Instructions:

- Logon to your bit bucket stash, you will find the repository “coding-assessment-3”
- Create a branch with your LLID
- Import the eclipse project into your Eclipse IDE
- Once you finish coding, commit your work to your respective branch
- VERY IMPORTANT: DO NOT COMMIT TO “master” BRANCH

Requirement #1:

In the file “dbscript.sql” write the commands to do the following:

- Write the command to create a database with your LLID
- Write the command to switch to the newly created database
- Write the CREATE TABLE command to create the CUSTOMERS table:

COLUMN NAME	DATA TYPE	EXTRA
id	int	primary key, auto increment
name	varchar (50)	not null
email	varchar (200)	not null, unique
phone	varchar (50)	not null, unique
city	varchar (50)	default “Bangalore”

- Write the INSERT commands to insert the following rows

name	email	phone	city
Ramesh	ramesh@xmpl.com	9731424900	Chennai
Rajesh	rajesh@xmpl.com	9731424800	Mangalore
Harish	harish@acme.com	9731424700	Bangalore
Vijay	vijay@qwert.com	9731424600	Bangalore
Mahesh	mahesh@qwert.com	9731424500	Bangalore

Requirement #2:

- Create the entity class com.ps.entity.Customer to represent the CUSTOMERS table with appropriate entity mappings. Use Lombok for default constructor, getters, setters and toString.
- Create an interface com.ps.repository.CustomerRepository that inherits from CrudRepository
    - Add a method to get the list of customers from a given city
    - Add a method to get the customer based on email
    - Add a method to get the customer based on phone
- Create a REST controller com.ps.controllers.CustomerRestController that has the following request mappings:
    - /api/customers (HTTP GET method)
        - should return data for all customers
        - Client should be able to negotiate content using “Accept” header for XML as well as JSON

- /api/customers/{id} (HTTP GET method)
  - should return details of one customer based on the id supplied as path variable
  - Client should be able to negotiate content using “Accept” header for XML as well as JSON
  - If the id is not found, should respond with status 404
- /api/customers (HTTP POST method)
  - should accept one customer data in either JSON or XML format and should save the data as a new customer in the database table.
  - should return details of the same customer data along with the newly generated id
  - Client should be able to negotiate content using “Accept” as well as “Content-type” header for XML as well as JSON
- /api/customers/{id} (HTTP PUT method)
  - should accept one customer data in either JSON or XML format and should update the data in the database table for the customer based on the id supplied as path variable
  - should return details of the same customer data along with the newly generated id
  - If the id is not found, should respond with status 404
  - Client should be able to negotiate content using “Accept” as well as “Content-type” header for XML as well as JSON

Marking scheme:

Slno	Fulfillment	Marks (50)
1	Create table command	2
2	Constraints on each column (2 marks for each constraint)	10
3	Insert SQL command	3
4	Customer entity table (JPA annotations - 3, Lombok - 2)	5
5	CustomerRepository interface	2
6	CustomerRepository - method for customers by city	2
7	CustomerRepository - method for customer by email	2
8	CustomerRepository - method for customer by phone	2
9	CustomerRestController - Spring annotations	2
10	Handler for /api/customers (GET JSON format)	2
11	Handler for /api/customers (GET XML format)	2
12	Handler for /api/customers/{id} (GET JSON format)	2
13	Handler for /api/customers/{id} (GET XML format)	2
14	Handler for /api/customers/{id} (GET invalid id)	2
15	Handler for /api/customers (POST JSON format)	2
16	Handler for /api/customers (POST XML format)	2
17	Handler for /api/customers/{id} (PUT JSON format)	2
18	Handler for /api/customers/{id} (PUT XML format)	2
19	Handler for /api/customers/{id} (PUT invalid id)	2