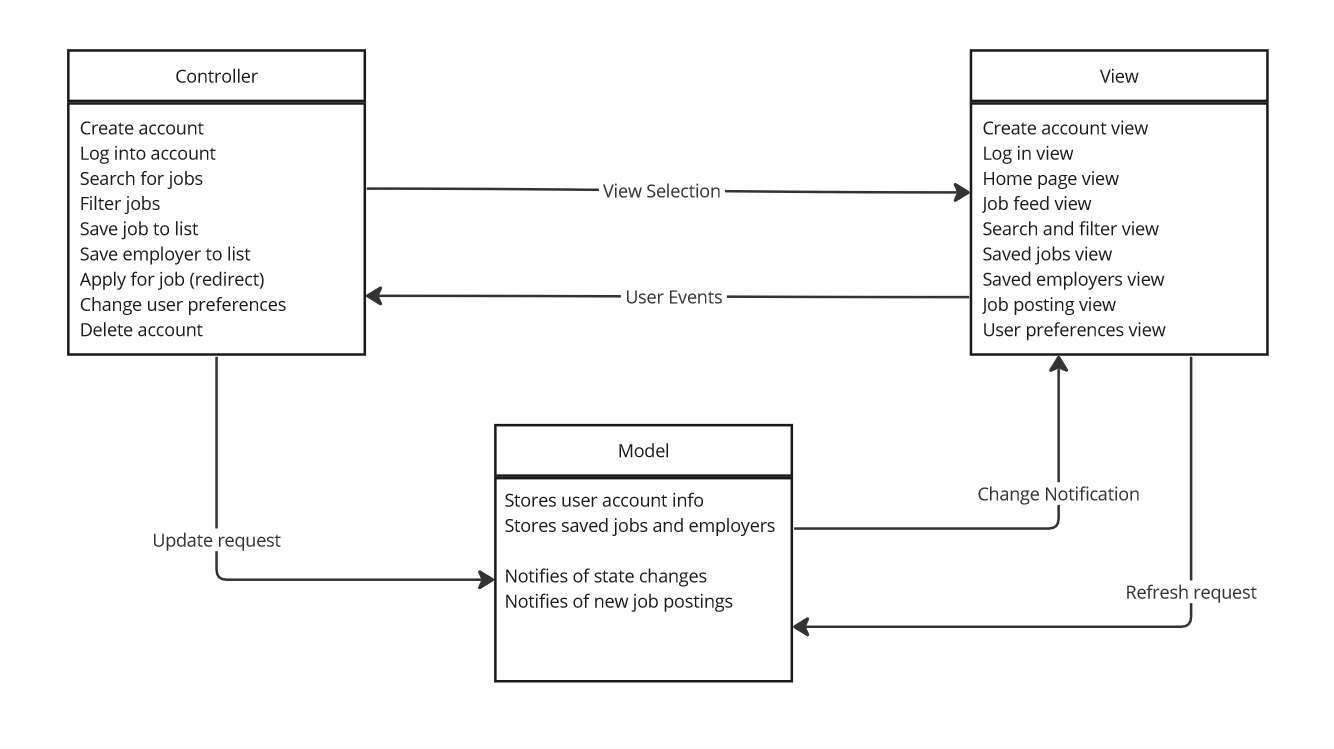
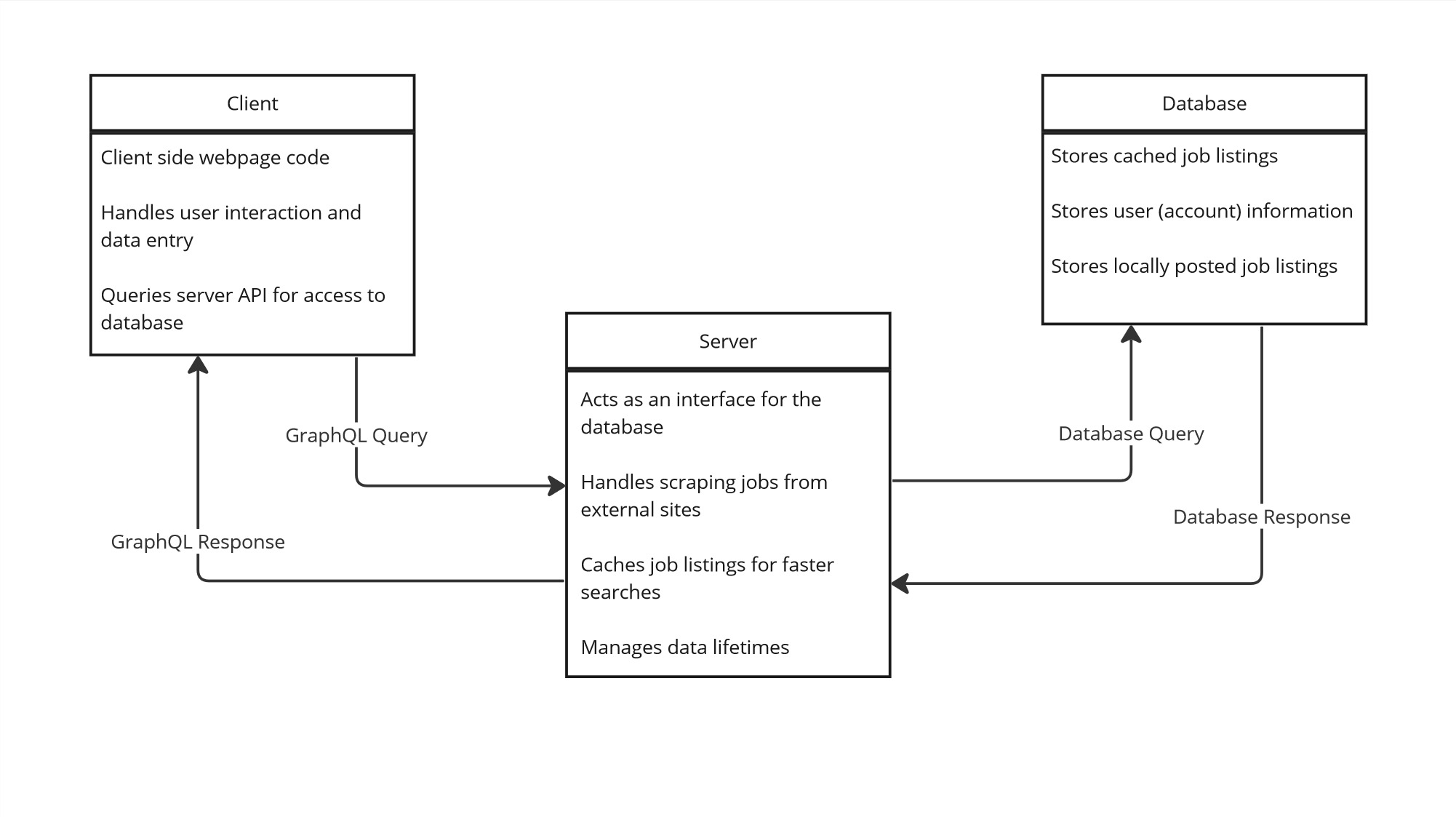
**Team: D - College Jobs**

**Architecture Design:**

For the architecture design of CollegeJobs, we use a combination of Model-View-Controller architecture and Client-Server architecture. MVC is ideal for building interactive web applications. We use React for our front-end, which is built on the MVC architecture, so that is how our user interface is architected. Since we scrape jobs from other sites, a time and compute intensive process, we decided to introduce a central server to scrape and cache job listings. This also simplifies and speeds up the client side web interface, because we can send everything through our server API, which offloads resources requirements from client to server.

**Architecture Design Diagrams**:

[](https://miro.com/app/board/uXjVNa4EnLM=/?share_link_id=15467409899)



**Design Description:**

The architectural pattern consists of five components; three for the user interface: the view, the controller and the model; two for the job scraping and data handling: the client (webpage), and the server.

**MODEL-VIEW-CONTROLLER (MVC)**

The View for CollegeJobs handles layout and display of the UI. React and Bootstrap do most of the heavy lifting here. The Home Page view will be the root view, and will be the default view for unregistered users. The Job Feed view will be the default view for registered users. Create account, log in, home page, job search, and job details views will be accessible to unregistered users, but all other views will redirect to the login view unless the user is logged into a valid session. All pages will have some version of a navigation component across the top of the screen to access the other pages on the site.

The view layer will include Bootstrap for css layout.

**[Excursus] List of Views and Components**

All views will have the same footer component with copyright and contact information, and all views will have a header bar that will include a navigation component, and sometimes a search bar. Listed below are the components that are unique to each view.

Home page view

- Simple home page for new users to land on

* BigCentralSearchBubble
* SiteSummary

Log in view

* LogInForm

Create account view

* CreateAccountForm

Search/Filter view

* SearchBar (in header)
* FilterBoxAside
* ResultsSection
  + JobSummary

Job feed view + Saved jobs view

* ResultsSection (same component, different job query)
  + JobSummary

Saved employers view

* EmployerList
  + EmployerSummary

Job posting view

* JobName
* JobDetails
* JobDescription

User preferences view

* PasswordChangeForm
* DefaultFilterBox
* DeleteAccount

The Controller handles events from the View and communication with the server API. Events from the view vary, and include button presses to perform actions like logging in or out, searching for jobs, and selecting job filters. When it detects input, the controller notifies the view of changes, and notifies the model of changes to state. The controller acts like the central nervous system to coordinate actions between the model and view. When the user clicks login, for example, the controller tells the model to check the entered credentials against the database of users.

The controller layer will include Passport.js to handle user authentication.

The Model handles all the data for the app. The view references the model to get any data needed to display elements in the app. When the controller notifies the model of a change in state, the model will push that change out to the view. The model does not only hold the current state of the application, but also handles fetching and handling data from the database. When the user searches for jobs, the model takes their search terms, builds a database query with them, sends that query to the database, and processes the results so that the view can display them.

**CLIENT SERVER**

The Client is the react app that runs in the user’s browser. It can be thought of as the user interface to access the data stored on the server. The client is responsible for sending requests to the server for data on jobs, and taking the data that the server returns and displaying it.

The Server is the node application that acts as a database interface and web scraper. Requests for data come in from the client, the server checks whether it needs to update the job listing database based on the time since last update, and then constructs API calls and webpage queries to get job data from other services. The return from those queries gets formatted for the local database and stored, and while the web processes run in the background the server constructs a database query for the local database. The data from the local database query then gets formatted for the client to digest, and sent as a response to the client. This client-server communication will take place using GraphQL.