Online Medical Management System

Online Medical Management System, titled “Virtual Medicine Home”, is a web-based project with the aim of providing a platform for online appointment between doctor and patient. It enables a doctor to give online appointment, e-prescription, save and view the patient’s previous health records, lab reports, etc. This system also facilitates to look for blood and eye donators.

Features:

- It has two modules — Admins module and Doctors module. Admin module is for controlling the online software system whereas doctors module is for interaction with patients.
- It is accessible from any part of the world via internet service.
- It ensures data security and accuracy with the maintenance of telecenter data.
- Users needn’t go anywhere for treatment; the communication between patients and doctors is established through chat.

Function Requirements:-

- The site should have proper users’ profile management and registration.
- There must be appointment of patients with doctors. For new patients, it is better to have data regrading previous health records.
- Required number of doctors should be available for service of patients.
- There should be a facility of making a complaint to the admin for any kind of medical error.
- The site needs to have number of medical manuals in local language.
- Admin is supposed to take backups of all the data and it should be capable of generating system reports.

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.**