Challenge: SkipTheDishes - Skip the Dishes

Name: Sylvio Antonio Pedroza Neto

Position: Java Developer

Email: sylvio.pedroza@gmail.com / Phone**:** +55 11 94252 5313

Linkedin**:** https://www.linkedin.com/in/sylvio-antonio-pedroza-neto-5a0b8150/

Github: https://github.com/sylvioneto/vanhack

Contents

Introduction 2

Database 4

Application 5

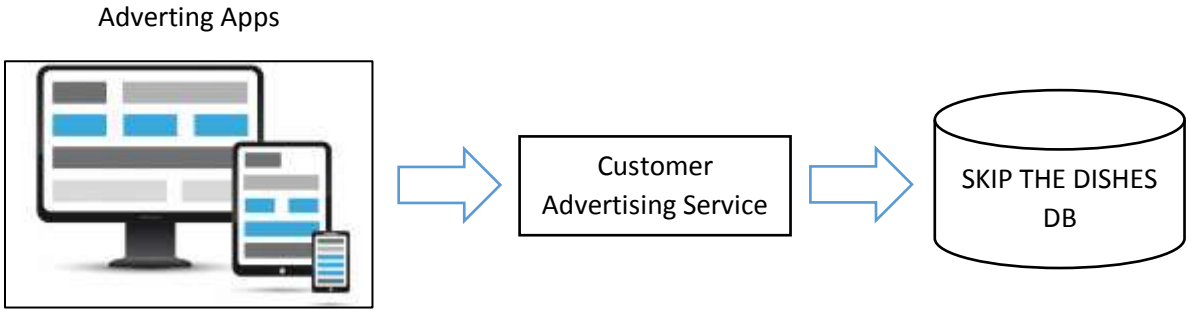
Testing 8

Introduction

I created a set of services that can be used for mobile and desktop clients in order to get customer information.

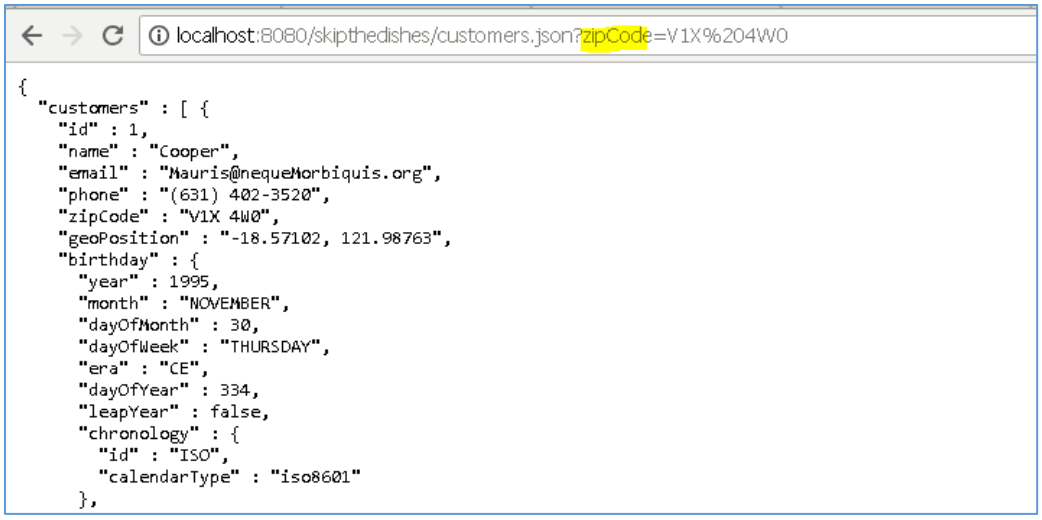
Advertising apps can invoke these APIs to find potential customers based on zip code or geo position for example and send them promotions or other advertising goals. For example, a restaurant would like to see all customers placed in a specific zip code where people don't order food very often in order to offer them a discount. Or, a new restaurant want to know who leaves in the neighborhood and offer a promotional price in its opening week.

Architecture



Examples

1: Using zip code



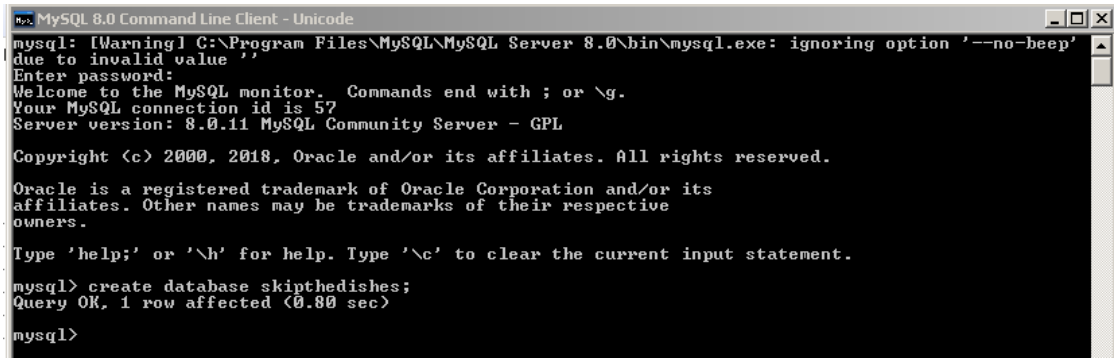
Example 2: Using geo position

Database

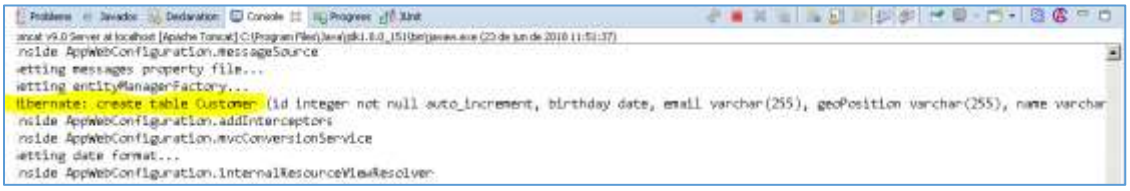
MYSQL 8

To create the database, table and insert the sample data, please do the following:

1. Login in MYSQL
2. Type “create database skipthedishes”

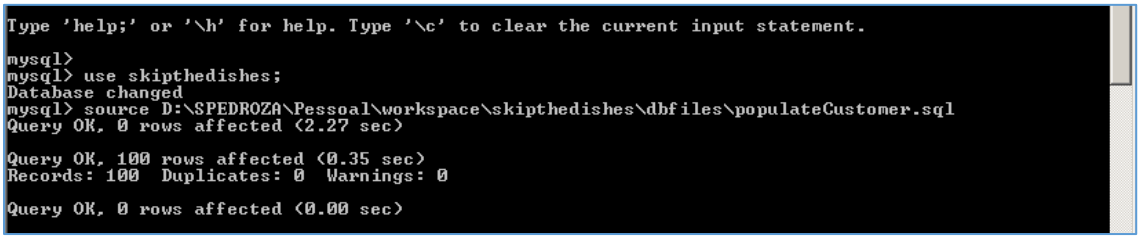


3. The application will create the tables automatically on start. You can check that in console:

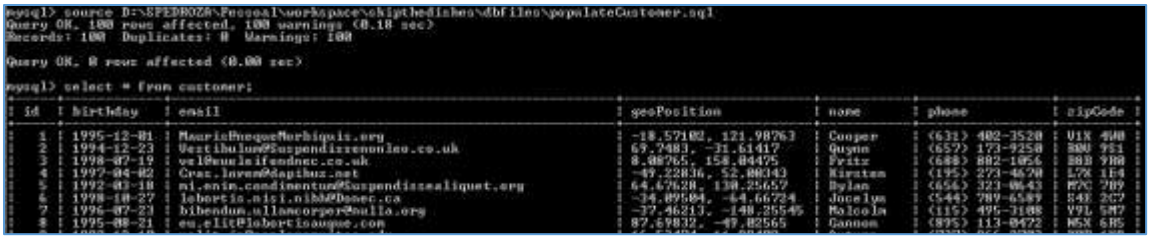


4. Run the *populateCustomer.sql* located in the dbfiles folder into the project. This script will create the customer table and load customer sample data:
   To run the script, type
   “source D:\SPEDROZA\Pessoal\workspace\skipthedishes\dbfiles\populateCustomer.sql”

   Replace “D:\SPEDROZA\Pessoal\workspace\skipthedishes\dbfiles\” according to your machine.



   To check inserted data, just type “select * from customer”:



Application

I used JAVA8 to compile my application, Spring MVC, Hibernate and JUNIT.

So, you will find some J8 features, for instance:

New java date api LocalDate

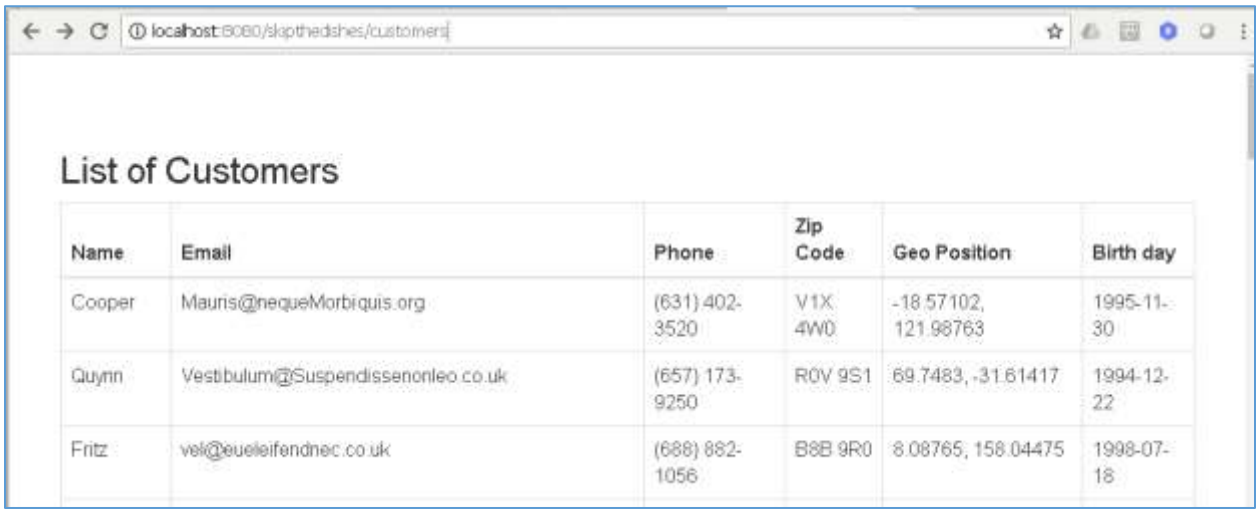
```
15  @Entity
16  public class Customer {
17
18      @Id
19      @GeneratedValue(strategy=GenerationType.IDENTITY)
20      private int id;
21
22      private String name;
23      private String email;
24      private String phone;
25      private String zipCode;
26      private String geoPosition;
27
28      @DateTimeFormat
29      private LocalDate birthday;
30
31      public int getId() {
32          return id;
33      }
```

Streams and lambda

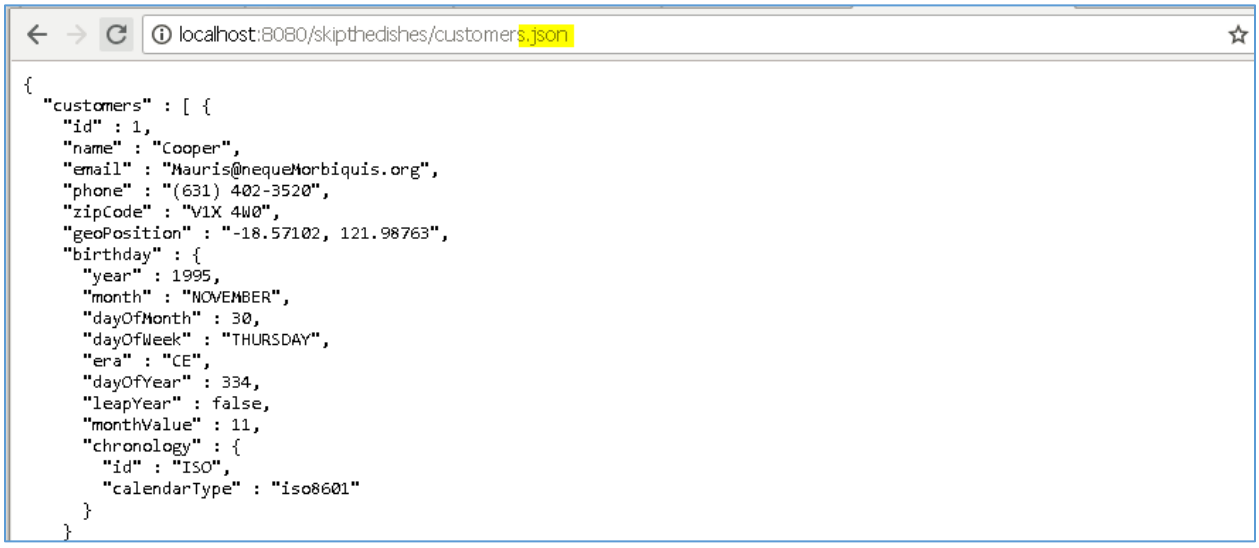
```
public ModelAndView find(String zipCode, String geoPosition) {
    System.out.println("Inside CustomerController.findAll");
    ModelAndView modelAndView = new ModelAndView("/customer/list");
    List<Customer> customers = customerDAO.findAll();
    System.out.println("List size before filter: " + customers.size());

    // check if the requester send any parameter
    if (zipCode != null && !zipCode.isEmpty()) {
        customers = customers.stream().filter(c -> c.getZipCode().equalsIgnoreCase(zipCode.trim()))
                .collect(Collectors.toList());
    }
    if (geoPosition != null && !geoPosition.isEmpty()) {
        customers = customers.stream().filter(c -> c.getGeoPosition().equalsIgnoreCase(geoPosition.trim()))
                .collect(Collectors.toList());
    }
```

I also used **internal resolvers** to return the requested response format. So, the requester can use the same URL to get html or json. Also, the developer doesn't have to maintain two different methods. It makes the application easy to maintain. Example testing both returns in a browser:

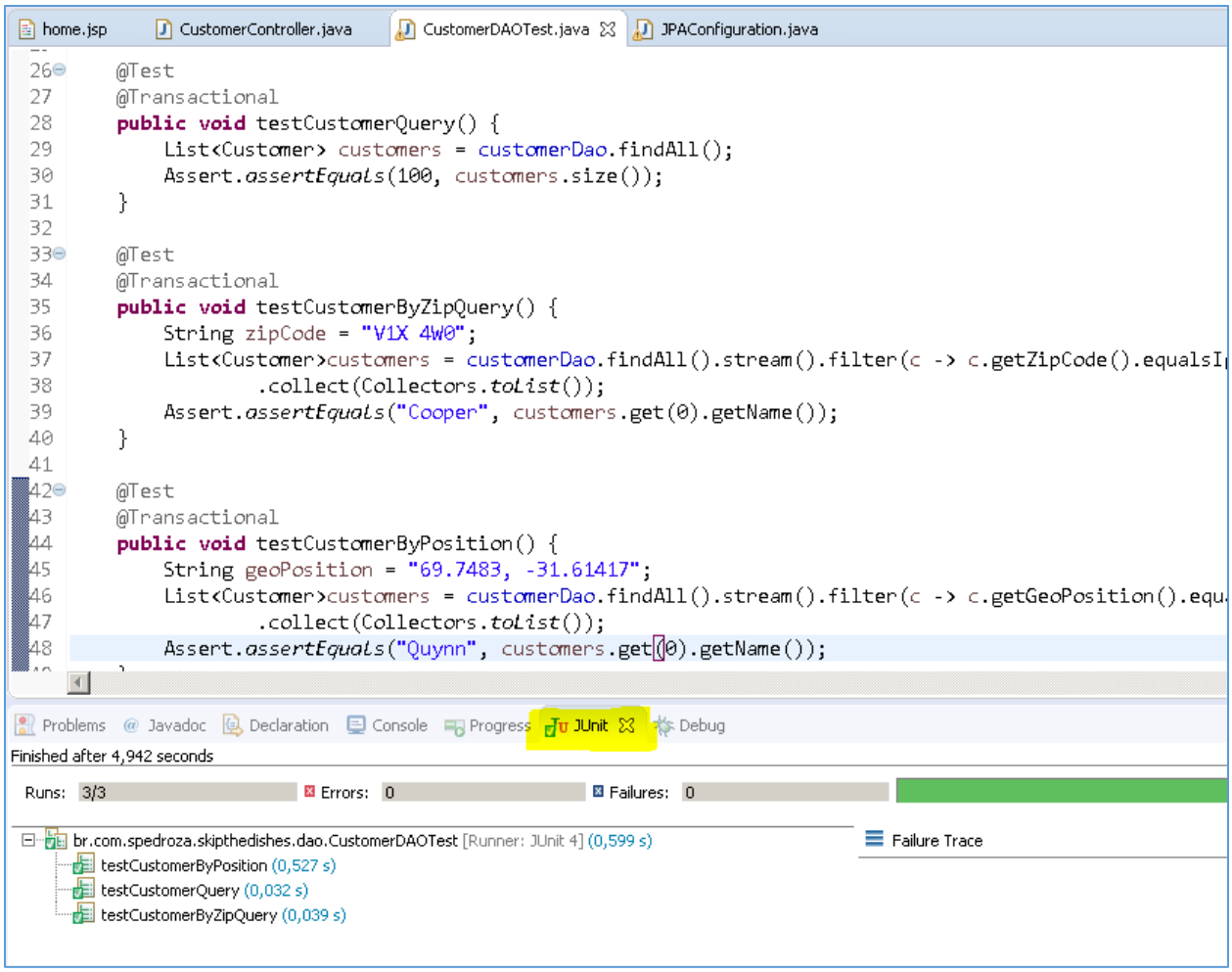
HTML



JSON

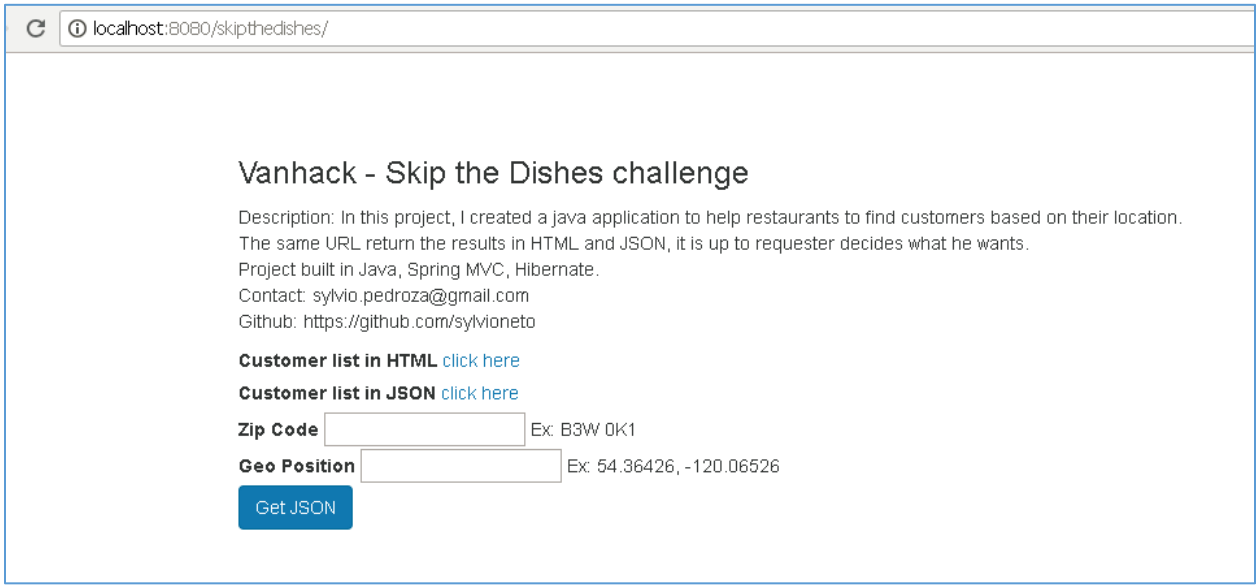


Finally, I wrote JUNIT classes to test the application. So, it can be used in Continuous Delivery workflow.



Testing

You can use a browser and the home page to test the application:



Or any tool you want: