**System-level Design**

**Introduction**

The TA Management Suite is structured to provide real-time responsiveness and efficient processing. This design considers the diverse functionalities and interactions needed for managing Teaching Assistants at North University.

**Modules**

**TA Application Module**

Responsibilities

* Manage incoming TA applications.
* Securely store CVs and other application data.

Triggers and Actions

* When a new application is submitted, trigger the initial evaluation process and notify department staff.

**Course Management Module**

Responsibilities

* Manage course listings requiring TAs.

Triggers and Actions

* When a new course is added or updated, notify the TA Matching Module to re-run its algorithms.

**TA Matching Module**

Responsibilities

* Automated matching of TA applicants with appropriate courses.

Triggers and Actions

* Triggered by new TA applications or course listings to run matching algorithms.

**Decision-Making Module**

Responsibilities

* Support the TA Committee in final decision-making for TA assignments.

Triggers and Actions

* Once the TA Matching Module finishes its task, notify the committee members to review and finalize decisions.

**TA Evaluation Module**

Responsibilities

* Collect evaluation data from instructors.

Triggers and Actions

* At the end of a term, trigger the evaluation submission process for instructors.

**Notification Module**

Responsibilities

* Send real-time notifications to various stakeholders.

Triggers and Actions

* Trigger notifications based on events such as new TA applications, committee decisions, and evaluation deadlines.

**Data Storage and Management**

* Structured data like application statuses and course details will be managed using Azure SQL Database.
* Unstructured data such as CVs will be stored in Azure Blob Storage.

**User Authentication and Access Control**

* Azure Active Directory will manage user authentication.
* Access will be role-based, with distinct roles for TA applicants, department staff, committee members, and instructors.

**Automation and Real-Time Processing**

* The system will utilize Azure Logic Apps or Azure Functions to handle automated tasks, such as generating preliminary TA assignment suggestions.
* Azure Notification Hubs will manage real-time notifications.

**Security Measures**

* All sensitive information will be encrypted and securely stored, adhering to data protection laws and regulations.

**Conclusion**

By integrating real-time responsiveness and process automation, this design ensures a streamlined, efficient, and user-friendly experience. The use of Azure services enhances the system’s reliability, scalability, and security.

**Detailed Design: TA Application Management Subsystem**

**Overview:**

The TA Application Management Subsystem is designed to facilitate the process of TA application management. It allows TA applicants to submit, view, and update applications, as well as upload associated CVs.

**Components:**

1. Frontend forms for application: UI components to capture user input.
2. Python backend for application processing: Handles the business logic.
3. Azure SQL Database: Stores structured application data.
4. Azure Blob Storage: Stores CVs and other associated documents.

**Data Model:**

TA Application Entity in Azure SQL Database

* Application ID (Primary Key)
* User ID (Foreign Key: Azure AD User ID)
* Status (Pending/Approved/Rejected)
* Courses applied for
* Previous TA Experience (if any)
* Timestamps (created\_at, updated\_at)
* CV URL (Pointer to Azure Blob Storage)

**Functionalities and Interactions:**

TA Application Submission

Frontend

* Displays a form capturing relevant details such as courses applied for, previous TA experience, and CV upload.

Python Backend

* Validates the input data.
* Stores the application details in Azure SQL Database.
* Uploads the CV to Azure Blob Storage.

Viewing Existing Applications

Frontend

* Allows the user to see a list of their existing applications.

Python Backend

* Queries the Azure SQL Database to retrieve the list of applications associated with the authenticated user.

Updating Existing Applications

Frontend

* Provides an option to edit existing applications.

Python Backend

* Updates the existing application in the Azure SQL Database.
* If there’s a new CV, updates it in Azure Blob Storage.

CV Management

Frontend

* Provides an option to upload or replace the CV during the application or updating process.

Python Backend

* Uploads new CV to Azure Blob Storage.
* Updates the CV URL in the Azure SQL Database.

**API Endpoints:**

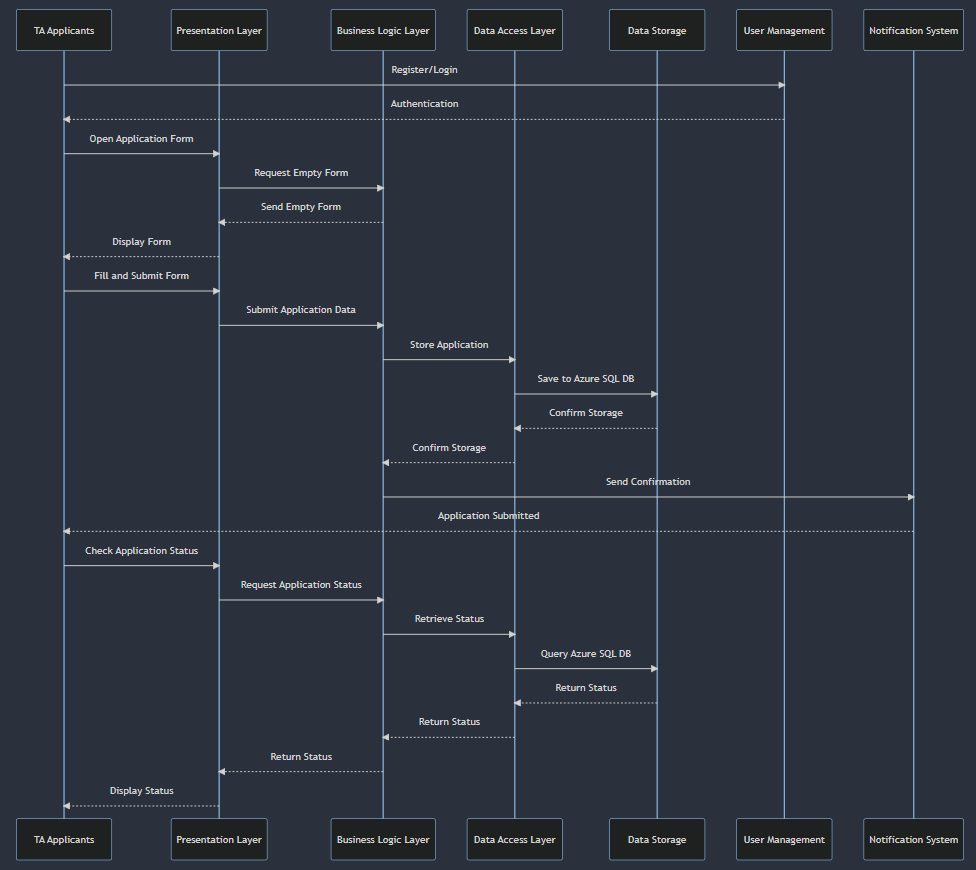
1. /applications/create - POST: Allows TA applicants to submit new applications.
2. /applications - GET: Retrieves the list of applications for the authenticated user.
3. /applications/<application\_id> - GET: Retrieves details of a specific application.
4. /applications/<application\_id>/update - PUT: Updates a specific application.
5. /applications/<application\_id>/delete - DELETE: Deletes a specific application.

**Security Measures:**

1. Data Validation: Validates data before it is saved to the database.
2. Authentication: Requires the user to be authenticated before accessing application functionalities.
3. Authorization: Checks if the user is authorized to view, create, or update the application.
4. Encryption: All data, including CVs, is encrypted in transit and at rest.

UML Diagrams:

>Sequence Diagram



In this sequence diagram:

* The actors and components involved are identified as TA Applicants, Presentation Layer, Business Logic Layer, Data Access Layer, Data Storage, User Management, and Notification System.
* The steps or methods in the sequence include registering and logging in (Register/Login), opening and filling out the TA application form (Open Application Form, Fill and Submit Form), storing the application in the Azure SQL Database (Store Application), and finally, checking the application's status (Check Application Status).

>Use Case

