|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Sarath Narayana | | |
| **Email** | Sharath.e125@gmail.com | | |
| **Current Role** | Manager | **Number of Years/Months in the role** | 1 year 6 months |
| **Current Responsibilities** | * Responsible for the end-to-end project execution which includes initiation, requirements, development, quality, scrum planning and delivery * Handling multiple projects as Product Manager & Innovation Lead - leading an entire team of Technical Architect, Solution Architect and client management . * My ability to balance hands-on development with team management underscores my versatility and commitment to delivering high-quality results in a collaborative environment * To Work with Business Analysts and users to understand the requirements and contribute to the requirements gathering, analysis, design, development, change management, release and post-production verification of the project. * I handle all the security validations that needs to be followed by InfoSec review * Also, to Integrate tableau and D3 Charts for visualization of the Dashboards | | |

|  |
| --- |
| **Please describe technical competences you specialize in.**  **Note: use N/A if no experience.** |
| **Front-end Technologies:** ASP.NET Web Forms, ASP.NET MVC, jQuery, AngularJS, Angular10  **Back-end Technologies:** Windows forms C#,Cefsharp ,Electron atom, window services, web API, Node JS  **Databases:** Microsoft SQL Server  **Cloud Platforms:** Azure (basics) |

|  |
| --- |
| **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.** |
| During my 9 years of experience in various organizations, I have worked on multiple projects and products involving the integration of web applications with AI/ML models  **API Integration approach**: In some projects, we enabled web applications to expose data via APIs to external applications. These applications consumed the data in JSON format and calibrated their models accordingly. Additionally, in other scenarios, data inputted into the system was sent to external services for AI-based OCR extraction (such as Azure Form Recognizer). The resulting data was then returned to the source application through JSON responses, where it was further processed and formatted as required for the destination application |
| **Context Diagram** |
|  |
| **Solution Architecture** |
|  |
| **Solution Approach :** |
| * **Uploading Data from an Excel Sheet**: It all starts when someone uploads information from an Excel file into the system. * **Using an API for Data Processing**: Once the data is uploaded, it's sent to an API. An API is like a middleman that helps different software programs to talk to each other and process the information. * **Hangfire Takes Over for Background Tasks:** After the API gets the data, Hangfire steps in. Hangfire is a tool that does certain jobs in the background, which means it can handle tasks that take a long time without making you wait. * **Storing Data in Microsoft SQL Azure**: Next, the data that has been processed gets stored in Microsoft SQL Azure. This is a type of database that's based in the cloud, meaning it can store lots of data and you can access it from anywhere. * **Searching for Data via a Web Interface:** When someone needs to find this data, they use a search query. This is like asking the system to find specific information stored in the database. * **Interacting through the Web User Interface (Web UI):** Finally, there's the Web UI. This is the part of the system that people interact with. They can use it to search for and view the data stored in the SQL Azure database, depending on what they need.   In short, this process involves uploading data, processing it through various tools, storing it in a cloud database, and accessing it through a user-friendly web interface. |
| **Design Decisions** |
| The SKU Price Feed Processing System is designed to manage and process large volumes of pricing data from multiple retail locations. The system must accommodate uploads from approximately 3000 stores, provide for parallel processing of this data, store it securely, and allow for real-time querying through a user interface.  **Component**: Excel Upload   * **Design Decision**: Utilize Excel spreadsheets for data upload. * **Justification**: Excel is a widely-adopted tool among retail partners, allowing for ease of use and minimizing the need for additional training. * **Implications**: Ensures a smooth transition and adoption by store personnel, limiting the potential for data entry errors.   **Component**: API   * **Design Decision**: Implement a RESTful API for file reception and initiation of processing tasks. * **Justification**: A RESTful API provides a scalable and flexible means to handle data ingestion and can be easily integrated with various front-end technologies. * **Implications**: Simplifies future integration with other systems and can scale to meet increased demand.   **Component**: Hangfire   * **Design Decision**: Use Hangfire for managing background jobs, specifically for processing uploads. * **Justification**: Hangfire supports complex job processing scenarios, including retries and failover strategies, which are essential for handling large-scale data processing reliably. * **Implications**: Enhances the robustness of data processing operations and allows for comprehensive monitoring and logging of background tasks.   **Component**: Microsoft SQL Azure   * **Design Decision**: Store and manage processed data in Microsoft SQL Azure. * **Justification**: Azure provides high availability, disaster recovery, and automated backups, ensuring data integrity and security. * **Implications**: Adds to the operational costs but significantly reduces the risk of data loss and improves data accessibility.   **Component**: Web UI   * **Design** Decision: Develop a Web UI for end-user interactions. * **Justification**: A Web UI is accessible from any device with internet access, providing flexibility and convenience to users. * **Implications**: Requires careful design to ensure usability and responsiveness, particularly on mobile devices.   **Throttling Strategy:**   * **Design Decision**: Implement an API throttling mechanism. * **Justification**: Throttling is necessary to manage the load on the system, preventing service degradation during peak usage. * **Details**: Define specific rate limits, such as requests per minute, and employ a queuing mechanism to handle excess requests. |
| **Non Functional Requirements** |
| * **Concurrency Handling**: The system must accommodate a high throughput, with the capability to manage 40-50 requests per microsecond during peak periods, based on anticipated demand from 3000 stores, with each store potentially generating 1000 requests per minute. * **Response Time**: The architecture should ensure a swift data retrieval process, with the objective of furnishing each request with a response time of under one second. * **Secure Data Insertion**: Insertion of data into the SFTP server must be fortified by strict authorization protocols to prevent unauthorized data uploads. * **API Security**: Access to the API should be stringently controlled, prohibiting all unauthorized attempts to retrieve or manipulate data. * **Data Privacy Compliance**: The system must be designed and operated in compliance with local data protection and privacy regulations, ensuring that all data handling procedures meet the requisite legal standards. |
| **Assumptions** |
| Critical assumptions are the following but are not limited to :   * Assuming that the system will be accessed by three types of categories of users – Admins, Retail Store Operators and End users * Assuming the deployment of the application will be done on a cloud based environmen * Assuming that the technology stack considered is Microsoft .NET, WebAPI, SQL Server, and Angular. |