Full-stack CRUD application using React with functional components on the frontend and a Spring Boot application with a layered architecture on the backend, secured with JWT-based stateless authentication.
The backend uses Spring Data JPA with PostgreSQL, and the frontend integrates Axios for API calls.

The application includes a login page and separate dashboards for users and admins.

Project Overview
- **Backend**: Spring Boot with layered architecture (Controller, Service, Repository), JWT authentication, and Spring Data JPA with PostgreSQL.
- **Frontend**: React with functional components, Axios for API calls, and routing for login, user dashboard, and admin dashboard.
- **Features**:
  - CRUD operations on an entity (e.g., `Task`).
  - Token-based authentication.
  - Role-based dashboards (User and Admin).

Backend: Spring Boot Application

1. Project Setup (`pom.xml`)
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.4</version>
    <relativePath/>
  </parent>
  <groupId>com.example</groupId>
  <artifactId>spring-boot-crud</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>spring-boot-crud</name>

  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>


    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

```xml
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-api</artifactId>
            <version>0.12.5</version>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-impl</artifactId>
            <version>0.12.5</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-jackson</artifactId>
            <version>0.12.5</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Configuration (`application.yml`)

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/taskdb
    username: yourusername
    password: yourpassword
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect

jwt:
  secret: your-very-secure-secret-key-1234567890
  expiration: 86400000

server:
  port: 8080
```

3. Entities

```java
// Task.java
package com.example.entity;

import jakarta.persistence.*;
import lombok.Data;

@Entity
@Data
@Table(name = "tasks")
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    private String description;

    private boolean completed;
}

// User.java
package com.example.entity;

import jakarta.persistence.*;
import lombok.Data;

import java.util.HashSet;
import java.util.Set;

@Entity
@Data
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles = new HashSet<>();
}

// Role.java
package com.example.entity;

import jakarta.persistence.*;
import lombok.Data;

@Entity
@Data
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String name;
}
```

4. Repositories

```
// TaskRepository.java
package com.example.repository;

import com.example.entity.Task;
import org.springframework.data.jpa.repository.JpaRepository;

public interface TaskRepository extends JpaRepository<Task, Long> {
}

// UserRepository.java
package com.example.repository;

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

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

// RoleRepository.java
package com.example.repository;

import com.example.entity.Role;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findByName(String name);
}
```

5. Security Configuration

```
// JwtTokenProvider.java
package com.example.config;

import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.stream.Collectors;

@Component
public class JwtTokenProvider {
  @Value("${jwt.secret}")
  private String jwtSecret;

  @Value("${jwt.expiration}")
  private long jwtExpiration;

  public String generateToken(Authentication authentication) {
    String username = authentication.getName();
    String roles = authentication.getAuthorities().stream()
        .map(GrantedAuthority::getAuthority)
        .collect(Collectors.joining(","));

    return Jwts.builder()
        .subject(username)
        .claim("roles", roles)
        .issuedAt(new Date())
        .expiration(new Date(System.currentTimeMillis() + jwtExpiration))
        .signWith(Keys.hmacShaKeyFor(jwtSecret.getBytes()))
        .compact();
  }

  public String getUsernameFromToken(String token) {
    return Jwts.parser()
        .verifyWith(Keys.hmacShaKeyFor(jwtSecret.getBytes()))
        .build()
        .parseSignedClaims(token)
        .getPayload()
        .getSubject();
  }
  public boolean validateToken(String token) {
    try {
      Jwts.parser()
        .verifyWith(Keys.hmacShaKeyFor(jwtSecret.getBytes()))
        .build()
        .parseSignedClaims(token);
      return true;
    } catch (JwtException e) {
      return false;
    }
  }
}

// JwtAuthenticationFilter.java
package com.example.config;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

  @Autowired
  private JwtTokenProvider jwtTokenProvider;

  @Autowired
  private UserDetailsService userDetailsService;

  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
      throws ServletException, IOException {
    String token = getTokenFromRequest(request);

    if (token != null && jwtTokenProvider.validateToken(token)) {
      String username = jwtTokenProvider.getUsernameFromToken(token);
      UserDetails userDetails = userDetailsService.loadUserByUsername(username);

      UsernamePasswordAuthenticationToken authentication =
          new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
      authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

      SecurityContextHolder.getContext().setAuthentication(authentication);
    }

    filterChain.doFilter(request, response);
  }
```

```
    private String getTokenFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return null;
    }
}
```

```
// SecurityConfig.java
package com.example.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .requestMatchers("/api/user/**").hasRole("USER")
                .anyRequest().authenticated()
            );

        http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
        return authConfig.getAuthenticationManager();
    }
}
```

6. Service Layer

```
// UserService.java
package com.example.service;

import com.example.config.JwtTokenProvider;
import com.example.entity.Role;
import com.example.entity.User;
import com.example.repository.RoleRepository;
import com.example.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Collections;

@Service
public class UserService implements UserDetailsService {

  @Autowired
  private UserRepository userRepository;

  @Autowired
  private RoleRepository roleRepository;

  @Autowired
  private PasswordEncoder passwordEncoder;
  @Autowired
  private JwtTokenProvider jwtTokenProvider;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username);
    if (user == null) {
      throw new UsernameNotFoundException("User not found: " + username);
    }
    return org.springframework.security.core.userdetails.User
        .withUsername(user.getUsername())
        .password(user.getPassword())
        .authorities(user.getRoles().stream()
            .map(role -> "ROLE_" + role.getName())
            .toArray(String[]::new))
        .build();
  }

  @Transactional
  public String registerUser(String username, String password, String roleName) {
    if (userRepository.findByUsername(username) != null) {
      throw new RuntimeException("Username already exists");
    }
```

```
        User user = new User();
        user.setUsername(username);
        user.setPassword(passwordEncoder.encode(password));

        Role role = roleRepository.findByName(roleName);
        if (role == null) {
            role = new Role();
            role.setName(roleName);
            roleRepository.save(role);
        }
        user.setRoles(Collections.singleton(role));

        userRepository.save(user);

        UserDetails userDetails = loadUserByUsername(username);
        return jwtTokenProvider.generateToken(
                new org.springframework.security.authentication.UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities()));
    }

    public String loginUser(String username, String password) {
        UserDetails userDetails = loadUserByUsername(username);
        if (passwordEncoder.matches(password, userDetails.getPassword())) {
            return jwtTokenProvider.generateToken(
                    new org.springframework.security.authentication.UsernamePasswordAuthenticationToken(
                            userDetails, null, userDetails.getAuthorities()));
        }
        throw new RuntimeException("Invalid credentials");
    }
}
```

```
// TaskService.java
package com.example.service;

import com.example.entity.Task;
import com.example.repository.TaskRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class TaskService {

    @Autowired
    private TaskRepository taskRepository;

    public List<Task> getAllTasks() {
        return taskRepository.findAll();
    }

    public Optional<Task> getTaskById(Long id) {
        return taskRepository.findById(id);
    }
    public Task createTask(Task task) {
        return taskRepository.save(task);
    }

    public Task updateTask(Long id, Task taskDetails) {
        Task task = taskRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Task not found"));
        task.setTitle(taskDetails.getTitle());
        task.setDescription(taskDetails.getDescription());
        task.setCompleted(taskDetails.isCompleted());
        return taskRepository.save(task);
    }

    public void deleteTask(Long id) {
        Task task = taskRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Task not found"));
        taskRepository.delete(task);
    }
}
```

7. Controller Layer

```
// AuthController.java
package com.example.controller;

import com.example.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private UserService userService;

    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestParam String username,
                                           @RequestParam String password,
                                           @RequestParam String role) {
        String token = userService.registerUser(username, password, role);
        return ResponseEntity.ok(token);
    }

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username,
                                        @RequestParam String password) {
        String token = userService.loginUser(username, password);
        return ResponseEntity.ok(token);
    }
}
```

```java
// TaskController.java
package com.example.controller;

import com.example.entity.Task;
import com.example.service.TaskService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
public class TaskController {

  @Autowired
  private TaskService taskService;

  @GetMapping("/user/tasks")
  public ResponseEntity<List<Task>> getAllTasksForUser() {
    return ResponseEntity.ok(taskService.getAllTasks());
  }

  @GetMapping("/admin/tasks")
  public ResponseEntity<List<Task>> getAllTasksForAdmin() {
    return ResponseEntity.ok(taskService.getAllTasks());
  }

  @GetMapping("/user/tasks/{id}")
  public ResponseEntity<Task> getTaskByIdForUser(@PathVariable Long id) {
    return taskService.getTaskById(id)
        .map(ResponseEntity::ok)
        .orElseGet(() -> ResponseEntity.notFound().build());
  }

  @PostMapping("/user/tasks")
  public ResponseEntity<Task> createTaskForUser(@RequestBody Task task) {
    return ResponseEntity.ok(taskService.createTask(task));
  }

  @PutMapping("/user/tasks/{id}")
  public ResponseEntity<Task> updateTaskForUser(@PathVariable Long id, @RequestBody Task task) {
    return ResponseEntity.ok(taskService.updateTask(id, task));
  }

  @DeleteMapping("/user/tasks/{id}")
  public ResponseEntity<Void> deleteTaskForUser(@PathVariable Long id) {
    taskService.deleteTask(id);
    return ResponseEntity.ok().build();
  }
@DeleteMapping("/admin/tasks/{id}")
    public ResponseEntity<Void> deleteTaskForAdmin(@PathVariable Long id) {
        taskService.deleteTask(id);
        return ResponseEntity.ok().build();
    }
}
```

8. Main Application

```
// SpringBootCrudApplication.java
package com.example;

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

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

Frontend: React Application

1. Project Setup

```
npx create-react-app react-crud-app
cd react-crud-app
npm install axios react-router-dom bootstrap
```

2. `src/App.js`

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Login from './components/Login';
import UserDashboard from './components/UserDashboard';
import AdminDashboard from './components/AdminDashboard';
import 'bootstrap/dist/css/bootstrap.min.css';

function App() {
  return (
    <Router>
      <div className="App">
        <Routes>
          <Route path="/" element={<Login />} />
          <Route path="/user-dashboard" element={<UserDashboard />} />
          <Route path="/admin-dashboard" element={<AdminDashboard />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

3. API Service (` src/services/api.js`)

```
import axios from 'axios';

const API_URL = 'http://localhost:8080/api';

const api = axios.create({
  baseURL: API_URL,
});

api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = ` Bearer ${token}` ;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

export const login = (username, password) =>
  api.post('/auth/login', null, { params: { username, password } });

export const register = (username, password, role) =>
  api.post('/auth/register', null, { params: { username, password, role } });

export const getTasks = () => api.get('/user/tasks');
export const getTaskById = (id) => api.get(` /user/tasks/${id}` );
export const createTask = (task) => api.post('/user/tasks', task);
export const updateTask = (id, task) => api.put(` /user/tasks/${id}` , task);
export const deleteTask = (id) => api.delete(` /user/tasks/${id}` );
export const deleteTaskAdmin = (id) => api.delete(` /admin/tasks/${id}` );
```

4. Login Component (` src/components/Login.js`)

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { login, register } from '../services/api';

const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [role, setRole] = useState('USER');
  const [isRegistering, setIsRegistering] = useState(false);
  const [error, setError] = useState('');
  const navigate = useNavigate();
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = isRegistering
      ? await register(username, password, role)
      : await login(username, password);
    localStorage.setItem('token', response.data);
    const tokenPayload = JSON.parse(atob(response.data.split('.')[1]));
    const roles = tokenPayload.roles.split(',');
    if (roles.includes('ROLE_ADMIN')) {
      navigate('/admin-dashboard');
    } else {
      navigate('/user-dashboard');
    }
  } catch (err) {
    setError('Invalid credentials or registration failed');
  }
};

return (
  <div className="container mt-5">
    <h2>{isRegistering ? 'Register' : 'Login'}</h2>
    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label>Username</label>
        <input
          type="text"
          className="form-control"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
      </div>
      <div className="mb-3">
        <label>Password</label>
        <input
          type="password"
          className="form-control"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
      </div>
      {isRegistering && (
        <div className="mb-3">
          <label>Role</label>
          <select
            className="form-control"
            value={role}
            onChange={(e) => setRole(e.target.value)}
          >
            <option value="USER">User</option>
            <option value="ADMIN">Admin</option>
          </select>
        </div>
      )}
    {error && <div className="alert alert-danger">{error}</div>}
    <button type="submit" className="btn btn-primary">
     {isRegistering ? 'Register' : 'Login'}
    </button>
    <button
     type="button"
     className="btn btn-link"
     onClick={() => setIsRegistering(!isRegistering)}
    >
     {isRegistering ? 'Switch to Login' : 'Switch to Register'}
    </button>
   </form>
  </div>
 );
};

export default Login;
```

5. User Dashboard (` src/components/UserDashboard.js`)

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { getTasks, createTask, updateTask, deleteTask } from '../services/api';

const UserDashboard = () => {
 const [tasks, setTasks] = useState([]);
 const [newTask, setNewTask] = useState({ title: '', description: '', completed: false });
 const [editingTask, setEditingTask] = useState(null);
 const navigate = useNavigate();

 useEffect(() => {
  fetchTasks();
 }, []);

 const fetchTasks = async () => {
  try {
   const response = await getTasks();
   setTasks(response.data);
  } catch (err) {
   console.error(err);
  }
 };

 const handleCreate = async () => {
  try {
   await createTask(newTask);
   setNewTask({ title: '', description: '', completed: false });
   fetchTasks();
  } catch (err) {
   console.error(err);
  }
 };


 const handleUpdate = async (id) => {
  try {
   await updateTask(id, editingTask);
   setEditingTask(null);
   fetchTasks();
  } catch (err) {
    console.error(err);
  }
};

const handleDelete = async (id) => {
  try {
    await deleteTask(id);
    fetchTasks();
  } catch (err) {
    console.error(err);
  }
};

const logout = () => {
  localStorage.removeItem('token');
  navigate('/');
};

return (
  <div className="container mt-5">
    <h2>User Dashboard</h2>
    <button className="btn btn-danger mb-3" onClick={logout}>Logout</button>

    <h3>Create Task</h3>
    <div className="mb-3">
      <input
        type="text"
        className="form-control"
        placeholder="Title"
        value={newTask.title}
        onChange={(e) => setNewTask({ ...newTask, title: e.target.value })}
      />
      <input
        type="text"
        className="form-control mt-2"
        placeholder="Description"
        value={newTask.description}
        onChange={(e) => setNewTask({ ...newTask, description: e.target.value })}
      />
      <button className="btn btn-primary mt-2" onClick={handleCreate}>Add Task</button>
    </div>

    <h3>Tasks</h3>
    <table className="table">
      <thead>
        <tr>
          <th>Title</th>
          <th>Description</th>
          <th>Completed</th>
          <th>Actions</th>
        </tr>
      </thead>
    <tbody>
     {tasks.map((task) => (
      <tr key={task.id}>
       {editingTask?.id === task.id ? (
        <>
         <td>
          <input
           type="text"
           value={editingTask.title}
           onChange={(e) => setEditingTask({ ...editingTask, title: e.target.value })}
          />
         </td>
         <td>
          <input
           type="text"
           value={editingTask.description}
           onChange={(e) => setEditingTask({ ...editingTask, description: e.target.value })}
          />
         </td>
         <td>
          <input
           type="checkbox"
           checked={editingTask.completed}
           onChange={(e) => setEditingTask({ ...editingTask, completed: e.target.checked })}
          />
         </td>
         <td>
          <button className="btn btn-success" onClick={() => handleUpdate(task.id)}>Save</button>
          <button className="btn btn-secondary" onClick={() => setEditingTask(null)}>Cancel</button>
         </td>
        </>
       ) : (
        <>
         <td>{task.title}</td>
         <td>{task.description}</td>
         <td>{task.completed ? 'Yes' : 'No'}</td>
         <td>
          <button className="btn btn-info" onClick={() => setEditingTask(task)}>Edit</button>
          <button className="btn btn-danger ms-2" onClick={() => handleDelete(task.id)}>Delete</button>
         </td>
        </>
       )}
      </tr>
     ))}
    </tbody>
   </table>
  </div>
 );
};

export default UserDashboard;
```

6. Admin Dashboard (` src/components/AdminDashboard.js`)

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { getTasks, deleteTaskAdmin } from '../services/api';

const AdminDashboard = () => {
 const [tasks, setTasks] = useState([]);
 const navigate = useNavigate();

 useEffect(() => {
  fetchTasks();
 }, []);

 const fetchTasks = async () => {
  try {
   const response = await getTasks();
   setTasks(response.data);
  } catch (err) {
   console.error(err);
  }
 };

 const handleDelete = async (id) => {
  try {
   await deleteTaskAdmin(id);
   fetchTasks();
  } catch (err) {
   console.error(err);
  }
 };

 const logout = () => {
  localStorage.removeItem('token');
  navigate('/');
 };

 return (
  <div className="container mt-5">
   <h2>Admin Dashboard</h2>
   <button className="btn btn-danger mb-3" onClick={logout}>Logout</button>

   <h3>All Tasks</h3>
   <table className="table">
    <thead>
     <tr>
      <th>Title</th>
      <th>Description</th>
      <th>Completed</th>
      <th>Actions</th>
     </tr>
    </thead>
    <tbody>
     {tasks.map((task) => (
      <tr key={task.id}>
       <td>{task.title}</td>
       <td>{task.description}</td>
       <td>{task.completed ? 'Yes' : 'No'}</td>
       <td>
        <button className="btn btn-danger" onClick={() => handleDelete(task.id)}>Delete</button>
       </td>
      </tr>
     ))}
    </tbody>
   </table>
  </div>
 );
};

export default AdminDashboard;
```

Running the Application

Backend
1. Update `application.yml` with your PostgreSQL credentials.
2. Run the Spring Boot app: `mvn spring-boot:run`.
3. Test endpoints:
   - Register: `POST http://localhost:8080/api/auth/register?username=user&password=user123&role=USER`
   - Login: `POST http://localhost:8080/api/auth/login?username=user&password=user123`

Frontend
1. Start the React app: `npm start`.
2. Open `http://localhost:3000` in your browser.
3. Register or log in, and you'll be redirected to the appropriate dashboard based on the role.

Notes
- **Security**: Use HTTPS in production and a stronger JWT secret.
- **Role-Based Access**: Admin can delete any task, while users can CRUD their tasks.
- **Layered Architecture**: The backend follows Controller → Service → Repository layers.
- **Stateless Authentication**: JWT ensures stateless sessions, validated by the `JwtAuthenticationFilter`.

This setup provides a complete CRUD application with authentication and role-based dashboards, ready for further customization!