**Vehicle Management System**

**Important Instructions:**

1. Please read the document thoroughly before you code.

2. Import the given skeleton code into your Eclipse.

3. Use Java 8 and Spring 5 for solving the code challenge.

4. You have to test the code and ensure there are no compilation errors before l

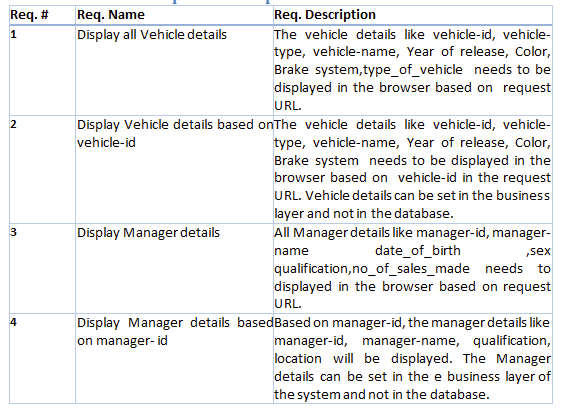
**1. Business Scenario:**

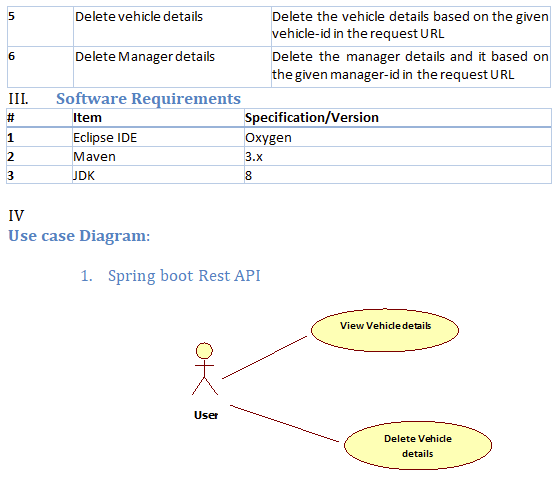
The objective of this vehicle management application is to provide and maintain information related to workflow of a vehicle and the various aspects associated with it. It is a collection of data in an organized manner to be easily be accessed, retrieved, managed and updated as per need. A Vehicle Database contains the details of Vehicle, Available numbers, Type and Manager with department head wise.

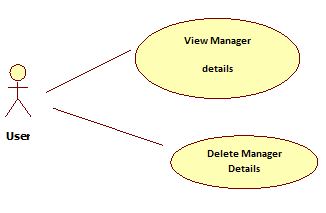
The scope and pilot test of this scenario is, to display only a minimum functional product that allow User (Admin) to View and Delete the details by providing some basic information like vehicle- id, Manager-id etc., The static data will be given in the Business Layer of the system. The sample data is given in vehicle.txt file. Using appropriate Rest API calls, the desired output should be shown in the browser. The vehicle and manager details will be retrieved or deleted based on URL parameters.

Develop a simple Spring MVC, with Spring Boot and Rest API.

**II. Functional Requirement Specification:**

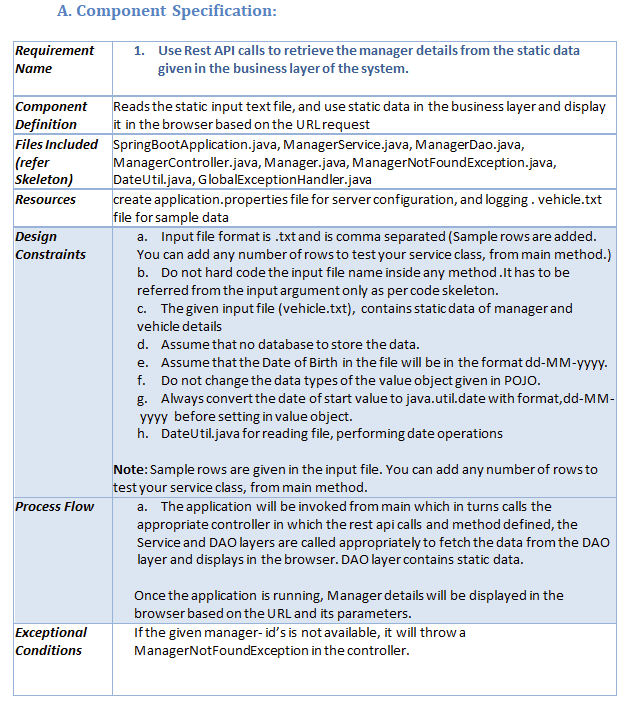


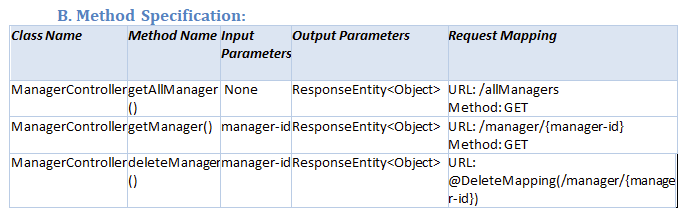


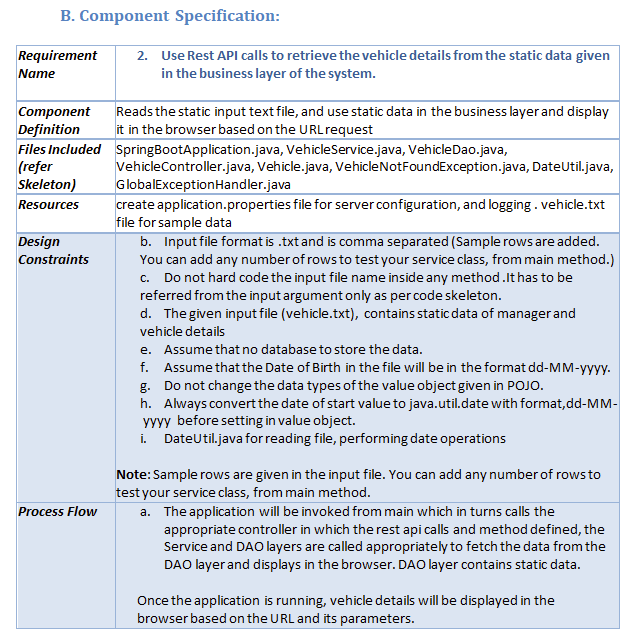


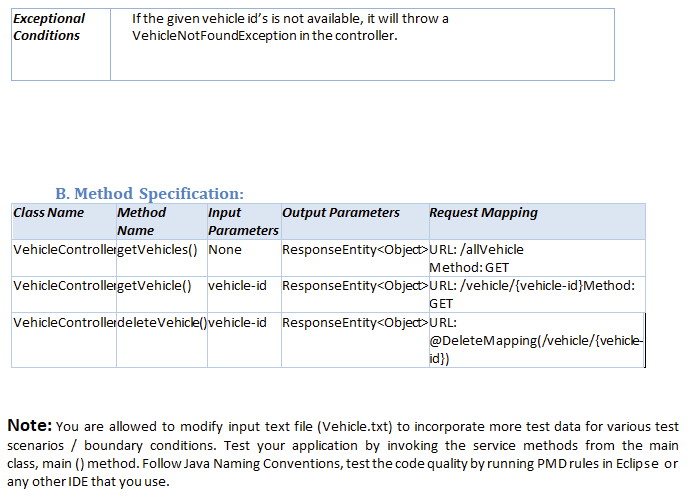
**V) Technical Requirements**

A. Component Specification:









Requirement Name 1. Use Rest API calls to retrieve the manager details from the static data given in the business layer of the system.

Component

Definition  Reads the static input text file, and use static data in the business layer and display it in the browser based on the URL request

Files Included

(refer

Skeleton)  SpringBootApplication.java, ManagerService.java, ManagerDao.java, ManagerController.java, Manager.java, ManagerNotFoundException.java, DateUtil.java, GlobalExceptionHandler.java

Resources create application.properties file for server configuration, and logging . vehicle.txt file for sample data

Design

Constraints  a. Input file format is .txt and is comma separated (Sample rows are added. You can add any number of rows to test your service class, from main method.) b. Do not hard code the input file name inside any method .It has to be referred from the input argument only as per code skeleton. c. The given input file (vehicle.txt), contains static data of manager and vehicle details d. Assume that no database to store the data. e. Assume that the Date of Birth in the file will be in the format dd-MM-yyyy.  f. Do not change the data types of the value object given in POJO.  g. Always convert the date of start value to java.util.date with format,dd-MM- yyyy before setting in value object.  h. DateUtil.java for reading file, performing date operations  Note: Sample rows are given in the input file. You can add any number of rows to test your service class, from main method.

Process Flow a. The application will be invoked from main which in turns calls the appropriate controller in which the rest api calls and method defined, the Service and DAO layers are called appropriately to fetch the data from the DAO layer and displays in the browser. DAO layer contains static data. Once the application is running, Manager details will be displayed in the browser based on the URL and its parameters.

Exceptional

Conditions If the given manager- id’s is not available, it will throw a ManagerNotFoundException in the controller.

B. Method Specification:

Class Name Method Name Input Parameters Output Parameters Request Mapping

ManagerController getAllManager() None ResponseEntity<Object> URL: /allManagers Method: GET

ManagerController getManager() manager-id ResponseEntity<Object> URL: /manager/{manager-id} Method: GET

ManagerController deleteManager() manager-id ResponseEntity<Object> URL: @DeleteMapping(/manager/{manager-id})

B. Component Specification:

Requirement Name 2. Use Rest API calls to retrieve the vehicle details from the static data given in the business layer of the system.

Component

Definition  Reads the static input text file, and use static data in the business layer and display it in the browser based on the URL request

Files Included

(refer

Skeleton)  SpringBootApplication.java, VehicleService.java, VehicleDao.java, VehicleController.java, Vehicle.java, VehicleNotFoundException.java, DateUtil.java, GlobalExceptionHandler.java

Resources create application.properties file for server configuration, and logging . vehicle.txt file for sample data

Design

Constraints  b. Input file format is .txt and is comma separated (Sample rows are added. You can add any number of rows to test your service class, from main method.) c. Do not hard code the input file name inside any method .It has to be referred from the input argument only as per code skeleton. d. The given input file (vehicle.txt), contains static data of manager and vehicle details e. Assume that no database to store the data. f. Assume that the Date of Birth in the file will be in the format dd-MM-yyyy.  g. Do not change the data types of the value object given in POJO.  h. Always convert the date of start value to java.util.date with format,dd-MM- yyyy before setting in value object.  i. DateUtil.java for reading file, performing date operations  Note: Sample rows are given in the input file. You can add any number of rows to test your service class, from main method.

Process Flow a. The application will be invoked from main which in turns calls the appropriate controller in which the rest api calls and method defined, the Service and DAO layers are called appropriately to fetch the data from the DAO layer and displays in the browser. DAO layer contains static data. Once the application is running, vehicle details will be displayed in the browser based on the URL and its parameters.

Exceptional Conditions If the given vehicle id’s is not available, it will throw a VehicleNotFoundException in the controller.

B. Method Specification:

Class Name Method Name Input Parameters Output Parameters Request Mapping

VehicleController getVehicles() None ResponseEntity<Object> URL: /allVehicle Method: GET

VehicleController getVehicle() vehicle-id ResponseEntity<Object> URL: /vehicle/{vehicle-id}Method: GET

VehicleController deleteVehicle() vehicle-id ResponseEntity<Object> URL: @DeleteMapping(/vehicle/{vehicle-id})

Note: You are allowed to modify input text file (Vehicle.txt) to incorporate more test data for various test scenarios / boundary conditions. Test your application by invoking the service methods from the main class, main () method. Follow Java Naming Conventions, test the code quality by running PMD rules in Eclipse or any other IDE that you use.