

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

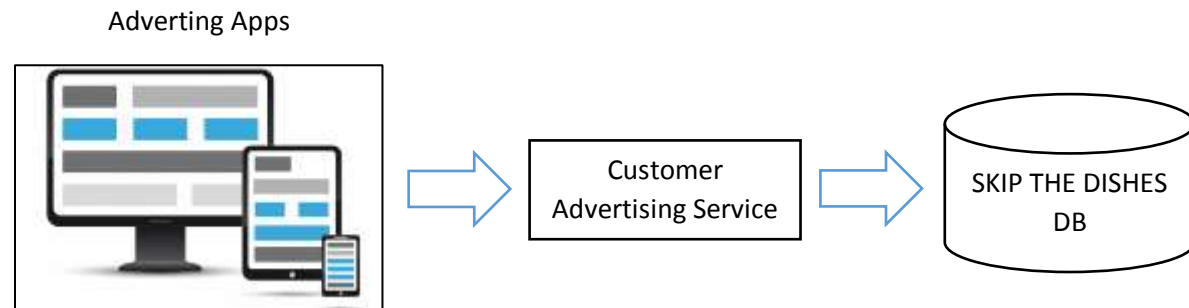
Introduction.....	2
Database	4
Application.....	5
Testing	7

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

```
← → ↻ ⓘ localhost:8080/skipthedishes/customers.json?zipCode=V1X%204W0

{
  "customers": [ {
    "id": 1,
    "name": "Cooper",
    "email": "Mauris@nequeMorbiquis.org",
    "phone": "(631) 402-3520",
    "zipCode": "V1X 4W0",
    "geoPosition": "-18.57102, 121.98763",
    "birthday": {
      "year": 1995,
      "month": "NOVEMBER",
      "dayOfMonth": 30,
      "dayOfWeek": "THURSDAY",
      "era": "CE",
      "dayOfYear": 334,
      "leapYear": false,
      "chronology": {
        "id": "ISO",
        "calendarType": "iso8601"
      }
    },
  },
}
```

Example 2: Using geo position

```
← → ↻ ⓘ localhost:8080/skipthedishes/customers.json?geoPosition=69.7483,%20-31.61417

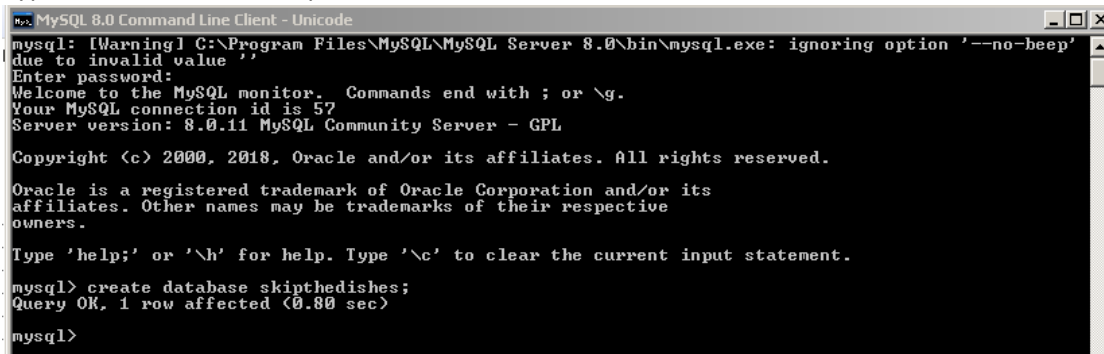
{
  "customers": [ {
    "id": 2,
    "name": "Quynn",
    "email": "Vestibulum@Suspendissenonleo.co.uk",
    "phone": "(657) 173-9250",
    "zipCode": "R0V 9S1",
    "geoPosition": "69.7483, -31.61417",
    "birthday": {
      "year": 1994,
      "month": "DECEMBER",
      "dayOfMonth": 22,
      "dayOfWeek": "THURSDAY",
      "era": "CE",
      "dayOfYear": 356,
      "leapYear": false,
      "chronology": {
        "id": "ISO",
        "calendarType": "iso8601"
      }
    },
    "monthValue": 12
  },
}
```

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”



```
mysql: [Warning] C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe: ignoring option '--no-beep' due to invalid value ''
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 57
Server version: 8.0.11 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database skipthedishes;
Query OK, 1 row affected (0.80 sec)

mysql>
```

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



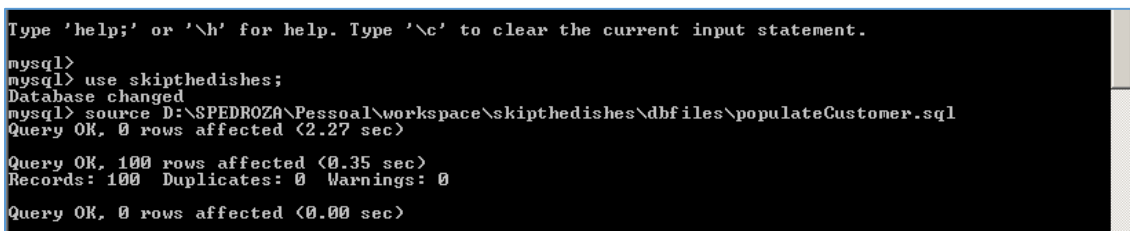
```
...
inside AppWebConfiguration.addInterceptors
inside AppWebConfiguration.mvcConversionService
setting date format...
inside AppWebConfiguration.InternalResourceViewResolver
...
Hibernate: create table Customer (id integer not null auto_increment, birthday date, email varchar(255), geoPosition varchar(255), name varchar(255))
...
inside AppWebConfiguration.addInterceptors
inside AppWebConfiguration.mvcConversionService
setting date format...
inside AppWebConfiguration.InternalResourceViewResolver
...
```

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.



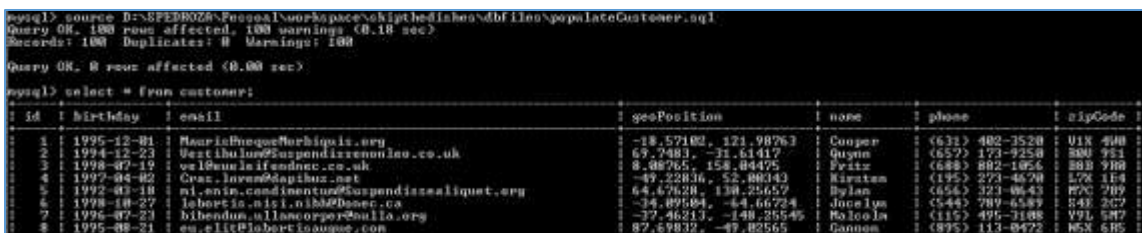
```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql> use skipthedishes;
Database changed
mysql> source D:\SPEDROZA\Pessoal\workspace\skipthedishes\dbfiles\populateCustomer.sql
Query OK, 0 rows affected (2.27 sec)

Query OK, 100 rows affected (0.35 sec)
Records: 100 Duplicates: 0 Warnings: 0

Query OK, 0 rows affected (0.00 sec)
```

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



```
mysql> source D:\SPEDROZA\Pessoal\workspace\skipthedishes\dbfiles\populateCustomer.sql
Query OK, 100 rows affected, 100 warnings (0.18 sec)
Records: 100 Duplicates: 0 Warnings: 100

Query OK, 0 rows affected (0.00 sec)

mysql> select * from customer;
+----+-----+-----+-----+-----+-----+
| id | birthday | email | geoPosition | name | phone | zipCode |
+----+-----+-----+-----+-----+-----+
| 1 | 1995-12-01 | Mauricio.goncalves@chipski.org | -18.57182, -121.98763 | Cooper | (632) 482-3520 | 018 400 |
| 2 | 1994-12-23 | Distribution@superdixennino.co.uk | 69.7483, -31.61417 | Gupta | (657) 173-9250 | 388 581 |
| 3 | 1998-07-19 | w10mule@fodnec.co.uk | 8.06765, 158.04475 | Fritz | (688) 882-1054 | 888 980 |
| 4 | 1997-04-02 | Crea.Loren@apthuz.net | -49.22836, 52.08043 | Kintan | (195) 273-4670 | 178 164 |
| 5 | 1992-03-18 | ni.enin.candiscentur@superdixenalliquet.org | 64.67620, 130.25657 | Bylan | (666) 323-0643 | 870 709 |
| 6 | 1998-10-27 | laberin.m@i.onibn00sec.ca | -24.09504, -64.66724 | Jones Jus | (544) 789-6589 | 540 352 |
| 7 | 1996-07-29 | b1bedun.allacorpor@nulla.org | -37.46213, -148.25545 | Molco la | (115) 495-3105 | 991 587 |
| 8 | 1995-08-21 | eu.elit@lobertisaupue.com | 87.69832, -49.02565 | Gannon | (895) 113-8472 | 858 685 |
| 9 | 1995-05-10 | ... | ... | ... | ... | ... |
| 10 | 1995-05-10 | ... | ... | ... | ... | ... |
+----+-----+-----+-----+-----+-----+
mysql>
```

Application

I used JAVA8 to compile my application. So, you will find some J8 features such as:

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     }
```

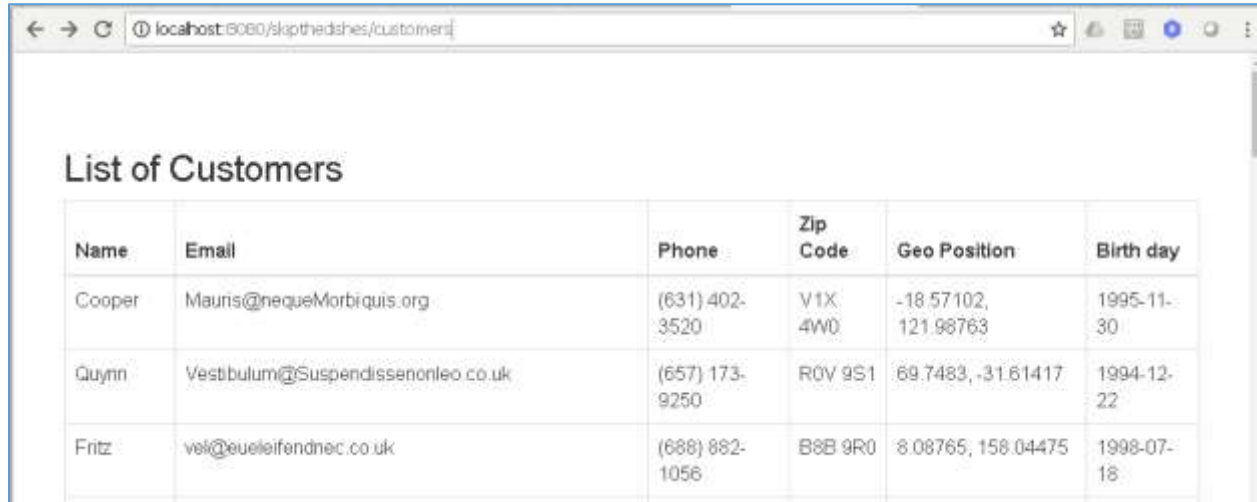
Streams and lambda

```
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:

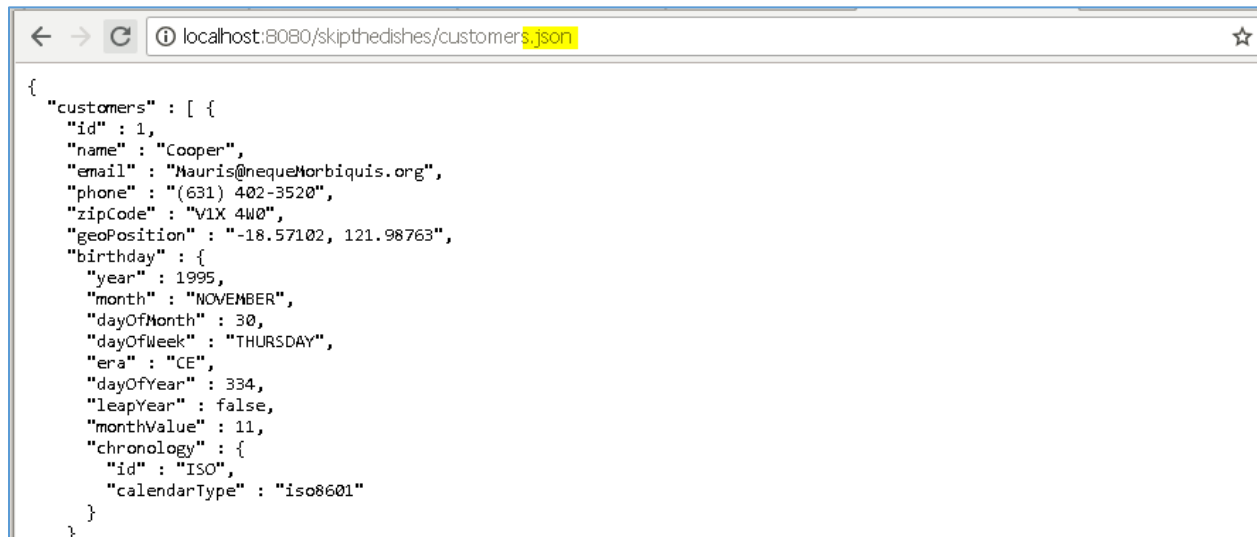
HTML



A screenshot of a web browser window. The address bar shows 'localhost:8080/skipthedishes/customers'. The page title is 'List of Customers'. Below the title is a table with 6 columns: Name, Email, Phone, Zip Code, Geo Position, and Birth day. The table contains 3 rows of customer data.

Name	Email	Phone	Zip Code	Geo Position	Birth day
Cooper	Mauris@nequeMorbiquis.org	(631) 402-3520	V1X 4W0	-18.57102, 121.98763	1995-11-30
Quynn	Vestibulum@Suspendissenonleo.co.uk	(657) 173-9250	R0V 9S1	69.7483, -31.61417	1994-12-22
Fritz	vel@eueleifendnec.co.uk	(688) 882-1056	B8B 9R0	8.08765, 158.04475	1998-07-18

JSON



A screenshot of a web browser window. The address bar shows 'localhost:8080/skipthedishes/customers.json'. The page displays a JSON object representing the customer data.

```
{
  "customers": [ {
    "id" : 1,
    "name" : "Cooper",
    "email" : "Mauris@nequeMorbiquis.org",
    "phone" : "(631) 402-3520",
    "zipCode" : "V1X 4W0",
    "geoPosition" : "-18.57102, 121.98763",
    "birthday" : {
      "year" : 1995,
      "month" : "NOVEMBER",
      "dayOfMonth" : 30,
      "dayOfWeek" : "THURSDAY",
      "era" : "CE",
      "dayOfYear" : 334,
      "leapYear" : false,
      "monthValue" : 11,
      "chronology" : {
        "id" : "ISO",
        "calendarType" : "iso8601"
      }
    }
  }
}
```

Testing

You can use the home page in order to test the application:

localhost:8080/skipthedishes/

Vanhack - Skip the Dishes challenge

Description: In this project, I created a java application to help restaurants to find customers based on their location. The same URL return the results in HTML and JSON, it is up to requester decides what he wants.
Project built in Java, Spring MVC, Hibernate.
Contact: sylvio.pedroza@gmail.com
Github: <https://github.com/sylvioneto>

Customer list in HTML [click here](#)
Customer list in JSON [click here](#)

Zip Code Ex: B3W 0K1
Geo Position Ex: 54.36426, -120.06526

[Get JSON](#)

Or any tool you want:

