliedPost, currStatus, profile picture, and resume - Job Seekers can view Recruiter Profiles Collaborations: - Users - Post - Profile for Job Seeker	**Post:** Parent class: Noe Subclasses: none Responsibilities: - Save all post information including companyName,role,description,numofApplicants,recruiter_id, Collaborations: - Recruiter

Post-Job Seeker: Parent class: none Subclasses: none Responsibilities: - Create the relationship between Job seeker and Post - Job seekers can view all job post - Job seekers can apply to many job Post - Job seekers can view all the job application post that they have applied to - Job seekers can view the application status for all the job application that they have applied to - Job seekers can select interview dates for each post if they get selected for an interview - Job seekers can select view OA links for each post if they get selected for an interview Collaborations: - Collaborations: - Profile for Job-seeker - Post	**Post-Recruiter:** Parent class: none Subclasses: none Responsibilities: - Create the relationship between Recruiter and Post - Recruiter can create many job post - Recruiters can view all the job applications for each of their Job Post that they have posted - Recruiters can update the application status of Job Seeker who have applied to their jo posts - Recruiter can send interview dates to JobSeeker - Recruiter can send OA links to JobSeeker Collaborations: - Profile for Recruiter - Post -	**Company:** Parent class: none Subclasses: none Responsibilities: - Save all post information including companyName,about, createrId,JobPosts, reviews, companyImage Collaborations: - Recruiter
JobSeeker-Company: Parent class: none Subclasses: none Responsibilities: - Create the relationship between Job seeker and Company	**Recruiter-Company:** Parent class: none Subclasses: none Responsibilities: - Create the relationship between Recruiter and Company	

- Job seekers can write reviews for company all job post - Job seekers can view reviews for a company - Job seekers can view Company Page. - Job Seeker can view Company Profile image Collaborations: - Company - Job Seeker	- Recruiter can create a company page - Recruiter can view company page - Recruiter can view Company Profile image Collaborations: - Company - Recruiter	

Description of System Interaction with Environment

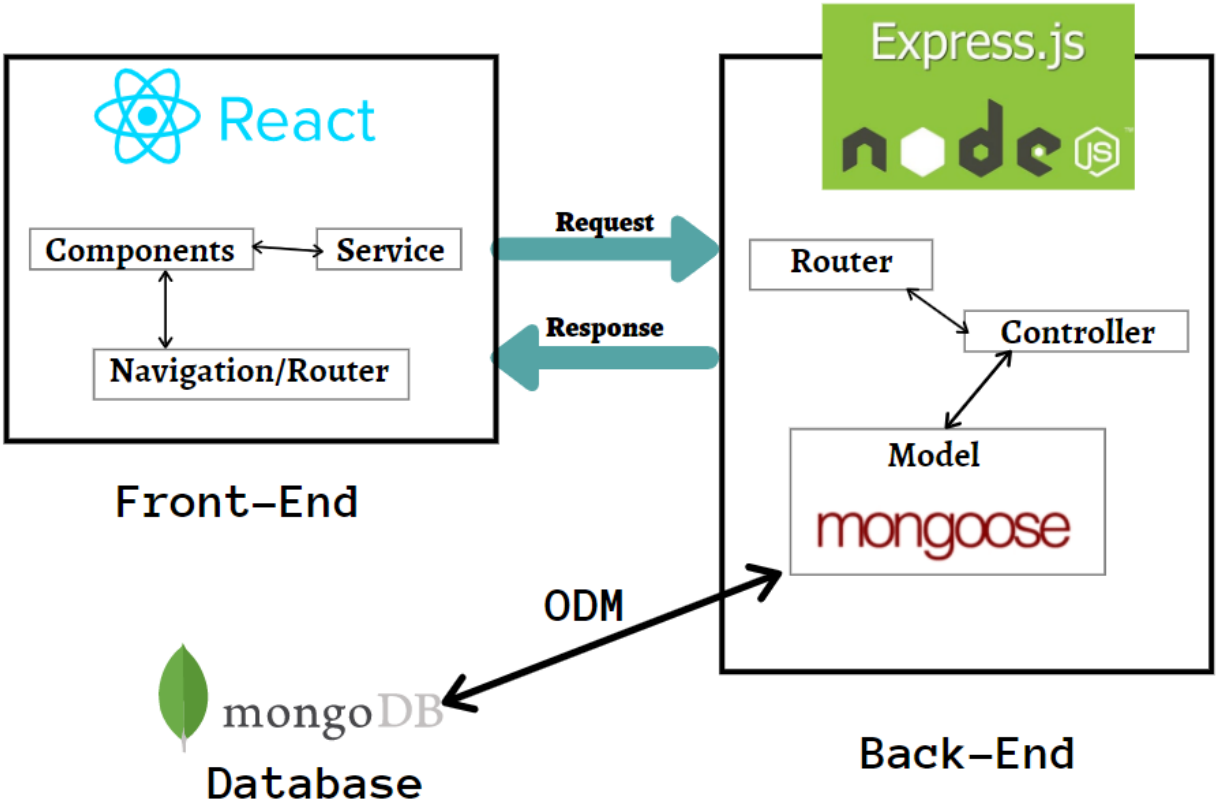
The user must have a working browser with internet access on any technological device. Through the browser, the user is able to access the site and can login or sign up using a valid email and password. Upon logging in or signing up successfully, the user can browse and access the application so long as they are not logged out or the session has timed-out. Every user will have a profile where their profile name, and other basic info can be added or changed at will. If the user is uploading any data from their device (i.e for profile pictures or uploading documents), access to their system file storage will be required.

To run the web application locally, the steps are as follows: clone the Git repo to your device and run “sh firstinstall.sh” to install all dependencies and then run “sh runapp.sh” to start the web app. The sh script takes care of the installation and activation. The default port for accessing the frontend by which the app can be accessed is http://localhost:3000

Description of Architecture of the System

The software architecture model being used is the three-tiered architecture , where we are specifically using the MERN Stack. MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack: React(.js) makeup the top (client-side /frontend) , Express and Node make up the middle (application/server) tier, and MongoDB makes up the

bottom(Database) tier. System Decomposition explains the relationship better below. The software architecture diagram below details the interaction of varying components in the system.



System Decomposition

The MERN architecture allows us to easily construct a three-tier architecture (front end, back end, database) .

The top tier of the MERN stack is React.js, the declarative JavaScript framework for creating dynamic client-side applications in HTML. React lets you build up complex interfaces through simple components, connect them to data on your back-end server, and render them as HTML.

The next level down is the Express.js server-side framework, running inside a Node.js server. Express.js bills itself as a “fast, unopinionated, minimalist web framework for Node.js,” and that is indeed exactly what it is. Express.js has powerful models for URL routing (matching an incoming URL with a server function), and handling HTTP requests and responses.

The next level down is the MongoDB database tier.MongoDB works extremely well with Node.js, and makes storing, manipulating, and representing JSON data at every tier of your application incredibly easy. For cloud-native applications, MongoDB Atlas makes it even easier, by giving you an auto-scaling MongoDB cluster on the cloud provider of your choice, as easy as a few button clicks.

There are several error prevention schemas in place, and a few are described in further detail: Errors during login and registration where common passwords are prevented from being used, as well as checking for valid email and password during each process. Username is also cross checked with the database to avoid creating duplicate usernames.