Course Project – System Architecture (Due on Oct 4, 11:59pm)

Bitbucket Link: <https://bitbucket.org/homework-acs560-vedanshu/jobtracker/src/main/>

Project Members: Rahul Trivedi, Vedanshu Joshi, Surbhi Sharma

(1) Identify the system architecture(s) that will be in use. Provide written explanation.

regarding your choice.

* Layered Architecture:
* The Layered Architecture is well-suited for the browser extension project due to its ability to bring a structured and organized approach to the development of the extension. This architectural style allows the project to be broken down into distinct layers, such as the presentation layer (UI), business logic layer, and data access layer. This separation of concerns ensures that each component has a clear and well-defined role, leading to modularity and ease of development.
* In the context of the browser extension, the Layered Architecture helps maintain a clean and maintainable codebase. The presentation layer handles the user interface elements, the business logic layer manages application functionality, and the data access layer takes care of storing and retrieving job application data. This separation simplifies testing, as each layer can be tested independently, ensuring a robust and reliable extension.
* Moreover, Layered Architecture facilitates scalability by allowing specific layers to be scaled independently as the project evolves. For example, if there is a need to enhance the user interface or improve data handling capabilities, these changes can be made within the relevant layer without affecting the entire system.
* MVC (Model-View-Controller) Architecture:
* In addition to Layered Architecture, MVC Architecture complements the project by providing a clear separation of concerns between the Model, View, and Controller components. The Model handles data management and business logic, the View manages the presentation and user interface, and the Controller handles user input and interactions.
* The MVC Architecture brings a high level of modularity and reusability to the browser extension project. Components within each category (Model, View, and Controller) can often be reused across different parts of the extension. For instance, a View designed to display job listings can be reused in multiple UI screens, promoting efficiency and consistency in the user interface.
* Testability is another advantage of MVC. Each component can be tested independently, ensuring that they perform their designated functions as expected. Additionally, MVC aligns well with scalability needs, as it allows for the expansion of specific components without affecting others. This adaptability is particularly valuable when the extension needs to evolve to meet changing requirements or accommodate new features.
* The combined use of Layered Architecture and MVC Architecture provides a solid foundation for your browser extension project. These architectural choices offer a structured and modular approach to development, enhance maintainability, promote reusability, and facilitate scalability. By adopting these architectures, you can ensure that your browser extension remains well-organized, efficient, and responsive to user needs.

(2) Provide at least 2 different views of the system architecture. Provide written.

explanation regarding your choice.

* Layered Architecture Components:
* **User Interface Layer**: The User Interface Layer is the part of the application that the user interacts with. In the context of LinkedIn, this includes the user interface of the LinkedIn website or application. It contains various views like the Job Listing View and browser extension involves User Profile View and Job Dashboard View User interactions, such as clicking on job listings, viewing profiles using the browser extension, occur within this layer.
* **Server/API**: The Server/API acts as a bridge between the user interface and the application logic. It handles requests from the user interface and processes them. It encompasses the Application Logic Layer and Business Logic Layer.
* **Application Logic Layer**: The Application Logic Layer is responsible for managing the core functionalities of the application. It includes Job Application Management, Data Handling, and Integration with LinkedIn and External APIs. Job Application Management involves handling the entire job application process, from tracking applications to updating their statuses. Data Handling focuses on processing and manipulating data within the application. Integration with LinkedIn and External APIs involves communication with external platforms to fetch and update data in real-time.
* **Business Logic Layer**: The Business Logic Layer is a sub-layer of the Application Logic Layer. It handles critical business-related functionalities. Services within this layer manage Application Status Updates, User Authentication and Authorization, and act as a repository for essential business logic.
* **Database layer**: The Database layer is responsible for storing and managing the application's data. In this case, it includes a Job Application Database and User Profile Storage. The Job Application Database stores information related to job applications, such as job titles, application statuses, and associated details. The User Profile Storage holds user-specific data, allowing for personalized experiences within the application. This architecture allows for a clear separation of concerns, with each layer handling specific tasks.
* The User Interface Layer focuses on providing a user-friendly experience, the Application Logic Layer manages the core functionalities, and the Database ensures data persistence and retrieval. The Server/API acts as the intermediary, processing requests and directing them to the appropriate layers. Additionally, the Business Logic Layer handles critical business operations, ensuring the application's functionality aligns with business requirements.
* Diagram:

A diagram of a company

Description automatically generated

* MVC Architecture Components:
* **Model (Backend in C#):**

1. **Job Data Storage**: This component will handle the storage of job-related data. It interacts with the database to store and retrieve information about the jobs users have applied to. It includes tables for job details, application status, user data, etc.
2. **User Authentication**: This component manages user authentication and authorization. It ensures that only authenticated users can access and modify their job application data.
3. **API**: The Model exposes APIs that the frontend (View and Controller) can use to interact with the database and perform operations like adding job applications, updating statuses, and retrieving job data.

* **View (React.js Frontend):**

1. **Browser Extension UI**: The View component represents the user interface of your browser extension. It includes UI elements for tracking jobs, displaying job details, and triggering reminders.
2. **Static Website**: The static website with getting started steps and user access links is also part of the View. It provides a user-friendly interface for users to interact with the extension.
3. **HTML Job Tile Element**: The View component will rewrite the HTML job tile element on job listing websites like LinkedIn, Handshake, and Indeed to include an application status tag if the job has already been applied for. This involves modifying the DOM of the web page.

* **Controller (Middleware):**

1. **Browser Extension Logic:** The Controller component handles user interactions within the browser extension UI. It captures user actions like clicking on jobs, applying for them, and updating application statuses. It communicates with the Model to update job data.
2. **Reminder Notification Logic:** The Controller is responsible for setting up reminder notifications based on user preferences. It triggers notifications to remind users of important application deadlines or follow-ups.
3. **Routing Logic:** For the static website, the Controller manages routing and navigation. It determines which content to display based on user actions (e.g., clicking on links in the navbar).

* **MVC Interaction:**
* The View displays job data fetched from the Model and provided a user interface for interacting with the Controller.
* The Controller handles user actions and communicates with the Model to update and retrieve job data.
* The Model stores and manages job data, handles user authentication, and provides APIs for the Controller to access the data.
* In summary, in the Model-View-Controller (MVC) architecture:
* **Controller to Model Interaction:**
* The Controller communicates directly with the Model.
* It retrieves and updates data from the Model, including tasks like fetching job information, managing user data, and handling authentication.
* **Controller to View Interaction:**
* The Controller instructs the View on how to present data in the user interface.
* It passes data obtained from the Model to the View for rendering.
* The Controller captures user actions within the View, updating the interface as needed.
* **View to Controller Interaction:**
* The View can send user input or actions back to the Controller for processing.
* For instance, the View can capture user input (e.g., setting reminders) and pass it to the Controller, which determines how it affects the Model and updates the View accordingly.
* Overall, the Controller serves as an intermediary, managing user interactions, data flow between the Model and View, and ensuring a seamless user experience by coordinating data retrieval, updates, and presentation in the application.
* Diagram:

A diagram of a computer program

Description automatically generated