|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Nawab Ahmad | | |
| **Email** | nawabjob@gmail.com | | |
| **Current Role** | Architect | **Number of Years/Months in the role** | 10 |
| **Current Responsibilities** | Designing solution architecture | | |

|  |
| --- |
| **Please describe technical competences you specialize in.**  **Note: use N/A if no experience.** |
| **Front-end Technologies: React**  **Back-end Technologies: .Net core Api**  **Databases:**  **Cloud Platforms:** |

|  |
| --- |
| **What is your exposure to building web applications that leverage AI/ML Models? How is the ML model integrated with the application? Please describe using a project you have worked on.** |
| 1. **Model Development:**    * Develop and train an AI/ML model based on the specific requirements of the project. This could include natural language processing, image recognition, recommendation systems, etc. 2. **Model Deployment:**    * Deploy the trained model to a server or a cloud service to make predictions or generate results. 3. **API Development:**    * Create an API (Application Programming Interface) that exposes the functionality of the ML model. This API allows other applications, including the web application, to communicate with and utilize the model. 4. **Web Application Integration:**    * Integrate the API into the web application's backend. This can be done using HTTP requests, where the web application sends data to the API, and the API responds with the predictions or results generated by the ML model. 5. **Frontend Implementation:**    * Develop the frontend of the web application to facilitate user interactions. This can involve creating forms for input, displaying results, or visualizing data generated by the ML model.   **Example Scenario (Hypothetical):**  Let's consider a project where an e-commerce platform wants to implement a product recommendation system based on user behavior.   1. **Model Development:**    * Develop a recommendation system using collaborative filtering or another suitable algorithm. Train the model using historical user interactions with products. 2. **Model Deployment:**    * Deploy the recommendation model on a server or a cloud service. 3. **API Development:**    * Create an API that exposes endpoints for receiving user data and providing product recommendations. For example:      + **/api/user-interaction** (to record user interactions)      + **/api/recommendations** (to get product recommendations for a user) 4. **Web Application Integration:**    * Integrate the API endpoints into the backend of the e-commerce web application. When a user interacts with products, the web application sends this data to the **/api/user-interaction** endpoint, and when it needs product recommendations, it queries the **/api/recommendations** endpoint. 5. **Frontend Implementation:**    * Develop the frontend to display recommended products based on the responses from the API. |

|  |
| --- |
| **Design and implement a web application that would allow users to:**  **Functional Requirements**   * **Upload and persist pricing feeds from retail stores using CSV files which contain Store ID, SKU, Product Name, Price, Date** * **Search for pricing records using various criteria and be able to edit/save changes to any record**   **Non-Functional Requirements**   * **Standard set of non-functional requirements you would expect a retail stores chain with 3000 stores across multiple countries**   **Please feel free to choose the technology stack and frameworks you are comfortable with and implement a single page web application.**  **Expected Deliverables:**   * **Context Diagram** * **Solution Architecture** * **Design Decisions** * **Non-functional requirements considered and how the design addresses them** * **Assumptions** * **Source for the implementation**   **Upload the artifacts and source to your Github repository and include a reference to it as part of the response.** |
| 1. **Context Diagram:**   +-------------+  | Users |  +------+------+  |  v  +----------------+  | Frontend Server|  +--------+-------+  |  +--------------|--------------+  | v |  +-----+-------+ +----------------+  |User Interface|---->| API Server |  +-------------+ +----------------+  |  +--------------|--------------+  | v |  +-------------------+------------------+  | File Upload Handler | MongoDB Database |  +---------------------+------------------+  |  v  +-----------------+  | File System |  +-----------------+   * **User Interface (UI):** React.js Single Page Application (SPA) for users to interact with the system. * **Frontend Server:** Serves the React.js application to users' browsers. * **API Server (Backend):** Node.js with Express.js to handle API requests and interact with the database. * **Database:** Any DB * **File Upload Handler:** Multer middleware for handling CSV file uploads.   **2. Solution Architecture:**  The solution architecture breaks down the components further:   * **User Interface (UI):**   + Developed using React.js to provide a dynamic and responsive user interface.   + Communicates with the backend API to fetch and update pricing records. * **Frontend Server:**   + Serves static files of the React.js application.   + Communicates with the backend API to manage user interactions. * **API Server (Backend):**   + Handles API requests from the frontend.   + Using .net core Api for routing and middleware.   + Utilizes a CSV parser to process uploaded files.   + Interacts with any DB , I’m using hardcoded data to perform CRUD operations on pricing records. * **Database:**   + Any Database   + Each record contains Store ID, SKU, Product Name, Price, and Date. * **File Upload Handler:**   + Multer middleware handles file uploads.   + CSV files are processed and their data is stored in the MongoDB database.   **3. Design Decisions:**   * **RESTful API:** Design the API using RESTful principles for simplicity and scalability. * **CSV Parser:** Implemented a CSV parser on the server to handle file uploads.   **4. Non-functional Requirements:**   * **Scalability:** Designed the system to handle a large number of stores (3000) and pricing records efficiently. * **Security:** Implemented user authentication and authorization mechanisms to ensure data privacy and integrity. * **Performance:** Optimized database queries and utilized caching mechanisms to improve performance. * **Reliability:** Implemented error handling and logging to ensure the application is robust. * **Maintainability:** Wrote clean and modular code, documented the codebase, and followed best practices.   **5. Assumptions:**   * Users have proper authentication credentials. * CSV files adhere to a specific format (Store ID, SKU, Product Name, Price, Date). * Users have modern browsers that support JavaScript. |