Variant 1

Bus ticket reservation system

A bus reservation system is a web software solution designed to provide customers with a personalized easy-to-utilize user experience for booking and purchasing tickets online. It stores scheduled routes data, frequent trips, drop points, and other information. You can implement real-time seat availability, multiple payment gateways, offer seat map functionality, and other features.

Features

- 24/7 availability for customers
- There are administrator profile and bus driver profile. Administrator profile to control and monitor payments, CRUD (Get Create Update Delete) trip, add bus and bus drivers.

Function Requirements :

- Trip ticket should contain: departure and arrival point and time, seat number, bus number
- Trip info is contained by departure and arrival point and time, number of available seats, bus and bus driver info, price, duration of trip
- User can buy ticket without any registration
- Admin and bus driver can login to the system
- Admin can CRUD driver and trip
- Driver has access to their schedule
- System should show available trips and seats of a bus
- Any user can see available trip and buy it
- System should print ticket for customer

1. Create a database in postgres or use h2 in memory database. Create 5 entity tables, where should be One-to-one, One-to-many, many-to-many relationships (join table won't be counted as entity table). Create a DATABASE UML diagram. Upload your diagram with project as PDF file. **FILE SHOULD BE LOCATED INSIDE YOUR PROJECT FOLDER**
2. Upload database backup file with your project, if you use **postgres** database. **Name your database/backup file. – {$your_variant_{$your_lastname}}.** For example **variant1_Urmanov.tar.**
   spring.datasource.url=jdbc:postgresql://localhost:5432/ variant1_Urmanov
3. Create Readme.MD file in project structure. In this file write your project's idea, functionality that you're going to implement etc. (https://github.com/tchapi/markdown-cheatsheet/blob/master/README.md):
4. Use different type of beans annotations.

5. Use different type of Dependency Injections. (ONLY CONSTRUCTOR and Setter injection. NO FIELD injection)
6. **Write good service logic in service classes.** (If your most port of code will consist only calling repository methods, -50% from your grade)
7. Use next annotations: @Configuration.
8. Use next annotations: @Bean with init and destroy methods.
9. Add AOP configuration. Use AspectJ annotation style.
10. Use next annotations: @Before, @Pointcut, @After.
11. **Add real service/business logic in AOP code.**
12. **Add Jpa repository support.**
13. Add cache configuration.
14. Use different type of Query creation (https://docs.spring.io/spring-data/jpa/docs/1.5.0.RELEASE/reference/html/jpa.repositories.html - 2.3.2 Query creation).
15. Use Annotation based named query configuration
16. Declare query at the query method using @Query
17. Use SpEL expressions
18. Use Transactionality/ Locking/Auditing
19. Use JSR-349 Bean Validation
20. Use **ALL** next attributes:

Attribute Name	Default Value	Possible Values

`propagation`	`Propagation.REQUIRED`	`Propagation.REQUIRED`
		`Propagation.SUPPORTS`
		`Propagation.MANDATORY`
		`Propagation.REQUIRES_NEW`
		`Propagation.NOT_SUPPORTED`
		`Propagation.NEVER`
		`Propagation.NESTED`

`isolation`	`Isolation.DEFAULT` (default isolation level of the underlying resource)	`Isolation.DEFAULT`
		`Isolation.READ_UNCOMMITTED`
		`Isolation.READ_COMMITTED`
		`Isolation.REPEATABLE_READ`
		`Isolation.SERIALIZABLE`

`timeout`	`TransactionDefinition.TIMEOUT_DEFAULT` (default transaction timeout in seconds of the underlying resource)	An integer value larger than zero; indicates the number in seconds for timeout
`readOnly`	false	{`true`, `false`}
`rollbackFor`	Exception classes for which the transaction will be rolled back	N/A
`rollbackForClassName`	Exception class names for which the transaction will be rolled back	N/A
`noRollbackFor`	Exception classes for which the transaction will not be rolled back	N/A
`noRollbackForClassName`	Exception class names for which the transaction will not be rolled back	N/A
`value`	"" (a qualifier value for the specified transaction)	N/A

21. Write scheduled method. Use @Scheduled annotations with attributes:
    - fixedDelay
    - fixedRate
    - initialDelay
22. Use all next methods:

HTTP Method	Description
GET	GET retrieves a representation of a resource.
HEAD	Identical to GET, without the response body. Typically used for getting a header.
POST	POST creates a new resource.
PUT	PUT updates a resource.
DELETE	DELETE deletes a resource.
OPTIONS	OPTIONS retrieves allowed HTTP methods.

23. Use next annotations:

Annotation	Old-Style Equivalent
`@GetMapping`	`@RequestMapping(method = RequestMethod.GET)`
`@PostMapping`	`@RequestMapping(method = RequestMethod.POST)`
`@PutMapping`	`@RequestMapping(method = RequestMethod.PUT)`
`@DeleteMapping`	`@RequestMapping(method = RequestMethod.DELETE)`

24. Use RequestBody and ResponseBody Annotations. Read HTTP Headers in Spring REST Controllers.
25. Use Spring @ResponseStatus to Set HTTP Status Code. Use Spring ResponseEntity to Manipulate the HTTP Response
26. **Add JUnit test with at least 60% code coverage.**
27. Use different type of Assertions.
28. Use ReflectionTestUtils.
29. Write JMS service. 1 method which send data to topic, second method which listen topic.
30. Use Spring Security Basic Authentication.
31. Use BasicAuthenticationEntryPoint
32. **Write CURL in README.md for your ALL endpoints, or upload in project folder POSTMAN collections.**