Name: Smruti Jagtap
RollNo: 198

DAY 4

```
package com.calc.pages;

import java.io.IOException;
import java.io.PrintWriter;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;


@WebServlet("/calculate")
public class CalculatorServlet extends HttpServlet {


	protected void doGet(HttpServletRequest request, HttpServletResponse
response)
			throws ServletException, IOException {

		// 1. set cont type
		response.setContentType("text/html");

		// 2 get writer - PW
		try (PrintWriter pw = response.getWriter()) {
			// 3 get req params

			int number1 =
Integer.parseInt(request.getParameter("num1"));
			int number2 =
Integer.parseInt(request.getParameter("num2"));

			String operation = request.getParameter("action");
```

```
double result = 0;

switch (operation) {
case "add":
        result = number1 + number2;
        break;
case "subtract":
        result = number1 - number2;
        break;
case "multiply":
        result = number1 * number2;
        break;
case "divide":
        result = number1 / number2;
        break;

default:
        break;

}

Cookie cookie;


if(result >=0) {

        cookie = new Cookie("CalCookie","Result="
                        + ""+String.valueOf(result));
        response.addCookie(cookie);
        response.sendRedirect("Result1Servlet");
}
else {

        cookie = new Cookie("CalCookie","Result="
                        + ""+String.valueOf(result));
        response.addCookie(cookie);
        response.sendRedirect("Result2Servlet");
}
```

```
//                      Cookie cookie = new Cookie("CalCookie","Result= "
//                                      + ""+result);
//                      response.addCookie(cookie);
                        pw.print("<h5>Result of "+operation+" is -
"+result+"</h5>");
                }
        }

}


package com.calc.pages;

import java.io.IOException;
import java.io.PrintWriter;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;


@WebServlet("/Result1Servlet")
public class Result1Servlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                response.setContentType("text/html");

                try(PrintWriter pw = response.getWriter()){

                        Cookie[] cookies = request.getCookies();

                        if(cookies != null) {
                                String value = cookies[0].getValue();
                                pw.print("Result is >= 0 " + value);
                        }
```

```
		}catch(Exception e) {
			e.printStackTrace();
		}
	}

}
```

```
package com.calc.pages;

import java.io.IOException;
import java.io.PrintWriter;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;


@WebServlet("/Result2Servlet")
public class Result2Servlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		response.setContentType("text/html");

		try(PrintWriter pw = response.getWriter()){

			Cookie[] cookies = request.getCookies();

			if(cookies != null) {
				String value = cookies[0].getValue();
				pw.print("Result is < 0 "+value);
			}

		}catch(Exception e) {
			e.printStackTrace();
```

```
		}
	}
}
```

DAY 10

```
package com.products.tester;

import static com.products.utils.HibernateUtils.getSessionFactory;

import java.time.LocalDate;
import java.util.List;
import java.util.Scanner;

import org.hibernate.SessionFactory;

import com.products.dao.ProductDao;
import com.products.dao.ProductDaoImpl;
import com.products.entities.Category;
import com.products.entities.Product;

public class TestProduct {

	public static void main(String[] args) {

		try(Scanner sc = new Scanner(System.in);
				SessionFactory sf = getSessionFactory()){

			ProductDao productDoa = new ProductDaoImpl();

			// add new product
			// pen 2024-02-15 20 50 stationary
			// coconut_oil 2025-10-26 155 40 oil
			// marker 2025-07-29 50 20 stationary

			System.out.println("Enter name, manufactureDate, price,
availableQuantity, Category: ");
			String msg = productDoa.addProduct(new
Product(sc.next(), LocalDate.parse(sc.next()), sc.nextDouble(), sc.nextInt(),
Category.valueOf(sc.next().toUpperCase())));
```

```
                System.out.println(msg);

                // display product details by id
                System.out.println("Enter product id to get details:");


        System.out.println(productDoa.displayProductDetails(sc.nextLong()));

                // display id , name , price  of all the products manufactured
before specified date and from specific category

                System.out.println("Enter date and category to get product
details:");
                //2024-03-15 stationary


        productDoa.getProductOfSpecificCategoryAndMfgDate(LocalDate.pars
e(sc.next()), Category.valueOf(sc.next().toUpperCase()))
                .forEach( p->System.out.println("Id : " + p.getId()+ " Product
name: " +p.getName()+ " Price: " +p.getPrice()));



                // change product price

                System.out.println("Enter product name and price to
change:");
                String message =
productDoa.changeProductPrice(sc.next(), sc.nextDouble());

                System.out.println(message);

                // apply discount

                System.out.println("Enter quantity and discount percentage
to apply:");

                String msg1 = productDoa.applyDiscount(sc.nextInt(),
sc.nextDouble());
                System.out.println(msg1);
            }
```

```
	}
}
package com.products.tester;

import static com.products.utils.HibernateUtils.getSessionFactory;

import java.time.LocalDate;
import java.util.List;
import java.util.Scanner;

import org.hibernate.SessionFactory;

import com.products.dao.ProductDao;
import com.products.dao.ProductDaoImpl;
import com.products.entities.Category;
import com.products.entities.Product;

public class TestProduct {

	public static void main(String[] args) {

		try(Scanner sc = new Scanner(System.in);
				SessionFactory sf = getSessionFactory()){

			ProductDao productDoa = new ProductDaoImpl();

			// add new product
			// pen 2024-02-15 20 50 stationary
			// coconut_oil 2025-10-26 155 40 oil
			// marker 2025-07-29 50 20 stationary

			System.out.println("Enter name, manufactureDate, price,
availableQuantity, Category: ");
			String msg = productDoa.addProduct(new
Product(sc.next(), LocalDate.parse(sc.next()), sc.nextDouble(), sc.nextInt(),
Category.valueOf(sc.next().toUpperCase())));

			System.out.println(msg);
```

```
                    // display product details by id
                    System.out.println("Enter product id to get details:");


        System.out.println(productDoa.displayProductDetails(sc.nextLong()));

                    // display id , name , price  of all the products manufactured
before specified date and from specific category

                    System.out.println("Enter date and category to get product
details:");
                    //2024-03-15 stationary


        productDoa.getProductOfSpecificCategoryAndMfgDate(LocalDate.pars
e(sc.next()), Category.valueOf(sc.next().toUpperCase()))
                    .forEach( p->System.out.println("Id : " + p.getId()+ " Product
name: " +p.getName()+ " Price: " +p.getPrice()));



                    // change product price

                    System.out.println("Enter product name and price to
change:");
                    String message =
productDoa.changeProductPrice(sc.next(), sc.nextDouble());

                    System.out.println(message);

                    // apply discount

                    System.out.println("Enter quantity and discount percentage
to apply:");

                    String msg1 = productDoa.applyDiscount(sc.nextInt(),
sc.nextDouble());
                    System.out.println(msg1);
                }

        }
```

```
}

package com.products.entities;

public enum Category {
	STATIONARY,SHOES,GRAINS,OIL;
}


package com.products.dao;

import java.time.LocalDate;
import java.util.List;

import com.products.entities.Category;
import com.products.entities.Product;

public interface ProductDao {

	String addProduct(Product product);

	String displayProductDetails(Long productId);

	List<Product> getProductOfSpecificCategoryAndMfgDate(LocalDate
mfgDate, Category category);


	String applyDiscount(int quantity, double discountPercentage);

	String changeProductPrice(String productName, double updatedPrice);
}

package com.products.dao;

import static com.products.utils.HibernateUtils.*;

import java.time.LocalDate;

import java.util.List;
```

```
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.products.entities.Category;
import com.products.entities.Product;

public class ProductDaoImpl implements ProductDao {

	@Override
	public String addProduct(Product product) {

		// get session
		Session session = getSessionFactory().getCurrentSession();

		// begin transaction
		Transaction tx = session.beginTransaction();

		try {

			session.persist(product);

			tx.commit();
		}
		catch(RuntimeException e) {

			if(tx != null)
				tx.rollback();

			throw e;
		}
		return "Product added...";
	}

	@Override
	public String displayProductDetails(Long productId) {

			Product product = null;

			// get session
```

```
                Session session =
getSessionFactory().getCurrentSession();

                // begin transaction
                Transaction tx = session.beginTransaction();

                try {

                        product = session.find(Product.class, productId);

                        if(product != null) {
                                return product.toString();
                        }

                }
                catch(RuntimeException e) {

                        if(tx != null)
                                tx.rollback();
                        throw e;
                }

                return "Product not found!!!";
        }

        @Override
        public List<Product>
getProductOfSpecificCategoryAndMfgDate(LocalDate date, Category
category) {

                List<Product> products = null;
                // get session
                Session session = getSessionFactory().getCurrentSession();

                // begin transaction
                Transaction tx = session.beginTransaction();

                String jpql = "select new com.products.entities.Product(p.id,
p.name, p.price) from Product p where p.manufactureDate <:date and
p.category =:cat";
```

```
		try {

			products = session.createQuery(jpql, Product.class)
					.setParameter("date", date)
					.setParameter("cat", category)
					.getResultList();

			tx.commit();
		}
		catch(RuntimeException e) {

			if(tx != null)
				tx.rollback();

			throw e;
		}

		return products;
	}

	@Override
	public String changeProductPrice(String productName, double
updateedPrice) {

		Product product=null;

		// get session
		Session session = getSessionFactory().getCurrentSession();

		// begin transaction
		Transaction tx = session.beginTransaction();

		String jpql = "select p from Product p where p.name =:pname";


		try {

			 product = session.createQuery(jpql, Product.class)
					.setParameter("pname", productName)
```

```
                                .getSingleResult();

                        // set new price
                        product.setPrice(updateedPrice);

                        tx.commit();

                }
                catch(RuntimeException e) {

                        if(tx != null)
                                tx.rollback();

                        throw e;
                }
                return "Password changed...";
        }

        @Override
        public String applyDiscount(int quantity, double discountPercentage) {


                // get session
                Session session = getSessionFactory().getCurrentSession();

                // begin transaction
                Transaction tx = session.beginTransaction();

                String jpql = "update Product p set p.price = p.price -(p.price *
:disPer/100) where p.availableQuantity >:qty ";

                try {

                        int rowCount = session.createMutationQuery(jpql)
                                        .setParameter("disPer", discountPercentage)
                                        .setParameter("qty", quantity)
                                        .executeUpdate();

                        tx.commit();
```

```
			}
			catch(RuntimeException e) {

				if(tx != null)
					tx.rollback();

				throw e;
			}

			return "Applied Discount successfully....";
		}

}
```

```
package com.products.utils;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;

public class HibernateUtils {
	private static SessionFactory sessionFactory;
	static {
		System.out.println("in static init block");
		sessionFactory=new
Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}

}
```

DAY 16

```
package com.cdac.controller;

import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.cdac.entities.Restaurant;
import com.cdac.services.RestaurantService;

@RestController
@RequestMapping("/restaurants")
public class RestaurantController {

	@Autowired
	private RestaurantService service;

	public RestaurantController() {
		System.out.println("in contr " +getClass());
	}

	//URL - http://host:port/restaurants
	// response - list of restaurants
	@GetMapping()
	public ResponseEntity<?> getAllAvailableRestaurants(){

		List<Restaurant> allAvailableRestaurants =
service.getAllAvailableRestaurants();

		if(allAvailableRestaurants.isEmpty()) {
			//In case of no available Restaurant- Status code 204 only
			// instead sending empty list

			return ResponseEntity.status(HttpStatus.NO_CONTENT)
					.build();
		}
```

```
		// returns status code 200 and list as response body
		return ResponseEntity.ok(allAvailableRestaurants);
	}

	// add new Restaurant
	// URL - http://host:port/restaurants, method = post
	// response message - success/ failure

	@PostMapping()
	public ResponseEntity<?> addNewRestaurant(@RequestBody
Restaurant newRestaurant){

		System.out.println("\n\t in add new Restaurant..."
+newRestaurant);

		try {

			String msg = service.addNewRestaurant(newRestaurant);
			// status code - 201, body - success msg
			return ResponseEntity.status(HttpStatus.CREATED)
					.body(msg);
		}
		catch(RuntimeException e) {

			System.out.println("Error: " +e);

			// status code - 400, body - error msg
			return ResponseEntity.status(HttpStatus.BAD_REQUEST)
					.body(e.getMessage());
		}


	}

	// delete existing Restaurant
	// URL - http://host:port/restaurants/{restaurantId}, method - delete
	// response - success ---> status code - 200 and message
	// failure ----> status code - 404
```

```
	//{restaurantId} is template variable
	@DeleteMapping("/{restaurantId}")
	public ResponseEntity<?> deletRestaurantDetails(@PathVariable Long
restaurantId){

		System.out.println("in delete details" +restaurantId);

		try {
			return
ResponseEntity.ok(service.deletRestaurantDetails(restaurantId));

		}
		catch(RuntimeException e) {
			return ResponseEntity.notFound()
					.build();
		}

	}


}

package com.cdac.services;

import java.util.List;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.cdac.entities.Restaurant;

@Service
@Transactional
public interface RestaurantService {

	// Get All available Restaurants.
	List<Restaurant> getAllAvailableRestaurants();

	String addNewRestaurant(Restaurant newRestaurant);
```

```
	String deletRestaurantDetails(Long restaurantId);
}

package com.cdac.services;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.cdac.dao.RestaurantDao;
import com.cdac.entities.Restaurant;
import com.cdac.exceptions.ResourceExistsException;
import com.cdac.exceptions.ResourceNotFoundException;

@Service
@Transactional
public class RestautantServiceImpl implements RestaurantService {

	@Autowired
	private RestaurantDao dao;

	@Override
	public List<Restaurant> getAllAvailableRestaurants() {

		return dao.findAllByStatusTrue();
	}

	@Override
	public String addNewRestaurant(Restaurant newRestaurant) {

		// check if the Restaurant with same name already exists
		if(dao.existsByName(newRestaurant.getName())) {
			// failure -  duplicate Restaurant name
			throw new ResourceExistsException("Restaurant with
same name already exists!!!!");

		}
```

```
		// set status to true
		newRestaurant.setStatus(true);

		// save the new Restaurant
		Restaurant restaurant = dao.save(newRestaurant);

		return "Restaurant added with id " +restaurant.getId();
	}

	@Override
	public String deletRestaurantDetails(Long restaurantId) {

		Restaurant rest = dao.findById(restaurantId)
				.orElseThrow(() -> new
ResourceNotFoundException("Restaurant not found with given id"));

		rest.setStatus(false);

		return "Restaurant status set to unavailable";
	}//

}

package com.cdac.dao;
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.cdac.entities.Restaurant;

@Repository
public interface RestaurantDao extends JpaRepository<Restaurant, Long> {

	// get all Restaurant with status true
	List<Restaurant>  findAllByStatusTrue();

	// check if the Restaurant with same name already exists - derived
query
```

```
	boolean existsByName(String name);

	// save Restaurant entity
	Restaurant save(Restaurant newRestaurant);

}
```

```
package com.cdac.entities;

import java.time.LocalDate;
import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import jakarta.persistence.Column;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

/*
 * Declares a common base class , w/o any table associated with it.
 * Add common fields here.
 */
@MappedSuperclass
@Getter
@Setter
@ToString
public abstract class BaseEntity {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;

	@CreationTimestamp
	private LocalDate creationDate;
```

```
	@UpdateTimestamp
	private LocalDateTime lastUpdated;


}

package com.cdac.entities;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Table;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Entity
@Table(name = "restaurants")
@NoArgsConstructor
@Getter
@Setter
@ToString(callSuper = true)
public class Restaurant extends BaseEntity {

	@Column(length = 100, unique = true)
	private String name;

	private String address;
	@Column(length = 20)
	private String city;

	private String description;
	private boolean status;

	public Restaurant(String name, String address, String city, String
description) {

		this.name = name;
		this.address = address;
		this.city = city;
```

```
		this.description = description;
		this.status=true;
	}

}
```

```
package com.cdac.exceptions;

public class ResourceExistsException extends RuntimeException {


	public ResourceExistsException(String msg) {
		super(msg);
	}
}
```

```
package com.cdac.exceptions;

public class ResourceNotFoundException extends RuntimeException {

	public ResourceNotFoundException(String msg) {
		super(msg);
	}
}
```

HEALTHCARE CASE STUDY

```
package com.healthcare;

import org.modelmapper.Conditions;
import org.modelmapper.ModelMapper;
import org.modelmapper.convention.MatchingStrategies;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
//one of the annotations - @SpringBootConfiguration => it's Spring boot config class , where you can add @Bean methods to declare spring beans
```

```
public class HealtcareBackendApplication {

	public static void main(String[] args) {
		SpringApplication.run(HealtcareBackendApplication.class, args);
	}

	// configure ModelMapper class as a spring bean
	@Bean // exactly equivalent to - <bean id...../>
	ModelMapper modelMapper() {
		ModelMapper mapper = new ModelMapper();
		mapper.getConfiguration() // get default config
				.setPropertyCondition(Conditions.isNotNull()) //
transfer only not null props from src-> dest
				.setMatchingStrategy(MatchingStrategies.STRICT);//
transfer the props form src -> dest which match by

				// name & data type

		return mapper;
	}

}

package com.healthcare.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.healthcare.dto.ApiResponse;
import com.healthcare.dto.AuthRequest;
import com.healthcare.service.UserService;

import jakarta.validation.Valid;
import lombok.AllArgsConstructor;

@RestController
```

```
@RequestMapping("/users")
@AllArgsConstructor
public class UserController {
	//depcy
	private final UserService userService;
	/*
	 * 1. Patient Login / Doctor Login(User Login)  common
- Controller
 - UserController
URL - http://host:port/users/signin
Method - POST  (for security , JWT generation, JSON payload)
Eg . Patient Logs in
Payload - email , password  (Auth Request DTO)
Success Resp -Sc 200 |201  Auth Resp DTO (user id ,name, email , role ,
message)
Failure Resp - SC 401 ApiResp DTO(status : succes | failure , timestamp ,
message)

	 */
	@PostMapping("/signin")
	public /* @ResponseBody */ ResponseEntity<?>
userAuthentication(@RequestBody @Valid AuthRequest dto)
	{
		System.out.println("in user auth "+dto);
		//invoker service layer method

			return ResponseEntity.ok(userService.signIn(dto));

	}


}

package com.healthcare.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.healthcare.dto.ApiResponse;
import com.healthcare.dto.BookAppointment;
import com.healthcare.dto.PatientRegDTO;
import com.healthcare.service.AppointmentService;
import com.healthcare.service.PatientService;

import lombok.AllArgsConstructor;

@RestController
@RequestMapping("/patients")
@AllArgsConstructor
public class PatientController {
	//depcy
	private final AppointmentService appointmentService;
	private PatientService patientService;
	/*
	 * Desc - List patient's upcoming appointments
End Point design
URL - http://host:port/patients/{patientId}/appointments/upcoming
Method- GET
resource id - path var.
Resp - List<AppointmentDTO>
Main highlight - custom query - JPQL
i/p - user id.
	 */
	@GetMapping("/{patientId}/appointments/upcoming")
	public  ResponseEntity<?> listUpcomingAppointments(@PathVariable
Long patientId )
	{
		System.out.println("in patient list appointments "+patientId);
		//invoker service layer method
		try {
			return
ResponseEntity.ok(appointmentService.listUpcomingPatientAppointments(pati
entId));
		} catch (RuntimeException e) {
```

```
                        return ResponseEntity.status(HttpStatus.NOT_FOUND)
//SC 404
                                        .body(new ApiResponse(e.getMessage(),
"failed"));
                }
        }
        /*
         * URL - http://host:port/patients/{pid}/appointments
Request Payload -
{

  "doctorId": 5,
  "appointmentDateTime": "2025-11-05T10:30:00",
 }

Controller  - PatientController
Method - POST
Response - AppointmentResp DTO

Resp SC 201
{
  "appointmentId": 101,
  "doctorName": "Dr. Priya Sharma",
  "appointmentDateTime": "2025-11-05T10:30:00",
  "status": "SCHEDULED",
  "message": "Appointment booked successfully"
}

Error Resp SC 400 | 404 | 409
Api Response - with error message.

         */
        @PostMapping("/{pid}/appointments")
        public ResponseEntity<?> bookAppointment(@PathVariable Long
pid,@RequestBody BookAppointment dto)
        {
                System.out.println("in book appointment "+pid +" "+dto);
                try {
                        //invoke service layer method
                        return ResponseEntity.status(HttpStatus.CREATED)
```

```
		.body(appointmentService.bookAppointment(pid,dto));
			} catch (RuntimeException e) {
				System.out.println("err "+e);
				return ResponseEntity.status(HttpStatus.BAD_REQUEST)
						.body(new ApiResponse(e.getMessage(),
"Failed"));
			}
		}
		/*
		 * Desc - Patient Registration
		 * URL - http://host:port/patients/signup
		 * Method - POST
		 * Payload - req dto
		 * Success resp - api resp + SC 201
		 * Failed -  api resp +  SC 400
		 */
		@PostMapping("/signup")
		public ResponseEntity<?> registerPatient(@RequestBody
PatientRegDTO dto) {
			System.out.println("in patient reg "+dto);
			try {
				return ResponseEntity.status(HttpStatus.CREATED)
						.body(patientService.registerNewPatient(dto));
			} catch (RuntimeException e) {
				System.out.println("err "+e);
				return ResponseEntity.status(HttpStatus.BAD_REQUEST)
						.body(new ApiResponse(e.getMessage(),
"Failed"));
			}
		}


}

package com.healthcare.entities;

import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import jakarta.persistence.Column;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

/*
 * Declares a common base class , w/o any table associated with it.
 * Add common fields here.
 */
@MappedSuperclass
@Getter
@Setter
@ToString
public abstract class BaseEntity {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	@CreationTimestamp // to specify creation date | time | TS
	@Column(name = "created_on")
	private LocalDate createdOn;
	@UpdateTimestamp //to specify updation date | time | TS
	@Column(name="last_updated")
	private LocalDateTime lastUpdated;

}

package com.healthcare.entities;

import java.time.LocalDateTime;

import jakarta.persistence.AttributeOverride;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
```

```
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import jakarta.persistence.UniqueConstraint;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

// Table columns - doctor id , patient id , appointment date time , status

@Entity
@Table(name = "appointments",uniqueConstraints
=@UniqueConstraint(
                columnNames = {"appointment_date_time","doctor_id"} ) )

//PK col name - appointment_id
//@AttributeOverrides - tp override -id & creation_date
@AttributeOverride(name = "id", column = @Column(name =
"appointment_id"))
//lombok annotations
@NoArgsConstructor
@Getter
@Setter
@ToString(callSuper = true,exclude = {"myDoctor","myPatient"})
public class Appointment extends BaseEntity {

        @Column(name = "appointment_date_time")
        private LocalDateTime appointmentDateTime;
        @Enumerated(EnumType.STRING)
        private Status status = Status.SCHEDULED;
        // Appointment *------>1 Doctor - many to one association between
entities
        @ManyToOne
        // FK col name , not null
        @JoinColumn(name = "doctor_id", nullable = false)
        private Doctor myDoctor;
```

```
	// Appointment * ------> 1 Patient - many to one association between
entities
	@ManyToOne
	// FK col name , not null
	@JoinColumn(name = "patient_id", nullable = false)
	private Patient myPatient;
}

package com.healthcare.entities;
//users table -column - id(PK) , first name , last name, email ,password ,
phone , dob:date , role:enum,image :blob

import java.time.LocalDate;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Entity // to declare the class as Entity - so that Hibernate can manage it's life
cycle
		// - mandatory annotation
@Table(name = "users") // to specify table name
@AttributeOverride(name = "id", column = @Column(name = "user_id"))

@NoArgsConstructor
@Getter
@Setter
@ToString(callSuper = true,   exclude = { "password", "image" })
public class User extends BaseEntity {

	@Column(name = "first_name", length = 30) // col name , varchar size
	private String firstName;
	@Column(name = "last_name", length = 40)
	private String lastName;
	@Column(length = 50, unique = true) // col : unique constraint
	private String email;
	// not null constraint , size=300 (for hashed password)
	@Column(length = 300, nullable = false)
	private String password;
```

```
	@Column(unique = true, length = 14)

	private String phone;
//	@Transient //skip from persistence -> no col generation
//	private String confirmPassword;
	private LocalDate dob;
	@Enumerated(EnumType.STRING) // col type - varchar | enum
	private UserRole role;
	@Lob // large object , Mysql col type - longblob
	private byte[] image;
	@Column(name = "reg_amount")
	private int regAmount;

	public User(String firstName, String lastName, String email, String
password, String phone, LocalDate dob,
			int amount) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
		this.password = password;
		this.phone = phone;
		this.dob = dob;
		this.regAmount = amount;
	}

	public User(String firstName, String lastName, LocalDate dob) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.dob = dob;
	}
}

package com.healthcare.entities;

public enum UserRole {
	ROLE_DOCTOR, ROLE_PATIENT, ROLE_ADMIN
}
package com.healthcare.entities;
```

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

import jakarta.persistence.AttributeOverride;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.FetchType;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;


//gender , blood group , family history
@Entity
@Table(name = "patients")
@AttributeOverride(name="id", column = @Column(name="patient_id"))
//lombok annotations
@NoArgsConstructor
@Getter
@Setter
@ToString(callSuper = true,exclude = {"userDetails","diagnosticTests"})
public class Patient  extends BaseEntity{

	@Enumerated(EnumType.STRING)
	private Gender gender;
	@Enumerated(EnumType.STRING)
	@Column(name = "blood_group")
	private BloodGroup bloodGroup;
	@Column(name = "family_history", length = 500)
	private String familyHistory;
```

```
	//user
	@OneToOne(cascade = CascadeType.ALL) //mandatory
	@JoinColumn(name="user_id",nullable = false)
	private User userDetails;
	/*
	 * Project Tip (Gavin King)
	 * 1. In configuring many-many association , use Set , instead of List
	 *  - to avoid extra ,  un necessary db queries
	 *  2. In case of ANY Collection(List | Set)
	 *  - init it always to an empty Collection(to avoid NullPointerExc)
	 */
	//Patient *----->* DiagTest - many to -many , uni dir association
	@ManyToMany /* (fetch = FetchType.EAGER) */ //mandatory - avoids
MappingExc.
	@JoinTable(name="patient_tests",joinColumns =
@JoinColumn(name="patient_id"),inverseJoinColumns =
@JoinColumn(name="test_id")) //optional
	private Set<DiagnosticTest> diagnosticTests=new HashSet<>();
	public Patient(Gender gender, BloodGroup bloodGroup, String
familyHistory) {
		super();
		this.gender = gender;
		this.bloodGroup = bloodGroup;
		this.familyHistory = familyHistory;
	}


}
```

```
package com.healthcare.entities;

public enum Gender {
	MALE, FEMALE, OTHER
}
```

```
package com.healthcare.entities;

import jakarta.persistence.AttributeOverride;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
```

```
import jakarta.persistence.Entity;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

/*
 * speciality ,experienceInYears , qualifications , fees
 */
@Entity
@Table(name = "doctors")
/*
 * To specify id column name as - doctor_id
 */
@AttributeOverride(name = "id", column = @Column(name = "doctor_id"))
@NoArgsConstructor
@Getter
@Setter

/*
 * Project Tip - Exclude association fields from to string (to avoid recursion)
 * - Gavin King
 */
@ToString(callSuper = true, exclude = "userDetails")
public class Doctor extends BaseEntity {
	@Column(length = 100)
	private String speciality;
	@Column(name = "experience_in_years")
	private int experienceInYears;
	private String qualifications;
	private int fees;
	// Doctor 1----->1 User(user details)
	@OneToOne(cascade = CascadeType.ALL) // mandatory - o.w
Hibernate throws - MappingException
	/*
	 * To specify FK column name & to add not null constraint
	 */
```

```
	@JoinColumn(name = "user_id", nullable = false)
	private User userDetails;
	public Doctor(String speciality, int experienceInYears, String
qualifications, int fees) {
		super();
		this.speciality = speciality;
		this.experienceInYears = experienceInYears;
		this.qualifications = qualifications;
		this.fees = fees;
	}


}
```

```
package com.healthcare.entities;

public enum Status {
	SCHEDULED, CANCELLED,COMPLETED
}
```

```
package com.healthcare.service;

import java.util.List;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.healthcare.custom_exceptions.ResourceAlreadyExists;
import com.healthcare.custom_exceptions.ResourceNotFoundException;
import com.healthcare.dto.AppointmentDTO;
import com.healthcare.dto.AppointmentResp;
import com.healthcare.dto.BookAppointment;
import com.healthcare.entities.Appointment;
import com.healthcare.entities.Doctor;
import com.healthcare.entities.Patient;
import com.healthcare.entities.Status;
import com.healthcare.repository.AppointmentRepository;
import com.healthcare.repository.DoctorRepository;
import com.healthcare.repository.PatientRepository;
```

```
import lombok.AllArgsConstructor;

@Service
@Transactional
@AllArgsConstructor
public class AppointmentServiceImpl implements AppointmentService {
	private final AppointmentRepository appointmentRepository;
	private final PatientRepository patientRepository;
	private final DoctorRepository doctorRepository;

	@Override
	public List<AppointmentDTO> listUpcomingPatientAppointments(Long
patientId) {
		// by patient id
		return
appointmentRepository.getPatientUpcomingAppointmentsByPatientId(patientI
d, Status.SCHEDULED);
		// by user id
//		return
appointmentRepository.getPatientUpcomingAppointmentsByUserId2(userId,St
atus.SCHEDULED,LocalDateTime.now());

	}

	@Override
	public AppointmentResp bookAppointment(Long pid, BookAppointment
dto) {
		// 1. validate patient id
		Patient patient = patientRepository.findById(pid)
				.orElseThrow(() -> new
ResourceNotFoundException("Invalid Patient ID!!!!"));
		// 2. validate doc id
		Doctor doctor = doctorRepository.findById(dto.getDoctorId())
				.orElseThrow(() -> new
ResourceNotFoundException("Invalid Doctor ID!!!!"));
		// 3. check doc's availability
		if
(!appointmentRepository.existsByMyDoctorIdAndAppointmentDateTime(dto.g
etDoctorId(),
```

```
                        dto.getAppointmentDateTime())) {
                    // => available -> book appointment
                    Appointment appointment = new Appointment();

        appointment.setAppointmentDateTime(dto.getAppointmentDateTime());
                    // Appointment * --->1 Patient
                    appointment.setMyPatient(patient);
                    // Appointment * --->1 Doctor
                    appointment.setMyDoctor(doctor);
                    Appointment entity =
appointmentRepository.save(appointment);
                    // create Resp DTO
                    AppointmentResp respDto = new
AppointmentResp(entity.getId(), doctor.getUserDetails().getLastName(),
                                    dto.getAppointmentDateTime(),
Status.SCHEDULED, "Appointment Booked !");
                    return respDto;
                } else
                    throw new ResourceAlreadyExists("Appointment un
available");

        }

}

package com.healthcare.service;

import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.healthcare.custom_exceptions.ResourceAlreadyExists;
import com.healthcare.dto.ApiResponse;
import com.healthcare.dto.PatientRegDTO;
import com.healthcare.entities.Patient;
import com.healthcare.entities.UserRole;
import com.healthcare.repository.PatientRepository;
import com.healthcare.repository.UserRepository;

import lombok.AllArgsConstructor;
```

```
@Service
@Transactional
@AllArgsConstructor
public class PatientServiceImpl implements PatientService {
	private final PatientRepository patientRepository;
	private final UserRepository userRepository;
	private final ModelMapper mapper;

	@Override
	public ApiResponse registerNewPatient(PatientRegDTO reqDTO) {
		// 1. validate for dup email

	if(userRepository.existsByEmail(reqDTO.getUserDetails().getEmail()))
			throw new ResourceAlreadyExists("Email alread Exists
!!!!!");
		// 2. in case of no dup -> dto -> entity (deep copy)
		Patient entity = mapper.map(reqDTO, Patient.class);
		//2.5 assign patient role
		entity.getUserDetails().setRole(UserRole.ROLE_PATIENT);
		//3. save patient entity (highlight - JPA cascade)
		Patient persistentEntity=patientRepository.save(entity);//users :
insert -> PK -> child rec -> patients : FK
		//4. ret api resp dto
		return new ApiResponse("New Patient registered with
ID="+persistentEntity.getId(), "Success");
	}

}

package com.healthcare.service;

import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.healthcare.custom_exceptions.AuthenticationException;
import com.healthcare.dto.AuthRequest;
import com.healthcare.dto.AuthResp;
import com.healthcare.entities.User;
```

```
import com.healthcare.repository.UserRepository;

import lombok.AllArgsConstructor;

@Service
@Transactional
@AllArgsConstructor
public class UserServiceImpl implements UserService {
	//depcy
	private final UserRepository userRepository;
	private final ModelMapper modelMapper;

	@Override
	public AuthResp signIn(AuthRequest dto) {
		// invoke dao's method
		User
entity=userRepository.findByEmailAndPassword(dto.getEmail(),
dto.getPassword()) //Optional<User>
		.orElseThrow(() -> new AuthenticationException("Invalid Email or
password"));
		//convert entity -> dto (resp)
		AuthResp respDTO = modelMapper.map(entity, AuthResp.class);
		respDTO.setMessage("Login Successful.....");
		return respDTO;
	}

}

package com.healthcare.exception_handler;

import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.modelmapper.ModelMapper;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import com.healthcare.custom_exceptions.AuthenticationException;
import com.healthcare.custom_exceptions.ResourceNotFoundException;
import com.healthcare.dto.ApiResponse;

@RestControllerAdvice // To declare a spring bean containing global
exception handling logic . SC is
                                        // offering global exc handling advice via
this bean -> to all the rest
                                        // controllers in this app.
//try block - rest controller methods
//catch block - exc handler
public class GlobalExceptionHandler {

        // add exception handling method - to handle
ResourceNotFoundException
        @ExceptionHandler(ResourceNotFoundException.class)
        public ResponseEntity<?>
handleResourceNotFoundException(ResourceNotFoundException e) {
                return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(new
ApiResponse(e.getMessage(), "Failed"));
        }

        // add exception handling method - to handle auth exc

        @ExceptionHandler(AuthenticationException.class)
        public ResponseEntity<?>
handleAuthenticationException(AuthenticationException e) {
                return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(new
ApiResponse(e.getMessage(), "Failed"));
        }

        // catch all - handle ANY unchecked exception
        @ExceptionHandler(RuntimeException.class)
        //@ResponseStatus
```

```
	public ResponseEntity<?> handleRuntimeException(RuntimeException
e) {
		return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(new
ApiResponse(e.getMessage(), "Failed"));
	}

	// add exception handling method - to handleP.L validation failure - for
req body (JSON payload)

		@ExceptionHandler(MethodArgumentNotValidException.class)
		public ResponseEntity<?>
handleMethodArgumentNotValidException(MethodArgumentNotValidExceptio
n e) {
			System.out.println("in handle @Valid ");
			//1. get list of rejected fields
			List<FieldError> fieldErrors = e.getFieldErrors();
			//2. Covert it to Map <Key - field Name , Value - err mesg>
			Map<String, String> errorFieldMap = fieldErrors.stream()
//Stream<FieldError>

	.collect(Collectors.toMap(FieldError::getField,FieldError::getDefaultMess
age));//f -> f.getField(), f -> f.getDefaultMessage()

			return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errorFieldMap);
		}

}
```