Pre-requisites

- Completion of Basic Java course

Setup

- Install STS
- Install SpringBoot Extension for Visual Studio Code
- Install Postman Application
- Install PostgreSQL 13 or later and PgAdmin

General Guidelines

- Code must be properly indented as per the indentation format shared with you.
- Code must use meaningful variable names.
- Code must compile without any errors/warnings.
- Code must produce output strictly in the expected format wherever such format is provided.
- Unless and otherwise specified, always print floating point numbers with two decimal points.
- Add appropriate code for exception handling and print a meaningful exception message in case of an exception.
- Use request and response JSON format as per the specification within this document.
- Use Postman application to test APIs
- Code for all exercises must be pushed within the “advanced-java” folder under the branch name allocated to you on a Git repository. Branch name must be Participant’s Merce username (e.g. “indulekhas” for Indulekha Singh)

JSON request format

{
“token”: a bearer token wherever task demands else empty string
“data”: {

} -- all API request attributes will be inside this element,
“_reqid”: “afkjekfsdkf454354” -- a long random string of characters a unique request ID,
“_client_ts”: “” -- ISO timestamp when client sent a request,
“_client_type”: “web” (possible values: “web”, “ios”, “android”),
}

JSON response format

{
“status”: “...” -- “success” or “error”,
“status_code”: “...” -- an error code -- absent or empty for a success message,
“status_msg”: “...” -- an error message which can be shown to an end user -- absent or empty for a success message,
“data”: {

} -- all API response attributes will be inside this element or empty incase of an error

"_reqid": "afkjekfsdkf45fst87" -- a long random string of characters a unique request ID sent by a client,
"_server_ts": "" -- ISO timestamp when server sent a response,
}

Assignments

All following programs shall be done using SpringBoot framework. Request (input) and response (output) for each API endpoint will be in JSON format unless a specific format is specified. Wherever applicable, refer to a similar problem under "Core Java" and use as much code written earlier for following problems.

Assumption is that you will use most of the code you have developed from the earlier course.

1. Write an API "/hello" (GET method) to return "Hello World" as output. **[SpringBoot Main, 2 hours]**
2. Write an API "/hello" (POST method) to accept a name as an input, and return "Hello <name>" as an output. **[API introduction & JSON data handling, 4 hours]**
3. Write an API "/mysum" to accept "operation", "num1" and "num2" as input requests and return output as per the operation specified. **[API revision, 2 hours]**
4. Write an API "/mycalc" to accept a list of numbers and an operation name in an input request. Refer to earlier example for list of permitted operation names (e.g. mean value, minimum value, maximum value etc.). The response should provide information about the operation executed and the result of an operation. **[JSON array handling, 4 hours]**
5. Write an API "/myproperties". Define a few property identifiers within a SpringBoot config file. An input request will have a list of property identifiers to be retrieved. Result will return the property identifier and its value from the property file. Wherever property identifier does not exist, returned value will be "NULL". **[SpringBoot property files, 6 hours]**
6. Create a database in PostgreSQL and create a few tables as: (i) "employee" ("id" not null -- sequence, "empid" not null -- unique employee id, "fname" not null -- first name, "fullname", "dob" not null -- date of birth, "doj" not null -- date of joining, "salary" not null -- integer, "reportsto" -- employee id of a reporting officer, "deptid" -- foreign key to "departments" table, "rankid" -- foreign key to "ranks" table id, "createdat" not null -- when a new record was inserted, "updatedat" -- when record was last updated, "client_reqid" not null -- Client request ID ("_reqid" in JSON request) which was responsible for the changes (ii) "departments" ("id" -- department id -- sequence, "deptname" -- name of department). (iii) "ranks" ("id": rank id -- sequence, "rankdesc" -- Description of a rank, "parentrankid" -- rank id of rank above this rank. Use PgAdmin to run SQL queries to populate "departments" and "ranks" tables with at least 10 records each with meaningful values. **[Database schema basics, 4 hours].**
7. Write an API "/myhr/employee/add" operation for "employee" using Hibernate ORM. Ensure that database connection properties are stored within a SpringBoot property file. An Input request will have all relevant values to add an employee. If an employee record can be added successfully, it will return "status" as "success", else it will return "status" as "error" and will also provide an error code and error message. Error message must clearly specify the reason for an error -- e.g. database connection error, duplicate id, data type violation (e.g. invalid date or string for an integer number). Remember that an input request won't have fields like create timestamp / update time stamp. **[Hibernate introduction, 12 hours]**
8. Extend the above project to write an API /myhr/employee/list" operation. The response will be an array of arrays -- employee id and employee name -- found for employees within an employee table. **[CRUD Basics, 6 hours]**

9. Extend the above project -- write an API “/myhr/employee/list” operation to support a filtered response. An input request will have an optional parameter “filter” -- a string. Response will have only those employee records which have a “filter” string within an employee name. Use case insensitive match. Use SQL query to filter out records instead of filtering within the Java code. **[CRUD & SQL Basics, 4 hours]**
10. Extend the above project -- write an API “/myhr/employee/get” operation to retrieve details of a particular employee based on employee id sent within an input request. Within the response, send a rank description, department name and supervisor name, instead of a foreign key id. You may use multiple queries or a SQL JOIN. **[CRUD & SQL Basics, 4 hours]**
11. Extend the above project -- write an API “/myhr/employee/update” operation to update details of a particular employee based on employee id and other details sent within the input request. Remember that an input request won’t have fields like create timestamp / update time stamp. **[CRUD & SQL Basics, 4 hours]**
12. Create an additional table “employee_shadow”. This table will have all the columns of the employee table but “empid” will be non unique. **[DB Schema, 1 hour]**
13. Extend the above project -- write an API “/myhr/employee/delete” operation to retrieve details of a particular employee based on employee id sent within an input request. Within the response, send a rank description, department name and supervisor name, instead of a foreign key id. You may use multiple queries or a SQL JOIN. **[CRUD & SQL Basics, 4 hours]**
14. Extend the above project -- update APIs “/myhr/employee/update” & “/myhr/employee/delete” to copy an existing employee record to “employee_shadow” table before updating/deleting the record within “employee” table. **[DB transaction, 6 hours]**
15. Extend the above project and add debug logs across all APIs developed so far. Understand various log levels, configure log levels through a property file, and how log files are rotated. **[Debug logging, 6 hours]**
16. Extend the above project -- log each API request received using the logging framework provided by SpringBoot. The logline shall be structured as follows: ISO 1601 timestamp|user=<userid>|api=<API>|status=|<status>|error_code=<error code>|error_message=<error message>|client_reqid=<_reqid>|req=<JSON representation of request>| **[Structured logging, 6 hours]**
17. Extend API “/myhr/employee/list” developed earlier to support “/myhr/employee/list?type=xml” operation. This API will perform all tasks mentioned in the earlier API. Instead of returning a JSON response, this API will return an XML response with appropriate content type. **[XML document, 8 hours]**
18. Extend API “/myhr/employee/list” developed earlier to support “/myhr/employee/list?type=xlsx” operation. This API will create a unique Excel file using Apache POI library. Filename prefix and directory path must be picked up from the property file. If file creation is successful, it will return an Excel file with an appropriate content type, else it will return a JSON response with an error message. **[Excel document, 12 hours]**
19. Extend API “/myhr/employee/list” developed earlier to support “/myhr/employee/list?type=pdf” operation. This API will create a unique PDF file using the HTML 2 PDF feature of Apache iTextPDF or OpenPDF library. Filename prefix and directory path must be picked up from the property file. If file creation is successful, it will return a PDF file with an appropriate content type, else it will return a JSON response with an error message. **[HTML2PDF document, 12 hours]**
20. Install Redis service, and use “redis-cli” to perform following operations: (i) Add a new key: user.<empname> for a few employees with default value as zero, (ii) View key value, (iii) Use increment and decrement operators to change value, (iv) set TTL. Write a SpringBoot

API and singleton class to connect to Redis and perform all operations tried using “redis-cli”. **[Redis Setup & Singleton pattern, 5 hours]**

21. Write API /myhr/employee/updateEmployeeContribution. This API will take department name, an employee ID and an optional count (a number) within input request. This API will add a new key “user.<dept>.<empID>” with a number (default is zero) if the key does not exist. If the key exists, it will increment the key value with the count. The API response will have the key name and latest count. **[Redis Key-Value, 3 hours]**
22. Write API /myhr/employee/getContribution. This API will take a department name as input and return the latest count for this employee. **[Redis Key-Value, 3 hours]**
23. Extend APIs which support view operations in above exercises to fetch the value from the Redis. If the value is not found in Redis, it will perform a DB operation, cache the value with TTL as 3min, and then return the response. **[Redis TTL: 5 hours]**
24. Update all APIs to check for a valid JWT token, and reject those API requests if the token is not valid, using Filters. **[JWT and authentication, 3 hours]**
25. Update all APIs to log request and response in a file, along with the time spent and status (success/fail). **[Logging and Interceptors, 3 hours]**