Blood Bank Management System - Project Documentation

1. Overview

The Blood Bank Management System is a Spring Boot-based application designed to streamline donor management, hospital records, blood requests, and inventory tracking. It uses a microservices architecture with API Gateway handling authentication and request routing. The system ensures secure operations, efficient inter-service communication, and detailed reporting through email notifications.

2. Key Features

- Donor registration and management.
- Hospital records management.
- Blood requests from hospitals.
- Inventory management for blood units.
- Email notifications for blood requests and donor registrations.
- Secure authentication with JWT-based role management.
- generate reports for admins.
- CSV-based user upload.

3. Eureka Server

- Service registry for microservices.
- Ensures service discovery and load balancing.
- Facilitates inter-service communication.

Eureka Server Endpoints

Method	Endpoint	Description
GET	/eureka/apps	Retrieve registered services
GET	/eureka/apps/{serviceId}	Retrieve a specific service by ID
GET	/eureka/status	Check the health of Eureka Server

4. Microservices & Functionalities

4.1 API Gateway

1. Framework: Spring Cloud Gateway
2.Security: JWT authentication filtering, role-based request modifications
3. Authentication: Spring Security with JWT, role-based access control

API Gateway Endpoints:

Method	Endpoint	Description
POST	http://localhost:8000/auth/login	User login and JWT token generation
POST	http://localhost:8000/auth/register	New user registration
GET	http://localhost:8000/auth/validate	Validate JWT token
POST	http://localhost:8000/user/donor/addDonor	Add a new donor
GET	http://localhost:8000/user/donor/getDonor	Retrieve all donors
GET	http://localhost:8000/user/donor/getDonorbyId/{id}	Get donor by ID
POST	http://localhost:8000/user/donor/update	Update an existing donor
POST	http://localhost:8000/user/donor/getByBloodGrouName/{Bloodgroup}	Get donors by blood group
POST	http://localhost:8000/user/donor/getByBloodGrouNameAndAge/{Bloodgroup}/{Age}	Get donors by blood group and age
GET	http://localhost:8000/user/donor/getByName/{name}	Get donor by name
POST	http://localhost:8000/user/hospitol/addHospitol	Add a new hospital
GET	http://localhost:8000/user/hospitol/getHospitol	Retrieve all hospitals
GET	http://localhost:8000/user/hospitol/getHospitolbyId/{id}	Get hospital by ID
POST	http://localhost:8000/user/hospitol/update	Update an existing hospital

POST	http://localhost:8000/user/hospitol/getHospitolbyName/{hospitolname}	Get hospital by name
POST	http://localhost:8000/request/addBloodRequest	Add a blood request
GET	http://localhost:8000/request/getAllHospitolRequest	Get all hospital requests
POST	http://localhost:8000/request/getRequest/{DATE}	Get blood requests by date
POST	http://localhost:8000/request/setStatusrequest/{name}/{status}/{Hospitolname}	Update blood request status
GET	http://localhost:8000/request/getStatusRequest	Get all status requests
POST	http://localhost:8000/request/RequestNotificationForBlood/{bloodGroup}	Send blood request notification via email
POST	http://localhost:8000/inventroy/addDonorInventory	Add donor to inventory
GET	http://localhost:8000/inventroy/getDonor	Retrieve all donors from inventory
GET	http://localhost:8000/inventroy/getBloodCount	Get total blood count
GET	http://localhost:8000/inventroy/getBloodCount/{bloodgroup}	Get blood count by blood group
POST	http://localhost:8000/inventroy/deleteUserFromInventory/{donorname}	Delete donor from inventory
POST	http://localhost:8000/inventroy/updateInventroy	Update donor inventory details
GET	http://localhost:8000/inventroy/getDonorsListGraterThanRequiredAge	Get donors older than specified age
GET	http://localhost:8000/inventroy/getDonorsListLesserThanRequiredAge	Get donors younger than or equal to specified age
POST	http://localhost:8000/auth/login	User login
POST	http://localhost:8000/auth/validatetoken	Validate JWT token
POST	http://localhost:8000/auth/refresh-token	Refresh token
POST	http://localhost:8000/auth/uploadUsers	Upload users from CSV file
POST	http://localhost:8000/api/email/send	Send email notification
GET	http://localhost:8000/eureka/apps	Retrieve registered services
GET	http://localhost:8000/eureka/apps/{serviceId}	Retrieve a specific service by ID

GET	http://localhost:8000/eureka/status	Check the health of Eureka Server

4.2 User Service

The User Service is responsible for managing donor and hospital information. It provides CRUD operations for both entities, allowing the system to maintain an updated record of blood donors and hospitals. This service interacts with the Request, Inventory, and Notification services to handle blood requests, manage inventory, and send notifications.

Technologies Used:

- **Framework:** Spring Boot
- **Database:** MySQL (JPA for data persistence)
- **Inter-Service Communication:** WebClient
- **Logging:** SLF4J with `@Slf4j`
- **Security:** JWT-based authentication and role-based access control
- **Exception Handling:** Global Exception Handling with custom exceptions

Donor Management:

The Donor Management module handles operations related to blood donors, including:

- Adding new donors.
- Retrieving donor information by **ID**, **name**, or **blood group**.
- Updating donor details.
- Filtering donors by **blood group** and **age**.

Donor Service Endpoints:

Method	Endpoint	Description
POST	http://localhost:8000/user/donor/addDonor	Add a new donor
GET	http://localhost:8000/user/donor/getDonor	Retrieve all donors
GET	http://localhost:8000/user/donor/getDonorbyId/{id}	Get donor by ID
POST	http://localhost:8000/user/donor/update	Update an existing donor
POST	http://localhost:8000/user/donor/getByBloodGrouName/{Bloodgroup}	Get donors by blood group

POST	http://localhost:8000/user/donor/getByBloodGrouNameAndAge/{Bloodgroup}/{Age}	Get donors by blood group and age
GET	http://localhost:8000/user/donor/getByName/{name}	Get donor by name

Hospital Management:

The Hospital Management module handles operations related to hospital data, including:

- Adding new hospitals.
- Retrieving hospital information by **ID** or **name**.
- Updating hospital details.

Hospital Service Endpoints:

Method	Endpoint	Description
POST	http://localhost:8000/user/hospitol/addHospitol	Add a new hospital
GET	http://localhost:8000/user/hospitol/getHospitol	Retrieve all hospitals
GET	http://localhost:8000/user/hospitol/getHospitolbyId/{id}	Get hospital by ID
POST	http://localhost:8000/user/hospitol/update	Update an existing hospital
POST	http://localhost:8000/user/hospitol/getHospitolbyName/{hospitolname}	Get hospital by name

Exception Handling:

The User Service includes global exception handling to manage errors gracefully:

- **`DataAlreadyPresent` Exception:** Thrown when attempting to add a donor or hospital that already exists.
- **`IDNotFoundException`:** Thrown when the requested donor or hospital ID is not found.

Security and Authorization

- **JWT-based authentication:**
  - Only `ADMIN` users can **add, update, and delete** donor and hospital records.
  - Both `ADMIN` and `USER` roles can **view** donor and hospital data.
  - **Role-based access control** ensures that unauthorized access attempts are blocked.
  - Requests without valid tokens or with invalid roles are denied with a `403 Forbidden` response.

4.3 Inventory Service

The Inventory Service is responsible for managing the blood stock by tracking:

- **Donor blood units** added to the inventory.
- **Blood count** grouped by blood type.
- Deletion and updates of **donor records** in the inventory.
- Fetching blood stock details by **blood group**.
- Filtering donors by **age** range.

Technologies Used

- **Framework:** Spring Boot
- **Database:** MySQL (JPA for data persistence)
- **Inter-Service Communication:** WebClient
- **Logging:** SLF4J with `@Slf4j`
- **Security:** JWT-based authentication with role-based access control
- **Exception Handling:** Global Exception Handling with custom exceptions

Functionalities

1. Add Donor to Inventory

- Retrieves donor details from the **User Service** using **WebClient**.
- Adds the donor with the specified **blood units** and donation date.
- Increases the **blood count** by blood group.
- Saves the donor record in the inventory.

2. Get Inventory Data

- Fetches all donors in the inventory.
- Retrieves the **total blood count** by blood group.
- Retrieves blood count based on **specific blood group**.

3. Delete Donor from Inventory

- Removes the donor from the inventory by **name**.
- Reduces the **blood count** accordingly.
- Logs the action.

4. Filter Donors by Age

- Filters donors by:
    - Age **greater than** a specified value.
    - Age **less than or equal to** a specified value.
- Retrieves the filtered donor list.

Blood Inventory Management

The Inventory Service manages the entire blood inventory by:

- Adding **donors** to the blood inventory.
- Managing blood stock by tracking:
    - **Total blood count** by blood group.
    - **Donor information** with blood units.
- Removing donors from inventory.
- Updating donor details.
- Fetching filtered donor lists by **age**.

Exception Handling

- **`DataAlreadyPresent` Exception:** Thrown when adding a donor that already exists.
- **`IDNotFoundException`:** Thrown when the requested donor or blood group is not found.
- **`ResultNotFoundException`:** Thrown when filtering yields no results.

Inventory Service Endpoints

Method	Endpoint	Description
POST	http://localhost:8000/inventroy/addDonorInventory	Add donor to inventory
GET	http://localhost:8000/inventroy/getDonor	Retrieve all donors from inventory
GET	http://localhost:8000/inventroy/getBloodCount	Get total blood count
GET	http://localhost:8000/inventroy/getBloodCount/{bloodgroup}	Get blood count by blood group
POST	http://localhost:8000/inventroy/deleteUserFromInventory/{donorname}	Delete donor from inventory
POST	http://localhost:8000/inventroy/updateInventroy	Update donor inventory details
GET	http://localhost:8000/inventroy/getDonorsListGraterThanRequiredAge	Get donors older than specified age
GET	http://localhost:8000/inventroy/getDonorsListLesserThanRequiredAge	Get donors younger than or equal to specified age

Security and Authorization

- **JWT-based authentication**:
    - Only `ADMIN` users can **add, delete, and update** blood inventory.
    - Both `ADMIN` and `USER` roles can **view** the inventory data.
- **Role-based access control** ensures that unauthorized access attempts are blocked.

4.4 Request Service Service

The Request Service handles **blood requests** made by **hospitals**. It allows:

- Adding **blood requests** with details like **hospital name**, **blood group**, and **date limit**.
- Retrieving all blood requests.
- Filtering requests by **date**.
- Updating the **status** of blood requests.
- Sending **email notifications** to donors based on blood group availability.
- Managing blood request statuses and tracking fulfilled or pending requests.

Technologies Used

- **Framework:** Spring Boot
- **Database:** MySQL (JPA for data persistence)
- **Inter-Service Communication:** WebClient
- **Logging:** SLF4J with `@Slf4j`
- **Security:** JWT-based authentication with role-based access control
- **Exception Handling:** Global Exception Handling with custom exceptions

Functionalities

1. Add Blood Request

- Accepts **hospital name**, **blood group**, **message**, and **date limit**.
- Validates the **user role** (`ADMIN` or `USER`).
- Adds the request to the database.
- Logs the action.

2. Retrieve Blood Requests

- Fetches all **blood requests** from the database.

- Supports **filtering by date**.
- Ensures only ADMIN users can access this data.

3. Update Blood Request Status

- Allows ADMIN users to **update the status** of a blood request.
- Retrieves the **donor details** from the **User Service** using **WebClient**.
- Updates the **status** (e.g., `Pending`, `Fulfilled`) for the corresponding request.
- Deletes the donor from the **Inventory Service** after fulfilling the request.
- Logs the status update.

4. Send Blood Request Notifications

- Sends **email notifications** to donors based on **blood group availability**.
- Uses **WebClient** to communicate with the **Notification Service**.
- Sends the blood request details to the donors.
- Logs the notification status.

5. Exception Handling

- **`IDNotFoundException`:** Thrown when the requested blood request ID is not found.
- **Global Exception Handling:** Catches and logs errors gracefully.

Request Service Endpoints:

Method	Endpoint	Description
POST	http://localhost:8000/request/addBloodRequest	Add a blood request
GET	http://localhost:8000/request/getAllHospitolRequest	Get all hospital requests
POST	http://localhost:8000/request/getRequest/{DATE}	Get blood requests by date
POST	http://localhost:8000/request/setStatusrequest/{name}/{status}/{Hospitolname}	Update blood request status
GET	http://localhost:8000/request/getStatusRequest	Get all status requests
POST	http://localhost:8000/request/RequestNotificationForBlood/{bloodGroup}	Send blood request notification via email

Inter-Service Communication

The *Request Service* uses WebClient to communicate with the:

- **User Service:** To fetch donor details.
- **Inventory Service:** To remove the donor after fulfilling a blood request.
- **Notification Service:** To send **email notifications** to donors.

Security and Authorization

- **JWT-based authentication:**
  - Only `ADMIN` users can **add, update, and delete** blood requests.
  - Both `ADMIN` and `USER` roles can **view** blood request data.
  - **Role-based access control** ensures that unauthorized access attempts are blocked.
  - Requests without valid tokens or with invalid roles are denied with a `403 Forbidden` response.

5. Security Service

The Security Service (also referred to as the Authentication Service) handles **user** authentication, authorization, and token management. It uses:

- JWT (JSON Web Token) for secure authentication.
- Role-Based Access Control (RBAC) to manage permissions based on user roles (`ADMIN`, `USER`, and `SuperAdmin`).
- CSV-based bulk user upload for easy user management.
- Token validation and refresh mechanisms to ensure secure access.

Technologies Used

- **Framework:** Spring Boot
- **Database:** MySQL (JPA for data persistence)

- **Security:** Spring Security with JWT
- **Inter-Service Communication:** WebClient
- **Logging:** SLF4J with `@Slf4j`
- **Exception Handling:** Global Exception Handling with custom exceptions

Key Functionalities

1. User Authentication

- Handles **user login** and password verification.
- Generates a **JWT token** upon successful login.
- Embeds the **user role** in the JWT token for authorization.
- Verifies user credentials against the database.

2. JWT Token Management

- **Token Generation:**
  - Generates JWT tokens with the user's role embedded.
  - Sets the token expiration time (30 minutes).
- **Token Validation:**
  - Validates JWT tokens on every API request.
  - Extracts and verifies the user's role from the token.
- **Token Refresh:**
  - Generates a new token before the old token expires.
  - Ensures continuous authentication without requiring re-login.

3. Role-Based Access Control (RBAC)

- Uses **RBAC** to grant or restrict access based on the user's role:
  - **`ADMIN`** → Full access: Add, delete, and update data.
  - **`USER`** → Read-only access.
  - **`SuperAdmin`** → Special privileges for uploading CSV files and managing multiple users.

4. CSV-Based Bulk User Upload

- Allows **SuperAdmin** users to upload multiple users from a **CSV file**.
- Automatically assigns roles (`USER`, `ADMIN`, or `SuperAdmin`) based on the CSV data.
- Ensures existing users are not duplicated.

Security Service Endpoints

Method	Endpoint	Description
POST	http://localhost:8000/auth/login	User login
POST	http://localhost:8000/auth/validatetoken	Validate JWT token
POST	http://localhost:8000/auth/refresh-token	Refresh token
POST	http://localhost:8000/auth/uploadUsers	Upload users from CSV file
POST	http://localhost:8000/auth/adduser	Add a new user
POST	http://localhost:8000/auth/userrole	Get user role from the token

Inter-Service Communication

The **Security Service** communicates with the **API Gateway** to:

- Validate tokens.
- Extract and verify user roles.
- Grant or deny access based on the user's role.

6.Conclusion

The Blood Bank Management System is a fully integrated solution for managing donor, hospital, and blood inventory data. By utilizing Spring Boot microservices, it ensures secure and scalable operations while maintaining a seamless user experience through APIs, email notifications, and efficient inter-service communication. Future enhancements could include AI-based donor-matching algorithms or mobile app integration.