Problem Statement-1
E-commerce Product Catalog

ProductCatalogApplication.java

```
package com.example.catalog;

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

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

entity/Product.java

```
package com.example.catalog.entity;

import jakarta.persistence.*;

@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long productId;

   private String name;
   private String description;
   private double price;
   private String category;
   private int stockQuantity;

   // Getters and Setters
}
```

repository/ProductRepository.java

```
package com.example.catalog.repository;

import com.example.catalog.entity.Product;
```

```
 import org.springframework.data.jpa.repository.JpaRepository; import
org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long>
   { List<Product> findByCategory(String category);
   List<Product> findByPriceBetween(double min, double max);
   List<Product> findByNameContainingIgnoreCase(String name);

   @Query("SELECT p FROM Product p WHERE " +
         "(:category IS NULL OR p.category = :category) AND "
         + "(:minPrice IS NULL OR p.price >= :minPrice) AND " +
         "(:maxPrice IS NULL OR p.price <= :maxPrice)")
   List<Product> filter(String category, Double minPrice, Double maxPrice);
}
```

controller/ProductController.java

```
package com.example.catalog.controller;

import com.example.catalog.entity.Product;
import com.example.catalog.repository.ProductRepository; 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/products")
public class ProductController {

   @Autowired
   private ProductRepository repo;

   @GetMapping
   public List<Product> getAll() {
      return repo.findAll();
   }

   @PostMapping
   public Product create(@RequestBody Product p) {
```

```
        return repo.save(p);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Product> update(@PathVariable Long id, @RequestBody Product p)
{
        return repo.findById(id).map(existing -> {
            existing.setName(p.getName());
            existing.setDescription(p.getDescription());
            existing.setPrice(p.getPrice());
            existing.setCategory(p.getCategory());
            existing.setStockQuantity(p.getStockQuantity());
            return ResponseEntity.ok(repo.save(existing));
        }).orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id)
        { return repo.findById(id).map(p -> {
            repo.delete(p);
            return ResponseEntity.ok().build();
        }).orElse(ResponseEntity.notFound().build());
    }

    @GetMapping("/search")
    public List<Product> searchByName(@RequestParam String name) {
        return repo.findByNameContainingIgnoreCase(name);
    }

    @GetMapping("/filter")
    public List<Product> filter(
        @RequestParam(required = false) String category,
        @RequestParam(required = false) Double minPrice,
        @RequestParam(required = false) Double maxPrice
    ) {
        return repo.filter(category, minPrice, maxPrice);
    }
}
```

config/SecurityConfig.java

```
package com.example.catalog.config;
```

```
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.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
      { http
         .csrf().disable()
         .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/products").hasAnyRole("ADMIN", "USER")
            .requestMatchers("/api/products/**").hasRole("ADMIN")
            .anyRequest().authenticated()
         )
         .httpBasic();
      return http.build();
   }
}
```

resources/application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/product_catalog

spring.datasource.username=root

spring.datasource.password=yourpassword spring.jpa.hibernate.ddl-

auto=update spring.jpa.show-sql=true
```

Description :

The E-commerce Product Catalog is a Spring Boot-based RESTful application designed to manage products in an online store. At the core of the application is the Product entity, representing individual catalog items with fields such as productId, name, description, price, category, and stockQuantity. These products are persisted using Spring Data JPA and stored in a MySQL database, with the repository layer (ProductRepository) handling operations like finding by category, price range, and partial name matches. A custom query is also provided to filter products by optional parameters such as category and price limits, enhancing user experience with flexible search functionality. The ProductController serves as the REST layer, exposing endpoints to perform all standard CRUD operations and additional search and filter capabilities. The API allows all users (roles ADMIN and USER) to view products, but restricts creation, update, and deletion operations to users with the ADMIN role only. This role-based access control is configured using SecurityConfig, which uses Spring Security to define authorization rules and enables HTTP Basic authentication. To secure communication and user roles, CSRF protection is disabled for simplicity, but this can be adapted for frontend integration

if needed. The application is configured via application.properties, which sets up MySQL connectivity and enables automatic schema updates and SQL logging. Overall, this project is structured with separation of concerns, follows best practices, and provides a strong foundation for a production-grade e-commerce catalog service.

Problem Statement-2
Library Management System

LibraryManagementSystemApplication.java

```
package com.example.library;

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

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

entity/Book.java

```
package com.example.library.entity;

import jakarta.persistence.*;

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long bookId;
    private String title;
    private String author;
    private String category;
    private boolean availability;
```

```
    // Getters and Setters
}
```

entity/User.java

```
package com.example.library.entity;

import jakarta.persistence.*;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String membershipType;

    // Getters and Setters
}
```

entity/Transaction.java

```
package com.example.library.entity;

import jakarta.persistence.*;
import java.time.LocalDate;

@Entity
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long transactionId;

    @ManyToOne
    private Book book;

    @ManyToOne
    private User user;

    private LocalDate issueDate;
    private LocalDate returnDate;
    private String status;

    // Getters and Setters
```

```
}
```

repository/BookRepository.java

```
package com.example.library.repository;

import com.example.library.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByTitleContainingIgnoreCase(String title);
    List<Book> findByAuthorContainingIgnoreCase(String author);
    List<Book> findByCategoryContainingIgnoreCase(String category);
}
```

repository/UserRepository.java

```
package com.example.library.repository;

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

public interface UserRepository extends JpaRepository<User, Long>
{ }
```

repository/TransactionRepository.java

```
package com.example.library.repository;

import com.example.library.entity.Transaction;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface TransactionRepository extends JpaRepository<Transaction, Long>
    { List<Transaction> findByStatus(String status);
    List<Transaction> findByUserUserId(Long userId);
}
```

config/SecurityConfig.java

```
package com.example.library.config;

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.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
      { http
         .authorizeHttpRequests(auth -> auth.anyRequest().authenticated())
         .formLogin()
         .and()
         .httpBasic();
      return http.build();
   }
}
```

resources/application.properties

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

Description :

The approach taken to build the Library Management System is a well-structured and optimistic strategy that leverages modern Spring Boot capabilities for streamlined development and maintenance. At its core, the application is designed to simplify and automate key library operations such as book lending, returns, inventory tracking, and user management. By using Spring Data JPA with an in-memory H2 database, the system enables fast prototyping and testing while ensuring that all entities—Book, User, and Transaction—are persistently and relationally managed with minimal configuration. The implementation ensures a clear separation of concerns: entities represent data models, repositories handle database operations, and security is enforced through Spring Security to protect access and authenticate librarian users. Optimistically, this foundation allows librarians to perform all necessary CRUD operations and gain insight into lending history, overdue reports, and book availability in real time. Additionally, advanced search capabilities are enabled through repository query methods, allowing filtering by title, author, or category—greatly improving user experience. The application is scalable and easily extendable, whether for integrating with a front-end UI or switching to a production-grade database. Altogether, this architecture provides a robust, secure, and efficient solution for managing library workflows with confidence in future adaptability.

Problem Statement-3
Student Management System

entity/Student.java

```
package com.example.sms.entity;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;

@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long studentId;

    @NotBlank(message = "Name is required")
    private String name;

    @Min(value = 5, message = "Age must be at least 5")
    @Max(value = 100, message = "Age must be at most 100")
    private int age;

    @NotBlank(message = "Class is required")
    private String studentClass;

    @NotBlank(message = "Email is required")
    @Email(message = "Invalid email format")
    private String email;

    @NotBlank(message = "Address is required")
    private String address;

    // Getters and Setters
}
```

StudentController.java

```
package com.example.sms.controller;

import com.example.sms.entity.Student;
```

```
import com.example.sms.repository.StudentRepository;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Controller; import
org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
public class StudentController {

    @Autowired
    private StudentRepository studentRepo;

    @GetMapping("/")
    public String home(Model model, @RequestParam(defaultValue = "0") int page)
        { Page<Student> students = studentRepo.findAll(PageRequest.of(page, 5));
        model.addAttribute("students", students);
        return "index";
    }

    @GetMapping("/add")
    public String addStudentForm(Model model) {
        model.addAttribute("student", new Student());
        return "add-student";
    }

    @PostMapping("/save")
    public String saveStudent(@Valid Student student, BindingResult result, Model model)
        { if (result.hasErrors()) {
            return "add-student";
        }
        studentRepo.save(student);
        return "redirect:/";
    }

    @GetMapping("/edit/{id}")
    public String editStudent(@PathVariable Long id, Model model) {
        model.addAttribute("student", studentRepo.findById(id).orElseThrow());
        return "add-student";
```

```
    }

    @GetMapping("/delete/{id}")
    public String deleteStudent(@PathVariable Long id)
        { studentRepo.deleteById(id);
        return "redirect:/";
    }

    @GetMapping("/search")
    public String search(@RequestParam String keyword, Model model) {
        List<Student> students = studentRepo.findByNameContainingIgnoreCase(keyword);
        model.addAttribute("students", students);
        return "index";
    }
}
```

Repository:
StudentRepository.java

```
package com.example.sms.repository;

import com.example.sms.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface StudentRepository extends JpaRepository<Student, Long> {
    List<Student> findByNameContainingIgnoreCase(String name); List<Student>
    findByStudentClassContainingIgnoreCase(String studentClass);
}
```

StudentManagementSystemApplication.java

```
package com.example.sms;

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

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

}

application.properties

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

Description :
The Student Management System is designed with an optimistic and scalable approach that emphasizes simplicity, usability, and maintainability. By leveraging Spring Boot**,** Spring Data JPA**,** and H2 in-memory database, the system ensures smooth CRUD operations for student records with minimal configuration. The use of Thymeleaf templates enables a user-friendly web interface where administrators can easily add, update, delete, view, and search student data. Field validations using annotations such as @Email, @NotBlank, and @Min/@Max ensure data integrity right at the input level, reducing errors and enhancing reliability. Pagination is implemented to handle large datasets efficiently, while search functionality by student name or class improves data accessibility. The modular controller logic, clear separation of concerns, and meaningful error handling contribute to a maintainable codebase that can be extended or migrated to other databases in the future. This optimistic and thoughtfully structured approach not only meets all project requirements but also lays the foundation for enhancements like role-based access, export features, or integration with external systems.

Problem Statement-4
Employee Attendance Tracker

EmployeeAttendanceTrackerApplication.java

```
package com.example.attendance;

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

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

entity/Employee.java

```
package com.example.attendance.entity;

import jakarta.persistence.*;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;

    private String name;
    private String department;
    private String designation;

    // Getters and Setters
}
```

entity/Attendance.java

```
package com.example.attendance.entity;

import jakarta.persistence.*;
import java.time.LocalDate;

@Entity
public class Attendance {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long attendanceId;

    @ManyToOne
    private Employee employee;

    private LocalDate date;
    private String status; // Present or Absent

    // Getters and Setters
    }
```

```
package com.example.attendance.entity;

import jakarta.persistence.*;
import java.time.LocalDate;

@Entity
public class Attendance {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long attendanceId;

    @ManyToOne
    private Employee employee;

    private LocalDate date;
    private String status; // Present or Absent

    // Getters and Setters
}
```

repository/EmployeeRepository.java

```
package com.example.attendance.repository;

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

public interface EmployeeRepository extends JpaRepository<Employee, Long>
{ }
```

repository/AttendanceRepository.java

```
package com.example.attendance.repository;

import com.example.attendance.entity.Attendance;
import com.example.attendance.entity.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import java.time.LocalDate;
import java.util.List;
public interface AttendanceRepository extends JpaRepository<Attendance, Long> {
List<Attendance> findByEmployee(Employee employee);
    List<Attendance> findByDate(LocalDate date);
}
```

config/SecurityConfig.java

```
package com.example.attendance.config;

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.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
        { http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/reports/**").hasRole("MANAGER")
                .requestMatchers("/api/attendance/**").hasAnyRole("EMPLOYEE", "MANAGER")
                .anyRequest().authenticated()
            ).formLogin()
            .and()
            .httpBasic();
        return http.build();
    }
}
```

resources/application.properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/attendance_db
spring.datasource.username=postgres
```

spring.datasource.password=yourpassword spring.jpa.hibernate.ddl-

auto=update spring.jpa.show-sql=true

spring.sql.init.mode=always

Description :

The Employee Attendance Tracker is built with a modern and optimistic approach, focusing on simplicity, scalability, and security. Leveraging Spring Boot**,** Spring Data JPA, the system ensures robust data persistence and smooth tracking of daily attendance. The application supports a clean separation of concerns with well-defined entities for Employee and Attendance, enabling precise tracking and reporting. Role-based access control through Spring Security ensures that employees can securely mark their attendance while managers have exclusive access to comprehensive reports. RESTful API endpoints make integration with front-end interfaces or third-party systems straightforward, while thoughtful database design supports efficient queries for individual and department-wide summaries. The code structure is modular, extensible, and adheres to best practices, allowing for future enhancements like leave management, biometric integration, or real-time dashboards. Overall, this implementation presents a forward-looking foundation for a dependable and secure attendance tracking system tailored to organizational needs.