Team Anything Works
System Design documentation

By:

Juefei Lu, 1006878602

Faizan Riaz, 1004927074

Xiaohan Lu, 1005683404

Parth Solanki, 1005244415

Tien-Thanh Le, 1004956906

Vanshika Virmani, 1006865251

Ahmed Elkady, 1005342988

Table of contents

Frontend Pages

Class Name: Landing	
Parent Class (if any): Subclasses (if any):	
Responsibilities: - Directs user to correct signup/login page depending on whether they are an employer or a user	**Collaborators:** - Footer
Class Name: LoginSignup	
Parent Class (if any): Subclasses (if any):	
Responsibilities: - Directs user to correct login/signup page depending on whether they are an employer or a user - Contains various fields for creating an account - Contains fields as well as a button for user to login - Send login and signup details to the appropriate endpoints in the backend	**Collaborators:** - NavBar - Footer - Redux
Class Name: Dashboard	
Parent Class (if any): Subclasses (if any):	
Responsibilities: - Allows applicants to see the job board - Allows recruiters to see the postings	**Collaborators:** - Redux - JobBoard
they listed - Directs applicants to individual job description - Directs users to their profile - Directs applicants to favorite jobs and applied jobs	- Calendar - NavBar - Footer - PostedJobs
Class Name: Profile	
Parent Class (if any): Subclasses (if any):	

Responsibilities: - Allows users to edit some of their personal information - Allows Employer to view information and elevator pitch - Allow users to add or edit their video elevator pitch - Allow user to add a default resume	**Collaborators:** - Footer - NavBar - Redux
Class Name: JobListing	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Allows users to view information regarding job posting - Allows Employers to edit the job information - Lets users click to submit a job application - Provides user with the option to add a custom elevator pitch video - Provides user with the option to add a new Resume or Cover Letter - Allows Employers to view applicants to that positions - Directs Employers to profile page of selected applicants - Allows Employers to request meetings with applicant	**Collaborators:** - Redux - Footer - NavBar
Class Name: JobPosting	
Parent Class (if any): Subclasses (if any):	
Responsibilities: - Allows Employers to post to employee job board - Contains various fields for Employer to add key information regarding the position	**Collaborators:** - Footer - NavBar
Class Name: Calendar	
Parent Class (if any): Subclasses (if any):	

Responsibilities: - Allows the recruiter to reschedule interviews with applicants - Displays all scheduled interviews - Displays interview requests - Displays Applicants' responses to interview requests (including available times)	**Collaborators:** - NavBar
Class name: JobHistory	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Allows an applicant to view all the jobs they've applied to before	**Collaborators:** - NavBar - Footer

Frontend Components:

Class Name: NavBar	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Have a button to go to the dashboard - Have a button to go to the calendar - Have a button to go to the profile - Have a button to sign out	**Collaborators:** -
Class Name: JobBoard	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Display all relevant jobs - Directs applicant to each job display (JobListing page) - Provides a search option to view specific jobs - Provides filtering option to filter searches - Displays favorited jobs	**Collaborators:** - JobCard
Class Name: JobCard	

Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Displays a brief summary of the job description - Directs applicant and employer to the JobListing	**Collaborators:**
Class Name: PostedJobs	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Displays all active and inactive listings posted by an employer - Directs Employer to JobPosting	**Collaborators:** - JobCard
Class Name: Footer	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Provides the users with social links to contact the developers	**Collaborators:** -

Frontend Services:

Class Name: UserService	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Create an account - Authenticate users - Get User Profile - Modify profile	**Collaborators:** - User
Class Name: JobService	
Parent Class (if any): **Subclasses (if any):**	

Responsibilities: - Create a job posting - Modify a job posting - Delete a job posting - Fetch a job posting - Apply to a job posting - Fetch applicants to a job posting	**Collaborators:** - Job

Backend Services:

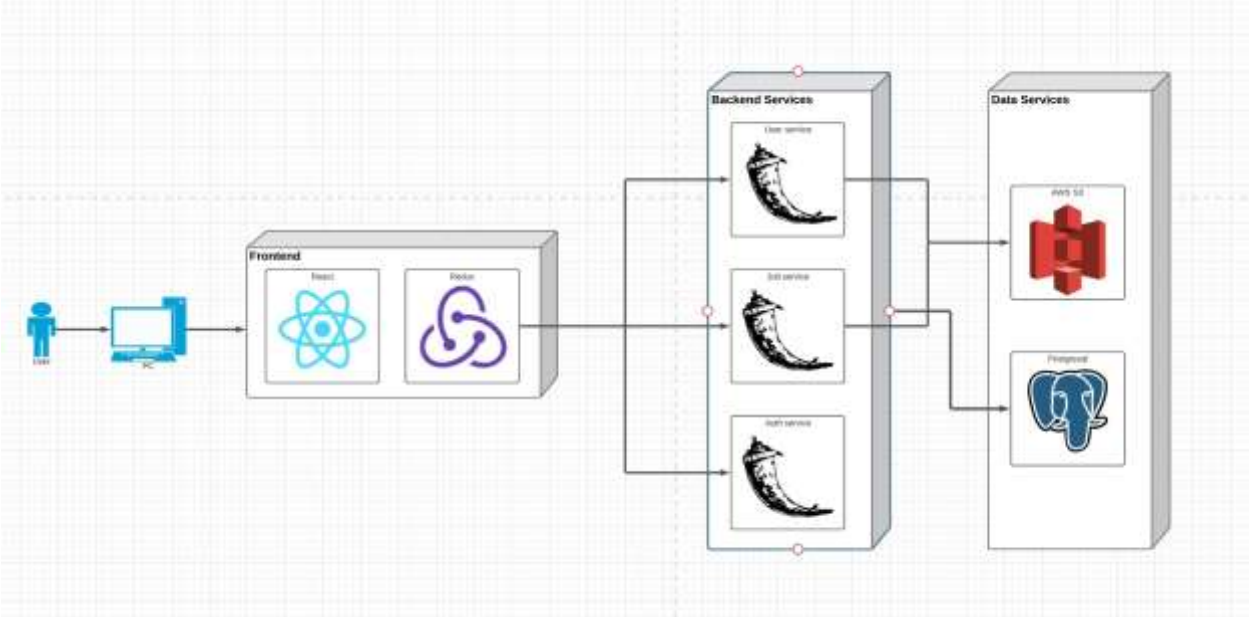
Class Name: User	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Standard CRUD Operations on the User model - Signup - Login - Get user profile - Modify profile - Authenticate	**Collaborators:** - UserService
Class Name: Job	
Parent Class (if any): **Subclasses (if any):**	

Responsibilities: - Standard CRUD operations on the jobs model - Create a job posting - Modify a job posting - Delete a job posting - Fetch a job posting - Apply to a job posting - Fetch applicants to a job posting	**Collaborators:** - JobService
Class Name: Job Schema	
Parent Class (if any): **Subclasses (if any):**	

Responsibilities: - Store job posting data - Columns: positionName,Company,DatePosted,JobDescription	**Collaborators:** - JobService
Class Name: User Schema	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Stores user data - Columns: Name,Age,Introduction,ProfilePicture	**Collaborators:** - UserService
Class Name: Auth Schema	
Parent Class (if any): **Subclasses (if any):**	
Responsibilities: - Stores user authentication data - Columns: Name,Email,Password,Role,Status	**Collaborators:** - AuthService

Architecture Design:

https://lucid.app/lucidchart/0e707c21-dcfa-4f5d-89fd-9a2b6f56de7c/edit?viewport_loc=-1542%2C-748%2C3290%2C1895%2C0_0&invitationId=inv_33d40371-777b-47ae-857c-4dbf9c24bf46



System Interaction with Environment:
The users of “easy apply” must have a device with an internet connection and a browser. They must sign up with a valid email and password. They will be able to access the web app whenever they log in. Applicants can upload a video elevator pitch to their profile and/or to any job postings. Any user will be able to upload a picture for their profile. Both of which require the user to allow the application to access their local file system.

The web application

User	The user is someone who interacts with the easy-apply application.
Browser	The browser allows the web application to be viewed.
React	The views of the web application.
Redux	The state control container for the views.
Flask Job service	The job service is a REST API that is accessed by Redux. It performs data manipulation on the postgresql server on the Job table.
Flask User service	The User service is a REST API that is accessed by Redux. It
	performs data manipulation on the postgresql server on the User table. It will be in charge of authentication as well.

Postgresql	A SQL database that is used to store long-term application data such as job postings, user information.
AWS S3	A cloud object store that is used to store media content / blobs such as images and videos.

System decomposition:

React

React acts as the view in the Redux/Flux architecture pattern. React creates the application's user interface for the User. When a user interacts with the interface, Redux will intervene accordingly to update the state of the application and update the UI. React will display errors that are returned by Redux when an action is dispatched. For example, whenever a user logs in with the wrong credentials. There will be a UI element to inform the user that they entered incorrect/invalid credentials.

Redux

Redux will be used as the state management container. When a triggers an event on the UI, (ie. interacts with the UI) this will send an action to the Redux Store more specifically the dispatcher. This action is linked to a reducer which will handle all API calls and state modification on the store. The reducer calls our flask services and retrieves data. This data is used to modify the state in the Redux Store. After the state is updated by the reducer, this new state will be used to update the component.

Backend

API calls from redux will hit the backend which hosts our 3 services, User, Authentication and Job.

This is responsible for sending application data back to Redux to store application states. Each service has a model for the table they are modifying, routes which defines all the api endpoints and lastly a controller to define endpoint logic.

User Service

The user service will have its controller to create, delete, update, and fetch user data which is going to be stored on the Postgresql server. The User service will be tied to the Authentication service. The user service will allow Redux to fetch and upload images and videos that will be stored in AWS S3.

Authentication Service

The authentication service will match username hashes and password hashes given by Redux inside our Postgres database to authenticate users.

Job Service

The job service will have its controller to create, delete, update and fetch jobs and their descriptions that are stored in the Postgresql server. The user service will allow Redux to fetch and upload images that will be stored in AWS S3.

A note on MVC, Flux and Redux:

We are using React for the frontend and the common architecture design pattern for React is called FLUX which is a modification on MVC by the engineers at facebook. More specifically, we will use Redux which is an implementation of this. MVC and Flux differ on several aspects: flux promotes unidirectional flow of data.

https://www.clariontech.com/blog/mvc-vs-flux-vs-redux-the-real-differences
https://facebook.github.io/flux/

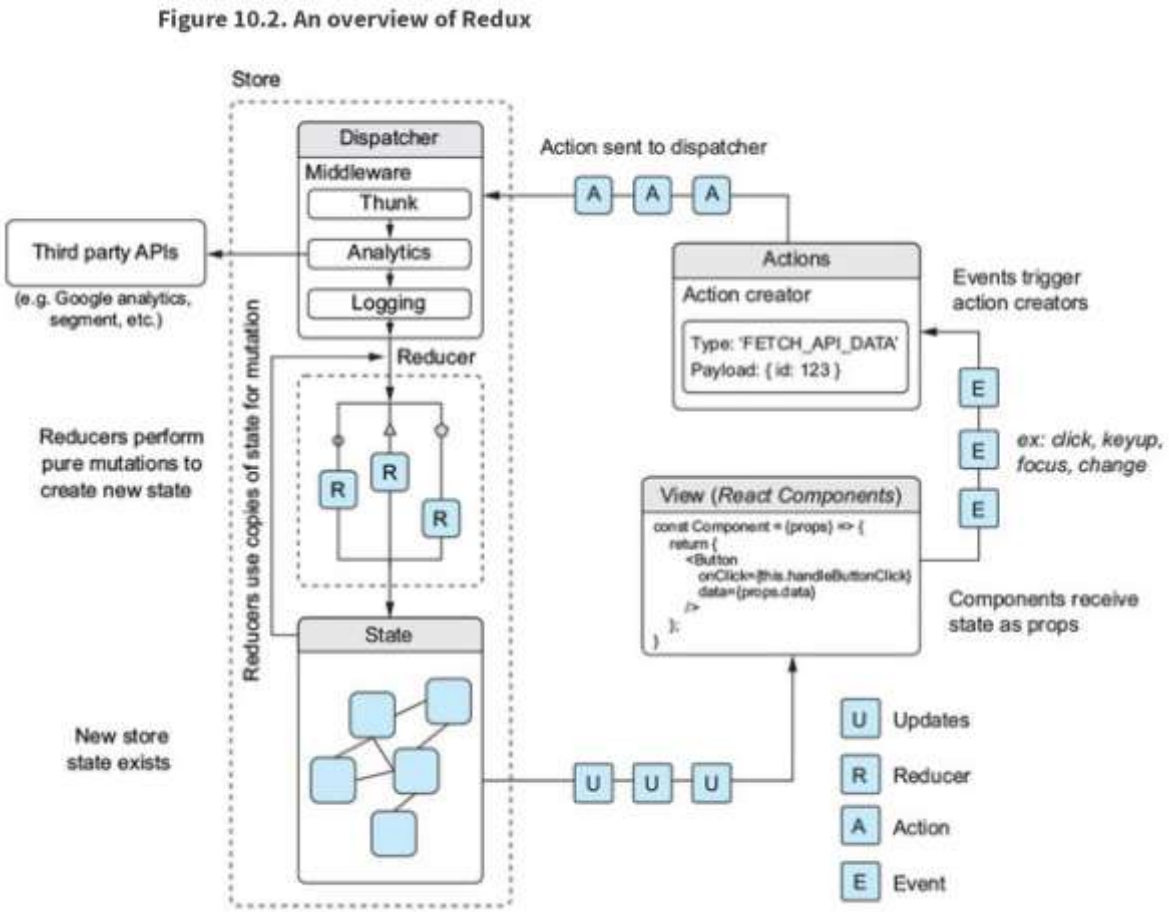


Figure 10.2 from Thomas, M.T. - React in Action-Manning Publ. (2018)