

Angular 1-Student Instructor Authentication

#app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { StudentDashboardComponent } from './student-dashboard/student-dashboard.component';
import { InstructorDashboardComponent } from './instructor-dashbaord/instructor-dashbaord.component';
import { AuthGuard } from './auth.guard';
```

```
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  {
    path: 'student-dashboard',
    component: StudentDashboardComponent,
    canActivate: [AuthGuard],
    data: { role: 'student' }
  },
  {
    path: 'instructor-dashbaord',
    component: InstructorDashboardComponent,
    canActivate: [AuthGuard],
    data: { role: 'instructor' }
  }
];
```

#instructor-dashbaord.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-instructor-dashbaord',
  templateUrl: './instructor-dashbaord.component.html',
  styleUrls: ['./instructor-dashbaord.component.css']
})
export class InstructorDashboardComponent {}
```

#login.component.html

```
<div class="login-container">
  <h2>Login</h2>
  <form (ngSubmit)="login()">
    <input type="text" [(ngModel)]="username" name="username"
    placeholder="Username" required />
    <input type="password" [(ngModel)]="password" name="password"
    placeholder="Password" required />
    <button type="submit">Login</button>
  </form>
  <p *ngIf="errorMessage" class="error">{{ errorMessage }}</p>
</div>
```

#login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from './auth.service';
```

```
@Component({
  selector: 'app-login',
```

```
}
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

#auth.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';
import { AuthService } from './auth.service';
```

```
@Injectable({
  providedIn: 'root'
})
```

```
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}
```

```
canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
  if (!this.authService.isLoggedIn()) {
    this.router.navigate(['/login']);
    return false;
  }
}
```

```
templateUrl: './login.component.html',
styleUrls: ['./login.component.css']
})
```

```
export class LoginComponent {
  username: string = '';
  password: string = '';
  errorMessage: string = '';
```

```
constructor(private authService: AuthService, private router: Router) {}
```

```
login() {
  if (this.authService.login(this.username, this.password)) {
    const role = this.authService.getUserRole();
    if (role === 'student') {
      this.router.navigate(['/student-dashboard']);
    } else if (role === 'instructor') {
      this.router.navigate(['/instructor-dashbaord']);
    }
  } else {
    this.errorMessage = 'Invalid username or password';
  }
}
}
```

#student-dashbaord.component.ts

```
import { Component } from '@angular/core';
```

```
@Component({
```

```
const requiredRole = next.data['role'];
if (requiredRole) {
  const userRole = this.authService.getUserRole();
  if (userRole !== requiredRole) {
    this.router.navigate(['/']);
    return false;
  }
}
}
```

#auth.service.ts

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
```

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private isAuthenticated = false;
  private userRole: 'student' | 'instructor' | null = null;
```

```
constructor(private router: Router) {}
```

```
selector: 'app-student-dashboard',
templateUrl: './student-dashboard.component.html',
styleUrls: ['./student-dashboard.component.css']
})
```

```
export class StudentDashboardComponent {}
```

React 2-Loan

#LoanForm.js

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import './App.css';
```

```
const LoanForm = () => {
  const navigate = useNavigate();
  const [formData, setFormData] = useState({
    fullName: '',
    loanAmount: '',
    purpose: 'House',
    tenure: ''
  });
  const [errors, setErrors] = useState({});
```

```
const handleChange = (e) => {
  const { name, value } = e.target;
```

```
login(username: string, password: string): boolean {
  if (username === 'student' && password === 'password') {
    this.isAuthenticated = true;
    this.userRole = 'student';
    return true;
  } else if (username === 'instructor' && password === 'password') {
    this.isAuthenticated = true;
    this.userRole = 'instructor';
    return true;
  }
  return false;
}
```

```
logout(): void {
  this.isAuthenticated = false;
  this.userRole = null;
  this.router.navigate(['/login']);
}
```

```
isLoggedIn(): boolean {
  return this.isAuthenticated;
}
```

```
getUserRole(): 'student' | 'instructor' | null {
  return this.userRole;
}
}
```

```
setFormData({ ...formData, [name]: value });
};
```

```
const handleSubmit = (e) => {
  e.preventDefault();
  const validationErrors = {};
```

```
if (!formData.fullName.trim()) {
  validationErrors.fullName = "Full Name is required";
}
```

```
const amount = parseFloat(formData.loanAmount);
if (!isNaN(amount) || amount < 1000 || amount > 1000000) {
  validationErrors.loanAmount = "Loan Amount must be between 1000 and 1000000";
}
```

```
const tenure = parseInt(formData.tenure);
if (!isNaN(tenure) || tenure < 1 || tenure > 30) {
  validationErrors.tenure = "Tenure must be between 1 and 30 years";
}
```

```
setErrors(validationErrors);
```

```
if (Object.keys(validationErrors).length === 0) {
  navigate("/welcome");
} else {
  navigate("/error");
}
```

```
};
```

```
return (
  <div>
    <h1 className="header">Bank Loan Form</h1>
    <form onSubmit={handleSubmit} className="form">
      <div>
        <label>Full Name:</label>
        <input
          type="text"
          name="fullName"
          value={formData.fullName}
          onChange={handleChange}
        />
        {errors.fullName && <p className="error">{errors.fullName}</p>}
      </div>

      <div>
        <label>Loan Amount:</label>
        <input
          type="number"
          name="loanAmount"
          value={formData.loanAmount}
          onChange={handleChange}
        />
        {errors.loanAmount && <p className="error">{errors.loanAmount}</p>}
      </div>
    </form>
  </div>
);
```

```
import './Dashboard.css';
```

```
function Dashboard() {
  const [totalSales, setTotalSales] = useState(0);
  const [totalCashSales, setTotalCashSales] = useState(0);
  const [totalCreditSales, setTotalCreditSales] = useState(0);
  const [mostSalesBuyer, setMostSalesBuyer] = useState({ buyerName: '', saleTotal: 0 });

  useEffect(() => {
    const fetchSales = async () => {
      const sales = await getSalesData();
      setTotalSales(calculateTotalSales(sales));
      setTotalCashSales(calculateTotalCashSale(sales));
      setTotalCreditSales(calculateTotalCreditSale(sales));
      setMostSalesBuyer(calculateBuyerWithMostSale(sales));
    };

    fetchSales();
  }, []);

  return (
    <div className="dashboard">
      <div className="card">
        <h2>Total Sales</h2>
        <p>{totalSales}</p>
      </div>
      <div className="card">
        <h2>Total Cash Sales</h2>
      </div>
    </div>
  );
}
```

```
<div>
  <label>Purpose of Loan:</label>
  <select
    name="purpose"
    value={formData.purpose}
    onChange={handleChange}
  >
    <option value="House">House</option>
    <option value="Car">Car</option>
    <option value="Personal">Personal</option>
    <option value="Education">Education</option>
  </select>
</div>

<div>
  <label>Tenure (in years):</label>
  <input
    type="number"
    name="tenure"
    value={formData.tenure}
    onChange={handleChange}
  />
  {errors.tenure && <p className="error">{errors.tenure}</p>}
</div>

<button type="submit">Apply</button>
</form>
</div>
```

```
<p>{totalCashSales}</p>
</div>
<div className="card">
  <h2>Total Credit Sales</h2>
  <p>{totalCreditSales}</p>
</div>
<div className="card">
  <h2>Buyer with Most Sales</h2>
  <p>{mostSalesBuyer.buyerName}</p>
  <p>{mostSalesBuyer.saleTotal}</p>
</div>
</div>
);
}

export default Dashboard;
.....
#report.js
import axios from 'axios';

export const getSalesData = async () => {
  let { data } = await axios.get('/sales.json');
  return data;
};

export const calculateTotalSales = (sales) => {
  return sales.reduce((total, sale) => total + sale.saleTotal, 0);
};
```

```
};

};

export default LoanForm;
.....
#app.js
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import LoanForm from './LoanForm';
import WelcomePage from './welcomePage';
import ErrorPage from './errorPage';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<LoanForm />} />
        <Route path="/welcome" element={<WelcomePage />} />
        <Route path="/error" element={<ErrorPage />} />
      </Routes>
    </Router>
  );
}

export default App;
.....
#errorPage.js
import React from 'react';
```

```
export const calculateTotalCashSale = (sales) => {
  return sales
    .filter((sale) => !sale.creditCard)
    .reduce((total, sale) => total + sale.saleTotal, 0);
};

export const calculateTotalCreditSale = (sales) => {
  return sales
    .filter((sale) => sale.creditCard)
    .reduce((total, sale) => total + sale.saleTotal, 0);
};

export const calculateBuyerWithMostSale = (sales) => {
  const buyerSales = sales.reduce((acc, sale) => {
    acc[sale.buyerName] = (acc[sale.buyerName] || 0) + sale.saleTotal;
    return acc;
  }, {});

  const topBuyer = Object.entries(buyerSales).reduce((max, [buyer, total]) => {
    return total > max.saleTotal ? { buyerName: buyer, saleTotal: total } : max;
  }, { buyerName: '', saleTotal: 0 });

  return topBuyer;
};

.....
package com.wecp.library.domain;
```

```
const ErrorPage = () => {
  return (
    <div>
      <h1>Error: Please check your loan application form for valid entries.</h1>
    </div>
  );
};

export default ErrorPage;
.....
#welcomepage.js
import React from 'react';

const WelcomePage = () => {
  return (
    <div>
      <h1>Welcome! Your loan application has been submitted successfully.</h1>
    </div>
  );
};

export default WelcomePage;
.....
React 3-Dashboard Report
.....
#dashboard.js
```

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.io.Serializable;

@Entity
public class Book implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String name;

  public Long getId() {
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }
}
```

```
public void setName(String name) {
    this.name = name;
}
}
```

```
package com.wecp.library.domain;
```

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import reactor.core.publisher.Mono;
```

```
import javax.persistence.*;
import java.io.Serializable;
import java.time.LocalDate;
```

```
@Entity
public class Issue implements Serializable {

    private static final long serialVersionUID = 1L;
```

```


    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
```

```
    private LocalDate issueDate;
```

```
    private LocalDate returnDate;
```

```
    private Integer period;
```

```
REST controller for managing library system process
```

```
*/
```

```
@RestController
```

```
@RequestMapping("/api/v1")
```

```
public class LibraryController {
```

```


    @Autowired
```

```
    private UserRepository userRepo;
```

```


    @Autowired
```

```
    private IssueRepository issueRepo;
```

```
/**
```

```
{@code POST /issue-book} : Create a new issue.
```

```


@param issue the issue to create.
```

```


@return the {@link ResponseEntity} with status {@code 200 (OK)} and with body
```

```
the issue, or throw {@link UserNotSubscribedException} if user is not subscribed.
```

```
*/
```

```
@PostMapping("/issue-book")
```

```
public ResponseEntity<Issue> issueBook(@RequestBody Issue issue) {
```

```
    Optional<User> userOpt = userRepo.findById(issue.getUser().getId());
```

```
    if (userOpt.isPresent()) {
```

```
        User user = userOpt.get();
```

```
        if (user.getSubscribed()) {
```

```
            private Integer fine;
```

```


            @ManyToOne
```

```
            @JsonIgnoreProperties(value = "issues", allowSetters = true)
```

```
            private Book book;
```

```


            @ManyToOne
```

```
            @JsonIgnoreProperties(value = "issues", allowSetters = true)
```

```
            private User user;
```

```


            public Long getId() {
                return id;
            }
        }
    }
}
```

```


            public void setId(Long id) {
                this.id = id;
            }
        }
    }
}
```

```


            public LocalDate getIssueDate() {
                return issueDate;
            }
        }
    }
}
```

```


            public void setIssueDate(LocalDate issueDate) {
                this.issueDate = issueDate;
            }
        }
    }
}
```

```


            public LocalDate getReturnDate() {
```

```

            Issue savedIssue = issueRepo.save(issue);
            return ResponseEntity.ok(savedIssue);
        } else {
            throw new UserNotSubscribedException("User subscription has expired");
        }
    }
    else {
        return ResponseEntity.noContent().build();
    }
}

/**
```

```
{@code POST /user} : Create a new user.
```

```


@param user the user to create.
```

```


@return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the
new user
```

```
*/
```

```
@PostMapping("/user")
```

```
public ResponseEntity<User> createUser(@RequestBody User user) {
```

```
    User savedUser = userRepo.save(user);
```

```
    return ResponseEntity.ok(savedUser);
```

```
}
```

```
/**
```

```

            return returnDate;
        }
    }
}
```

```


            public void setReturnDate(LocalDate returnDate) {
                this.returnDate = returnDate;
            }
        }
    }
}
```

```


            public Integer getPeriod() {
                return period;
            }
        }
    }
}
```

```


            public void setPeriod(Integer period) {
                this.period = period;
            }
        }
    }
}
```

```


            public Integer getFine() {
                return fine;
            }
        }
    }
}
```

```


            public void setFine(Integer fine) {
                this.fine = fine;
            }
        }
    }
}
```

```


            public Book getBook() {
                return book;
            }
        }
    }
}
```

```


            public void setBook(Book book) {
```

```
{@code GET /renew-user-subscription/:id} : Set user subscription to true
```

```


@param id the id of the user to renew subscription.
```

```


@return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the
updated user
```

```
*/
```

```
@GetMapping("/renew-user-subscription/{id}")
```

```
public ResponseEntity<User> renewUserSubscription(@PathVariable Long id) {
```

```
    Optional<User> userOpt = userRepo.findById(id);
```

```
    if (userOpt.isPresent()) {
```

```
        User user = userOpt.get();
```

```
        user.setSubscribed(true);
```

```
        userRepo.save(user);
```

```
        return ResponseEntity.ok(user);
```

```
    } else {
```

```
        return ResponseEntity.noContent().build();
```

```
    }
```

```
}
```

```
}

package com.wecp.library.domain;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
import java.io.Serializable;
```

```


@Entity
```

```

            this.book = book;
        }
    }
}
```

```


            public User getUser() {
                return user;
            }
        }
    }
}
```

```


            public void setUser(User user) {
                this.user = user;
            }
        }
    }
}
```

```
package com.wecp.library.controller;
```

```
import com.wecp.library.controller.exception.UserNotSubscribedException;
```

```
import com.wecp.library.domain.Issue;
```

```
import com.wecp.library.domain.User;
```

```
import com.wecp.library.repository.IssueRepository;
```

```
import com.wecp.library.repository.UserRepository;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import java.util.Optional;
```

```
/**
```

```
public class User implements Serializable {

    private static final long serialVersionUID = 1L;
```

```


    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    private String username;
```

```
    private boolean subscribed = false;
```

```


    public boolean getSubscribed() {
        return subscribed;
    }
}
```

```


    public void setSubscribed(boolean subscribed) {
        this.subscribed = subscribed;
    }
}
```

```


    public Long getId() {
        return id;
    }
}
```

```


    public void setId(Long id) {
        this.id = id;
    }
}
```

```


    public String getUsername() {
```

```

        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
}

package com.weep.library.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class WebSecurityConfigurer extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/api/v1/issue-book").permitAll()

            .anyRequest().authenticated()
            .and().httpBasic();
    }

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

```