Flight Reservation System (Monolithic Application)

application.properties

```
spring.datasource.url=jdbc:h2:mem:flightdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true
springdoc.api-docs.path=/api-docs
```

Flight.java

```
@Entity
public class Flight {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String flightNumber;

    private String origin;
    private String destination;
    private LocalDateTime departureTime;
    private int seatsAvailable;
}
```

Reservation.java

```
@Entity
public class Reservation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String passengerName;
    private String passengerEmail;
    private int seatsBooked;
    private LocalDateTime reservedAt;

    @ManyToOne
    private Flight flight;
}
```

FlightRepository.java

```
public interface FlightRepository extends JpaRepository<Flight,
Long> {
    Optional<Flight> findByFlightNumber(String flightNumber);
}
```

ReservationRepository.java

```
public interface ReservationRepository extends
JpaRepository<Reservation, Long> {
    List<Reservation> findByFlightId(Long flightId);
}
```

FlightNotFoundException.java

```
public class FlightNotFoundException extends RuntimeException {
    public FlightNotFoundException(String message) {
        super(message);
    }
}
```

NotEnoughSeatsException.java

```
public class NotEnoughSeatsException extends RuntimeException {
    public NotEnoughSeatsException(String message) {
        super(message);
    }
}
```

GlobalExceptionHandler.java

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(FlightNotFoundException.class)
    public ResponseEntity<String>
handleFlightNotFound(FlightNotFoundException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }

    @ExceptionHandler(NotEnoughSeatsException.class)
    public ResponseEntity<String>
handleNoSeats(NotEnoughSeatsException ex) {
        return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(ex.getMessage());
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleOther(Exception ex) {
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An
error occurred: " + ex.getMessage());
    }
}
```

FlightService.java

```
@Service
public class FlightService {
    @Autowired
    private FlightRepository repo;

    public Flight addFlight(Flight f) { return repo.save(f); }
    public List<Flight> getAllFlights() { return repo.findAll(); }
    public Flight getFlight(Long id) {
        return repo.findById(id).orElseThrow(() -> new
FlightNotFoundException("Flight not found"));
    }
    public Flight updateFlight(Long id, Flight flightDetails) {
        Flight f = getFlight(id);
        f.setFlightNumber(flightDetails.getFlightNumber());
        f.setOrigin(flightDetails.getOrigin());
        f.setDestination(flightDetails.getDestination());
        f.setDepartureTime(flightDetails.getDepartureTime());
        f.setSeatsAvailable(flightDetails.getSeatsAvailable());
        return repo.save(f);
    }
    public void deleteFlight(Long id) { repo.deleteById(id); }
    public void increaseSeats(Long flightId, int seats) {
        Flight f = getFlight(flightId);
        f.setSeatsAvailable(f.getSeatsAvailable() + seats);
        repo.save(f);
    }
    public void decreaseSeats(Long flightId, int seats) {
        Flight f = getFlight(flightId);
        if (f.getSeatsAvailable() < seats) throw new
NotEnoughSeatsException("Not enough seats available.");
        f.setSeatsAvailable(f.getSeatsAvailable() - seats);
        repo.save(f);
    }
}
```

ReservationService.java

```
@Service
public class ReservationService {
    @Autowired private ReservationRepository repo;
    @Autowired private FlightService flightService;

    public Reservation makeReservation(Long flightId, Reservation r)
{
        Flight f = flightService.getFlight(flightId);
        if (f.getSeatsAvailable() < r.getSeatsBooked()) {
            throw new NotEnoughSeatsException("Seats not available
for booking");
        }
        r.setFlight(f);
        r.setReservedAt(LocalDateTime.now());
        flightService.decreaseSeats(flightId, r.getSeatsBooked());
        return repo.save(r);
    }

    public List<Reservation> getAllReservations() { return
repo.findAll(); }
    public List<Reservation> getReservationsForFlight(Long flightId)
{
        return repo.findByFlightId(flightId);
    }
    public void cancelReservation(Long id) {
        Reservation r = repo.findById(id).orElseThrow(() -> new
RuntimeException("Reservation not found"));
        flightService.increaseSeats(r.getFlight().getId(),
r.getSeatsBooked());
        repo.deleteById(id);
    }
}
```

FlightController.java

```
@RestController
@RequestMapping("/api/flights")
public class FlightController {
    @Autowired private FlightService service;

    @PostMapping public Flight add(@RequestBody Flight f) { return
service.addFlight(f); }
    @GetMapping public List<Flight> getAll() { return
service.getAllFlights(); }
    @GetMapping("/{id}") public Flight get(@PathVariable Long id) {
return service.getFlight(id); }
    @PutMapping("/{id}") public Flight update(@PathVariable Long id,
@RequestBody Flight f) {
        return service.updateFlight(id, f);
    }
    @DeleteMapping("/{id}") public void delete(@PathVariable Long
id) { service.deleteFlight(id); }
}
```

ReservationController.java

```
@RestController
@RequestMapping("/api/reservations")
public class ReservationController {
    @Autowired private ReservationService service;

    @PostMapping
    public Reservation make(@RequestParam Long flightId,
@RequestBody Reservation r) {
        return service.makeReservation(flightId, r);
    }
    @GetMapping public List<Reservation> getAll() { return
service.getAllReservations(); }
    @GetMapping("/flight/{flightId}")
    public List<Reservation> getByFlight(@PathVariable Long
flightId) {
        return service.getReservationsForFlight(flightId);
    }
    @DeleteMapping("/{id}") public void cancel(@PathVariable Long
id) { service.cancelReservation(id); }
}
```

Restaurant Service - Microservice Code

application.properties

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/restaurantdb
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Restaurant.java

```
@Entity
public class Restaurant {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String location;
    private String contactNumber;

    @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL)
    private List<MenuItem> menuItems = new ArrayList<>();
}
```

MenuItem.java

```
@Entity
public class MenuItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String description;
    private double price;

    @ManyToOne
    @JoinColumn(name = "restaurant_id")
    private Restaurant restaurant;
}
```

RestaurantRepository.java

```
public interface RestaurantRepository extends
JpaRepository<Restaurant, Long> {}
```

MenuItemRepository.java

```
public interface MenuItemRepository extends JpaRepository<MenuItem,
Long> {
    List<MenuItem> findByRestaurantId(Long restaurantId);
}
```

RestaurantService.java

```
@Service
public class RestaurantService {
    @Autowired
    private RestaurantRepository restaurantRepo;

    public Restaurant addRestaurant(Restaurant r) {
        return restaurantRepo.save(r);
    }

    public List<Restaurant> getAllRestaurants() {
        return restaurantRepo.findAll();
    }

    public Restaurant getRestaurant(Long id) {
        return restaurantRepo.findById(id).orElseThrow(() -> new
RuntimeException("Restaurant not found"));
    }
}
```

MenuItemService.java

```
@Service
public class MenuItemService {
    @Autowired
    private MenuItemRepository menuRepo;

    @Autowired
    private RestaurantRepository restaurantRepo;

    public MenuItem addMenuItem(Long restaurantId, MenuItem item) {
        Restaurant r = restaurantRepo.findById(restaurantId)
            .orElseThrow(() -> new RuntimeException("Restaurant not
found"));
        item.setRestaurant(r);
        return menuRepo.save(item);
    }

    public List<MenuItem> getMenuItemsByRestaurant(Long
restaurantId) {
        return menuRepo.findByRestaurantId(restaurantId);
    }
}
```

RestaurantController.java

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

    @Autowired
    private RestaurantService restaurantService;

    @PostMapping
    public Restaurant addRestaurant(@RequestBody Restaurant r) {
        return restaurantService.addRestaurant(r);
    }

    @GetMapping
    public List<Restaurant> getAllRestaurants() {
        return restaurantService.getAllRestaurants();
    }

    @GetMapping("/{id}")
    public Restaurant getRestaurant(@PathVariable Long id) {
        return restaurantService.getRestaurant(id);
    }
}
```

MenuItemController.java

```
@RestController
@RequestMapping("/restaurants/{restaurantId}/menu-items")
public class MenuItemController {

    @Autowired
    private MenuItemService menuItemService;

    @PostMapping
    public MenuItem addItem(@PathVariable Long restaurantId,
@RequestBody MenuItem item) {
        return menuItemService.addMenuItem(restaurantId, item);
    }

    @GetMapping
    public List<MenuItem> getItems(@PathVariable Long restaurantId)
{
        return
menuItemService.getMenuItemsByRestaurant(restaurantId);
    }
}
```