Database Frameworks – Spring Data Exam

Alara Restaurant

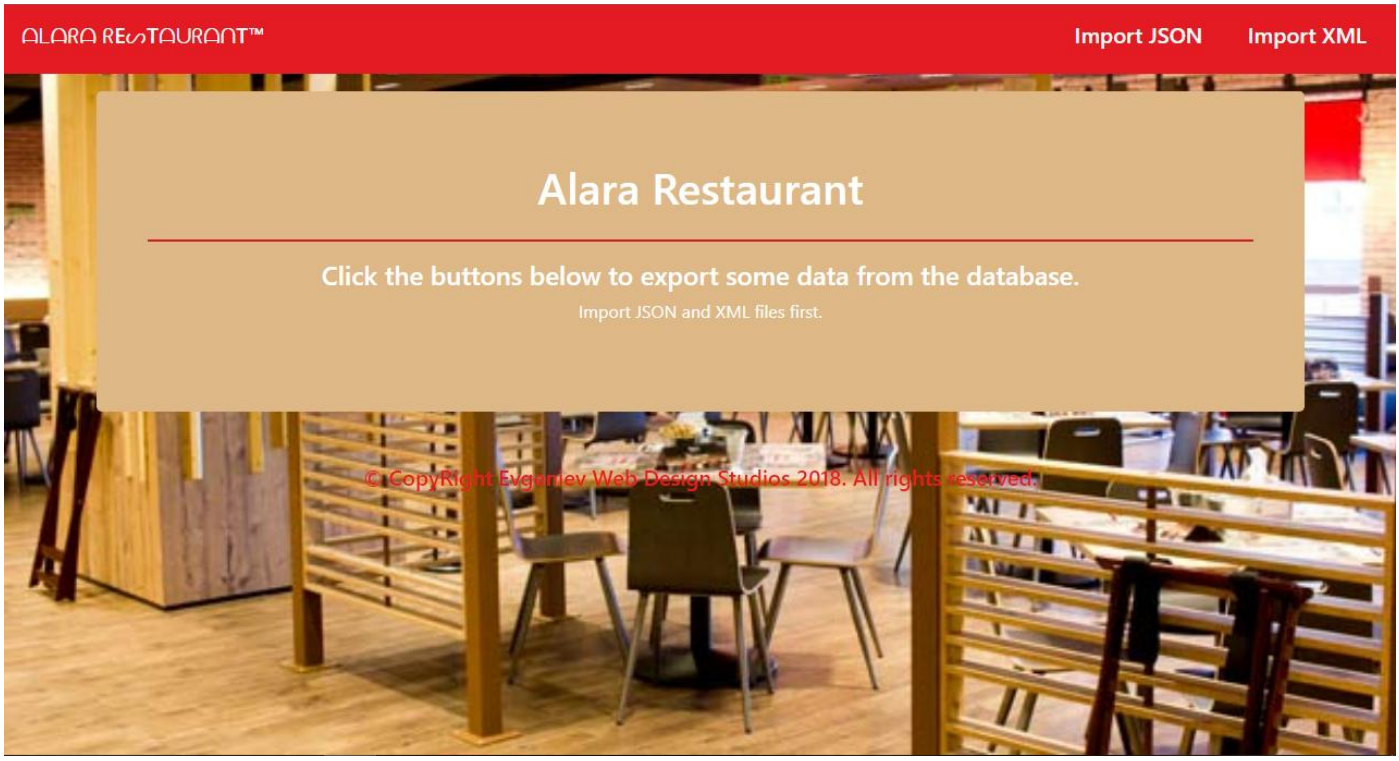
On the planet Alara, one of the first colonists and great friend of yours, decided to establish a restaurant. The restaurant is named after the planet – Alara Restaurant. You have been asked by your mate to finish the database layer, which supports basic functionality like importing JSON and XML data and exporting some results.

1. Functionality Overview

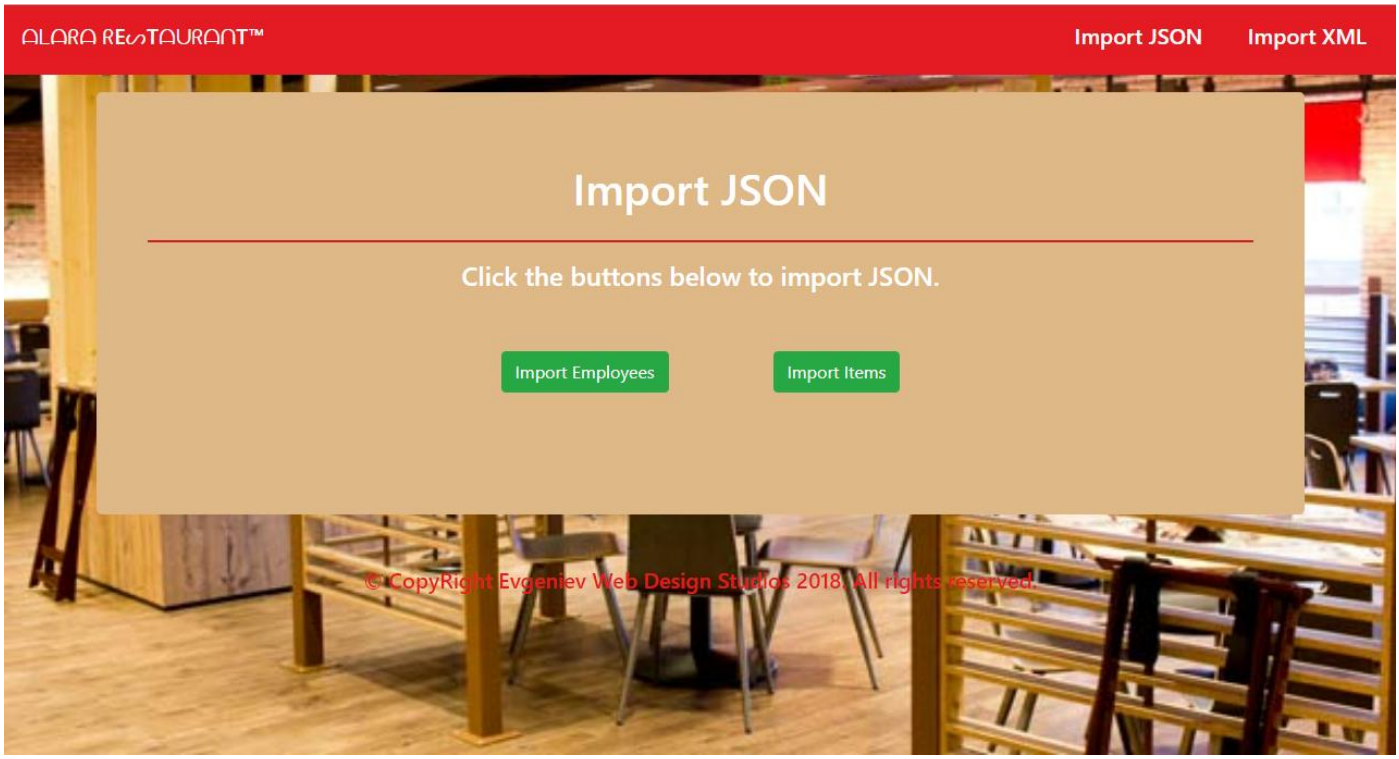
After finishing the Colonial Council Bank application, your friend has asked you to implement the **database layer**. The application should be able to easily **import** hard-formatted data from **XML** and **JSON** and **support functionality** for also **exporting** the imported data. The application is called – **Alara Restaurant**.

Look at the pictures below to see what must happen:

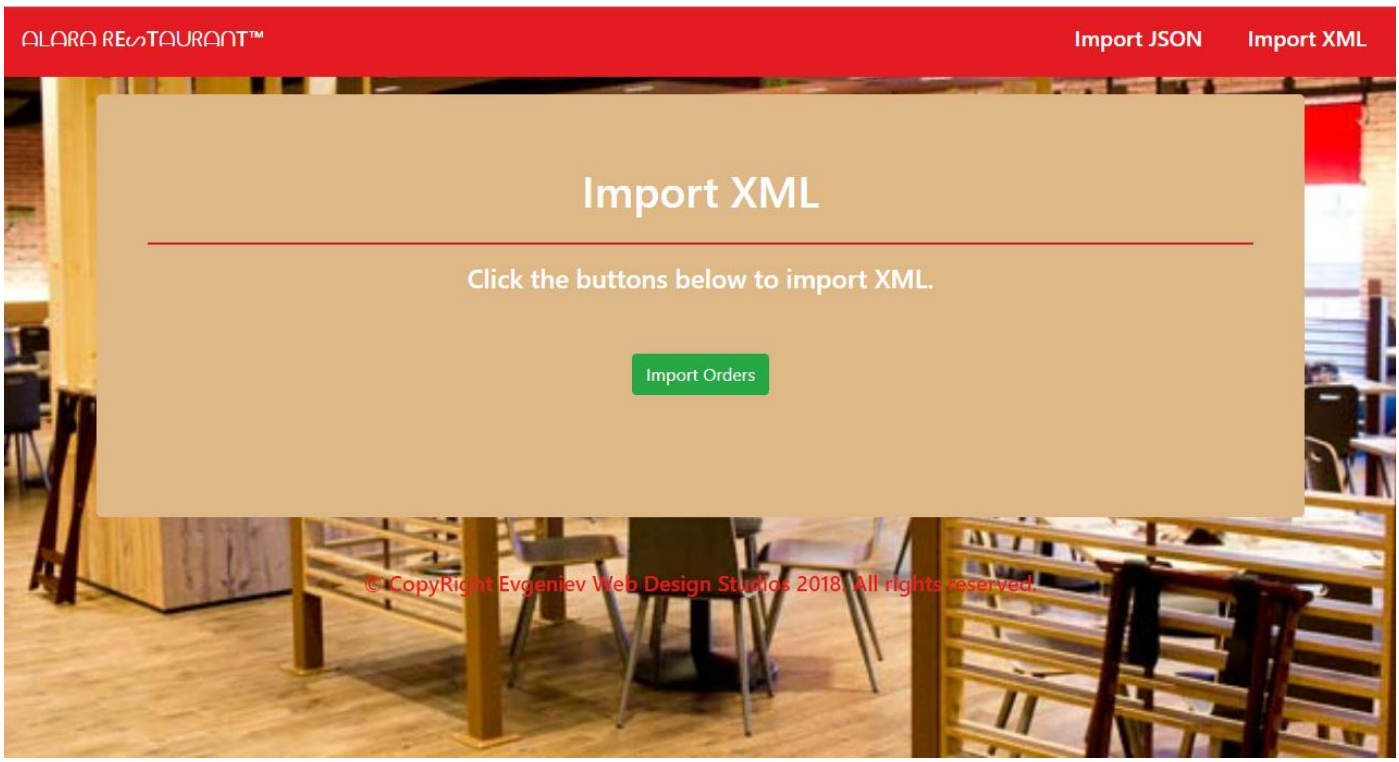
- Home page before importing anything:



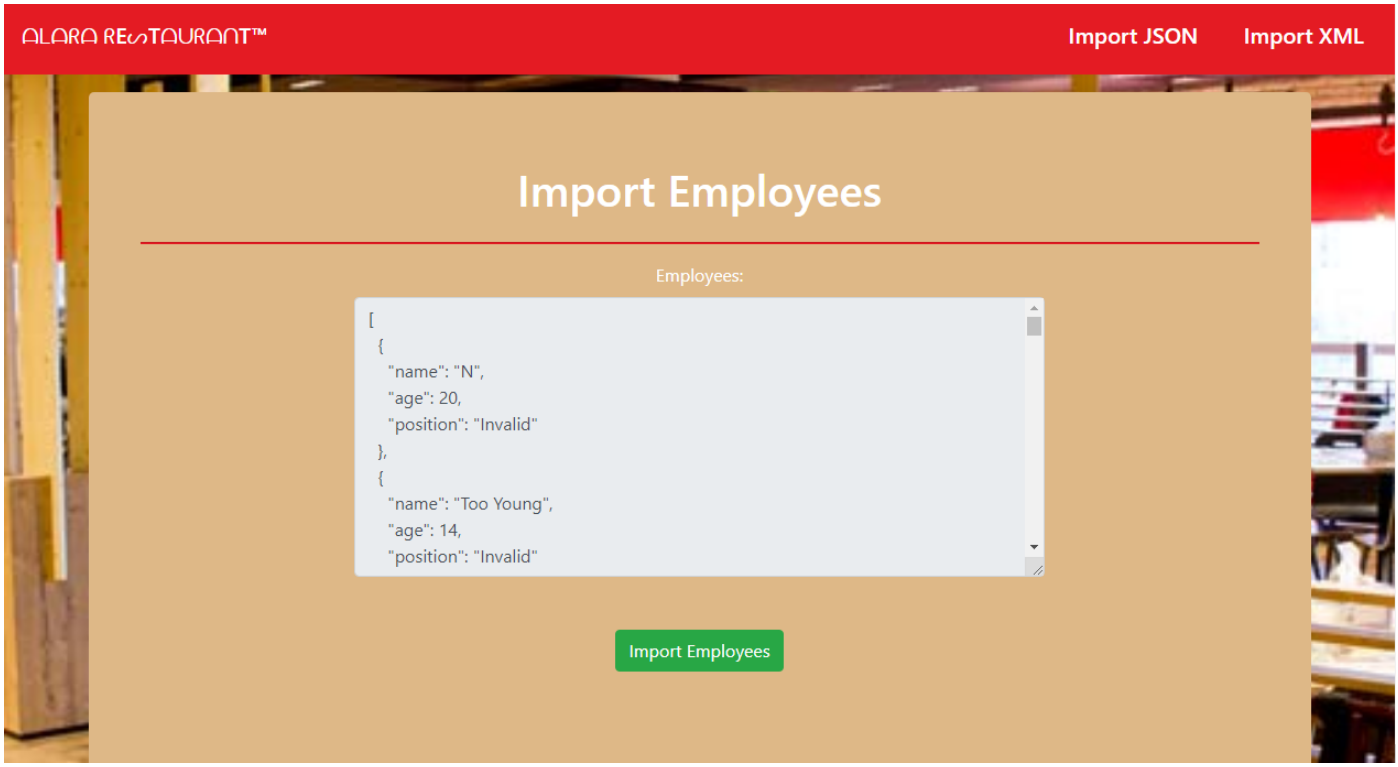
- Import JSON page before importing anything:



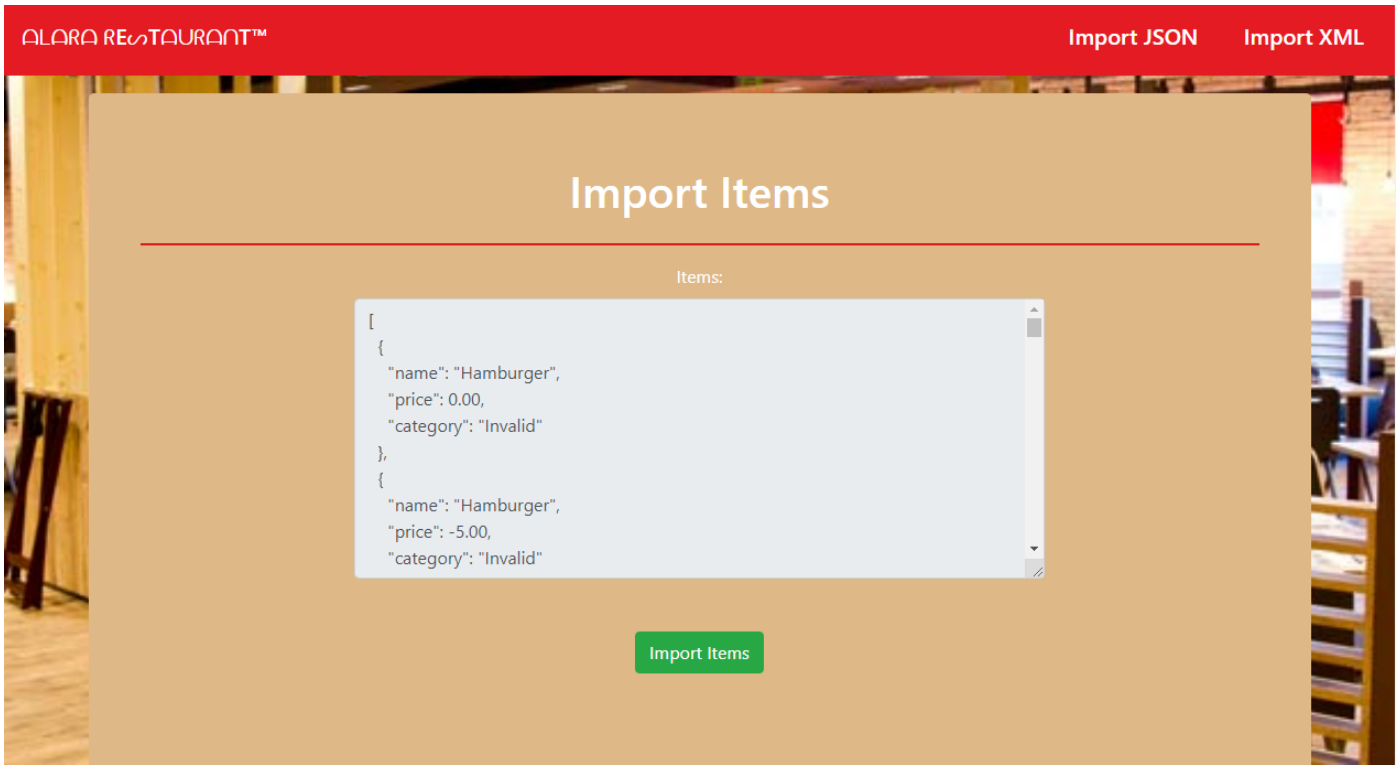
- Import XML page before importing anything:



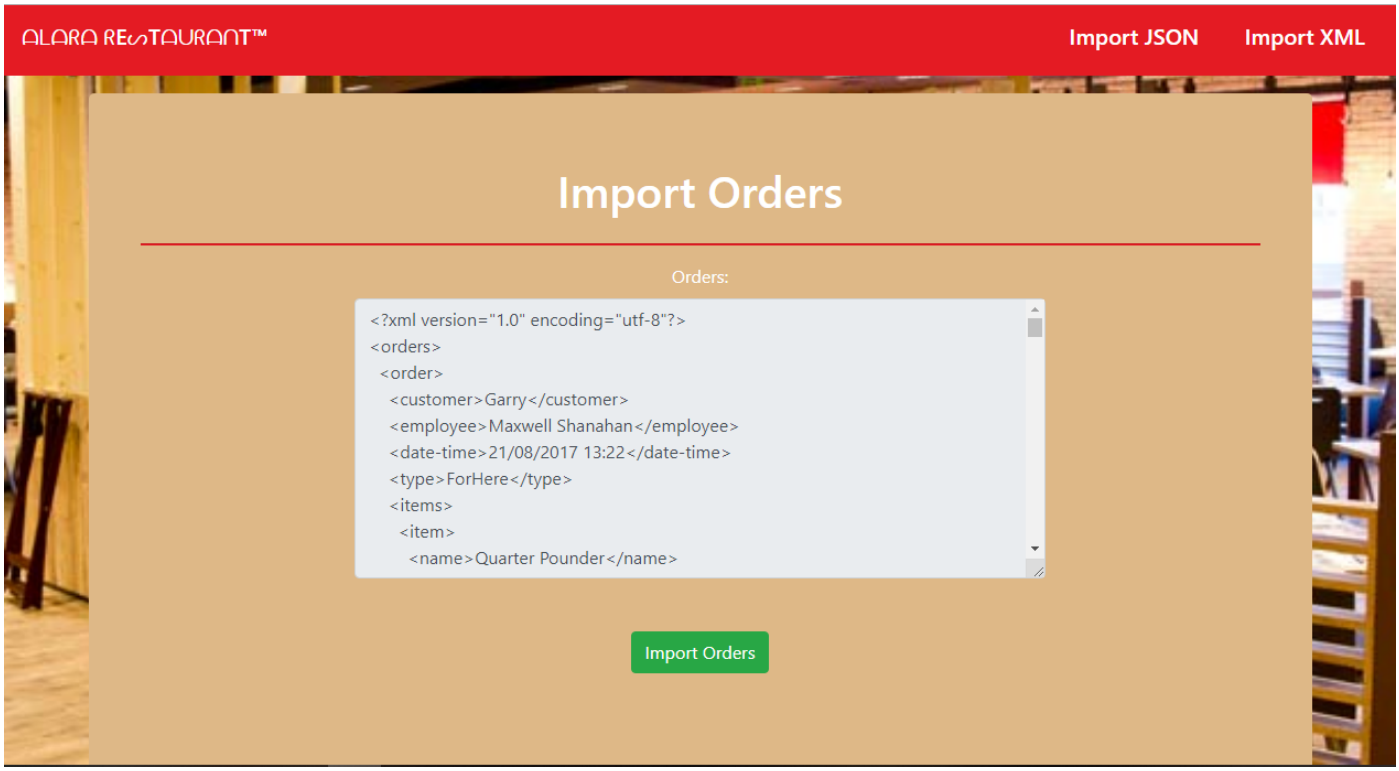
- Import Employees page after reading the **employees.json** file:



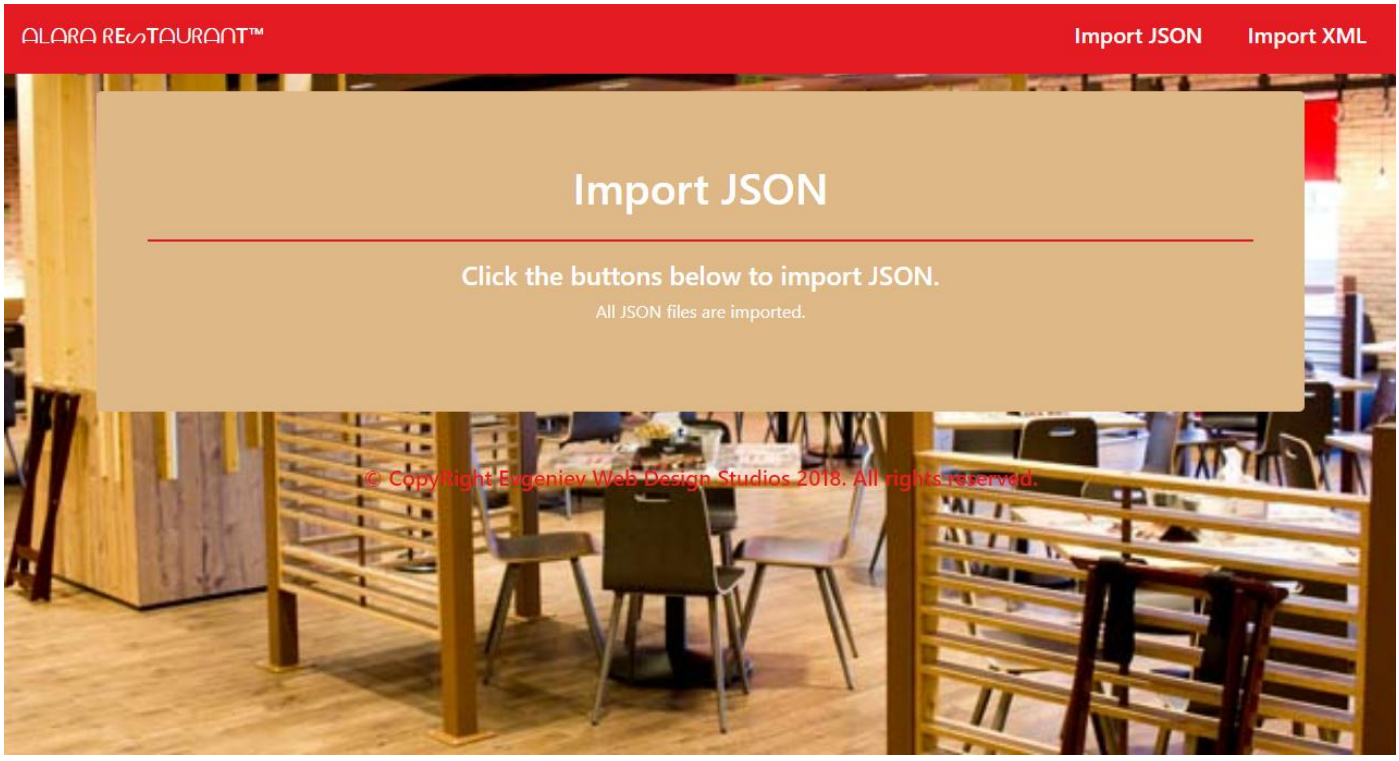
- Import Items page after reading the **items.json** file:



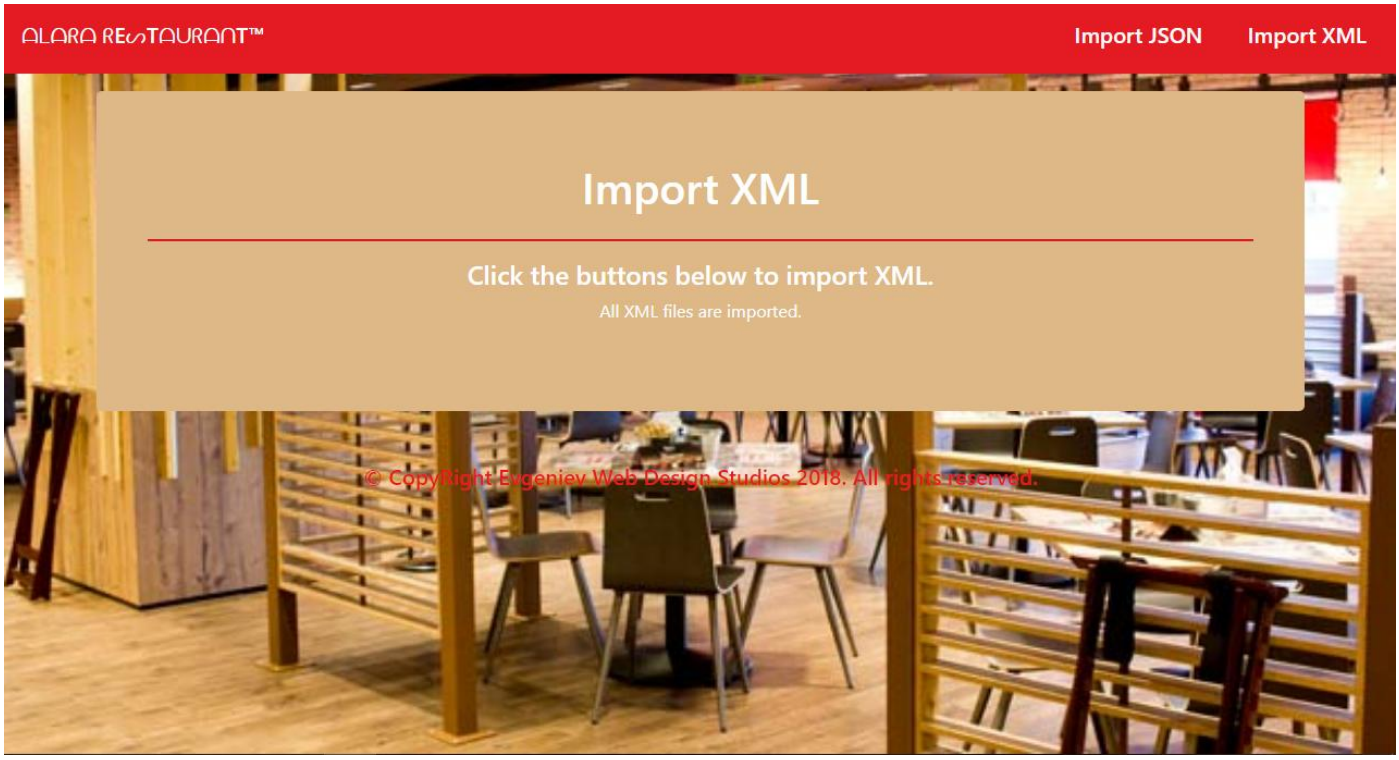
- Import Orders page after reading **orders.xml** file:



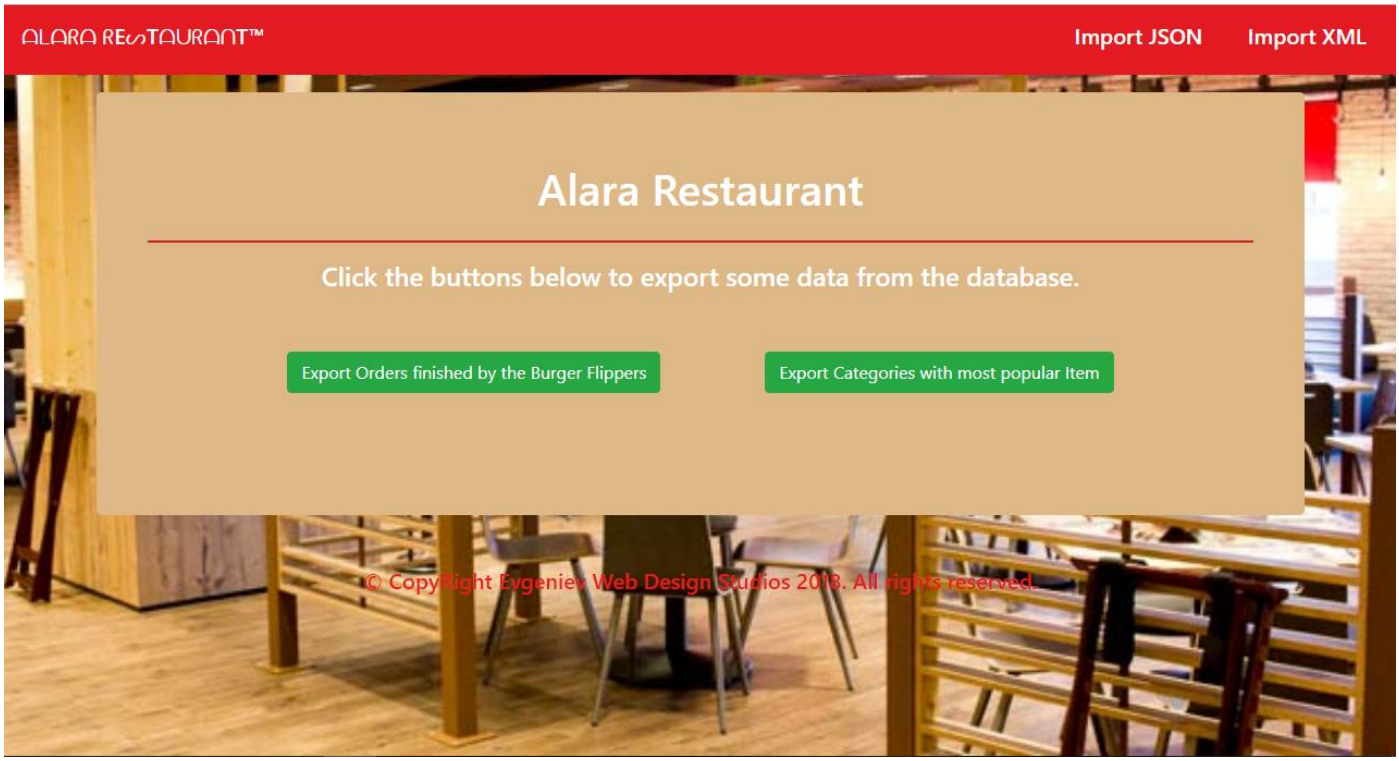
- Import JSON page after importing the given data:



- Import XML page after importing the given data:

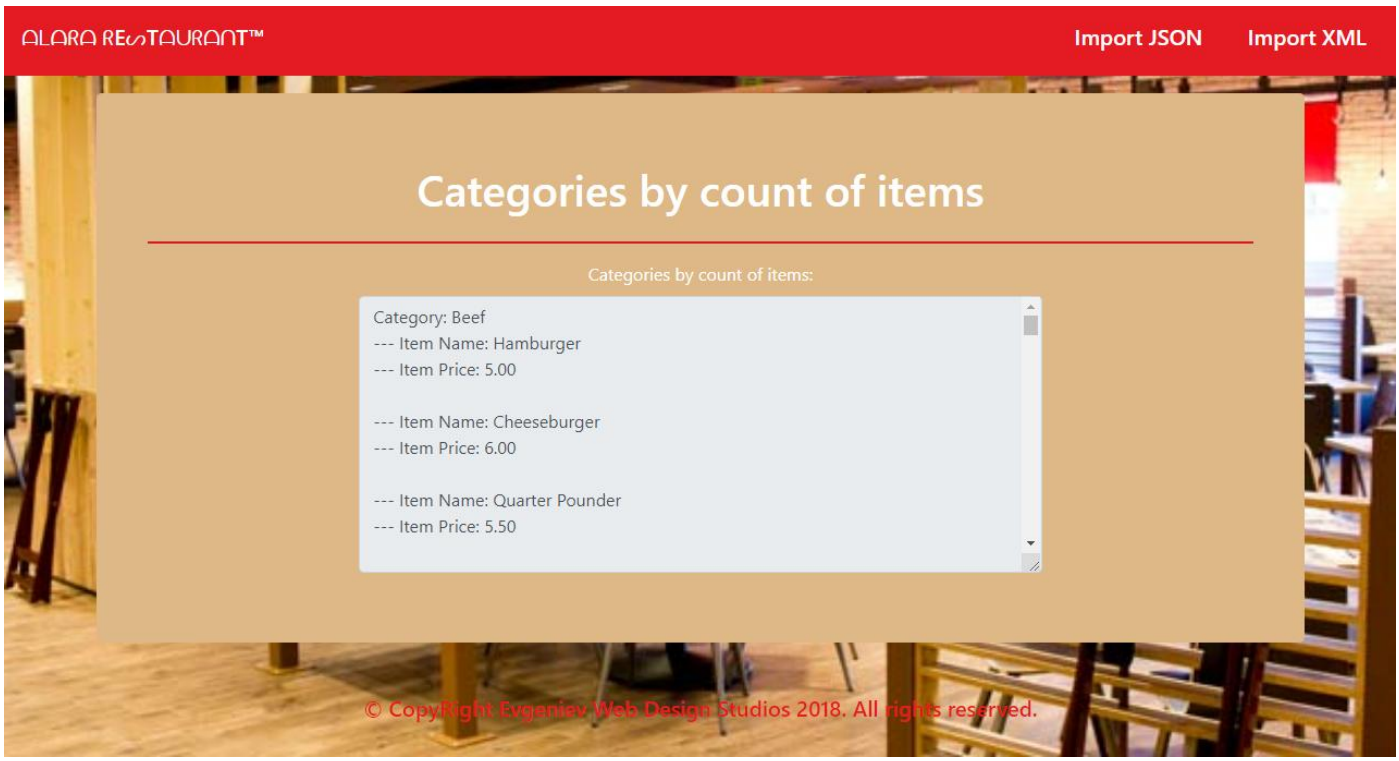


- Home page after importing the given data:



- Export Orders finished by the Burger Flippers page:

- Export Categories by count of items page:



NOTE: You will be able to finish the **first** export task after importing both of the JSON files.

2. Project Skeleton Overview

You will be given a **Skeleton**, containing a **certain architecture(MVC)** with **several classes**, some of which – completely empty. The **Skeleton** will include the **files** with which you will **seed** the **database**.

3. Model Definition

Every employee has a **position** and **orders**, which they need to process. Every **order** has a **customer**, **order date** and a **list of items**. Every item has a **category**, a **name** and a **price**. **Categories** have a **list of items**.

The application needs to store the following data:

Employee

- **`id`** – integer, **Primary Key**
- **`name`** – text with **min length 3** and **max length 30** (**required**)
- **`age`** – integer **in the range [15, 80]** (**required**)
- **`position`** – the employee's **position** (**required**)
- **`orders`** – the **orders** the employee has processed

Position

- **`id`** – integer, **Primary Key**
- **`name`** – text with **min length 3** and **max length 30** (**required, unique**)
- **`employees`** – Collection of type **`Employee`**

Category

- **`id`** – integer, **Primary Key**
- **`name`** – text with **min length 3** and **max length 30** (**required**)
- **`items`** – collection of type **`Item`**

Item

- **`id`** – integer, **Primary Key**
- **`name`** – text with **min length 3** and **max length 30** (**required, unique**)
- **`category`** – the item's **category** (**required**)
- **`price`** – decimal (**non-negative, minimum value: 0.01**, **required**)
- **`orderItems`** – collection of type **`OrderItem`**

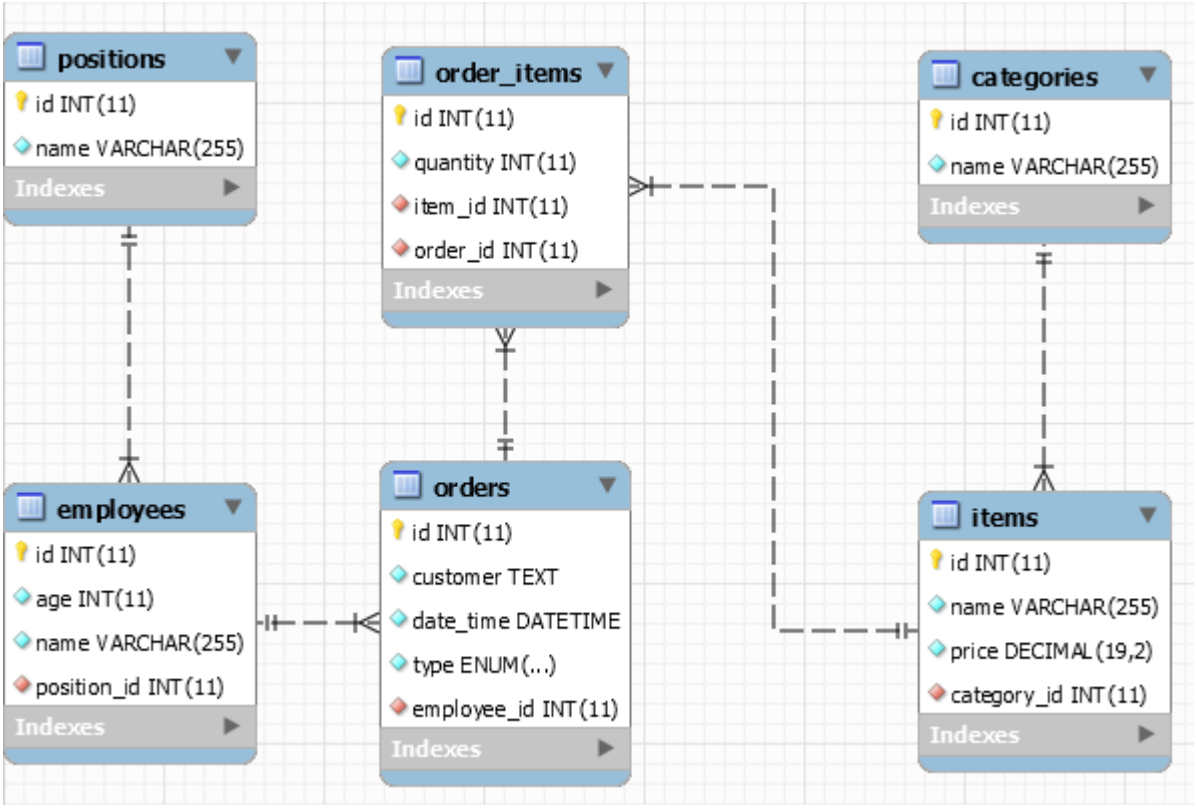
Order

- **`id`** – integer, **Primary Key**
- **`customer`** – text (**required**)
- **`dateTime`** – date and time of the order (**required**)
- **`type – OrderType`** enumeration with possible values: "**`ForHere`**, **`ToGo`** (default: **`ForHere`**)" (**required**)
- **`employee`** – The employee who will process the order (**required**)
- **`orderItems`** – collection of type **`OrderItem`**

OrderItem

- **`id`** – integer, **Primary Key**
- **`order`** – the item's **order** (**required**)
- **`item`** – the order's **item** (**required**)
- **`quantity`** – the quantity of the **item** in the **order** (**required**, **non-negative** and **non-zero**)

Your friend has given you an E/R Diagram for better understanding of the database:



4. Data Import

Use the provided **JSON** and **XML** files to populate the database with data. Import all the information from those files into the database.

You are not allowed **to modify the provided JSON and XML files.**

If a record does not meet the requirements from the first section, print an error message:

Error message
`Invalid data format.`

JSON Import

Import Employees

Using the file **employees.json**, import the data from that file into the database. Print information about each imported object in the format described below.

Constraints

- If any validation errors occur (such as if their **name or position are too long/short** or their **age** is out of range) proceed as described above
- If a position **doesn't exist yet** (and the position and rest of employee data is **valid**), **create it**.
- If an employee is **invalid**, **do not** import their **position**.

Example

employees.json

```
[
  {
    "name": "N",
    "age": 20,
    "position": "Invalid"
  },
  {
    "name": "Too Young",
    "age": 14,
    "position": "Invalid"
  },
  {
    "name": "Too Old",
    "age": 81,
    "position": "Invalid"
  },
  {
    "name": "Invalid Position",
    "age": 20,
    "position": ""
  },
  {
    "name": "InvalidPosition",
    "age": 20,
    "position": "Invalidddddddddddddddddddddddddd"
  },
  {
    "name": "Magda Bjork",
    "age": 44,
    "position": "CEO"
  },
…
```

Output

```
Invalid data format.
Invalid data format.
Invalid data format.
Invalid data format.
```

```
Invalid data format.
Record Magda Bjork successfully imported.
…
```

Import Items

Using the file **items.json**, import the data from that file into the database. Print information about each imported object in the format described below.

Constraints

- If any validation errors occur (such as invalid item name or invalid category name), **ignore** the entity and **print an error message**.
- If an item with the same name **already exists**, **ignore** the entity and **do not import it**.
- If an item's category **doesn't exist**, **create it** along with the item.

Example

items.json

```
[
  {
    "name": "Hamburger",
    "price": 0.00,
    "category": "Invalid"
  },
  {
    "name": "Hamburger",
    "price": -5.00,
    "category": "Invalid"
  },
  {
    "name": "x",
    "price": 1.00,
    "category": "Invalid"
  },
  {
    "name": "Invalidddddddddddddddddddddddddd",
    "price": 1.00,
    "category": "Invalid"
  },
  {
    "name": "Invalid",
    "price": 1.00,
    "category": "x"
  },
```

```
  {
    "name": "Invalid",
    "price": 1.00,
    "category": "Invalidddddddddddddddddddddddddd"
  },
  {
    "name": "Hamburger",
    "price": 5.00,
    "category": "Beef"
  },
  {
    "name": "Hamburger",
    "price": 1.00,
    "category": "Beef"
  },
  {
    "name": "Cheeseburger",
    "price": 6.00,
    "category": "Beef"
  },
…
```

Output

```
Invalid data format.
Invalid data format.
Invalid data format.
Invalid data format.
Invalid data format.
Invalid data format.
Record Hamburger successfully imported.
Invalid data format.
Record Cheeseburger successfully imported.
```

XML Import

Import Orders

Using the file **orders.xml**, import the data from the file into the database. Print information about each imported object in the format described below.

If any of the model requirements is violated continue with the next entity.

Constraints

- The order dates will be in the format "**dd/MM/yyyy HH:mm**".
- If the order's **employee** doesn't exist, **do not** import the order.

- If **any** of the **order's items** do not exist, **do not** import the order.
- Every employee will have a **unique name**

Example

orders.xml

```
<?xml version="1.0" encoding="utf-8"?>
<orders>
  <order>
    <customer>Garry</customer>
    <employee>Maxwell Shanahan</employee>
    <date-time>21/08/2017 13:22</date-time>
    <type>ForHere</type>
    <items>
      <item>
        <name>Quarter Pounder</name>
        <quantity>2</quantity>
      </item>
      <item>
        <name>Premium chicken sandwich</name>
        <quantity>2</quantity>
      </item>
      <item>
        <name>Chicken Tenders</name>
        <quantity>4</quantity>
      </item>
      <item>
        <name>Just Lettuce</name>
        <quantity>4</quantity>
      </item>
    </items>
  </order>
...
</orders>
```

Output

```
Order for Garry on 21/08/2017 13:22 added
```

5. Data Export

Get ready to export the data you've imported in the previous task. Here you will have some pretty complex database querying. Export the data in the formats specified below.

Categories by count of items

Export all categories by count of **items**:

- Extract from the database, **categories' names** and **items** in the categories with their **name** and **price.**
- **Order** them **descending** by **`count of items`** in each category, and if two or more categories have same number of items, order them **descending** by **sum of the items' prices** in each category.
- The format is described below**:**

Category: {category1Name}
--- Item Name: {item1Name}
--- Item Price: {item1Price}

--- Item Name: {item2Name}
--- Item Price: {item2Price}

--- Item Name: {item3Name}
--- Item Price: {item3Price}

…

Category: {category2Name}
--- Item Name: {item1Name}
--- Item Price: {item1Price}

…

```
Category: Beef
--- Item Name: Hamburger
--- Item Price: 5.00

--- Item Name: Cheeseburger
--- Item Price: 6.00

--- Item Name: Quarter Pounder
--- Item Price: 5.50
```

Orders finished by the Burger Flippers

Export all orders which are finished by the **Burger Flippers**:

- Extract from the database, **employees' names**, **orders' customers** and **items** in the orders with their **name**, **price** and **quantity**.
- **Order** them by `employee name`, and by `order id`.
- The format is described below**:**

Name: {employee1Name}
Orders:
Customer: {customerName}
Items:
Name: {item1Name}
Price: {item1Price}
Quantity: {item1Quantity}

Name: {item2Name}
Price: {item2Price}
Quantity: {item2Quantity}

Name: {employee2Name}
Orders:
Customer: {customerName}
Items:
…

```
Name: Avery Rush
Orders:
  Customer: Pablo
  Items:
    Name: Double Cheeseburger
    Price: 6.50
    Quantity: 3

    Name: Bacon Deluxe
    Price: 9.00
```