~~~

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-dashboard/instructor-dashboard.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-dashboard',
  component: InstructorDashboardComponent,
  canActivate: [AuthGuard],
  data: { role: 'instructor' }
 }
];

@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;
  }

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

  return true;
 }
}

` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `
#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) {}

 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;
 }
}
` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `

#instructor-dashboard.component.ts
import { Component } from '@angular/core';
@Component({
 selector: 'app-instructor-dashboard',
 templateUrl: './instructor-dashboard.component.html',
 styleUrls: ['./instructor-dashboard.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',
 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-dashboard']);
   }
  } else {
   this.errorMessage = 'Invalid username or password';
  }
 }
}
` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `
#student-dashboard.component.ts
import { Component } from '@angular/core';

@Component({
 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;
  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>

    <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>
 );
};

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';

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 "./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>
    <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 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;

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;

  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() {
    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) {
    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;

/**
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()) {
    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);
}

/**

{@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
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.wecp.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.WebSecurityConfigu
rerAdapter;
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();
   }
}

` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `
export class Employee {
 empId: number = 0;
 empName: string = "";
 empSalary: number = 0;

 constructor(empId: number, empName: string, empSalary: number) {
  this.empId = empId;
  this.empName = empName;
  this.empSalary = empSalary;
 }

 displayDetails(): boolean {
  console.log(` Employee Details:
Employee ID: ${this.empId}
Employee Name: ${this.empName}
Employee Salary: ${this.empSalary}`);
  return true;
 }
}

// Example usage:
let emp: Employee = new Employee(1, 'John Doe', 50000);
emp.displayDetails();

// Define two arrays: one for names and another for marks
const names: string[] = ["A", "B"];
const marks: number[] = [10, 20];

// Display names and marks using a for loop
console.log("Student Names and Marks");
for (let i = 0; i < names.length; i++) {
 console.log(` ${names[i]}: ${marks[i]}`);
}

// Function to calculate average
export function findAvg(marks: number[]): number {
 let tot = 0;
 for (let i = 0; i < marks.length; i++) {
  tot += marks[i];
 }
 const averageMarks = tot / marks.length;
 return averageMarks;
}

// Display the average
console.log(` \nAverage Marks: ${findAvg(marks)}`);

` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `
// Function to generate a random number between min and max
const getRandomNumber = (min, max) => Math.floor(Math.random() * (max - min + 1)) +
min;

// Function to create an array of 10 random numbers between 1 and 100
const createRandomNumbersArray = () => {
 const numbers = [];
 for (let i = 0; i < 10; i++) {
  numbers.push(getRandomNumber(1, 100));
 }
 return numbers;
};

// Arrow function to calculate the sum of array values
const calculateSum = (arr) => arr.reduce((sum, num) => sum + num, 0);

// Arrow function to calculate average
const calculateAverage = (arr) => calculateSum(arr) / arr.length;

// Function to print array elements using an iterator
const printArrayElements = (arr) => {
 console.log('Generated Array:', arr);
 console.log('Elements:');
 const iterator = arr[Symbol.iterator]();
 let index = 1;
 let result = iterator.next();
 while (!result.done) {
  console.log(` Element ${index}: ${result.value}`);
  result = iterator.next();
  index++;
 }
};

// Exporting all necessary modules
module.exports = {
 getRandomNumber,
 calculateSum,
 calculateAverage,
 printArrayElements,
 createRandomNumbersArray
};
` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `
const products = [
  { id: 1, name: "Product 1", price: 10 },
  { id: 2, name: "Product 2", price: 20 },
  { id: 3, name: "Product 3", price: 30 }
];

const shoppingCart = {
  items: [],
  coupon: null,

  // Add a product to the cart
  addToCart: function(productId, quantity) {
    const product = products.find(p => p.id === productId);
    if (!product) {
      console.log("Product not found.");
      return;
    }

    const existingItem = this.items.find(item => item.product.id === productId);
    if (existingItem) {
      existingItem.quantity += quantity;
    } else {
      this.items.push({ product, quantity });
    }
    console.log(` ${product.name} added to cart (x${quantity}).`);
  },

  // View current items in cart
  viewCart: function() {
    console.log("Cart Contents:");
    if (this.items.length === 0) {
      console.log("Cart is empty.");
      return;
    }
    this.items.forEach(item => {
      console.log(` ${item.product.name} - Quantity: ${item.quantity} - Price:
$${item.product.price}`);
    });
  },

  // Apply a discount coupon
  applyCoupon: function(couponCode) {
    if (this.coupon) {
      console.log("A coupon has already been applied.");
      return;
    }
    if (couponCode === "DISCOUNT10") {
      this.coupon = "DISCOUNT10";
      console.log("Coupon applied: DISCOUNT10");
    } else {
      console.log("Invalid coupon code.");
    }
  },

  // Calculate total amount payable
  calculateTotalAmount: function() {
    let totalAmount = 0;
    this.items.forEach(item => {
      totalAmount += item.product.price * item.quantity;
    });

    if (this.coupon === "DISCOUNT10") {
      totalAmount *= 0.9; // 10% off
      console.log("Coupon Applied: 10% discount");
    }

    console.log(` Total payable amount: $${totalAmount.toFixed(2)}`);
  }
};

// Example usage:
shoppingCart.addToCart(1, 2);   // Adds 2 units of Product 1
shoppingCart.addToCart(2, 1);   // Adds 1 unit of Product 2
shoppingCart.viewCart();   // Displays current cart
shoppingCart.applyCoupon("DISCOUNT10");   // Applies 10% discount
shoppingCart.calculateTotalAmount();   // Shows final total

module.exports = shoppingCart;
` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` ` `
function solve(N, M) {
return N % M;
}
const N = BigInt(gets().trim());
const M = BigInt(gets().trim());
const result = solve(N, M);
print(result);
~~~