MANDEM ASHA PRAVEENA

User Access Management System Requirements Document

To develop the User Access Management (UAM) system outlined in the requirements document.

1. Planning and Design

- **Define Scope:** Clearly identify the features and functionalities to be implemented.
- **User Stories:** Create user stories for regular users and managers to understand their needs and interactions with the system.
- **Wireframes:** Design wireframes or mockups for the user interfaces to visualize the layout and flow.

2. Technical Specifications

- **Choose Technology Stack:** Decide on programming languages, frameworks, and databases (e.g., HTML/CSS for frontend, Node.js/Python for backend, and SQL/NoSQL for the database).
- **Architecture Design:** Plan the system architecture, including how different components will interact (e.g., frontend, backend, database).

3. Development

- **User Registration Module:** Implement user registration with email verification.
- **Access Request Module:** Develop features for users to request access to applications and for managers to review these requests.
- **Notification System:** Set up email notifications for users and managers about request statuses.
- **Audit Trail Implementation:** Ensure all actions related to access requests are logged.

4. Security Measures

- **Data Protection:** Implement encryption for sensitive data and secure user authentication (e.g., password hashing).

- **Authorization:** Define user roles and permissions to restrict access to different parts of the system.

5. Testing

- **Unit Testing:** Test individual components to ensure they function as expected.
- **Integration Testing:** Test how different components work together, especially the interaction between users and managers.
- **User Acceptance Testing (UAT):** Involve potential users to test the system and provide feedback.

6. Deployment

- **Choose Hosting:** Select a hosting environment (cloud-based or on-premises).
- **Deployment Process:** Set up a deployment pipeline for seamless updates and maintenance.

7. Documentation

- **User Documentation:** Create guides for users and managers on how to navigate the system.
- **Technical Documentation:** Document the codebase, API specifications, and architecture for future reference.

8. Maintenance and Updates

- **Monitor Performance:** Keep track of system performance and user feedback for ongoing improvements.
- **Regular Updates:** Plan for regular updates and security patches to keep the system secure and functional.

9. Training

- **Train Users and Managers:** Conduct training sessions to help users and managers familiarize themselves with the system.

Step 1: Set Up the Project

1. **Create a new Spring Boot project** using Spring Initialize with the following dependencies:
   - Spring Web

- Spring Data JPA
- H2 Database or MySQL
- Spring Boot DevTools
- Spring Mail (for email notifications)

Step 2: Project Structure

user-access-management

```
|
├── src
|   ├── main
|   |   ├── java
|   |   |   └── com
|   |   |       └── example
|   |   |           └── useraccess
|   |   |               ├── UserAccessApplication.java
|   |   |               ├── controller
|   |   |               |   └── UserController.java
|   |   |               ├── model
|   |   |               |   └── User.java
|   |   |               ├── repository
|   |   |               |   └── UserRepository.java
|   |   |               └── service
|   |   |                   └── UserService.java
|   |   └── resources
|   |       ├── application.properties
|   |       └── templates
|   └── test
```

Step 3: Code Implementation

1.UserAccessApplication.java

```
package com.example.useraccess;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UserAccessApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserAccessApplication.class, args);
    }
}
```

2. User.java

```
package com.example.useraccess.model;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String email;
    private String password;

    @ElementCollection
    private List<String> accessRequests = new ArrayList<>();

    // Getters and Setters
}
```

3. UserRepository.java

```
package com.example.useraccess.repository;

import com.example.useraccess.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

4. UserService.java

```
package com.example.useraccess.service;

import com.example.useraccess.model.User;
import com.example.useraccess.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;
import java.util.List;


@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;


    @Autowired
    private JavaMailSender emailSender;


    public User registerUser(User user) {
        return userRepository.save(user);
    }


    public void requestAccess(String username, String appName) {
        User user = userRepository.findByUsername(username);
        if (user != null) {
            user.getAccessRequests().add(appName);
            userRepository.save(user);


            // Send notification email
            sendEmailNotification(user.getEmail(), appName);
        }
    }


    private void sendEmailNotification(String email, String appName) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(email);
        message.setSubject("Access Request Submitted");
        message.setText("Your request for access to " + appName + " has been submitted.");
        emailSender.send(message);
    }
}
```

5. UserController.java

```
package com.example.useraccess.controller;

import com.example.useraccess.model.User;
import com.example.useraccess.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class UserController {
    @Autowired
    private UserService userService;

    @PostMapping("/register")
    public String register(@RequestBody User user) {
        userService.registerUser(user);
        return "User registered successfully";
    }

    @PostMapping("/request-access")
    public String requestAccess(@RequestParam String username, @RequestParam String appName) {
        userService.requestAccess(username, appName);
        return "Access request submitted successfully";
    }
}
```

Step 4: Configuration

application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update

# Email configuration
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=your_email@gmail.com
spring.mail.password=your_email_password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Step 5: Running the Application

1. Run the Application:

```
mvn spring-boot:run
```

2. Testing the API:

Register User

```
POST http://localhost:8080/api/register
```

```
Content-Type: application/json
{
  "username": "testuser",
  "email": "testuser@example.com",
  "password": "password123"
}
```

Request Access:

POST http://localhost:8080/api/request-access?username=testuser&appName=SomeApplication

THANK YOU