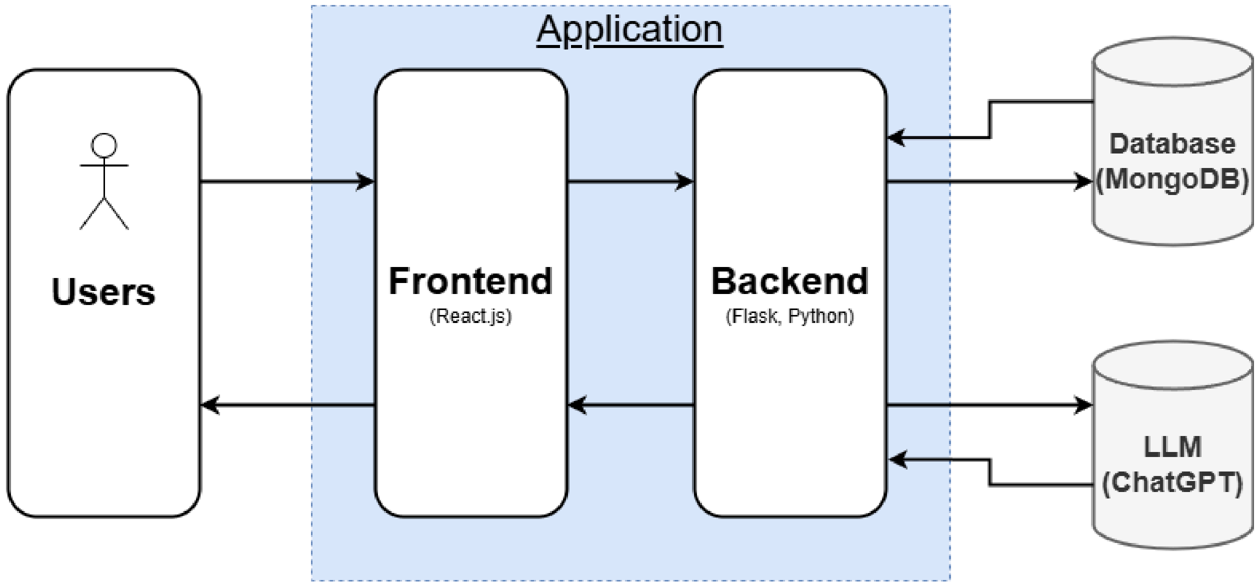
B.R.U.H Design Document

Group 9, SENG 401 W25
Adam Yuen, Rodolfo Gil-Pereira, Zaid Shaikh, Fahmi Sardar, Alexander Lai, Odin Fox

Architecture Overview

The architecture design used for this application is layered architecture. The frontend and backend portions of the application are each contained in their own directories in order to maintain high cohesion. Below is a high-level representation of the application and its interactions between each layer.



Components

Component	Language/Framework
Frontend	React.js
Backend	Flask
Database	MongoDB
Hosting Infrastructure	AWS

Layer Design

Frontend:

- Responsibilities:
    - Accept user input
    - Send data to backend
    - Display generated data
    - Handle user experience
- Key Classes/Interfaces:
    - Homepage.js
    - App.js
    - CoverLetter.js
    - Resume.js
    - Loginpage.js
- Interactions:
    - User inputs data for the frontend to send to the backend via an API POST request to the LLM or Database.
    - The frontend displays the UI in order to help the user navigate the site.

Backend:

- Responsibilities:
    - Receives data from frontend
    - Hashes user passwords and emails for security
    - Creates prompts for the LLM to use
    - Queries database to update or validate information
- Key Classes/Interfaces:
    - JobAppMaterial
        - Resume
        - Cover Letter
    - Flask backend (app.py)
    - gptPromptingUtilities.py
- Interactions:
    - Connects to database or LLM to transfer and validate information
    - Hashes emails and password for security

Design Patterns

Singleton Design Pattern

- The flask app and database connection are based on the singleton design pattern as only one instance of each is able to exist in the application. Singleton design pattern works best for these parts of the application as we only need one instance of each for the application to function properly. It also ensures that a second connection or a second instance of the flask app is not created so that no errors will be caused.

Observer Design Pattern

- The frontend contains a useEffect and useState functions that automatically refresh parts of the page when the variables they are connected to change. These act like observers when we use them to update a value like a text box, then it tells the page to refresh that component to display the edited text on the browser. This removes the need to consistently poll a variable for its value, which would reduce the efficiency of the webpage and possibly even make it crash.

SOLID Principles

Single Responsibility Principle

- The Resume and Cover Letter classes each have the single responsibility of accepting information for either a Resume or Cover Letter and not both.

Open/Close Principle

- The Resume and Cover Letter classes also implement OCP. This helps with upgradability if we want to improve the applications ability to add more documents, like a CV, or potentially a transcript.

Liskov Substitution Principle

- The CoverLetter and Resume classes have the capability to replace their base/super class JobAppMaterial as they don’t have major deviations. Since the classes inherit properly from the JobAppMaterial, and the parent method are overridden while also using the super() call the base functionality remains intact ensuring that LSP is adhered to.

Interface Segregation Principle

- Although at initial inspection the JobAppMaterial class (which functionally acts as an interface; a job app material class is never instantiated) may appear to violate ISP, the way our code works requires all of the fields to be functional. As a result, since all of the required fields are in the JobAppMaterial class, no unnecessary details exist within an interface.

Dependency Inversion Principle

- In the frontend, we used an overarching Context Class to save the values of email and password of a registered user. This allowed the other pages to access the values in the context and not depend on the other pages to send this data to each other. As a result, any page can be loaded in any order and they will still have access to the user data.