Electricity Billing System

Electricity Billing System is a software-based application. The project aims at serving the department of electricity by computerizing the billing system. It mainly focuses on the calculation of Units consumed during the specified time and the money to be paid to electricity offices. This computerized system will make the overall billing system easy, accessible, comfortable and effective for consumers.

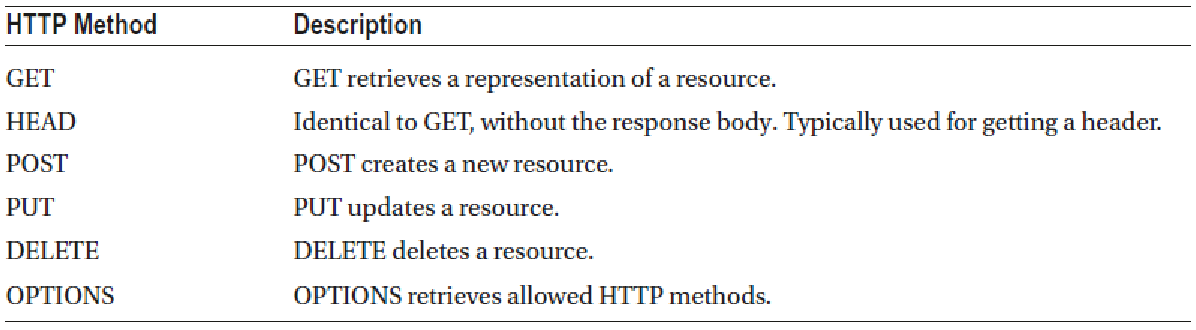
Features:

To make the billing system more service-oriented and simple, the following features should be implemented in the project.

- The application has high speed of performance with accuracy and efficiency.
- The software provides facility of data sharing.
- It doesn’t require any staffs as in the conventional system. Once it is installed on the system, only the meter readings are to be given by the customer.
- The electricity billing software calculates the units consumed by the customer and makes bills.
- It has the provision of security restriction.
- It requires small storage for installation and functioning.
- There is provision for debugging if any problem is encountered in the system.

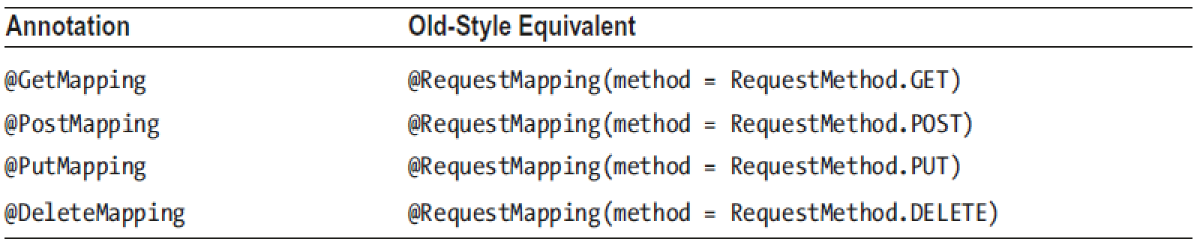
1. Create a database in postgres or use h2 in memory database. Create 10 entity tables, where should be One-to-one, One-to-many, many-to-many relationships (join table won’t be counted as entity table). Create a DATABASE UML diagram. Upload your diagram with project as PDF file. **FILE SHOULD BE LOCATED INSIDE YOUR PROJECT FOLDER.**
2. Upload database backup file with your project, if you use postgres database. **Name your database – {$your_variant_{$your_lastname}}.** For example **variant3_Urmanov.tar**
   spring.datasource.url=jdbc:postgresql://localhost:5432/ variant3_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 repo method, -50% from your grade)
7. Use @PropertySource, @Lazy, @Scope, @DependsOn.
8. Use field injection, use application.properties file In configuration class.
9. Add at least 2 configuration classes.
10. Add AOP configuration. Use AspectJ annotation style.
11. Use next annotations: @Before, @Pointcut, @After, @Aspect, @AfterReturning, @Around, @AfterThrowing.
12. Add real service/business logic in AOP code.
13. For your repository classes, use different and more complex methods/code for JdbcTemplate class. **DON'T USE JPA Repositories.**
14. Use batch operations.
15. Implement a Custom Converter (org.springframework.core.convert.converter.Converter)
16. Implement a Custom Formatter (FormattingConversionServiceFactoryBean)
17. Use AssertTrue for Custom Validation (@AssertTrue(message="ERROR! Individual customer should have gender and last name defined")
18. Write scheduled method. Use @Scheduled annotations with attributes:
    - fixedDelay
    - fixedRate
    - initialDelay
19. Parameterizing the Schedule. Parameters should be in application.props file.
20. **Run Tasks in Parallel.**
21. Use all next methods:



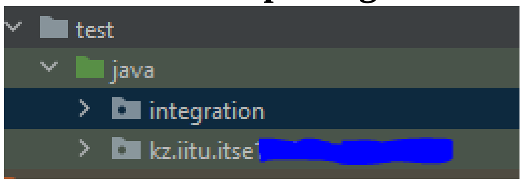
HTTP Method	Description
GET	GET retrieves a representation of a resource.
HEAD	Identical to GET, without the response body. Typically used for getting a header.
POST	POST creates a new resource.
PUT	PUT updates a resource.
DELETE	DELETE deletes a resource.
OPTIONS	OPTIONS retrieves allowed HTTP methods.

22. 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)`

23. Use RequestBody and ResponseBody Annotations. Read HTTP Headers in Spring REST Controllers.
24. **Setting Up Spring openapi**
25. Use Spring @ResponseStatus to Set HTTP Status Code. Use Spring ResponseEntity to Manipulate the HTTP Response
26. Add REST Pagination support
27. Add Upload and Download file methods.
28. **Add JUnit test with at least 80% code coverage.**
29. **Write integration test for controller classes. Integration test should be located in other package:**



```
test
  java
    integration
    kz.iitu.itse[redacted]
```

30. Write JMS service. 3 methods which send data to topic, 3 methods which listen topic.
31. **Add JUnit test with at least 80% code coverage.**
32. **Use OAuth2 and JWT**
33. **DO NOT USE** in memory authentication (auth.inMemoryAuthentication())
34. Prevent Brute Force Authentication Attempts with Spring Security
35. Control the Session with Spring Security
    - always
    - ifRequired
    - never
    - stateless
36. Fix 401s with CORS Preflights and Spring Security
37. Prevent Cross-Site Scripting (XSS) in a Spring Application
38. Add 1-2 pages which supports websocket technology. Example - https://www.baeldung.com/websockets-spring
39. **Write CURL in README.md for your ALL endpoints, or upload in project folder POSTMAN collections.**