CoBuild

System Design Document

Ansh Aneel

Andrew Aucie

Ashwin Malik

Dhruvin Patel

Vikram Nara

Vedat Gotkepe

Minjun Kim

Contents

Front-End

Back-End

Software Architecture

Front-end

Class Name: CreatePost	
Parent Class (if any): N/A **Subclasses (if any):** N/A	
Responsibilities: ● Extracts text from fields in create job posting page and runs API call to connect to backend. ● Render a form for creating a job posting ● Handle user input for job details such as title, location, job description, company name, deadline, and skills ● Allow users to upload files for the job posting ● Handle the submission of the job posting form ● Display a grid of job postings from a predefined list	**Collaborators:** ● React ● RxOpenInNewWindow (from 'react-icons/rx') ● Container, Grid, Box, Typography, alpha, Button, TextField, IconButton (from '@mui/material') ● CloudUpload (from '@mui/icons-material') ● './CreatePost.css' (custom CSS file) ● JobGrid component

React Component(s): JobPosting	
Parent Class (if any): React.Component **Subclasses (if any):** N/A	
Responsibilities: ● Render the view for a job posting ● Calculate the time remaining until the application deadline ● Format time values for display ● Fetch job post data from the server using the provided API URL and post ID ● Set the state variables for position name, company name, location, description, tags, and target date ● Display the remaining time until the application deadline ● Display the position name, company name, location, description, and tags of the job posting	**Collaborators:** ● React (useState, useEffect) ● react-router-dom (useParams)

React Component(s): Jobs

Parent Class (if any): React.Component
Subclasses (if any): N/A

Responsibilities:	**Collaborators:**
● Render the view for a list of jobs ● Fetch job IDs from the server using the provided API URL ● Set the state variable for job IDs ● Pass the job IDs as props to the JobPosting component	● React (useState, useEffect) ● react-router-dom (Link) ● JobPosting component

React Component(s): Login

Parent Class (if any): React.Component
Subclasses (if any): N/A

Responsibilities:	**Collaborators:**
● Render the view for the login page ● Manage state variables for username, password, and login error message ● Handle changes in the username and password fields ● Handle form submission for login ● Make a POST request to the server to authenticate the user ● Set the login error message if authentication fails ● Use the react-auth-kit library to handle user authentication and store the access token ● Redirect the user to the jobs page upon successful login	● React (useState) ● @mui/material (AppBar, Avatar, Typography, Button, TextField, BottomNavigation, Toolbar, InputAdornment) ● react-router-dom (Link, useNavigate) ● react-auth-kit (useSignIn

React Component(s): Signup

Parent Class (if any): N/A
Subclasses (if any): N/A

Responsibilities:	**Collaborators:**
● Render a sign-up form with input fields for name, email, username, password, and other information. ● Handle user input changes for name, email, username, password, and other fields.	● React: Used to create functional components and

● Handle form submission and send a sign-up request to the server. ● Reset input fields after successful sign-up or display an error message on failure. ● Apply styles and animations to the form and input fields. ● Navigate to the login page when the "Sign In" button is clicked.	manage component state. ● react-router-dom: Used for navigation to the login page. ● useState: Hook for managing state variables. ● Fetch API: Used for making HTTP requests to the server

React Component(s): Assessment

Parent Class (if any): N/A
Subclasses (if any): N/A

Responsibilities:	**Collaborators:**
● Assess users' code submission and run their code against test cases. ● Displays a leetcode type problem and a code editor for users to solve the problem.	● java: programming language for running test cases. ● useState: Hook for managing state variables. ● Backend API: Collaborates with the backend to fetch code and submission data

React Component(s): Leaderboard

Parent Class (if any): N/A
Subclasses (if any): N/A

Responsibilities:	**Collaborators:**
● Displays a leaderboard of users who completed a specific leetcode type problem. ● Ranks you amongst other applicants, displaying statistics like time taken, complexity, and the number of test cases passed.	● JobPosting: Represents a job posting with details like title, description, and requirements. ● User: Represents the list of users in the leaderboard. ● Backend API: Collaborates with the backend to fetch scores of users

React Component(s): SignUpRecruiter	
Parent Class (if any): N/A **Subclasses (if any):** N/A	
Responsibilities: ● Allows recruiters to sign up using their company name, job position, recommended skills, field, etc. ● Upload relevant files or documents	**Collaborators:** ● useState: Hook for managing state variables. ● materialUI

React Component(s): SubmissionForm	
Parent Class (if any): React.Component **Subclasses (if any):** N/A	

Responsibilities:
● Render the form for users to submit their application for a specific job posting
● Validate the input data entered by the user
● Submit the application to the backend for processing
	Collaborators:
	`User` (model)
	`JobPosting`
	`BackendAPI`

React Component(s): ApplicationDialog	
Parent Class (if any): React.Component **Subclasses (if any):** N/A	
Responsibilities: ● Display dialogues for when you apply to a job posting.	**Collaborators:** `User` (model): Provides information about the user receiving the

	notification `Recruiter` (model): Provides information about the recruiter receiving the notification `BackendAPI`

React Component(s): BottomNavBar	
Parent Class (if any): N/A **Subclasses (if any):** N/A	
Responsibilities: ● Provides a bottom navigation bar to be used throughout the app.	**Collaborators:** ● @mui/materialUI

React Component(s): JobBox	
Parent Class (if any): React.Component **Subclasses (if any):** N/A	
Responsibilities: ● Provides a card template for job postings, a company image above a job description	**Collaborators:** ● makeStyles ● @mui/material

React Component(s): Step1
Parent Class (if any): React.Component **Subclasses (if any):** N/A

React Component(s): UserProfile
Parent Class (if any): React.Component **Subclasses (if any):** N/A

Responsibilities: ● Displays a profile of a user, showing saved and applied applications as well as their user information such as university, courses, etc.	**Collaborators:** ● makeStyles ● JobBox: to incorporate job postings

Back-end

Class Name: JobSchema	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Store and define the structure of a job document in the database. ● Define the fields and their data types for a job document, including: o jobId: String o title: String o location: String o jobDescription: String o deadline: String o companyName: String o datePosted: String o skills: Array of strings	**Collaborators:** mongoose

Class Name: UserLoginSchema	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Store and define the structure of a user login document in the database. ● Define the fields and their data types for a user login document, including: o username: String (required) o password: String (required)	**Collaborators:** mongoose

Class Name: SignUpSchema	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: ● Store and define the structure of a sign-up document in the database. ● Define the fields and their data types for a sign-up document, including: o username: String o email: String o name: String o password: String o resume: String o transcript: String	**Collaborators:** mongoose

Class Name: Server.js	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: ● Create and configure an Express application. ● Set up routes for various API endpoints. ● Establish a connection to the MongoDB database. ● Start the server and listen on port 8000.	**Collaborators:** express: Express framework mongoose: MongoDB library cors: Cross-origin resource sharing library app: Express application instance

● Handle incoming HTTP requests and route them to the corresponding API handlers.	assesmentApi: "/createassesment" endpoint handler getPost: "/getpost" endpoint handler createPost: "/createpost" endpoint handler userLogin: "/login" endpoint handler signUpRequest: "/signup" endpoint handler

Class Name: Assessment	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: ● Create and define the assessment schema in MongoDB ● Handle the POST request for creating a new assessment ● Generate a UUID for the assessment ● Save the new assessment in the database	**Collaborators:** mongoose uuidv4 AssesmentModel

Class Name: getpost.js	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: ● Define the job schema in MongoDB ● Handle the POST request for creating a new job posting ● Generate a unique job ID using short-uuid and uuidv4 ● Convert the deadline to ISO format ● Save the new job posting in the database	**Collaborators:** mongoose uuidv4 short jobschema job

Class Name: Loginuser.js
Parent Class (if any): **Subclasses (if any):**

Responsibilities:	**Collaborators:**
● Define the user login schema in MongoDB ● Handle the POST request for user login ● Validate the provided username and password ● Generate an access token using JWT ● Return the access token upon successful login	userLoginSchema mongoose jwt dotenv

Class Name: Signup.js	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: ● Define the user sign-up schema in MongoDB ● Handle the POST request for user sign-up ● Validate the uniqueness of the email and username ● Create a new user instance and save it to the database ●	**Collaborators:** signUpSchema mongoose

Class Name: Assessments	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Maintain a leaderboard of users based on their assessment scores ● Calculate and update user rankings based on assessment scores ● Store user information, such as username and assessment scores ● Provide methods to add new users and update their assessment scores ● Retrieve and display the leaderboard in descending order of scores	**Collaborators:** User: Represents a user in assessments Assessment: Represents an assessment with score access Database: Stores user info and assessment scores Sorting algorithm: Sorts leaderboard based on scores

Dependencies:

```
"@codemirror/lang-cpp": "^6.0.2",
"@codemirror/lang-java": "^6.0.1",
"@codemirror/lang-javascript": "^6.1.9",
"@codemirror/lang-python": "^6.1.3",
"@emotion/react": "^11.11.1",
"@emotion/styled": "^11.11.0",
"@fontsource/work-sans": "^5.0.3",
"@monaco-editor/react": "^4.5.1",
"@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.13.6",
    "@mui/styles": "^5.13.2",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "@uiw/codemirror-theme-dracula": "^4.21.7",
    "@uiw/react-codemirror": "^4.21.7",
    "monaco-editor": "^0.39.0",
    "react": "^18.2.0",
    "react-auth-kit": "^2.12.2",
    "react-dom": "^18.2.0",
    "react-icons": "^4.9.0",
    "react-router-dom": "^6.12.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
```

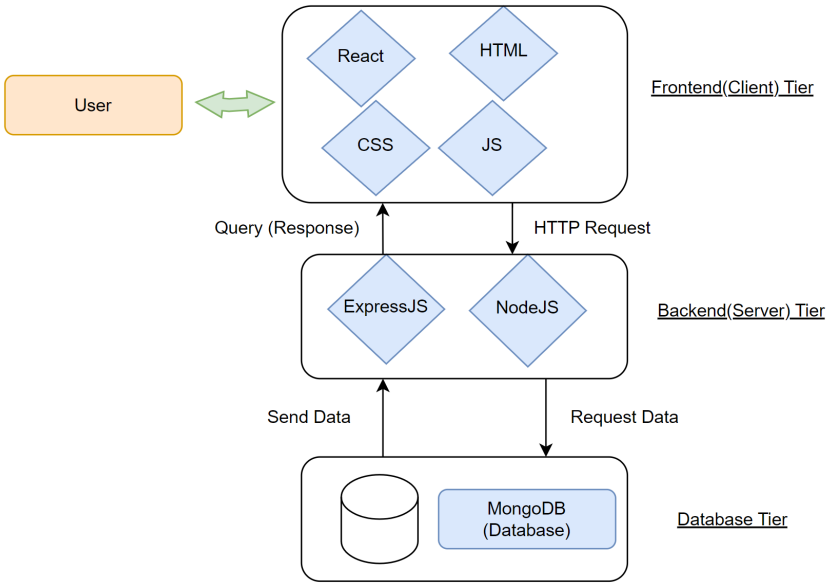
Project Configuration Summary:

Operating System: Any
Languages: JS, JSX
Compiler: BABEL
Virtual Machine: None
Databases: MongoDB, S3 Blob Storage
Server port: 8000
Frontend port: 3000

Software Architecture

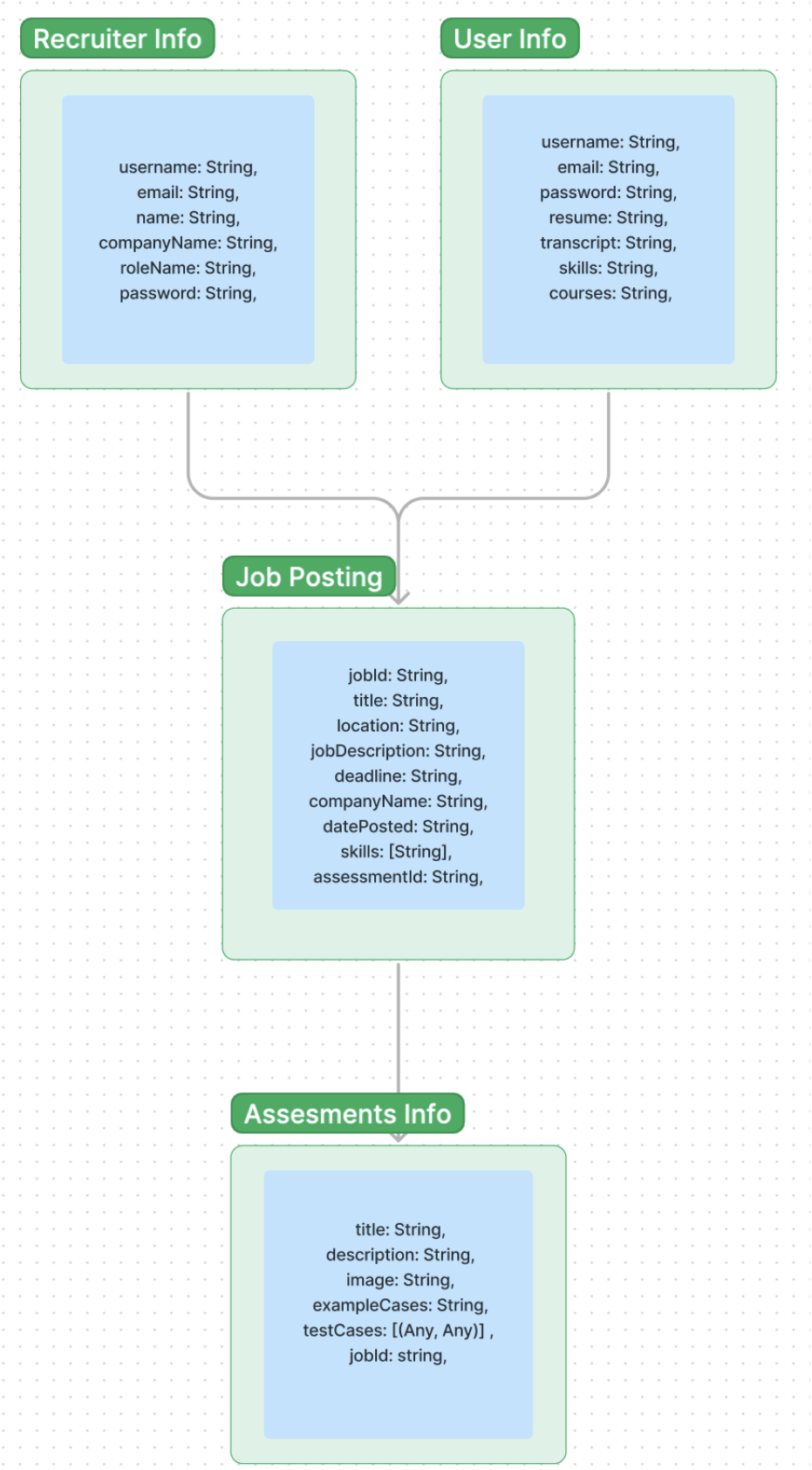
Three-Tier Architecture

This project utilized the Three-Tier Architecture (MERN Stack). The user interacts with React (the client) frontend of the web application. Any user requests will enable React to send a HTTP request to the application server which is written using NodeJS and ExpressJS. The server then serves the client by accessing the MongoDB database.



System Decomposition

The database, the front end, and the back end are the three distinct components of the system architecture. The users will be engaging with the frontend. React and its components are used in the front-end development. For universal styling and the development of components with smoother operation, we also employed Material UI. Using the fetch call, we can also access the API endpoints that the backend offers. In this manner, we can enable the introduction of CRUD properties in our application. The app's backend was developed using Node.js and Express.js. Express is a well-liked node framework with several advantages. Writing handlers for HTTP requests at various URL paths (routes) is one of the ones we used. This is how we configure the API endpoints that communicate with our MongoDB database. We have created distinct collections in our database, including job postings, assessments, and user profiles. To prevent receiving invalid inputs, we have placed input sanitation code in place.


Recruiter Info
username: String,
email: String,
name: String,
companyName: String,
roleName: String,
password: String,
User Info
username: String,
email: String,
password: String,
resume: String,
transcript: String,
skills: String,
courses: String,
Job Posting
jobId: String,
title: String,
location: String,
jobDescription: String,
deadline: String,
companyName: String,
datePosted: String,
skills: [String],
assessmentId: String,
Assesments Info
title: String,
description: String,
image: String,
exampleCases: String,
testCases: [(Any, Any)] ,
jobId: string,