Spring Boot: Upload & Read CSV file into H2 Database

I. Introduction:

In this documentation, we are going to create a project which uploads and reads a dataset (**CSV file**) into a database and analyze queries given by the user and return results.

II. Technologies:

- Java 8
- Spring Boot (with Spring Web MVC, Spring Data JPA)
- Maven
- Apache Commons CSV
- H2 Database

III. Spring Boot Rest APIs for uploading CSV Files:

-As we know, we have a **.csv** file that contains Orders data as following:

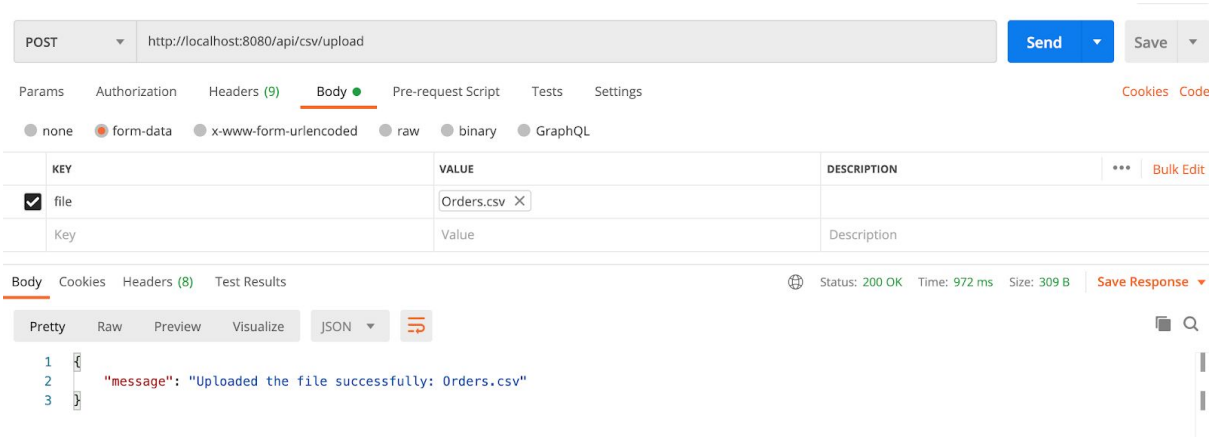
```
Order ID,Order Date,Order Quantity,Sales,Ship Mode,Profit,Unit
Price,Customer Name,Customer Segment,Product Category
3,10-13-2010,6,261.54,Regular Air,-213.25,38.94,Muhammed MacIntyre,Small
Business,Office Supplies
6,02-20-2012,2,6.93,Regular Air,-4.64,2.08,Ruben Dartt,Corporate,Office
Supplies
```

```
32,07-15-2011,26,2808.08,Regular Air,1054.82,107.53,Liz
Pelletier,Corporate,Furniture
32,07-15-2011,24,1761.4,Delivery Truck,-1748.56,70.89,Liz
Pelletier,Corporate,Furniture
32,07-15-2011,23,160.2335,Regular Air,-85.129,7.99,Liz
Pelletier,Corporate,Technology
```

-We're going create a Spring Boot Application that provides APIs for:

- uploading CSV File and storing data in H2 Database
- getting a list of orders from H2 table
- Getting Order By Customer Name from H2 table
- Getting Order By Customer Name and Order Date from H2 table

-After the CSV file is uploaded successfully:

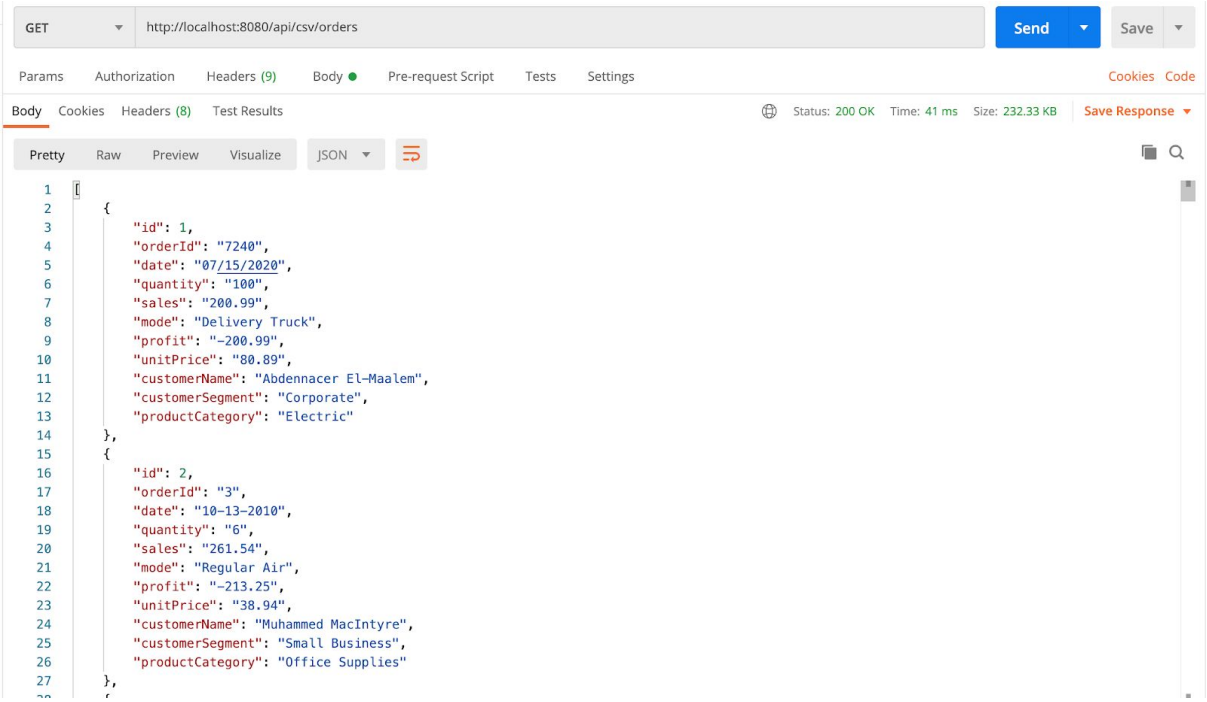


Orders table in H2 database will look like this:

select * from orders;

ID	CUSTOMER_NAME	CUSTOMER_SEGMENT	DATE	MODE	ORDER_ID	PRODUCT_CATEGORY	PROFIT	QUANTITY	SALES	UNIT_PRICE
1	Abdennacer El-Maalem	Corporate	07/15/2020	Delivery Truck	7240	Electric	-200.99	100	200.99	80.89
2	Muhammed MacIntyre	Small Business	10-13-2010	Regular Air	3	Office Supplies	-213.25	6	261.54	38.94
3	Ruben Dartt	Corporate	02-20-2012	Regular Air	6	Office Supplies	-4.64	2	6.93	2.08
4	Liz Pelletier	Corporate	07-15-2011	Regular Air	32	Furniture	1054.82	26	2808.08	107.53
5	Liz Pelletier	Corporate	07-15-2011	Delivery Truck	32	Furniture	-1748.56	24	1761.4	70.89
6	Liz Pelletier	Corporate	07-15-2011	Regular Air	32	Technology	-85.129	23	160.2335	7.99
7	Liz Pelletier	Corporate	07-15-2011	Regular Air	32	Technology	-128.38	15	140.56	8.46
8	Julie Creighton	Corporate	10-22-2011	Regular Air	35	Office Supplies	60.72	30	288.56	9.11
9	Julie Creighton	Corporate	10-22-2011	Regular Air	35	Technology	48.987	14	1892.848	155.99
10	Sample Company A	Home Office	11-02-2011	Regular Air	36	Technology	657.477	46	2484.7455	65.99
11	Tamara Dahlen	Corporate	03-17-2011	Regular Air	65	Technology	1470.3	32	3812.73	115.79
12	Liz Pelletier	Corporate	07-15-2008	Regular Air	32	Furniture		26		107.53
13	Liz Pelletier	Corporate	07-15-2008	Delivery Truck	32	Furniture		24		70.89
14	Liz Pelletier	Corporate	07-15-2008	Regular Air	32	Technology		23		7.99
15	Liz Pelletier	Corporate	07-15-2008	Regular Air	32	Technology		15		8.46
16	Julie Creighton	Corporate	10-22-2008	Regular Air	35	Office Supplies		30		9.11
17	Julie Creighton	Corporate	10-22-2008	Regular Air	35	Technology		14		155.99
18	Sample Company A	Home Office	10-22-2008	Regular Air	36	Technology		46		65.99
19	Tamara Dahlen	Corporate	10-22-2008	Regular Air	65	Technology		32		115.79
20	Arthur Gainer	Consumer	01-19-2009	Regular Air	66	Office Supplies	7.57	41	108.15	2.88

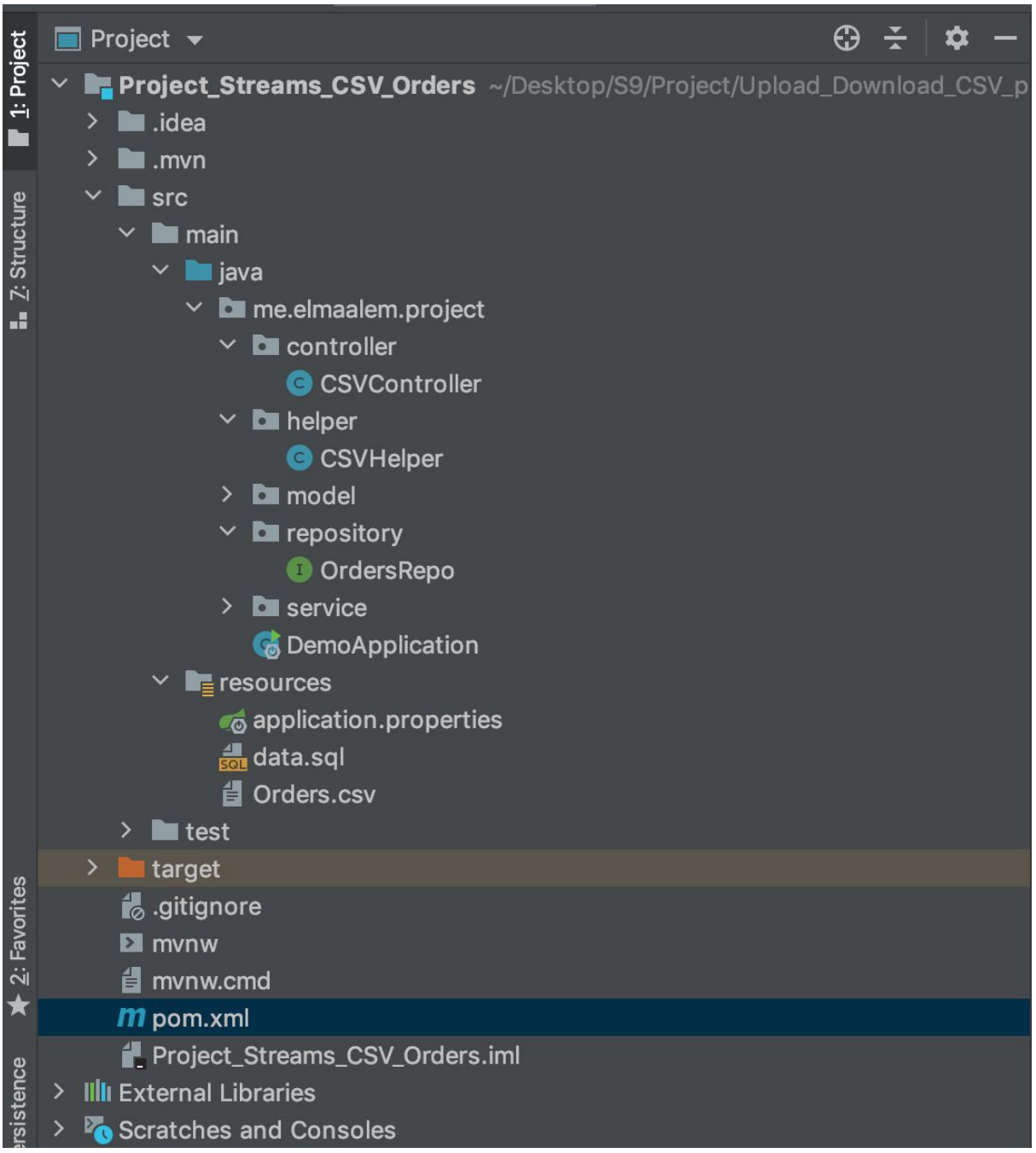
If we get a list of Orders, the Spring Rest-API will return:



-These are APIs to be exported:

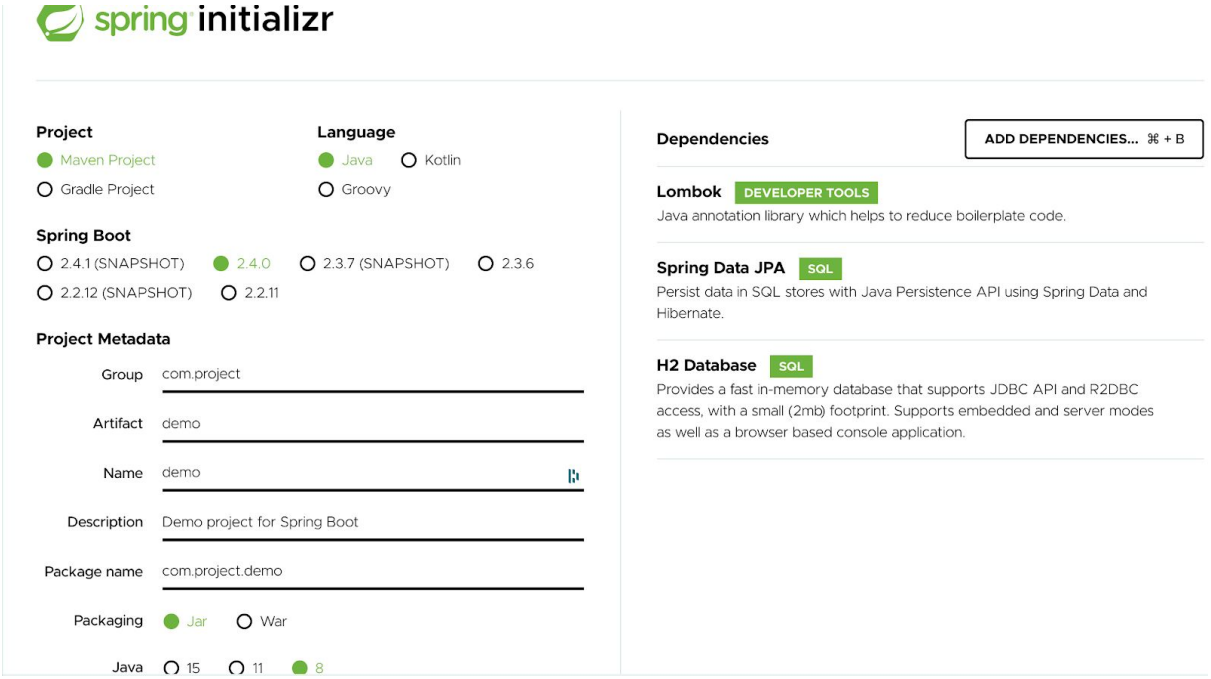
Methods	URL	Definition
POST	/api/csv/upload	upload a CSV File
GET	/api/csv/orders	get a List of orders in H2 table
GET	/api/csv/orders/findByCustomerName/{customerName}	get orders by customer name from H2 table
GET	/api/csv/orders/findByCustomerNameAndDate/{customerName}/{orderDate}	get orders by customer name and date of order from H2 table
GET	/api/csv/orders/findByProductCategoryAndMustHaveProfitPositive/{productCategory}	Get orders by product category when the profit was greater than 0

IV. Project Structure:



V. Setup Spring Boot CSV File Upload project:

In the first step, we use spring initializr to generate the first configuration of the project:



Use this link to get this configuration: here

Finally, we get this file **pom.xml:**

```
    </parent>
    <groupId>me.elmaalem</groupId>
    <artifactId>Project_Streams_CSV_Orders</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Project_Streams_CSV_Orders</name>
    <description>Spring Boot: Upload And Read CSV file into H2 Database</descriptio

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-csv</artifactId>
            <version>1.8</version>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
```

VI. Configure Spring Datasource, JPA, Hibernate

Under **src/main/resources** folder, open **application.properties** and add this code:

```
spring.datasource.url= jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.h2.console.enabled=true
spring.datasource.username= sa
spring.datasource.password=

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.H2Dialect
```

```
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto= update
```

VII. Define Data Model

In the **model** package, we define **Orders** class.
model/Orders.class:

```
package me.elmaalem.project.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Orders {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String orderId;
    private String date;
    private String quantity;
    private String sales;
    private String mode;
    private String profit;
    private String unitPrice;
    private String customerName;
    private String customerSegment;
    private String productCategory;

   public Orders(String orderId, String date, String quantity, String
sales, String mode, String profit, String unitPrice, String
customerName, String customerSegment, String productCategory) {
        this.orderId = orderId;
```

```
        this.date = date;
        this.quantity = quantity;
        this.sales = sales;
        this.mode = mode;
        this.profit = profit;
        this.unitPrice = unitPrice;
        this.customerName = customerName;
        this.customerSegment = customerSegment;
        this.productCategory = productCategory;
    }
  }
```

VIII. Create a Data Repository for working with Database:

Let's create a **repository** to interact with **Orders** from the database.
In the repository package, create an **OrdersRepo** interface that extends **JpaRepository**.

repository/OrdersRepo.java:

```
package me.elmaalem.project.repository;

import me.elmaalem.project.model.Orders;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface OrdersRepo extends JpaRepository<Orders,Long> {
}
```

IX. Implement Read/Write CSV Helper Class:

Under **helper** package, we create **CSVHelper** class with 3 methods:

- **hasCSVFormat()**: check if a file has CSV format or not
- **csvToOrders()**: read **InputStream** of a file, return a list of Orders

We're gonna use **Apache Commons CSV** classes such as: **CSVParser**, **CSVRecord**, **CSVFormat**.
Here is the class **helper/CSVHelper.java**:

```
package me.elmaalem.project.helper;
```

```java
import me.elmaalem.project.model.Orders;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

@Component
public class CSVHelper {
    public static String TYPE = "text/csv";
    static String[] HEADERs = { "Order ID","Order Date","Order
Quantity","Sales","Ship Mode","Profit","Unit Price","Customer
Name","Customer Segment","Product Category" };

    public static boolean hasCSVFormat(MultipartFile file) {
        if (!TYPE.equals(file.getContentType())) {
            return false;
        }
        return true;
    }

    public static List<Orders> csvToOrders(InputStream is) {
        try (BufferedReader fileReader = new BufferedReader(new
InputStreamReader(is, "UTF-8"));
             CSVParser csvParser = new CSVParser(fileReader,

CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreHeaderCase().withT
rim());){

            List<Orders> orders = new ArrayList<Orders>();

            Iterable<CSVRecord> csvRecords = csvParser.getRecords();

            for (CSVRecord csvRecord : csvRecords) {
                Orders order = new Orders(
                        Long.parseLong(csvRecord.get(0)),
                        csvRecord.get(1),
                        Integer.parseInt(csvRecord.get(2)),
                        Double.parseDouble( !csvRecord.get(3).isEmpty()
```

```
? csvRecord.get(3) : "0.00"),
                        csvRecord.get(4),
                        Double.parseDouble( !csvRecord.get(5).isEmpty()
? csvRecord.get(5) : "0.00"),
                        Double.parseDouble( !csvRecord.get(6).isEmpty()
? csvRecord.get(6) : "0.00"),
                        csvRecord.get(7),
                        csvRecord.get(8),
                        csvRecord.get(9)
                );
                orders.add(order);
            }
            return orders;
        } catch (IOException e) {
            throw new RuntimeException("fail to parse CSV file: " +
e.getMessage());
        }
    }
}
```

X. Create CSV File Service:

CSVService class uses **CSVHelper** and **OrdersRepository** for 4 functions:

→ **save(MultipartFile file)**: store CSV data to database
→ **getAllOrders()**: read data from database and return **List<Orders>**
→ **getOrdersByCustomerName(String customerName)**: select orders which contain the string of customer name and return **List<Orders>.**
→ **getOrdersByCustomerNameAndDate(String name,String date):** ……

Here is the code of **service/CSVService.java:**

```
package me.elmaalem.project.service;

import me.elmaalem.project.helper.CSVHelper;
import me.elmaalem.project.model.Orders;
import me.elmaalem.project.repository.OrdersRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
```

```java
import java.util.List;
import java.util.stream.Collectors;

@Service
public class CSVService {

    @Autowired
    OrdersRepo repository;

    public void save(MultipartFile file) {
        try {
            List<Orders> orders =
CSVHelper.csvToOrders(file.getInputStream());
            repository.saveAll(orders);
        } catch (IOException e) {
            throw new RuntimeException("fail to store csv data: " +
e.getMessage());
        }
    }

    public List<Orders> getAllOrders() {
        return repository.findAll();
    }


    public List<Orders> getOrdersByCustomerName(String customerName){

        return repository.findAll().stream()

.filter(s->s.getCustomerName().contentEquals(customerName))
                .collect(Collectors.toList());
    }

    public List<Orders> getOrdersByCustomerNameAndDate(String
name,String date) {


        return repository.findAll().stream()
                .filter(s->s.getCustomerName().contentEquals(name))
                .filter(s->s.getDate().contentEquals(date))
                .collect(Collectors.toList());
    }
}
```

XI. Create Controller for Upload CSV Files:

In the controller package, we create a **CSVController** class for **RestAPIs**.
– **@CrossOrigin** is for configuring allowed origins.
– **@Controller** annotation indicates that this is a controller.
– **@GetMapping** and **@PostMapping** annotation is for mapping **HTTP GET** & **POST** requests.

– We use **@Autowired** to inject implementation of **CSVService** bean to local variables.

controller/CSVController.java:

```
package me.elmaalem.project.controller;

import me.elmaalem.project.helper.CSVHelper;
import me.elmaalem.project.model.Orders;
import me.elmaalem.project.service.CSVService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import java.util.List;

@CrossOrigin("http://localhost:8080")
@Controller
@RequestMapping("/api/csv")
public class CSVController {

        @Autowired
        CSVService fileService;

        @PostMapping("/upload")
        public ResponseEntity<String> uploadFile(@RequestParam("file")
MultipartFile file) {
            String message = "";

            if (CSVHelper.hasCSVFormat(file)) {
                fileService.save(file);

                try {
                    fileService.save(file);
                    message = "Uploaded the file successfully: " +
file.getOriginalFilename();
                    return ResponseEntity.status(HttpStatus.OK).body(
"\" message \": \" "+ message +" \"");
```

```
                } catch (Exception e) {
                    message = "Could not upload the file: " +
file.getOriginalFilename() + "!";
                    return
ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body("\" message
\": \" "+ message +" \"");
                }
            }
            message = "Please upload a csv file!";
            return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body("\" message \": \" "+
message +" \"");
        }
    //1-SELECT * FROM Order;
    @GetMapping("/orders")
    public ResponseEntity<List<Orders>> getAllOrders () {
        try {
            List<Orders> orders = fileService.getAllOrders();

            if (orders.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NO_CONTENT);
            }

            return new ResponseEntity<List<Orders>>(orders,
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
    //2-Stream :Get Order By Customer Name
    @GetMapping(value = "/orders/findByCustomerName/{customerName}")
    public ResponseEntity<List<Orders>> getOrdersByCustomerName
(@PathVariable String customerName) {
        try {
            List<Orders> orders =
fileService.getOrdersByCustomerName(customerName);

            if (orders.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NO_CONTENT);
            }

            return new ResponseEntity<List<Orders>>(orders,
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null,
```

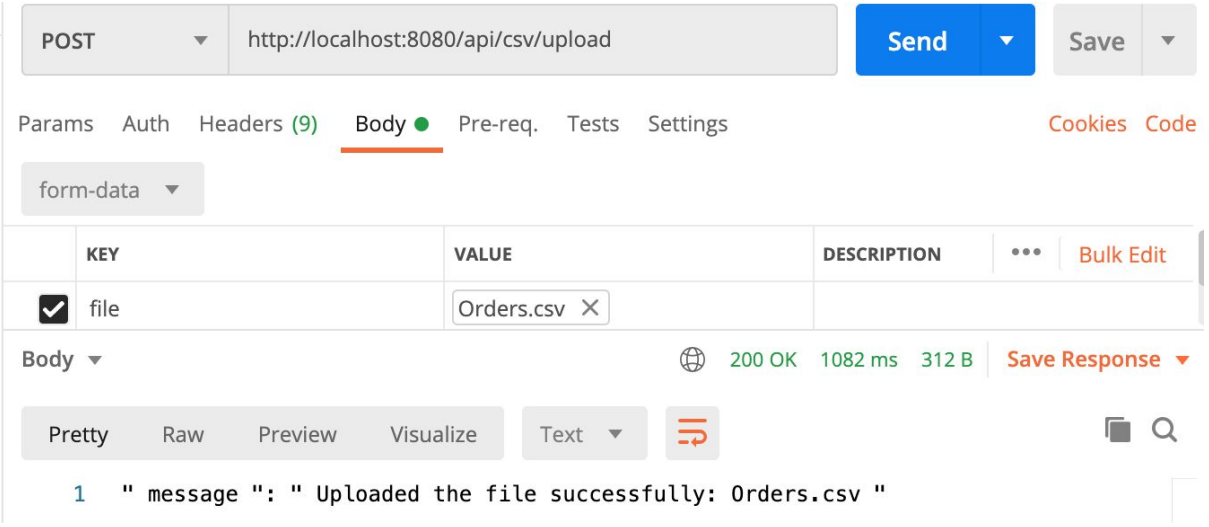
```
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
    //3-Stream :Get Order By Customer Name and Order Date
    @GetMapping("/orders/findByCustomerNameAndDate/{name}/{orderDate}")
    public ResponseEntity<List<Orders>> getOrdersByCustomerNameAndDate
(@PathVariable String customerName,@PathVariable String orderDate) {
        try {
            List<Orders> orders =
fileService.getOrdersByCustomerNameAndDate(customerName,orderDate);

            if (orders.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NO_CONTENT);
            }

            return new ResponseEntity<List<Orders>>(orders,
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

XII. Run & Check:

Let's use **Postman** to make some **requests**.



GET http://localhost:8080/api/csv/orders Send Save

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies Code

Body 200 OK 1146 ms 445.74 KB Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "orderId": 7240,
        "date": "07-15-2020",
        "quantity": 100,
        "sales": 200.99,
        "mode": "Delivery Truck",
        "profit": -200.99,
        "unitPrice": 80.89,
        "customerName": "Abdennacer El-Maalem",
        "customerSegment": "Corporate",
        "productCategory": "Electric"
    },
    {
        "id": 2,
        "orderId": 3,
        "date": "10-13-2010",
        "quantity": 6,
        "sales": 261.54,
        "mode": "Regular Air",
        "profit": -213.25,
        "unitPrice": 38.94,
        "customerName": "Muhammed MacIntyre",
        "customerSegment": "Small Business",
        "productCategory": "Office Supplies"
    },
    {
```

GET http://localhost:8080/api/csv/orders/findByCustomerName/Muhammed M... Send Save

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies Code

Body Cookies Headers (8) Test Results 200 OK 75 ms 721 B Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 2,
        "orderId": 3,
        "date": "10-13-2010",
        "quantity": 6,
        "sales": 261.54,
        "mode": "Regular Air",
        "profit": -213.25,
        "unitPrice": 38.94,
        "customerName": "Muhammed MacIntyre",
        "customerSegment": "Small Business",
        "productCategory": "Office Supplies"
    },
    {
        "id": 1009,
        "orderId": 3,
        "date": "10-13-2010",
        "quantity": 6,
        "sales": 261.54,
        "mode": "Regular Air",
        "profit": -213.25,
        "unitPrice": 38.94,
        "customerName": "Muhammed MacIntyre",
        "customerSegment": "Small Business",
        "productCategory": "Office Supplies"
    }
```

GET http://localhost:8080/api/csv/orders/findByCustomerNameAndDate/Muhammed MacIntyre/10-13-2010

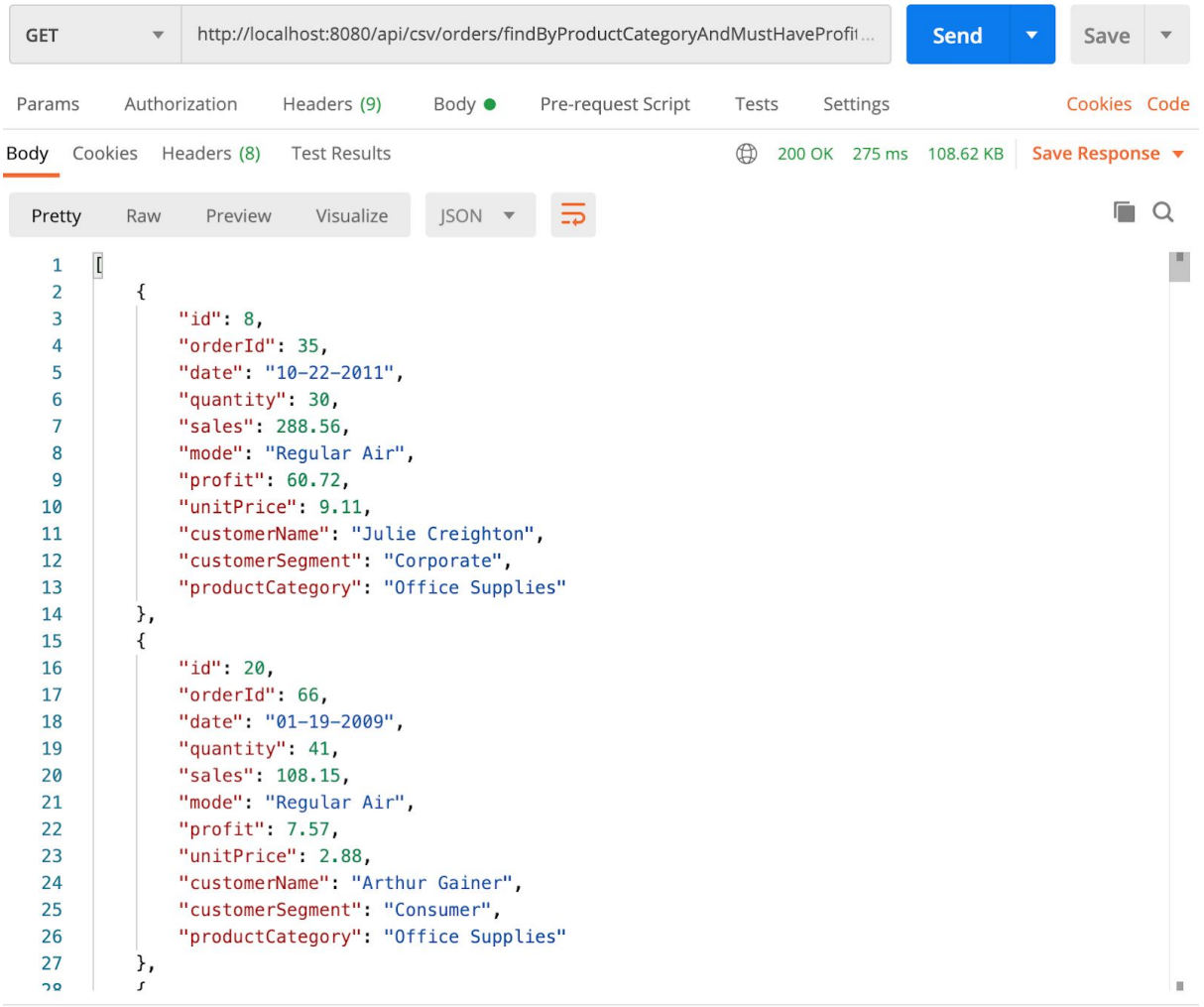
Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

Body Cookies Headers (8) Test Results 200 OK 459 ms 721 B Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 2,
        "orderId": 3,
        "date": "10-13-2010",
        "quantity": 6,
        "sales": 261.54,
        "mode": "Regular Air",
        "profit": -213.25,
        "unitPrice": 38.94,
        "customerName": "Muhammed MacIntyre",
        "customerSegment": "Small Business",
        "productCategory": "Office Supplies"
    },
    {
        "id": 1009,
        "orderId": 3,
        "date": "10-13-2010",
        "quantity": 6,
        "sales": 261.54,
        "mode": "Regular Air",
        "profit": -213.25,
        "unitPrice": 38.94,
        "customerName": "Muhammed MacIntyre",
        "customerSegment": "Small Business",
        "productCategory": "Office Supplies"
    }
]
```



XIII. Conclusion:

Thank you for reading! If you enjoyed it.

If you want to test the examples above, you will find my Github code link:
[Upload & Read CSV file into database]