**Design “Class” Diagram**

[**Miro Board**](https://miro.com/app/board/uXjVLEpku5Y=/)

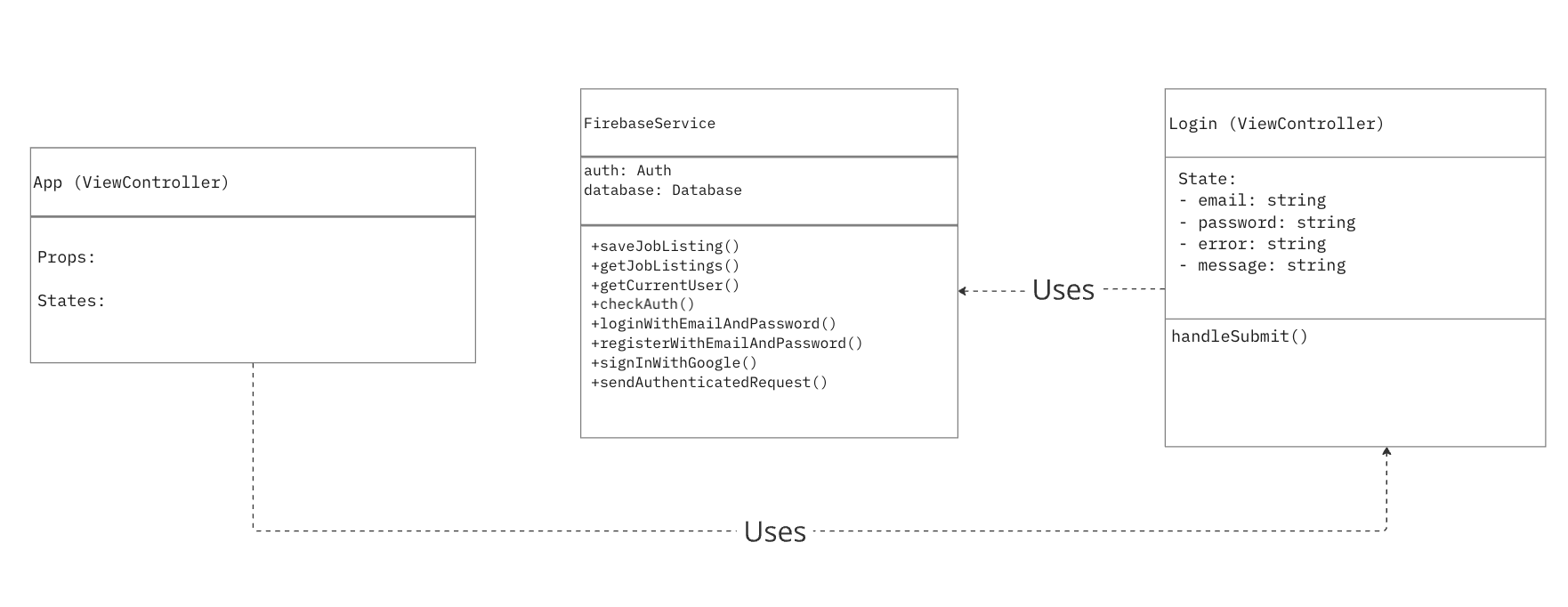
The diagram illustrates a detailed class design of a React application, highlighting component relationships through dotted arrows that indicate dependencies between classes. The arrow direction flows from the dependent class to the class being used. At the core of the architecture is the App component, which serves as the application's root and handles URL-based routing to appropriate components.

Overview:

* Login
  + Allows the user to login to the web application
* Register
  + Allows the user to register a new account on the web application
* Homepage
  + Shows a list of helpful resources to the user and serves as a welcome page after login
* Create Job Listing
  + Allows the user to create and publish a job listing
* View Job Listings
  + Allows the user to search for and apply to job listings

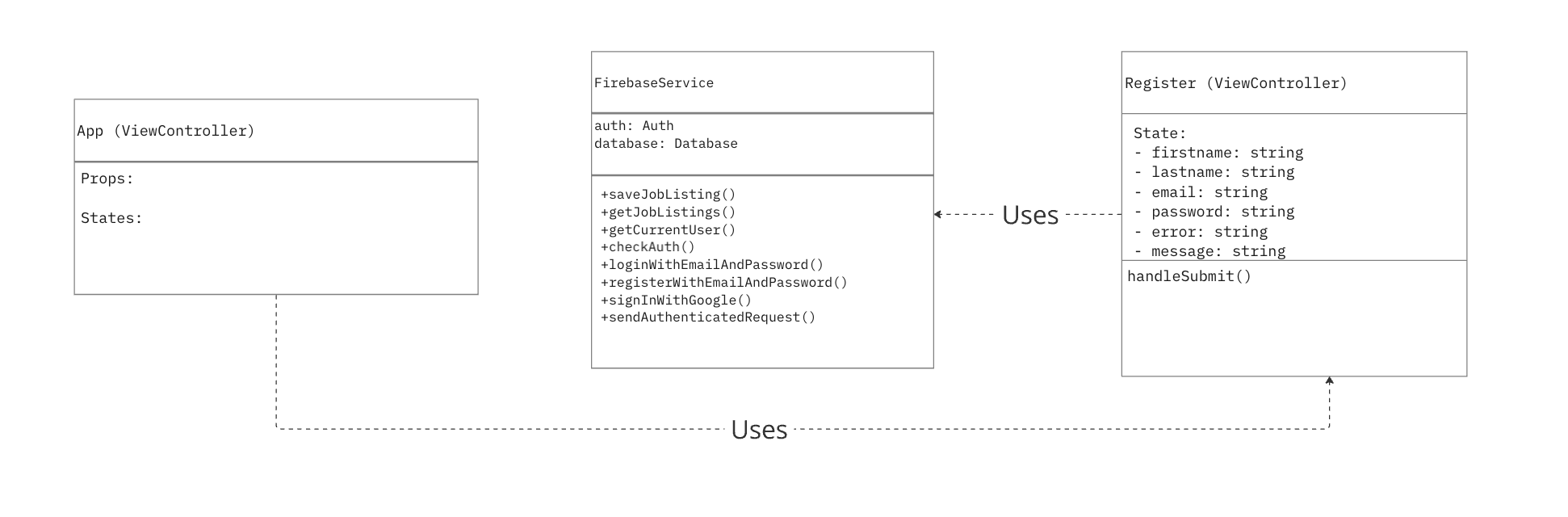
To reduce the complexity of the diagram, we’ve split up the DCD into diagrams representing the pages listed in the overview:

**Login**

****

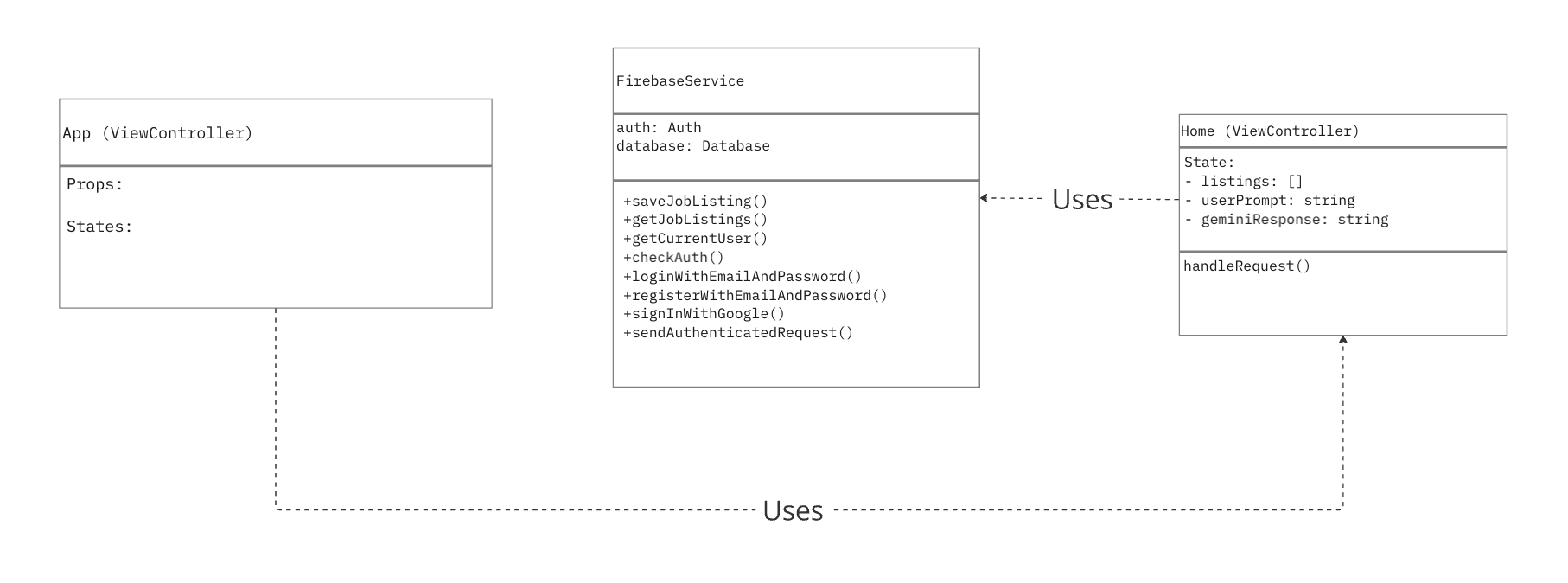
The login page serves as the first page that the user will encounter on page load. The user has the option to sign in with a username or password or sign in with a Google account. Additionally, there is the option to register a new account if the user does not already have one. The login process uses the loginWithEmailAndPassword and signInWithGoogle methods of the FirebaseService class to log the user in.

**Register**

****

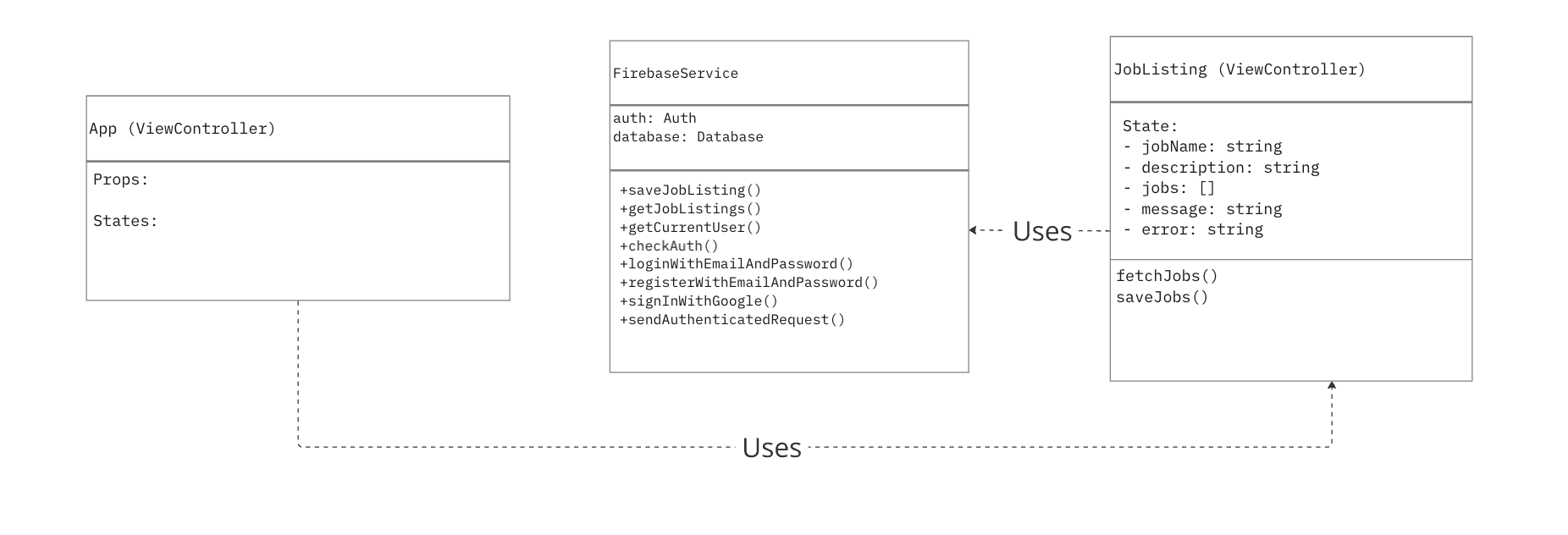
The registration page allows the user to create a new account on the web app. They create a new account with a first name, last name, email, and password. The user has the alternative option to register an account with Google, which in return signs them into the account. The registration process uses the registerWithEmailAndPassword method of the FirebaseService class.

**Homepage**

****

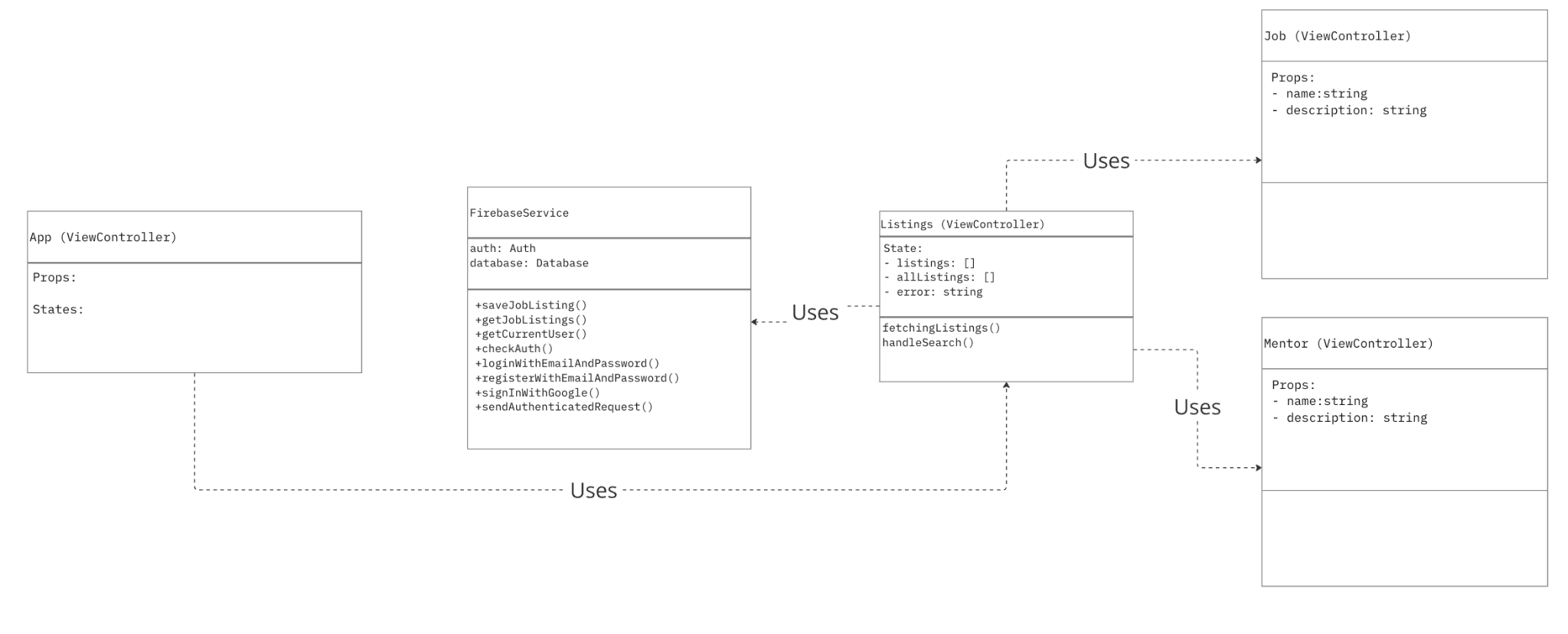
The homepage acts as the primary landing destination following successful user authentication. This welcoming interface serves as a central hub, presenting users with a curated collection of job-seeking resources and interview preparation materials. The page leverages the FirebaseService to fetch and display personalized content, ensuring users have immediate access to valuable tools and materials to support their job search journey.

**Create Job Listing**

****

The Create Job Listing page allows the user to create a job listing with a name and description that describes the listing. The job listing is saved in the database using the saveJobListing method of the FirebaseService class. The listings can later be retrieved with the getJobListings method of the FirebaseService class.

**View Job Listing**

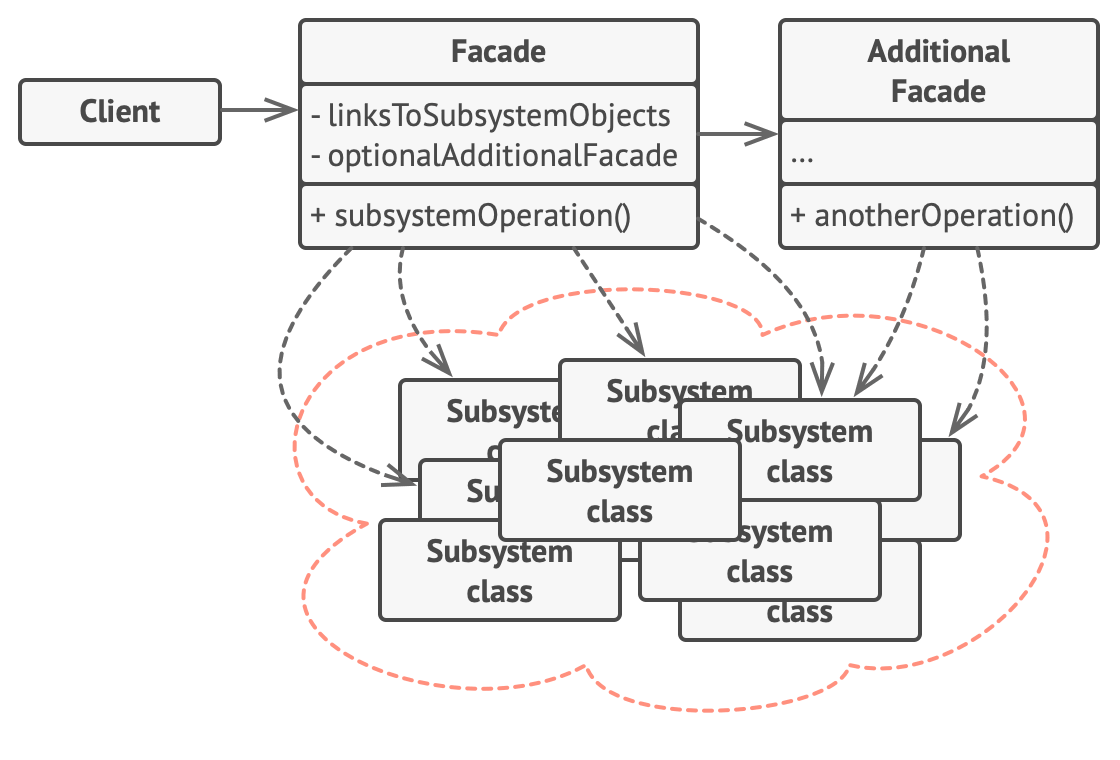
****

On the View Job Listing page, the job listings saved into Firebase are retrieved with the getJobListings method of the FirebaseService class and rendered into the page view using the Job component. The View Job Listing page has the option to search for job listings by their name, which will filter out the matching job listings and render them on the page. The user has the option to click on the job listings and view their full information along with having the option to apply to the job listing.

**Design Patterns**

**Design Pattern: Facade**

<https://refactoring.guru/design-patterns/facade>





The Facade pattern, implemented by FirebaseService in this architecture, is a structural design pattern that provides a simplified interface to a complex subsystem. Just as a building's facade presents a unified exterior while hiding the complex internal structure, the FirebaseService presents a clean, unified API while abstracting away the complexities of Firebase's authentication and database systems. This pattern is valuable in this design as it reduces dependencies on external services, simplifies the component code, and makes the system more adaptable to changes in the underlying Firebase implementation.