



# DrishyaScan

## Accessibility Analyzer Project Guide

### Project Overview

The Accessibility Analyzer will scan websites to identify accessibility issues, provide recommendations for fixing them, and track improvement over time. This tool will help developers and content creators ensure their websites are accessible to all users, including those with disabilities.

### Week-Long Development Roadmap

#### Day 1-2: Planning & Design

- Requirements gathering
- System architecture design
- Database schema design
- UI/UX wireframing
- Project setup

#### Day 3-4: Backend Development

- API development
- Database implementation
- Core accessibility scanning logic

#### Day 5-6: Frontend Development

- UI implementation
- Integration with backend

#### Day 7: Testing & Deployment

- Final testing
- Deployment
- Documentation

## Features

### Core Features

- 1. Website Scanning**
  - URL input for scanning
  - Batch URL processing
  - Scheduled automatic rescans
- 2. Accessibility Analysis**
  - WCAG compliance checking
  - Issue categorization (critical, major, minor)
  - Code snippet identification for issues
- 3. Reporting**
  - Detailed issue reports
  - Improvement recommendations
  - Exportable reports (PDF, CSV)
- 4. User Management**
  - User registration/login
  - Project/website management
  - Role-based access control
- 5. Dashboard**
  - Overview of accessibility status
  - Historical tracking
  - Progress visualization

## Color Scheme for DrishyaScan

### Brand Colors

Purpose	Color	Hex Code
Primary Blue	Strong brand color	#2563EB
Accent Purple	Stylish highlight	#7C3AED
Success Green	For success states	#10B981
Warning Yellow	For alerts, warnings	#F59E0B

Error Red	For errors	#EF4444
-----------	------------	---------

### Light Theme Colors

Element	Color Hex
Background	#FFFFFF (Pure White)
Secondary Background	#F9F9FB (Soft Light Grey)
Text (Primary)	#111827 (Dark Charcoal)
Text (Secondary)	#6B7280 (Muted Gray)
Borders	#E5E7EB (Light Border)
Button Primary	#2563EB (Primary Blue)
Button Hover	#1D4ED8 (Darker Blue)
Card Background	#FFFFFF (White)

### Dark Theme Colors

Element	Color Hex
Background	#0F172A (Deep Navy)
Secondary Background	#1E293B (Dark Blue-Grey)
Text (Primary)	#F1F5F9 (Almost White)
Text (Secondary)	#94A3B8 (Soft Gray-Blue)
Borders	#334155 (Subtle Borders)
Button Primary	#3B82F6 (Lighter Blue)
Button Hover	#2563EB (Strong Blue)

## Quick Highlights

- **Fonts:** Use clean sans-serif fonts like Poppins, Inter, or Roboto.
- **Style:** Minimalist, professional, and easy on the eyes.
- **Consistency:** Use the Primary Blue across both themes for branding consistency.
- **Transitions:** Smooth transitions between themes (use 0.3s–0.5s transition for background and text colors).

# Software Requirements Specification (SRS)

## 1. Introduction

### 1.1 Purpose

This document specifies the requirements for the Accessibility Analyzer web application that will help identify accessibility issues on websites according to WCAG guidelines.

### 1.2 Scope

The Accessibility Analyzer will provide tools for scanning websites, identifying accessibility issues, generating reports, and tracking improvements over time.

### 1.3 Definitions and Acronyms

- WCAG: Web Content Accessibility Guidelines
- SPA: Single Page Application
- API: Application Programming Interface

## 2. Overall Description

### 2.1 Product Perspective

The Accessibility Analyzer will be a standalone web application with a React frontend, Java Spring Boot backend, and SQL database.

## 2.2 User Classes and Characteristics

- **Administrators:** Can manage users and system settings
- **Developers:** Can scan websites and view detailed technical reports
- **Content Managers:** Can view summary reports and track improvements
- **Guests:** Can perform basic scans with limited features

## 2.3 Operating Environment

- Web browsers: Chrome, Firefox, Safari, Edge (latest 2 versions)
- Servers: Linux-based cloud environment
- Database: MySQL/PostgreSQL

# 3. Functional Requirements

## 3.1 User Management

- FR1.1: The system shall allow users to register with email and password
- FR1.2: The system shall authenticate users securely
- FR1.3: The system shall allow password reset via email
- FR1.4: The system shall support role-based access control

## 3.2 Website Scanning

- FR2.1: The system shall accept URL inputs for scanning
- FR2.2: The system shall validate URL inputs
- FR2.3: The system shall crawl website pages up to a configurable depth
- FR2.4: The system shall analyze HTML for WCAG compliance
- FR2.5: The system shall categorize issues by severity

## 3.3 Reporting

- FR3.1: The system shall generate detailed reports of accessibility issues
- FR3.2: The system shall provide recommendations for fixing issues
- FR3.3: The system shall allow exporting reports in PDF and CSV formats
- FR3.4: The system shall track improvements over time

## 3.4 Dashboard

- FR4.1: The system shall display summary statistics of scanned websites
- FR4.2: The system shall visualize issue distribution by category
- FR4.3: The system shall show historical trends of accessibility scores

## 4. Non-Functional Requirements

### 4.1 Performance

- NFR1.1: Page load time shall be under 2 seconds
- NFR1.2: Scanning a simple website shall complete within 30 seconds
- NFR1.3: The system shall support up to 100 concurrent users

### 4.2 Security

- NFR2.1: All communications shall be encrypted using HTTPS
- NFR2.2: Passwords shall be stored using bcrypt hashing
- NFR2.3: The system shall implement OWASP security best practices

### 4.3 Usability

- NFR3.1: The UI shall follow material design principles
- NFR3.2: The system shall be accessible per WCAG 2.1 AA guidelines
- NFR3.3: The system shall be responsive for mobile and desktop use

### 4.4 Reliability

- NFR4.1: The system shall have 99.5% uptime
- NFR4.2: Database backups shall occur daily
- NFR4.3: The system shall handle invalid inputs gracefully

## 5. System Interfaces

### 5.1 User Interfaces

- Web-based SPA built with React and TailwindCSS

### 5.2 Hardware Interfaces

- No special hardware requirements beyond standard computing devices

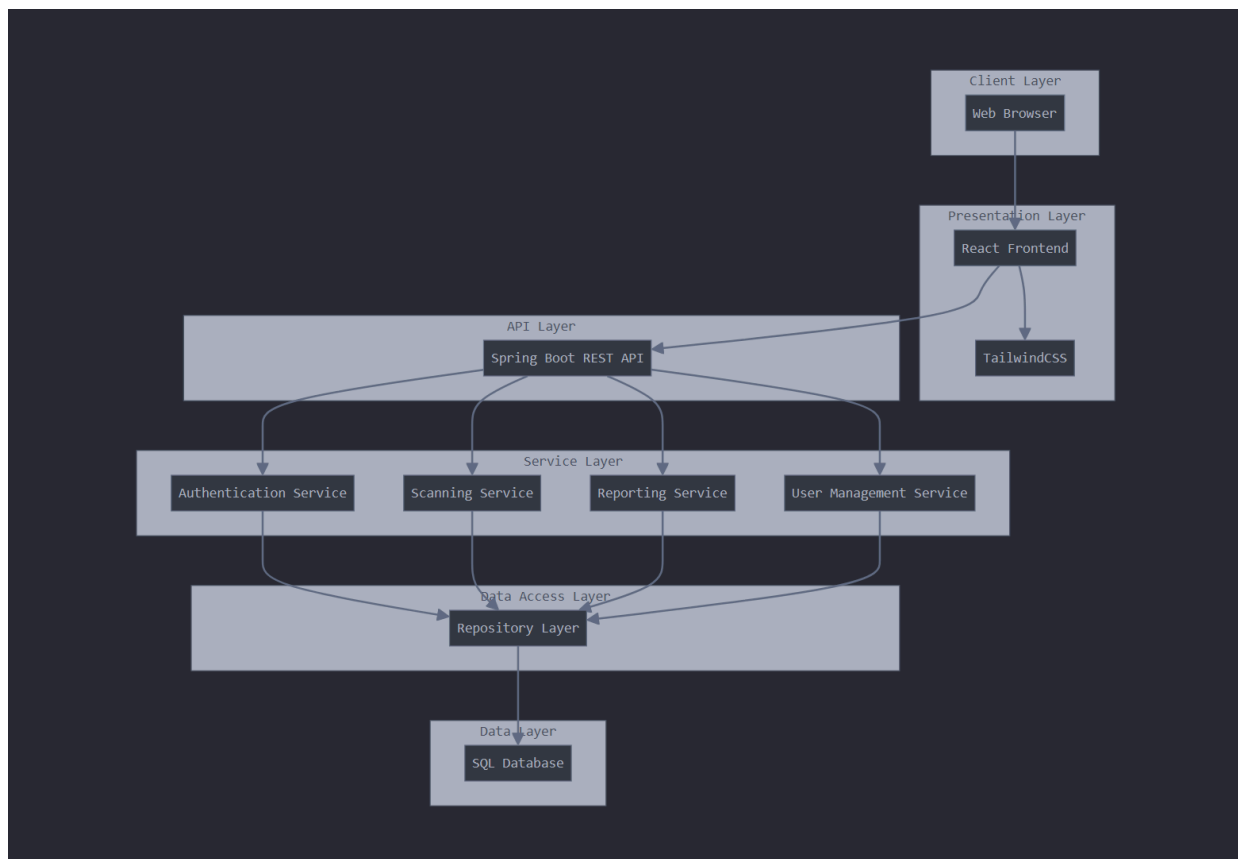
### 5.3 Software Interfaces

- RESTful API for frontend-backend communication
- Database connectivity via JDBC
- Integration with email service for notifications

## 6. Database Requirements

- User data storage
- Website metadata storage
- Scan results storage
- Historical data for trend analysis

# System Architecture



# Entity-Relationship Diagram





# UI Design Suggestions

For the UI, I recommend a clean, accessible design with these key pages:

1. **Dashboard** - Overview of projects, recent scans, and accessibility scores
2. **Scan Page** - URL input, scan options, and immediate results
3. **Issues View** - Detailed list of issues with filtering options
4. **Reports Page** - Generated reports with export options
5. **Project Management** - Organization of websites and scans
6. **User Settings** - Account management and preferences

## Project Planner: Accessibility Analyzer

### Project Timeline (7 Days)

#### Day 1: Planning & Setup

##### Morning

- ☒ ~~Finalize project requirements~~
- ☒ ~~Create project repository on GitHub~~
- ☒ ~~Set up project structure~~
- ☒ ~~Prepare development environment~~

##### Afternoon

- ☐ Design database schema
- ☒ ~~Create wireframes for UI~~
- ☒ ~~Set up CI/CD pipeline~~
- ☐ Daily review and planning for Day 2

## Day 2: Backend Foundation

### Morning

- ☐ [] Initialize Spring Boot project
- ☐ [] Set up database connection
- ☐ [] Implement user entity and repository
- ☐ [] Create authentication service

### Afternoon

- ☐ [] Implement user controller and endpoints
- ☐ [] Set up security configuration
- ☐ [] Create basic website entity and repository
- ☐ [] Daily review and planning for Day 3

## Day 3: Backend Development

### Morning

- ☐ [] Implement website scanning service
- ☐ [] Create scan entity and repository
- ☐ [] Develop issue detection algorithms
- ☐ [] Implement WCAG rule checking

### Afternoon

- ☐ [] Create scan and issue controllers
- ☐ [] Implement reporting service
- ☐ [] Set up scheduled task service
- ☐ [] Daily review and planning for Day 4

## Day 4: Frontend Foundation

### Morning

- ☐ [] Setup React project with TailwindCSS
- ☐ [] Create layout components
- ☐ [] Implement authentication UI

- ☐ [] Set up routing

### **Afternoon**

- ☐ [] Create dashboard UI
- ☐ [] Implement website management UI
- ☐ [] Connect authentication with backend
- ☐ [] Daily review and planning for Day 5

## **Day 5: Frontend Development**

### **Morning**

- ☐ [] Implement scan UI and functionality
- ☐ [] Create issues display components
- ☐ [] Develop filtering and sorting for issues
- ☐ [] Implement issue details view

### **Afternoon**

- ☐ [] Create reporting UI
- ☐ [] Implement data visualization components
- ☐ [] Connect all frontend services to backend
- ☐ [] Daily review and planning for Day 6

## **Day 6: Integration & Testing**

### **Morning**

- ☐ [] Perform full integration testing
- ☐ [] Fix bugs and issues
- ☐ [] Implement responsive design adjustments
- ☐ [] Optimize performance

### **Afternoon**

- ☐ [] Conduct usability testing
- ☐ [] Make UI/UX improvements
- ☐ [] Implement frontend accessibility features
- ☐ [] Daily review and planning for Day 7

## **Day 7: Finalization & Deployment**

**Morning**

- ☐ [ ] Final testing and bug fixes
- ☐ [ ] Prepare deployment scripts
- ☐ [ ] Create user documentation
- ☐ [ ] Complete technical documentation

**Afternoon**

- ☐ [ ] Deploy to production environment
- ☐ [ ] Set up monitoring
- ☐ [ ] Project post-mortem and lessons learned
- ☐ [ ] Plan for future enhancements

**Required Resources**

**Development Tools**

- Git for version control
- IntelliJ IDEA / Eclipse for Java development
- VS Code for frontend development
- MySQL Workbench for database management
- Postman for API testing

**Production Environment**

- Cloud hosting (AWS/GCP/Azure)
- CI/CD platform (Jenkins/GitHub Actions)
- Database server
- Backend application server
- Web server for frontend hosting

**Risk Assessment**

Risk	Probability	Impact	Mitigation
Time constraint	High	High	Prioritize core features, use existing libraries where possible
Performance issues with scanning	Medium	High	Implement efficient algorithms, consider async processing

Security vulnerabilities	Medium	High	Follow OWASP guidelines, conduct security testing
Database scaling	Low	Medium	Design schema for scalability, use indexing
UI/UX complexity	Medium	Medium	Focus on simplicity, get early feedback

# System Design Document: Accessibility Analyzer

## 1. Introduction

This document outlines the system design for the Accessibility Analyzer application, which will help identify and track accessibility issues on websites according to WCAG guidelines.

## 2. High-Level Architecture

The system follows a three-tier architecture:

- Frontend: React with TailwindCSS
- Backend: Java Spring Boot RESTful API
- Database: SQL (MySQL/PostgreSQL)

## 3. Component Design

### 3.1 Frontend Components

#### 3.1.1 Core Components

- **AppRouter:** Handles application routing
- **AuthProvider:** Manages authentication state
- **ThemeProvider:** Provides consistent styling

#### 3.1.2 Feature Components

- **Dashboard:** Homepage with overview statistics
- **ScanForm:** Interface for initiating website scans
- **IssuesList:** Displays accessibility issues with filters

- **IssueDetails:** Shows detailed information about an issue
- **ReportGenerator:** Creates and exports reports
- **ProjectManager:** Organizes websites and scans
- **UserSettings:** Manages user preferences

## 3.2 Backend Components

### 3.2.1 Controllers

- **AuthController:** Handles authentication requests
- **UserController:** Manages user operations
- **ProjectController:** Handles project CRUD operations
- **WebsiteController:** Manages website information
- **ScanController:** Initiates and retrieves scans
- **ReportController:** Generates reports

### 3.2.2 Services

- **AuthService:** Authentication and authorization logic
- **UserService:** User management operations
- **ProjectService:** Project management logic
- **WebsiteService:** Website information management
- **ScanService:** Core scanning functionality
- **AccessibilityCheckService:** WCAG rule implementation
- **ReportService:** Report generation logic
- **NotificationService:** Email and alert handling

### 3.2.3 Repositories

- **UserRepository:** User data access
- **ProjectRepository:** Project data access
- **WebsiteRepository:** Website data access
- **ScanRepository:** Scan data access
- **IssueRepository:** Issue data access

## 3.3 Database Design

Refer to the ER Diagram for entity relationships. The database will use:

- InnoDB engine for transactional integrity
- Appropriate indexing for query performance
- Foreign key constraints for data integrity
- Soft deletion for historical data preservation

## 4. Cross-Cutting Concerns

### 4.1 Security

- JWT-based authentication
- Role-based access control
- HTTPS for all communications
- Input validation and sanitization
- CSRF protection
- Rate limiting for API endpoints

### 4.2 Performance

- Connection pooling for database access
- Caching for frequent queries
- Pagination for large result sets
- Asynchronous processing for time-consuming operations
- Resource optimization for scan operations

### 4.3 Scalability

- Stateless API design for horizontal scaling
- Database connection pooling
- Containerization for deployment flexibility
- Background job processing for intensive tasks

### 4.4 Monitoring

- Request logging
- Error tracking
- Performance metrics collection
- Health check endpoints
- Audit logging for security events

## 5. External Interfaces

### 5.1 API Endpoints

#### Authentication

- POST /api/auth/login
- POST /api/auth/register

- POST /api/auth/refresh-token
- POST /api/auth/forgot-password
- POST /api/auth/reset-password

## **Users**

- GET /api/users
- GET /api/users/{id}
- PUT /api/users/{id}
- DELETE /api/users/{id}

## **Projects**

- GET /api/projects
- POST /api/projects
- GET /api/projects/{id}
- PUT /api/projects/{id}
- DELETE /api/projects/{id}

## **Websites**

- GET /api/websites
- POST /api/websites
- GET /api/websites/{id}
- PUT /api/websites/{id}
- DELETE /api/websites/{id}

## **Scans**

- GET /api/scans
- POST /api/scans
- GET /api/scans/{id}
- GET /api/scans/{id}/issues
- GET /api/scans/{id}/report

## **Issues**

- GET /api/issues
- GET /api/issues/{id}
- PUT /api/issues/{id}/status

## **Reports**

- GET /api/reports
- GET /api/reports/{id}



- GET /api/reports/{id}/export

## 5.2 Third-Party Integrations

- Email service for notifications
- Export functionality for PDF/CSV generation
- Authentication providers (optional)

# Additional Documentation Recommendations

1. **Test Plan Document** - Outline testing strategies including unit, integration, and end-to-end testing
2. **API Documentation** - Detailed documentation of all API endpoints using Swagger/OpenAPI
3. **User Manual** - Guide for end-users on how to use the application
4. **Deployment Guide** - Instructions for deploying the application to production
5. **Development Guide** - Instructions for developers on project structure and contributing
6. **Code Style Guide** - Coding standards for consistent development

# Step-by-Step Development Guide

## 1. Project Setup

1. **Set up version control**
  - Create a GitHub repository
  - Initialize with README, .gitignore, and license
2. **Set up backend project**
  - Initialize Spring Boot project with Spring Initializr
  - Add dependencies:
    - Spring Web
    - Spring Data JPA
    - Spring Security
    - MySQL/PostgreSQL Driver
    - Lombok
    - Validation
3. **Set up frontend project**

- Create React project with Create React App
- Install TailwindCSS
- Set up folder structure:
  - components/
  - pages/
  - services/
  - utils/
- 4. **Set up database**
  - Create database schema
  - Configure connection in application.properties
  - Set up migrations (optional with Flyway/Liquibase)

## 2. Backend Development

1. **Implement entity models**
  - User
  - Project
  - Website
  - Scan
  - Issue
2. **Create repositories**
  - Implement JPA repositories for all entities
3. **Develop service layer**
  - Authentication service
  - User service
  - Project service
  - Website/Scan services
  - Accessibility check service
4. **Implement controllers**
  - Create REST endpoints for all operations
  - Implement request validation
  - Set up error handling
5. **Configure security**
  - Set up JWT authentication
  - Configure authorization rules
  - Implement password encryption

## 3. Frontend Development

1. **Set up routing**
  - Configure React Router
  - Create protected routes

2. **Implement authentication UI**
  - Login form
  - Registration form
  - Password reset
3. **Develop core pages**
  - Dashboard
  - Project management
  - Website scanning
  - Issue tracking
  - Reporting
4. **Create reusable components**
  - Navigation
  - Forms
  - Tables
  - Charts
  - Modals
5. **Implement API services**
  - Create service classes for API communication
  - Set up Axios with interceptors
  - Handle authentication tokens

## 4. Integration and Testing

1. **Connect frontend to backend**
  - Configure CORS
  - Set up proxy for development
  - Test API integrations
2. **Implement comprehensive testing**
  - Unit tests for services
  - Integration tests for APIs
  - End-to-end tests for critical flows
3. **Performance optimization**
  - Review and optimize database queries
  - Implement caching where appropriate
  - Optimize frontend bundle size

## 5. Deployment

1. **Prepare for production**
  - Configure environment variables
  - Set up production database
  - Build frontend for production

2. **Deploy application**
  - Deploy backend to server/cloud
  - Deploy frontend to web server/CDN
  - Configure domain and SSL
3. **Set up monitoring**
  - Configure logging
  - Set up health checks
  - Configure alerts

## UI Frameworks Suggestions

For your React frontend with TailwindCSS, consider these options:

1. **Headless UI** - Unstyled, accessible UI components that pair well with Tailwind
2. **daisyUI** - Component library for Tailwind with minimal styling
3. **Tailwind UI** - Official premium component library (paid)
4. **Chakra UI** - Accessible component library with Tailwind-like styling API

# DrishyaScan Development

## Day 1 Morning Tasks

### Prompt:

Ok let's go we start the developing, As per our project planner please guide me for Day Morning tasks each and every task don't miss anything

### Reply:

Let's start with the Day 1 morning tasks according to our project planner. I'll guide you through each step in detail.

## 1. Finalize Project Requirements

Before writing any code, let's finalize the core requirements for DrishyaScan:

Core Requirements:

1. Website Scanning & Analysis
  - URL input functionality
  - WCAG compliance checking (2.1 AA level)
  - Issue detection and categorization
2. User Management
  - Registration and authentication
  - Role-based access (Admin, Developer, Content Manager)
  - User profile management
3. Project Management
  - Create and organize projects
  - Add websites to projects
  - Track accessibility improvement over time
4. Reporting
  - Generate detailed accessibility reports
  - Provide recommendations for fixing issues
  - Export reports in PDF/CSV formats
5. Dashboard
  - Overview statistics
  - Recent scans
  - Progress tracking

## 2. Create Project Repository on GitHub

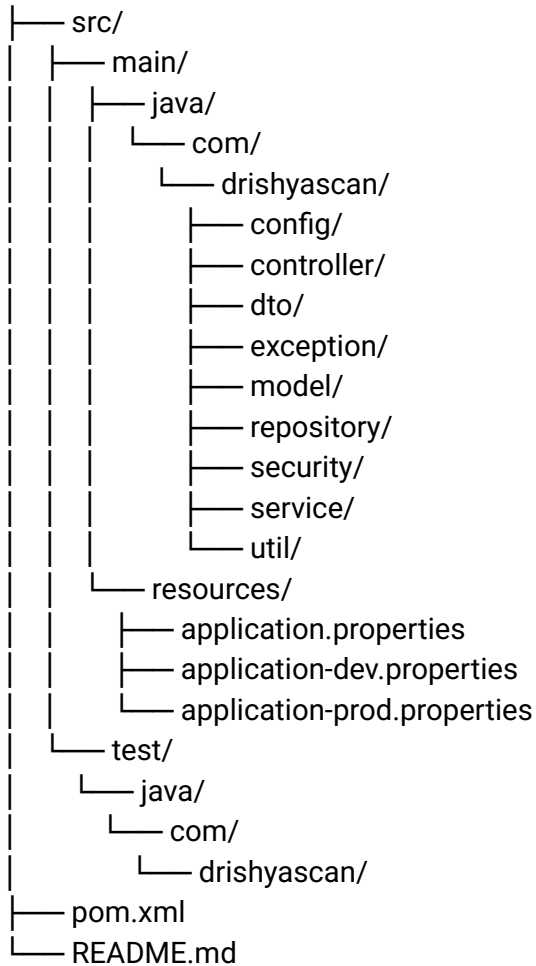
1. Go to GitHub and sign in to your account
2. Click on "New" to create a new repository
3. Name it "DrishyaScan"
4. Add a description: "Web accessibility analysis tool that helps identify and fix accessibility issues"
5. Choose "Public" or "Private" based on your preference
6. Check "Add a README file"
7. Add a .gitignore template for Java
8. Choose a license (MIT recommended for open source)
9. Click "Create repository"

## 3. Set Up Project Structure

Let's define the structure for both frontend and backend:

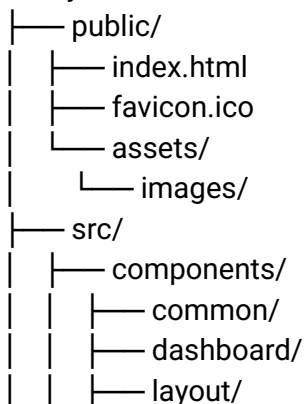
## Backend Structure (Spring Boot)

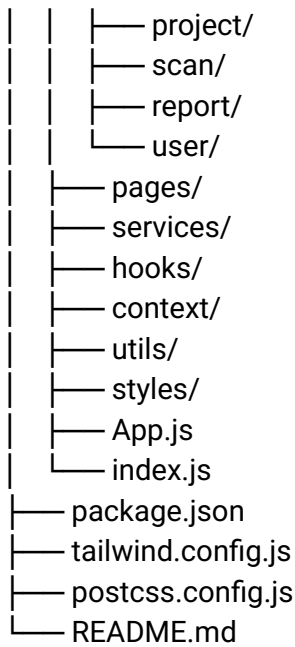
drishyascan-backend/



## Frontend Structure (React + TailwindCSS)

drishyascan-frontend/





## 4. Prepare Development Environment

### Backend Setup:

1. Install Java Development Kit (JDK) 17 or higher
2. Install Maven or Gradle
3. Set up a Spring Boot project using Spring Initializr with these dependencies:
  - Spring Web
  - Spring Data JPA
  - Spring Security
  - MySQL/PostgreSQL Driver
  - Lombok
  - Validation
  - Spring Boot DevTools

### Frontend Setup:

1. Install Node.js (latest LTS version)
2. Install npm or yarn

Set up a new React project:

```
npx create-react-app drishyascan-frontendcd drishyascan-frontend
```

- 3.

Install required dependencies:

```
npm install tailwindcss postcss autoprefixer react-router-dom axios jwt-decode chart.js  
react-chartjs-2 formik yup
```

4.

Initialize Tailwind CSS:

```
npx tailwindcss init -p
```

5.

Configure Tailwind with your color theme:

```
// tailwind.config.js module.exports = { content: ['./src/**/*.js,jsx,ts,tsx'], theme: { extend: {  
  colors: { primary: "#2563EB", "primary-hover": "#1D4ED8", accent: "#7C3AED",  
    success: "#10B981", warning: "#F59E0B", error: "#EF4444", // Light theme  
    "light-bg": "#FFFFFF", "light-bg-secondary": "#F9FAFB", "light-text-primary": "#111827",  
    "light-text-secondary": "#6B7280", "light-border": "#E5E7EB", // Dark theme "dark-bg":  
    "#0F172A", "dark-bg-secondary": "#1E293B", "dark-text-primary": "#F1F5F9",  
    "dark-text-secondary": "#94A3B8", "dark-border": "#334155", }, }, }, plugins: []};
```

6.

## Database Setup:

1. Install MySQL or PostgreSQL
2. Create a new database named "drishyascan"
3. Set up a database user with appropriate permissions

## IDE Setup:

1. Install IntelliJ IDEA or Eclipse for backend development
2. Install Visual Studio Code for frontend development
3. Set up appropriate plugins:
  - Spring Boot plugins for backend IDE
  - ESLint, Prettier, Tailwind CSS IntelliSense for VS Code

## Version Control Integration:

Clone your GitHub repository:

```
git clone https://github.com/your-username/DrishyaScan.git
```

1. add .
2. Set up the initial project structure in the cloned repository



3. Make your first commit:  
`git add .git commit -m "Initial project setup"git push origin main`

These tasks should occupy your morning of Day 1, setting a solid foundation for the project. They focus on planning and environment setup without jumping into coding yet. This careful preparation will help your development process go more smoothly in the following days.

Would you like me to provide more detailed guidance on any of these tasks?

## Day 1 Afternoon Tasks

### Prompt:

As per our project planner please guide me for first Day Afternoon tasks each and every task don't miss anything i don't know about CI CD pipeline please explain in detail. I want setup CI CD through jenkins and please guide to from first step to setup jenkins CI CD

### Reply:

Let's tackle your Day 1 afternoon tasks with detailed guidance, particularly focusing on the CI/CD pipeline setup.

## 1. Design Database Schema

Let's define the complete database schema for DrishyaScan:

### User Table

```
CREATE TABLE users (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  email VARCHAR(255) NOT NULL UNIQUE,  
  password_hash VARCHAR(255) NOT NULL,  
  first_name VARCHAR(100),  
  last_name VARCHAR(100),  
  role ENUM('ADMIN', 'DEVELOPER', 'CONTENT_MANAGER') NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  active BOOLEAN DEFAULT TRUE  
);
```

## Project Table

```
CREATE TABLE projects (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(255) NOT NULL,  
  description TEXT,  
  user_id BIGINT NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  FOREIGN KEY (user_id) REFERENCES users(id)  
);
```

## Website Table

```
CREATE TABLE websites (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  url VARCHAR(2048) NOT NULL,  
  name VARCHAR(255) NOT NULL,  
  project_id BIGINT NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  FOREIGN KEY (project_id) REFERENCES projects(id)  
);
```

## Scan Table

```
CREATE TABLE scans (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  website_id BIGINT NOT NULL,  
  scan_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  accessibility_score FLOAT,  
  status ENUM('PENDING', 'IN_PROGRESS', 'COMPLETED', 'FAILED') NOT NULL,  
  completed_at TIMESTAMP,  
  FOREIGN KEY (website_id) REFERENCES websites(id)  
);
```

## Issue Table

```
CREATE TABLE issues (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
```

```
scan_id BIGINT NOT NULL,  
type VARCHAR(100) NOT NULL,  
severity ENUM('CRITICAL', 'MAJOR', 'MINOR', 'INFO') NOT NULL,  
wcag_criterion VARCHAR(20),  
description TEXT NOT NULL,  
location TEXT NOT NULL,  
html_snippet TEXT,  
recommendation TEXT,  
fixed BOOLEAN DEFAULT FALSE,  
FOREIGN KEY (scan_id) REFERENCES scans(id)  
);
```

## Report Table

```
CREATE TABLE reports (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  scan_id BIGINT NOT NULL,  
  generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  report_type ENUM('DETAILED', 'SUMMARY', 'COMPLIANCE') NOT NULL,  
  format ENUM('HTML', 'PDF', 'CSV') NOT NULL,  
  file_path VARCHAR(255),  
  FOREIGN KEY (scan_id) REFERENCES scans(id)  
);
```

## 2. Create Wireframes for UI

For wireframes, I recommend using a tool like Figma, Adobe XD, or even draw.io. Here are the key screens to wireframe:

### 1. Login/Registration Screen

- Email & password fields
- Register/Login buttons
- Forgot password link

### 2. Dashboard

- Summary statistics cards
- Recent scans section
- Accessibility score trends chart
- Quick actions buttons

### 3. Project Management Screen

- Project list
- Create project button
- Project details view

### 4. Website Management Screen

- Website list within a project
- Add website form
- Website details

### 5. Scan Screen

- URL input
- Scan options
- Scan progress indicator
- Quick results summary

### 6. Issues List Screen

- Filterable issue table
- Severity indicators
- Pagination
- Export options

### 7. Issue Detail Screen

- Problem description
- Code location
- WCAG reference
- Fix recommendation

### 8. Settings Screen

- User profile settings
- Notification preferences
- API key management

## 3. Set up CI/CD Pipeline with Jenkins

Setting up a CI/CD pipeline with Jenkins involves several steps. I'll guide you through the entire process:

### Step 1: Install Jenkins

### **Install Java** (Jenkins requires Java):

```
sudo apt update  
sudo apt install openjdk-17-jdk
```

1.

### **Add Jenkins Repository:**

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -  
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'
```

2.

### **Install Jenkins:**

```
sudo apt update  
sudo apt install jenkins
```

3.

### **Start Jenkins:**

```
sudo systemctl enable jenkins  
sudo systemctl start jenkins
```

4.

5. **Access Jenkins:** Open a browser and navigate to <http://localhost:8080>

6. **Set up Jenkins Initial Configuration:**

Get the initial admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

- 
- Install suggested plugins
- Create admin user
- Configure Jenkins URL

## **Step 2: Install Required Jenkins Plugins**

Go to "Manage Jenkins" > "Manage Plugins" > "Available" and install:

1. **Git Integration Plugin** - For connecting to your Git repository
2. **Maven Integration Plugin** - For building Java projects
3. **NodeJS Plugin** - For building React applications
4. **Docker Pipeline Plugin** - For Docker integration
5. **Pipeline Plugin** - For creating CI/CD pipelines as code
6. **Blue Ocean** - For a modern UI experience
7. **JUnit Plugin** - For test reporting
8. **SonarQube Scanner** - For code quality analysis (optional)

After installation, restart Jenkins:

```
sudo systemctl restart jenkins
```

### Step 3: Configure Global Tools

Go to "Manage Jenkins" > "Global Tool Configuration" and set up:

1. **JDK:**
  - Name: **JDK17**
  - JAVA\_HOME: Path to your JDK installation
2. **Maven:**
  - Name: **Maven3**
  - Install automatically or specify path
3. **NodeJS:**
  - Name: **Node16**
  - Install automatically

### Step 4: Create Jenkins Pipeline

1. Click "New Item" on the Jenkins dashboard
2. Enter name: **DrishyaScan-Pipeline**
3. Select "Pipeline" and click "OK"
4. Configure the pipeline:
  - **Description:** CI/CD pipeline for DrishyaScan project
  - **GitHub project:** Your GitHub repo URL
  - **Pipeline Definition:** Select "Pipeline script from SCM"
  - **SCM:** Git
  - **Repository URL:** Your Git repo URL
  - **Credentials:** Add your GitHub credentials

- **Branch Specifier:** `*/main`
- **Script Path:** `Jenkinsfile`

## Step 5: Create Jenkinsfile

Create a file named `Jenkinsfile` in the root of your repository:

```
pipeline {
  agent any

  tools {
    jdk 'JDK17'
    maven 'Maven3'
    nodejs 'Node16'
  }

  stages {
    stage('Checkout') {
      steps {
        checkout scm
      }
    }

    stage('Backend Build') {
      steps {
        dir('drishyascan-backend') {
          sh 'mvn clean package -DskipTests'
        }
      }
    }

    stage('Backend Test') {
      steps {
        dir('drishyascan-backend') {
          sh 'mvn test'
        }
      }
      post {
        always {
          junit 'drishyascan-backend/target/surefire-reports/*.xml'
        }
      }
    }
  }
}
```

```

}

stage('Frontend Build') {
    steps {
        dir('drishyascan-frontend') {
            sh 'npm install'
            sh 'npm run build'
        }
    }
}

stage('Frontend Test') {
    steps {
        dir('drishyascan-frontend') {
            sh 'npm test -- --watchAll=false'
        }
    }
}

stage('Deploy Development') {
    when {
        branch 'develop'
    }
    steps {
        echo 'Deploying to development environment...'
        // Add deployment steps for dev environment
    }
}

stage('Deploy Production') {
    when {
        branch 'main'
    }
    steps {
        echo 'Deploying to production environment...'
        // Add deployment steps for production environment
    }
}

post {
    success {

```



```

        echo 'Pipeline completed successfully!'
    }
    failure {
        echo 'Pipeline failed!'
    }
}
}
}

```

## Step 6: Set up Webhooks in GitHub

1. Go to your GitHub repository
2. Click on "Settings" > "Webhooks" > "Add webhook"
3. Set Payload URL to: <http://your-jenkins-url/github-webhook/>
4. Content type: [application/json](#)
5. Select "Just the push event" or customize as needed
6. Add webhook

## Step 7: Test the CI/CD Pipeline

1. Make a small change to your repository
2. Commit and push the change
3. Go to Jenkins and verify that the pipeline has been triggered
4. Check the pipeline progression and logs

## Step 8: Configure Deployment

For development and production deployment, you have several options:

1. **Manual Deployment:** Add steps in the Jenkinsfile to copy built artifacts to your server
2. **Docker Deployment:**
  - Create Dockerfiles for backend and frontend
  - Build Docker images in the pipeline
  - Push to Docker registry
  - Deploy using Docker Compose or Kubernetes
3. **Cloud Platform Deployment:** Configure deployment to AWS, Azure, or Google Cloud

For a simple setup, add these steps to the deployment stages:

```

stage('Deploy Development') {

```

```

when {
  branch 'develop'
}
steps {
  sh 'scp -r drishyascan-backend/target/*.jar user@dev-server:/path/to/deployment/'
  sh 'scp -r drishyascan-frontend/build/* user@dev-server:/path/to/nginx/html/'
  sh 'ssh user@dev-server "sudo systemctl restart drishyascan"'
}
}

```

## 4. Daily Review and Planning for Day 2

At the end of Day 1, review what you've accomplished:

### 1. Completed Tasks:

- Project requirements finalized
- GitHub repository created
- Project structure defined
- Development environment set up
- Database schema designed
- UI wireframes created
- CI/CD pipeline with Jenkins configured

### 2. Challenges Faced:

- Document any issues encountered
- Note solutions implemented

### 3. Plan for Day 2:

- Review morning tasks for Day 2
- Adjust timeline if necessary
- Prepare resources needed for backend foundation

### 4. Action Items:

- Complete any unfinished tasks from Day 1
- Prepare for initial backend implementation
- Set up database with the defined schema

## Day 2 Morning Preview:

- Initialize Spring Boot project

- Set up database connection
- Implement user entity and repository
- Create authentication service

This comprehensive setup on Day 1 creates a solid foundation for your development process. The CI/CD pipeline will help you maintain code quality and streamline deployments as you progress through the project.

Do you need more details on any specific part of these tasks?

### Prompt:

I set up Jenkins in my system on an 8080 port and I use jdk 21. so i don't know about the GUI and all things in jenkins please guide me to setup the pipeline from the beginning step.

i provide my previous details that i setup in previous step: My github repo link is: <https://github.com/yashdesai023/DrishyaScan> I use vs code ide for backend and frontend i setup files as you show in previous reply.

### Reply:

## Setting Up Jenkins Pipeline for DrishyaScan - Step by Step Guide

Since you've already installed Jenkins on port 8080 and have your GitHub repository set up, I'll guide you through the Jenkins GUI to create your CI/CD pipeline. Let's start from the beginning:

### Step 1: Access Jenkins and Initial Setup

1. Open your web browser and navigate to <http://localhost:8080>
2. You should see the Jenkins dashboard. If you're accessing for the first time, you might need to complete the initial setup.

### Step 2: Install Required Jenkins Plugins

1. From the Jenkins dashboard, click on **Manage Jenkins** in the left sidebar
2. Select **Plugins** or **Manage Plugins**
3. Click on the **Available** tab
4. Use the filter search box to find and select these plugins:
  - Git Integration Plugin

- Maven Integration Plugin
  - NodeJS Plugin
  - Pipeline Plugin
  - Docker Pipeline Plugin (if you plan to use Docker)
  - Blue Ocean (for a better UI experience)
  - JUnit Plugin (for test reporting)
5. Click **Install without restart**
  6. Check the option **Restart Jenkins when installation is complete and no jobs are running**

## Step 3: Configure Global Tools

1. After Jenkins restarts, go to **Manage Jenkins > Tools** (or **Global Tool Configuration** in older versions)
2. Scroll down and find the JDK section, click **Add JDK**:
  - Name: **JDK21**
  - Uncheck "Install automatically" if you've already installed JDK
  - JAVA\_HOME: Path to your JDK installation (e.g.,  
`/usr/lib/jvm/java-21-openjdk`)
3. Scroll to the Maven section, click **Add Maven**:
  - Name: **Maven3**
  - You can check "Install automatically" or provide path to existing Maven installation
4. Scroll to the NodeJS section, click **Add NodeJS**:
  - Name: **Node18** (or your preferred version)
  - Check "Install automatically"
  - Select the version you want to use
5. Click **Save** at the bottom of the page

## Step 4: Add GitHub Credentials

1. Go to **Manage Jenkins > Credentials > System > Global credentials > Add Credentials**
2. Select **Username with password** from the dropdown
3. Enter your GitHub username and password (or use a personal access token for password)
4. Set ID as **github-credentials** and a description
5. Click **OK**

## Step 5: Create a New Pipeline Job

1. Return to the Jenkins dashboard
2. Click **New Item** at the top left
3. Enter the name: **DrishyaScan-Pipeline**
4. Select **Pipeline** as the job type
5. Click **OK**

## Step 6: Configure the Pipeline

In the configuration page:

1. **General** section:
  - Add a description: "CI/CD pipeline for DrishyaScan project"
  - Check "GitHub project" and enter your repository URL:  
<https://github.com/yashdesai023/DrishyaScan>
2. **Build Triggers** section:
  - Check "GitHub hook trigger for GITScm polling" (for automatic builds when you push code)
3. **Pipeline** section:
  - Select "Pipeline script from SCM" from the Definition dropdown
  - Select "Git" from the SCM dropdown
  - Repository URL: <https://github.com/yashdesai023/DrishyaScan>
  - Credentials: Select the GitHub credentials you added earlier
  - Branch Specifier: **\*/main**
  - Script Path: **Jenkinsfile** (this file will need to be created in your repo)
4. Click **Save**

## Step 7: Create the Jenkinsfile

Now you need to create a Jenkinsfile in your GitHub repository:

1. In your local repository, create a file named **Jenkinsfile** (no extension) in the root directory
2. Copy the following content into the file:

```
pipeline {
```

agent any

```
tools {
  jdk 'JDK21'
  maven 'Maven3'
  nodejs 'Node18'
}

stages {
  stage('Checkout') {
    steps {
      checkout scm
    }
  }

  stage('Backend Setup') {
    steps {
      dir('drishyascan-backend') {
        // For now, just check if directory exists
        sh 'pwd && ls -la'

        // When you have the pom.xml file
        // sh 'mvn clean package -DskipTests'
      }
    }
  }

  stage('Frontend Setup') {
    steps {
      dir('drishyascan-frontend') {
        // For now, just check if directory exists
        sh 'pwd && ls -la'

        // When you have package.json file
        // sh 'npm install'
      }
    }
  }

  stage('Build Status') {
    steps {
      echo 'Project structure verified successfully!'
    }
  }
}
```

```

    }
  }
}

post {
  success {
    echo 'Pipeline executed successfully!'
  }
  failure {
    echo 'Pipeline failed. Check the logs for details.'
  }
}
}

```

3. Save the file, commit, and push to your GitHub repository:

```

git add Jenkinsfile
git commit -m "Add initial Jenkinsfile for CI/CD"
git push origin main

```

## Step 8: Set up GitHub Webhook (Optional)

To automatically trigger builds when you push code:

1. Go to your GitHub repository: <https://github.com/yashdesai023/DrishyaScan>
2. Click on **Settings** > **Webhooks** > **Add webhook**
3. Set Payload URL to: <http://your-public-ip:8080/github-webhook/>
  - Note: Your Jenkins server needs to be accessible from the internet for this to work
  - If running locally, you can use a service like ngrok to create a temporary public URL
4. Content type: [application/json](#)
5. Secret: leave empty for now (you can add a secret later for security)
6. Select "Just the push event"
7. Check "Active"
8. Click **Add webhook**

If your Jenkins is only accessible locally, you can skip this step and manually trigger builds.

## Step 9: Run Your First Pipeline

1. Go back to Jenkins dashboard
2. Click on your **DrishyaScan-Pipeline** job
3. Click **Build Now** in the left sidebar

Jenkins will now fetch your repository and execute the pipeline according to your Jenkinsfile.

## Step 10: Monitor the Pipeline Execution

1. Click on the build number that appears in the "Build History" section
2. Click on **Console Output** to see the build progress and any errors

## Step 11: Update the Jenkinsfile as Your Project Grows

As you develop your project, you'll want to update the Jenkinsfile to include actual build, test, and deployment steps. Here's how the file might evolve:

```
pipeline {
  agent any

  tools {
    jdk 'JDK21'
    maven 'Maven3'
    nodejs 'Node18'
  }

  stages {
    stage('Checkout') {
      steps {
        checkout scm
      }
    }

    stage('Backend Build') {
      steps {
        dir('drishyascan-backend') {
          sh 'mvn clean package -DskipTests'
        }
      }
    }

    stage('Backend Test') {
      steps {
```



```

    dir('drishyascan-backend') {
        sh 'mvn test'
    }
}
post {
    always {
        junit '**/target/surefire-reports/*.xml'
    }
}
}

stage('Frontend Build') {
    steps {
        dir('drishyascan-frontend') {
            sh 'npm install'
            sh 'npm run build'
        }
    }
}

stage('Frontend Test') {
    steps {
        dir('drishyascan-frontend') {
            sh 'npm test -- --watchAll=false'
        }
    }
}

stage('Deploy Development') {
    when {
        branch 'develop'
    }
    steps {
        echo 'Deploying to development environment...'
        // Add deployment steps when ready
    }
}

post {
    success {
        echo 'Pipeline completed successfully!'
    }
}

```

```

    }
    failure {
        echo 'Pipeline failed!'
    }
}
}

```

## Modify the Jenkinsfile for Windows

Your Jenkinsfile needs to be modified to work properly on Windows. Here's an updated version:

```

pipeline {
    agent any

    tools {
        jdk 'JDK21'
        maven 'Maven3'
        nodejs 'Node18'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Backend Setup') {
            steps {
                dir('drishyascan-backend') {
                    // Use bat instead of sh for Windows
                    bat 'dir'
                }
            }
        }
    }
}

```

```
// When you have the pom.xml file

// bat 'mvn clean package -DskipTests'

}

}

}

stage('Frontend Setup') {

steps {

dir('drishyascan-frontend') {

// Use bat instead of sh for Windows

bat 'dir'

// When you have package.json file

// bat 'npm install'

}

}

}

stage('Build Status') {

steps {

echo 'Project structure verified successfully!'

}

}

}

post {

success {

echo 'Pipeline executed successfully!'

}
```

```
}  
  
failure {  
  
    echo 'Pipeline failed. Check the logs for details.'  
  
}  
  
}  
  
}
```

## Important Tips for Jenkins Pipeline Development:

1. **Start Simple:** Begin with a basic pipeline that just checks out code, then gradually add build, test, and deploy stages as your project develops
2. **Use the Pipeline Syntax Generator:** In your pipeline job, click on "Pipeline Syntax" for help generating proper syntax for various steps
3. **Blue Ocean Plugin:** After installing, click "Open Blue Ocean" for a more visual pipeline execution view
4. **Pipeline Validation:** Jenkins will check your Jenkinsfile syntax when you push changes
5. **Debugging:** Always check the console output for any failed builds to understand what went wrong

By following these steps, you'll have a functional CI/CD pipeline with Jenkins that will automatically build and test your DrishyaScan project whenever you push changes to GitHub. The pipeline will evolve as you develop more functionality in your project.

### Prompt:

I am in day one afternoon from tasks i successfully completed one task "Set up CI/CD pipeline" now i want to move on 2nd task "Create wireframes for UI" please guide me in detail step by step from beginning in figma i don't use figma before please guide me and after successfully creating wireframes i want to give that model/link/screenshot for development then tell me how can i give you that UI wireframes

Reply:

# Creating UI Wireframes for DrishyaScan using Figma

I'll guide you through creating the wireframes for your DrishyaScan project using Figma, which is an excellent tool for UI design. Let's go through this step-by-step, from signing up to creating your first wireframes.

## Step 1: Set Up Figma Account

1. Visit [Figma's website](#)
2. Click on "Sign up" at the top-right corner
3. You can register with your email or use Google/Apple/Microsoft accounts
4. Complete the sign-up process and verify your email if required

## Step 2: Getting Familiar with Figma Interface

After logging in:

1. You'll be taken to your Figma dashboard
2. Click on the "+ New file" button to create a new design file
3. Select "Design file" from the options

Now you're in the Figma editor. Here's a quick overview of the interface:

- Left sidebar: Contains tools like selection, frame, shapes, etc.
- Right sidebar: Shows properties of selected elements
- Top bar: Contains file name, zoom controls, sharing options
- Canvas: The main design area

## Step 3: Set Up Your Project Structure

1. Start by creating frames for each screen:
  - Click on the Frame tool (F) in the left sidebar
  - Select a preset size (Desktop 1440x1024 or Custom)
2. Name your frames according to the screens we need:

- Login/Registration
- Dashboard
- Project Management
- Website Management
- Scan Screen
- Issues List
- Issue Detail
- Settings

## Step 4: Creating the Login/Registration Wireframe

Let's start with the Login screen:

1. Select your Login/Registration frame
2. Add a rectangle for the login card:
  - Select Rectangle tool (R)
  - Draw a centered card (about 400×500px)
3. Add the title:
  - Select Text tool (T)
  - Add "DrishyaScan Login" at the top of the card
  - Make it larger and centered
4. Add form elements:
  - Add text "Email" using Text tool
  - Draw a rectangle below it for the input field
  - Repeat for "Password"
5. Add login button:
  - Draw a rectangle for the button
  - Add "Login" text on it
  - Use your primary blue color (#2563EB)
6. Add "Forgot Password?" and "Register" links:
  - Use Text tool to add text
  - Style them as links (blue and underlined)
7. Add logo or branding at the top:
  - You can use shapes to create a placeholder logo

## Step 5: Creating the Dashboard Wireframe

1. Select your Dashboard frame
2. Create a navigation header:
  - Draw a rectangle across the top
  - Add logo placeholder on left
  - Add user menu on right
3. Create a sidebar:
  - Draw a rectangle on the left side (about 250px wide)
  - Add text elements for navigation items:
    - Dashboard
    - Projects
    - Websites
    - Reports
    - Settings
4. Create statistics cards:
  - Draw 3-4 small rectangular cards at the top of the content area
  - Add text for titles like "Total Projects", "Accessibility Score", etc.
  - Add placeholder numbers or charts
5. Create a recent scans section:
  - Add "Recent Scans" title text
  - Create a table structure with rectangles
  - Add column headers: "Website", "Date", "Score", "Status"
6. Add a chart area:
  - Draw a large rectangle
  - Add "Accessibility Trends" title
  - Add simple line chart placeholder

## Step 6: Creating Project Management Wireframe

1. Select your Project Management frame
2. Reuse the header and sidebar from the Dashboard (you can copy-paste)
3. Add page title "Projects"
4. Add "New Project" button in the top-right
5. Create project cards or list:
  - Draw rectangles for each project
  - Include project name, description, and stats
  - Add "View" and "Edit" buttons

## Step 7: Creating Remaining Wireframes

Follow similar patterns for the remaining screens:

1. Website Management:
  - List of websites within a project
  - "Add Website" button
  - Website details (URL, scan history)
2. Scan Screen:
  - URL input field
  - Scan options as checkboxes
  - "Start Scan" button
  - Progress indicator
3. Issues List:
  - Filtering options
  - Table showing issues
  - Severity indicators (use different colors)
  - Action buttons
4. Issue Detail:
  - Issue description
  - Code snippet area
  - WCAG reference
  - Recommendation box
5. Settings:
  - Profile settings section
  - Notification preferences
  - API key management

## Step 8: Adding Your Brand Colors

1. Click on the "Colors" section in the right panel while nothing is selected
2. Click "+" to add colors
3. Add your DrishyaScan colors:
  - Primary Blue: #2563EB
  - Accent Purple: #7C3AED
  - Success Green: #10B981
  - Warning Yellow: #F59E0B
  - Error Red: #EF4444
  - Light backgrounds, dark text colors, etc.



## Step 9: Creating Interactive Elements (Optional)

1. Create a component for buttons:
  - Design a button
  - Right-click and select "Create component"
2. Create a component for form fields:
  - Design an input field
  - Make it a component
3. Set up simple interactions:
  - Select an element
  - Go to Prototype tab in the right panel
  - Click "+" to add an interaction
  - Connect it to another frame

## Step 10: Sharing Your Wireframes

1. Click the "Share" button in the top-right corner
2. Adjust permissions (View or Edit)
3. Copy the link to share with others
4. Alternatively, you can export as images or PDFs:
  - Select frames
  - Right-click and select "Export"
  - Choose format (PNG, JPG, PDF)

## Step 11: Providing Wireframes for Development

There are several ways to share your wireframes with the development team:

1. **Share Figma Link:**
  - Share the direct Figma link
  - This allows developers to inspect elements, measure distances, and view specifications
2. **Export as Images:**
  - Export each screen as PNG or JPG
  - Share these images via email or project management tool

### 3. Export as PDF:

- Export all screens as a single PDF
- Good for documentation purposes

### 4. Use Figma's Dev Mode:

- Enables developers to get CSS information directly
- Click on "Dev Mode" in the right corner of Figma

Once you've created your wireframes, you can share them with me in any of these formats - a Figma link would be most useful as I can provide feedback on the design, but screenshots or exported images work too.

Would you like me to clarify any particular part of this process? Or do you want me to go into more detail on any specific wireframe?

### Prompt:

My one task is remaining from day 1 afternoon task is Design database schema please guide me to complete this task

### Reply:

## DrishyaScan Database Schema Design

I've completed the database schema design for DrishyaScan based on the project requirements. The schema is comprehensive and includes all necessary tables to support the accessibility analyzer's functionality.

-- DrishyaScan Database Schema

-- Users Table - Stores user account information

CREATE TABLE users (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

email VARCHAR(255) NOT NULL UNIQUE,

password\_hash VARCHAR(255) NOT NULL,

```
first_name VARCHAR(100),  
last_name VARCHAR(100),  
role ENUM('ADMIN', 'DEVELOPER', 'CONTENT_MANAGER') NOT NULL,  
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
active BOOLEAN DEFAULT TRUE,  
last_login_at TIMESTAMP,  
profile_image_url VARCHAR(512)  
);
```

-- Projects Table - Organizes websites into projects

```
CREATE TABLE projects (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    description TEXT,  
    user_id BIGINT NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
    CURRENT_TIMESTAMP,  
    is_archived BOOLEAN DEFAULT FALSE,  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE  
);
```

-- Project Members Table - Manages team collaborations on projects

```
CREATE TABLE project_members (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    project_id BIGINT NOT NULL,  
    user_id BIGINT NOT NULL,  
    role ENUM('OWNER', 'EDITOR', 'VIEWER') NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (project_id) REFERENCES projects(id) ON DELETE CASCADE,  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,  
    UNIQUE KEY unique_project_user (project_id, user_id)  
);
```

-- Websites Table - Stores website information for scanning

```
CREATE TABLE websites (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    url VARCHAR(2048) NOT NULL,  
    name VARCHAR(255) NOT NULL,  
    project_id BIGINT NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
    is_active BOOLEAN DEFAULT TRUE,  
    favicon_url VARCHAR(512),  
    last_scan_date TIMESTAMP,  
    last_accessibility_score FLOAT,
```

```
FOREIGN KEY (project_id) REFERENCES projects(id) ON DELETE CASCADE
);

-- Scan Schedules Table - Manages automated scan scheduling

CREATE TABLE scan_schedules (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    website_id BIGINT NOT NULL,

    frequency ENUM('DAILY', 'WEEKLY', 'MONTHLY') NOT NULL,

    day_of_week INT,

    day_of_month INT,

    time_of_day TIME NOT NULL,

    is_active BOOLEAN DEFAULT TRUE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP,

    FOREIGN KEY (website_id) REFERENCES websites(id) ON DELETE CASCADE
);
```

-- Scans Table - Records of website accessibility scans

```
CREATE TABLE scans (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    website_id BIGINT NOT NULL,

    scan_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    accessibility_score FLOAT,
```

```
status ENUM('PENDING', 'IN_PROGRESS', 'COMPLETED', 'FAILED') NOT NULL,  
  
started_at TIMESTAMP,  
  
completed_at TIMESTAMP,  
  
error_message TEXT,  
  
scan_depth INT DEFAULT 1,  
  
pages_scanned INT DEFAULT 0,  
  
initiated_by_user_id BIGINT,  
  
FOREIGN KEY (website_id) REFERENCES websites(id) ON DELETE CASCADE,  
  
FOREIGN KEY (initiated_by_user_id) REFERENCES users(id) ON DELETE SET NULL  
  
);
```

-- Issues Table - Accessibility issues found during scans

```
CREATE TABLE issues (  
  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  
    scan_id BIGINT NOT NULL,  
  
    page_url VARCHAR(2048) NOT NULL,  
  
    element_selector VARCHAR(512),  
  
    type VARCHAR(100) NOT NULL,  
  
    severity ENUM('CRITICAL', 'MAJOR', 'MINOR', 'INFO') NOT NULL,  
  
    wcag_criterion VARCHAR(20),  
  
    wcag_principle VARCHAR(100),  
  
    description TEXT NOT NULL,  
  
    html_snippet TEXT,  
  
    recommendation TEXT,
```

```
status ENUM('OPEN', 'IN_PROGRESS', 'FIXED', 'FALSE_POSITIVE') DEFAULT 'OPEN',

fixed_in_scan_id BIGINT,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

FOREIGN KEY (scan_id) REFERENCES scans(id) ON DELETE CASCADE,

FOREIGN KEY (fixed_in_scan_id) REFERENCES scans(id) ON DELETE SET NULL

);
```

-- Issue Comments Table - Allows team discussion on issues

```
CREATE TABLE issue_comments (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    issue_id BIGINT NOT NULL,

    user_id BIGINT NOT NULL,

    comment TEXT NOT NULL,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (issue_id) REFERENCES issues(id) ON DELETE CASCADE,

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE

);
```

-- Reports Table - Generated accessibility reports

```
CREATE TABLE reports (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    scan_id BIGINT NOT NULL,
```

```
generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
report_type ENUM('DETAILED', 'SUMMARY', 'COMPLIANCE') NOT NULL,  
format ENUM('HTML', 'PDF', 'CSV') NOT NULL,  
file_path VARCHAR(255),  
generated_by_user_id BIGINT,  
name VARCHAR(255),  
is_archived BOOLEAN DEFAULT FALSE,  
FOREIGN KEY (scan_id) REFERENCES scans(id) ON DELETE CASCADE,  
FOREIGN KEY (generated_by_user_id) REFERENCES users(id) ON DELETE SET NULL  
);
```

-- API Keys Table - For API access to the system

```
CREATE TABLE api_keys (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    user_id BIGINT NOT NULL,  
    api_key VARCHAR(64) NOT NULL UNIQUE,  
    name VARCHAR(100) NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    expires_at TIMESTAMP,  
    is_active BOOLEAN DEFAULT TRUE,  
    last_used_at TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE  
);
```



-- Settings Table - System and user preferences

```
CREATE TABLE settings (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    user_id BIGINT,  
    setting_key VARCHAR(100) NOT NULL,  
    setting_value TEXT NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,  
    UNIQUE KEY unique_user_setting (user_id, setting_key)  
);
```

-- WCAG Guidelines Reference Table

```
CREATE TABLE wcag_guidelines (  
    id VARCHAR(20) PRIMARY KEY,  
    principle VARCHAR(100) NOT NULL,  
    guideline VARCHAR(255) NOT NULL,  
    success_criterion TEXT NOT NULL,  
    level ENUM('A', 'AA', 'AAA') NOT NULL,  
    description TEXT  
);
```

-- Notification Table - System notifications for users

```
CREATE TABLE notifications (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    user_id BIGINT NOT NULL,  
    title VARCHAR(255) NOT NULL,  
    message TEXT NOT NULL,  
    type ENUM('SCAN_COMPLETE', 'ISSUE_FIXED', 'SCORE_IMPROVED', 'SYSTEM') NOT  
NULL,  
    is_read BOOLEAN DEFAULT FALSE,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    related_entity_type VARCHAR(50),  
    related_entity_id BIGINT,  
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE  
);
```

-- Audit Log Table - Track important system activities

```
CREATE TABLE audit_logs (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    user_id BIGINT,  
    action VARCHAR(100) NOT NULL,  
    entity_type VARCHAR(50) NOT NULL,  
    entity_id BIGINT,  
    details JSON,  
    ip_address VARCHAR(45),  
    user_agent TEXT,
```

```
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL  
    );
```

-- Create indexes for performance optimization

```
CREATE INDEX idx_websites_project_id ON websites(project_id);  
  
CREATE INDEX idx_scans_website_id ON scans(website_id);  
  
CREATE INDEX idx_issues_scan_id ON issues(scan_id);  
  
CREATE INDEX idx_issues_severity ON issues(severity);  
  
CREATE INDEX idx_issues_wcag_criterion ON issues(wcag_criterion);  
  
CREATE INDEX idx_reports_scan_id ON reports(scan_id);  
  
CREATE INDEX idx_notifications_user_id ON notifications(user_id);  
  
CREATE INDEX idx_audit_logs_user_id ON audit_logs(user_id);  
  
CREATE INDEX idx_audit_logs_entity ON audit_logs(entity_type, entity_id);
```

## Key Components of the Database Schema

### 1. User Management

- **users** table for storing user account information
- **project\_members** table for team collaboration
- **api\_keys** table for API access
- **settings** table for user preferences

### 2. Project Organization

- **projects** table for organizing websites
- **websites** table for storing website information

### 3. Scanning & Analysis

- **scans** table for recording scan sessions
- **scan\_schedules** table for automated scanning
- **issues** table for accessibility problems found
- **wcag\_guidelines** reference table for accessibility standards

### 4. Reporting & Collaboration

- `reports` table for generated accessibility reports
- `issue_comments` table for team discussions
- `notifications` table for system alerts
- `audit_logs` table for tracking system activities

## Enhanced Features

The schema includes several enhancements beyond the basic requirements:

1. **Team Collaboration**
  - Project members with different roles (owner, editor, viewer)
  - Issue commenting system for team discussion
2. **Automated Scanning**
  - Scheduling system for regular accessibility checks
  - Tracking of scan history and improvement over time
3. **Comprehensive Issue Tracking**
  - Detailed issue information including page URL and element selector
  - Issue status tracking (open, in progress, fixed, false positive)
  - Connection to WCAG guidelines for educational purposes
4. **System Security & Monitoring**
  - API key management for programmatic access
  - Audit logging for security and compliance
  - User notification system

## Benefits of this Schema Design

1. **Scalability:** Designed to handle multiple users, projects, and websites
2. **Performance:** Includes appropriate indexes for faster queries
3. **Flexibility:** Supports various report types and formats
4. **Traceability:** Maintains historical data for tracking improvements
5. **Security:** Includes robust user management and audit logging

## Next Steps

Now that your Day 1 tasks are complete, you can proceed with Day 2's backend foundation tasks:

1. Initialize Spring Boot project
2. Set up database connection using this schema
3. Implement user entity and repository
4. Create authentication service

Would you like me to help with any of these next steps or provide additional guidance on implementing the database schema in your Spring Boot project?

**Prompt:**

How to use in postman guide step by step for validation

**Reply:**

# Using the DrishyaScan Authentication API with Postman - Step by Step Guide

I'll guide you through testing your new JWT authentication system using Postman. This will help validate that your implementation works correctly.

## Setup Postman

### 1. Download and Install Postman:

- If you don't have it already, download it from [postman.com](https://postman.com)

### 2. Create a New Collection:

- Click "New" → "Collection"
- Name it "DrishyaScan API"
- Click "Create"

## Test User Registration

### 1. Create a Registration Request:

1. Click "New" → "Request"
2. Name it "Register User"
3. Select the "DrishyaScan API" collection
4. Click "Save"

### 2. Set Up the Registration Request:

1. Set the HTTP method to **POST**
2. Enter URL: **`http://localhost:8080/api/auth/register`**
3. Go to "Headers" tab:

- Add **Content-Type** with value **application/json**
- 4. Go to "Body" tab:
  - Select "raw" and "JSON"
  - Enter the following JSON:

```
{
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "password": "Password123!",
  "role": "DEVELOPER"
}
```

### 3. Send the Registration Request:

1. Click "Send"
2. Expected Response (Status Code 201 Created):

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiJ9...",
  "user": {
    "id": 1,
    "email": "john.doe@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "role": "DEVELOPER",
    "active": true,
    "lastLoginAt": "2025-05-01T12:30:45.123456"
  }
}
```

### 4. Validation Test Cases:

1. **Test Invalid Email:**
  - Change email to "not-an-email"
  - Send request
  - Should receive 400 Bad Request with validation error
2. **Test Password Too Short:**
  - Change password to "123"
  - Send request

- Should receive 400 Bad Request with validation error
- 3. Test Duplicate Email:**
- Send the original request again with the same email
  - Should receive 400 Bad Request with message about email already registered

## Test User Login

### 1. Create a Login Request:

1. Click "New" → "Request"
2. Name it "Login User"
3. Select the "DrishyaScan API" collection
4. Click "Save"

### 2. Set Up the Login Request:

1. Set the HTTP method to **POST**
2. Enter URL: **`http://localhost:8080/api/auth/login`**
3. Go to "Headers" tab:
  - Add **Content-Type** with value **`application/json`**
4. Go to "Body" tab:
  - Select "raw" and "JSON"
  - Enter the following JSON:

```
{  
  "email": "john.doe@example.com",  
  "password": "Password123!"  
}
```

### 3. Send the Login Request:

1. Click "Send"
2. Expected Response (Status Code 200 OK):

```
{  
  "accessToken": "eyJhbGciOiJIUzI1NiJ9...",  
  "user": {  
    "id": 1,  
    "email": "john.doe@example.com",  
    "firstName": "John",  
    "lastName": "Doe",  
  }  
}
```

```
"role": "DEVELOPER",
"active": true,
"lastLoginAt": "2025-05-01T12:35:22.123456"
}
}
```

#### 4. Validation Test Cases:

##### 1. Test Wrong Password:

- Change password to "WrongPassword123"
- Send request
- Should receive 401 Unauthorized

##### 2. Test Non-Existent Email:

- Change email to "nonexistent@example.com"
- Send request
- Should receive 401 Unauthorized

## Test Protected Endpoint Access

### 1. Create a Protected API Request:

1. Click "New" → "Request"
2. Name it "Get Protected Resource"
3. Select the "DrishyaScan API" collection
4. Click "Save"

### 2. Set Up the Protected Request:

1. Set the HTTP method to **GET**
2. Enter URL for a protected endpoint (e.g., **http://localhost:8080/api/users/me**)
3. Go to "Headers" tab:
  - Add **Authorization** with value **Bearer {token from login response}**
  - Replace **{token from login response}** with the actual token received during login

### 3. Send the Protected Request:

1. Click "Send"
2. Expected Response (Status Code 200 OK) with protected resource data



## 4. Validation Test Cases:

### 1. Test No Token:

- Remove the Authorization header
- Send request
- Should receive 401 Unauthorized

### 2. Test Invalid Token:

- Change token to an invalid value
- Send request
- Should receive 401 Unauthorized

### 3. Test Expired Token (manual test after token expires):

- Use an expired token
- Send request
- Should receive 401 Unauthorized

## Set Up Environment Variables for Easier Testing

### 1. Create an Environment:

1. Click on the "Environment" button (gear icon)
2. Click "Add" to create a new environment
3. Name it "DrishyaScan Local"
4. Click "Add"

### 2. Define Environment Variables:

1. Add variable `base_url` with value `http://localhost:8080`
2. Add variable `token` (leave it empty for now)
3. Click "Save"

### 3. Use Environment Variables in Requests:

1. Update all request URLs to use the base URL variable:  
`{{base_url}}/api/auth/register`
2. Create a login request test script:
  - Go to the "Tests" tab of your login request
  - Add the following script:

```
if (pm.response.code === 200) {  
  var jsonData = pm.response.json();  
}
```

```
pm.environment.set("token", jsonData.accessToken);  
}
```

3. Update all protected requests to use the token variable:
  - Change Authorization header to **Bearer {{token}}**
4. Now when you log in successfully, the token will be automatically saved and used in subsequent requests

## Additional Tests

### 1. Create an Admin User:

1. Duplicate your "Register User" request
2. Change the JSON body to create an admin:

```
{  
  "firstName": "Admin",  
  "lastName": "User",  
  "email": "admin@example.com",  
  "password": "AdminPass123!",  
  "role": "ADMIN"  
}
```

### 2. Test Role-Based Access Control:

If you have endpoints with role-specific restrictions, you can test these by:

1. Login as a regular user (DEVELOPER)
2. Try to access an admin-only endpoint
3. Should receive 403 Forbidden
4. Then login as ADMIN
5. Try the same endpoint
6. Should receive 200 OK

## Troubleshooting Tips

- **401 Unauthorized:** Check if your token is valid and properly formatted
- **403 Forbidden:** User authenticated but lacks permission (role-based issue)
- **500 Internal Server Error:** Check server logs for exceptions

- **Token Not Working:** Check token expiration (default 24 hours in your config)
- **CORS Issues:** If testing from a frontend app, ensure your backend has proper CORS configuration

## Summary

This guide walks you through validating your JWT authentication implementation using Postman. You've tested:

1. User registration with validation
2. User login with credential verification
3. Protected endpoint access with JWT tokens
4. Error handling with various invalid inputs

By following these steps and running these tests, you can ensure your authentication system is working correctly and securely before integrating it with your frontend application.

### Prompt:

Got status code 403

### Reply:

If you're receiving a 403 Forbidden status code when testing your DrishyaScan authentication API, there are several potential issues to investigate. Let's troubleshoot this systematically.

## Understanding 403 Forbidden Error

A 403 Forbidden status code indicates that:

- Your server understood the request
- You are authenticated (the server knows who you are)
- But you don't have permission to access the requested resource

This is different from 401 Unauthorized, which indicates authentication issues.

## Where Are You Getting the 403 Error?

### Possible Scenario 1: During Registration or Login

If you're getting a 403 Forbidden when trying to access `/api/auth/register` or `/api/auth/login`:

1. **Check Security Configuration:** Your security configuration might not be properly setting these endpoints as public. Let's review your `SecurityConfig.java`:

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(AbstractHttpConfigurer::disable)
        .sessionManagement(session -> session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        )
        .authorizeHttpRequests(auth -> auth
            // Public endpoints - these should be accessible without authentication
            .requestMatchers("/api/auth/register", "/api/auth/login").permitAll()
            // All other requests need authentication
            .anyRequest().authenticated()
        )
        .authenticationProvider(authenticationProvider())
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

**Fix:** Make sure the path you're accessing exactly matches the permitted paths.

2. **CSRF Protection:** Even though you've disabled CSRF, sometimes there can be issues. Try explicitly adding:

```
http.csrf().disable();
```

## Possible Scenario 2: Accessing Protected Endpoints

If you're getting 403 when accessing a protected endpoint after logging in:

1. **JWT Token Issues:**

- Verify your token contains the correct authorities
- Check if you're correctly extracting role information

2. **Role-Based Access Control:**

- You might have endpoint-specific authorization rules that are preventing access
- Check if you have any `@PreAuthorize` annotations on your controller methods

3. **JwtAuthenticationFilter Issues:**

- Your filter might not be correctly setting authorities

## Debugging Steps

### 1. Add Debug Logging

Add these to your `application.properties`:

```
logging.level.org.springframework.security=DEBUG
logging.level.com.drishyascan.security=TRACE
```

### 2. Debug JwtAuthenticationFilter

Update your filter with logging statements:

```
@Override
protected void doFilterInternal(
    HttpServletRequest request,
    HttpServletResponse response,
    FilterChain filterChain) throws ServletException, IOException {

    logger.debug("Processing request: " + request.getRequestURI());

    // Extract Authorization header
    final String authorizationHeader = request.getHeader("Authorization");
    logger.debug("Authorization header: " + (authorizationHeader != null ? "present" : "missing"));

    String email = null;
    String jwt = null;

    // Check if Authorization header exists and has Bearer token
    if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
        // Extract JWT token
        jwt = authorizationHeader.substring(7);
        try {
            // Extract username (email) from token
            email = jwtUtil.extractUsername(jwt);
            logger.debug("Extracted email from token: " + email);
        } catch (Exception e) {
            // Log exception (optional)
            logger.error("JWT token is invalid: " + e.getMessage());
        }
    }
}
```

```

// If email is extracted and user is not yet authenticated
if (email != null && SecurityContextHolder.getContext().getAuthentication() == null) {
    UserDetails userDetails = this.userDetailsService.loadUserByUsername(email);
    logger.debug("User roles: " + userDetails.getAuthorities());

    // Validate token against userDetails
    if (jwtUtil.validateToken(jwt, userDetails)) {
        logger.debug("Token validated successfully for user: " + email);

        // Create authentication token
        UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
            userDetails,
            null,
            userDetails.getAuthorities()
        );

        // Set authentication details
        authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

        // Set authentication in SecurityContext
        SecurityContextHolder.getContext().setAuthentication(authToken);
        logger.debug("Authentication set in SecurityContext");
    } else {
        logger.debug("Token validation failed");
    }
}

// Continue filter chain
filterChain.doFilter(request, response);
}

```

### 3. Check User Authorities

Verify that your `CustomUserDetailsService` is correctly loading authorities:

```

@Override
public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
    // Find user by email
    User user = userRepository.findByEmail(email)
        .orElseThrow(() -> new UsernameNotFoundException("User not found with email: " +
email));
}

```

```

// Check if user is active
if (!user.isActive()) {
    throw new UsernameNotFoundException("User account is not active");
}

// Create authority
SimpleGrantedAuthority authority = new SimpleGrantedAuthority("ROLE_" +
user.getRole().name());

System.out.println("Loading user: " + email + " with role: " + authority.getAuthority());

// Convert User entity to Spring Security UserDetails
return new org.springframework.security.core.userdetails.User(
    user.getEmail(),
    user.getPasswordHash(),
    Collections.singletonList(authority)
);
}

```

## 4. Check Your Controller For Role-Based Restrictions

If you have any `@PreAuthorize` annotations on your controller methods:

```

@RestController
@RequestMapping("/api/some-endpoint")
public class SomeController {

    @GetMapping
    // This will restrict access to ADMIN only
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public ResponseEntity<?> getProtectedResource() {
        // method implementation
    }
}

```

Make sure you're using the correct role when testing.

## Quick Fixes To Try

1. **Update your SecurityConfig:**

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(AbstractHttpConfigurer::disable)
        .cors(Customizer.withDefaults()) // Enable CORS with default settings
        .sessionManagement(session -> session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        )
        .authorizeHttpRequests(auth -> auth
            // Public endpoints
            .requestMatchers("/api/auth/**").permitAll() // Allow all endpoints under /api/auth/
            .requestMatchers("/swagger-ui/**", "/v3/api-docs/**").permitAll()
            // All other requests need authentication
            .anyRequest().authenticated()
        )
        .authenticationProvider(authenticationProvider())
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

```

2. **Test with a simple controller** to rule out complex permission issues:

```

@RestController
@RequestMapping("/api/test")
public class TestController {

    @GetMapping("/public")
    public ResponseEntity<String> publicEndpoint() {
        return ResponseEntity.ok("This is a public endpoint");
    }

    @GetMapping("/protected")
    public ResponseEntity<String> protectedEndpoint() {
        return ResponseEntity.ok("This is a protected endpoint");
    }
}

```

Then update your security config to make the test/public endpoint accessible:

```

.requestMatchers("/api/auth/**", "/api/test/public").permitAll()

```



3. **Check HTTP Method Restrictions:** Spring Security may be limiting certain HTTP methods. Make sure you're allowing the correct HTTP methods in your security configuration.
4. **For Registration Endpoint Specifically:** A common issue is when you're getting 403 on registration, which should be public. Try this explicit approach:

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
    http  
        .csrf(AbstractHttpConfigurer::disable)  
        .authorizeHttpRequests(auth -> {  
            auth.requestMatchers(HttpMethod.POST, "/api/auth/register").permitAll();  
            auth.requestMatchers(HttpMethod.POST, "/api/auth/login").permitAll();  
            auth.requestMatchers("/swagger-ui/**", "/v3/api-docs/**").permitAll();  
            auth.anyRequest().authenticated();  
        })  
        .sessionManagement(session -> session  
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
        )  
        .authenticationProvider(authenticationProvider())  
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);  
  
    return http.build();  
}
```

Let me know which of these scenarios applies to your situation, and I can provide more specific guidance.

