

# Java Programming

## Report #2: Java IO/Databases

### ARM

### (Application for Restaurant Management)

**Class : 18CLC2-KTPM**

**Group 07:**

**Dinh Hoang Duong – 18127084**

**Duong Tran Man Duy – 18127087**

**Huynh Duc Lee – 18127126**

**Github:**

[tarzanchemgio/Java-Group07-ARM: Application for Restaurant Management](https://github.com/tarzanchemgio/Java-Group07-ARM: Application for Restaurant Management)  
[\(github.com\)](https://github.com)

---

## Table of content

Revision History .....	3
Introduction.....	4
Analysis and design .....	5
Implementation .....	7
Sample data .....	9
Result .....	10
Plan.....	12
References .....	14

-

---

### Revision History

Date	Version	Description	Author
21/11/2020	0.0.1	Login and Sign-up prototype	Hoàng Dương
5/12/2020	1.0.0	Complete report	Hoàng Dương Đức Lê Mẫn Duy

-

---

## Introduction

Our application is tended to be run on multiple machines so we need some ways to transfer data around as needed.

E.g.

- When administrator create new dish, all machines should see it and update to Menu.
- There are many orders from many machines, bills need to be stored so manager can see them later on

The list go on and on.

We decide to choose MongoDB for the sake of easy to design schema and implementation.

Data are stored as JSON format in one server which implement RestAPI for CRUD operation. Other machines would connect to and make request as needed

-

---

## Analysis and design

Stored information:

- User account of managers and employees
- List of dishes
- List of all bills
- List of customer

JSON schema:

- User: (user's password has been hashed by sha256)

```
{
  "listOfUsers": [
    {
      "ID": "MAN-123",
      "username": "String",
      "password": "7eae8733c19872feef9b12bf11bf216ce2283614c306d994b2732f29ad3af9c8",
      "name": "String",
      "phoneNumber": "String",
      "DoB": "String",
      "gender": "String",
      "email": "String",
      "citizenID": "String",
      "salary": [
        {
          "date": 3,
          "amount": 4
        },
        {
          "date": 5,
          "amount": 6
        }
      ]
    }
  ],
}
```

- 
- Item:

```
{
  "listOfItems": [
    {
      "type": "Item_1",
      "name": "Item_name",
      "price": "100000",
      "description": "String",
      "imgPath": "String"
    }
  ]
}
```

- Customer (Customer does not have DiscountPolicy yet):

```
{
  "listOfCustomers": [
    {
      "customerID": "CUS-0000123",
      "name": "String",
      "phoneNumber": "String",
      "point": "String"
    },
    {
      "customerID": "CUS-0000124",
      "name": "String",
      "phoneNumber": "String",
      "point": "String"
    }
  ]
}
```

---

## Implementation

The data is stored on a cluster in MongoDB, 2 drivers needed to be able to connect, load, save data: mongodb-driver-legacy and mongodb-driver-async.

First of all, to connect to the database, we will access the database through an account. MongoDB will provide a connection string which is used to connect to a database in Java. After connecting, depends on the purpose that a specific collection in the database will be chosen to read. Below is the example way to connect to our database.

```
private ModelManager() {
    ConnectionString connString = new ConnectionString(
        "mongodb+srv://ManDuy:ManDuy177013@rootcluster.7m3s7.mongodb
        .net/RestatouilleDB?retryWrites=true&w=majority"
    );
    MongoClientSettings settings = MongoClientSettings.builder()
        .applyConnectionString(connString)
        .retryWrites(true)
        .build();
    MongoClient mongoClient = MongoClient.create(settings);
    database = mongoClient.getDatabase(s: "RestatouilleDB");

    // deptraicogilasai
}
```

And connect to each collection in the database (here is Item)

```
MongoDatabase db = ModelManager.getInstance().getDatabase();
MongoCollection<Document> doc = db.getCollection(s: "Item");
```

To find and delete a specific items from database

```
Document temp = doc.findOneAndDelete(Filters.eq( fieldName: "name", name));
Items deleteItem = new Items(temp.getString( key: "type"),
    temp.getString( key: "name"),
    temp.getString( key: "price"),
    temp.getString( key: "description"));
```

The data will be splitted into collections. Currently, we have: Customer, Item and User

```
ConnectionString connString = new ConnectionString(
    "mongodb+srv://ManDuy:ManDuy177013@rootcluster.7m3s7.mongodb
.net/RestatouilleDB?retryWrites=true&w=majority"
);
MongoClientSettings settings = MongoClientSettings.builder()
    .applyConnectionString(connString)
    .retryWrites(true)
    .build();
MongoClient mongoClient = MongoClient.create(settings);
MongoDatabase database = mongoClient.getDatabase(s: "RestatouilleDB");
MongoCollection<Document> d = database.getCollection(s: "Item");

Document temp = new Document();
temp.append("type", items.getType());
temp.append("name", items.getName());
temp.append("price", items.getPrice().toString());
temp.append("description", items.getDescription());
temp.append("imgPath", items.getImgPath());

d.insertOne(temp);
```

First is connecting to the database. After that is connected to the needed collection, for example is "Item". Create a new Document object, add necessary information like the picture, insert the object into the collection, and that is finished. The way they add User's data and Customer's data is similar to Item's data.



-

---

## Sample data

All the data stored on MongoDB Cloud

Item data sample:

```
_id: ObjectId("5fc20749804adfd3e301266")
type: "Pizza"
name: "Margherita"
price: "150000"
description: "Pizza với sốt cà chua và phô mai."
imgPath: "https://api.alfrescos.com.vn/uploads/images/Alfrescos-Pizza-Margherit..."
```

User (Employee) data sample:

```
_id: ObjectId("5fba833ea4e00d3e5ce58fcc")
ID: "EMP-002"
username: "dragonboy0602"
password: "dad2ca39aba3d0b39208d0f41d7559a53aed6f2a6ca77486d793e71922c43abd"
name: "Đinh Dương Hoàng"
phoneNumber: "0964351247"
DoB: "27-2-1995"
gender: "Male"
email: "windows7@yahoo.com"
citizenID: "031501457"
salary: Array
  0: Object
    date: "9-2020"
    amount: "5850000"
  1: Object
    date: "10-2020"
    amount: "5850000"
  2: Object
    date: "11-2020"
    amount: "5850000"
```

Customer data will be imported later.

-

---

## Result

At this point, we have completed Login and Sign-Up function for both source code and UI design.

Also, a database server is set up and ready to use. This would force users to have an internet connection but this requirement is normal.

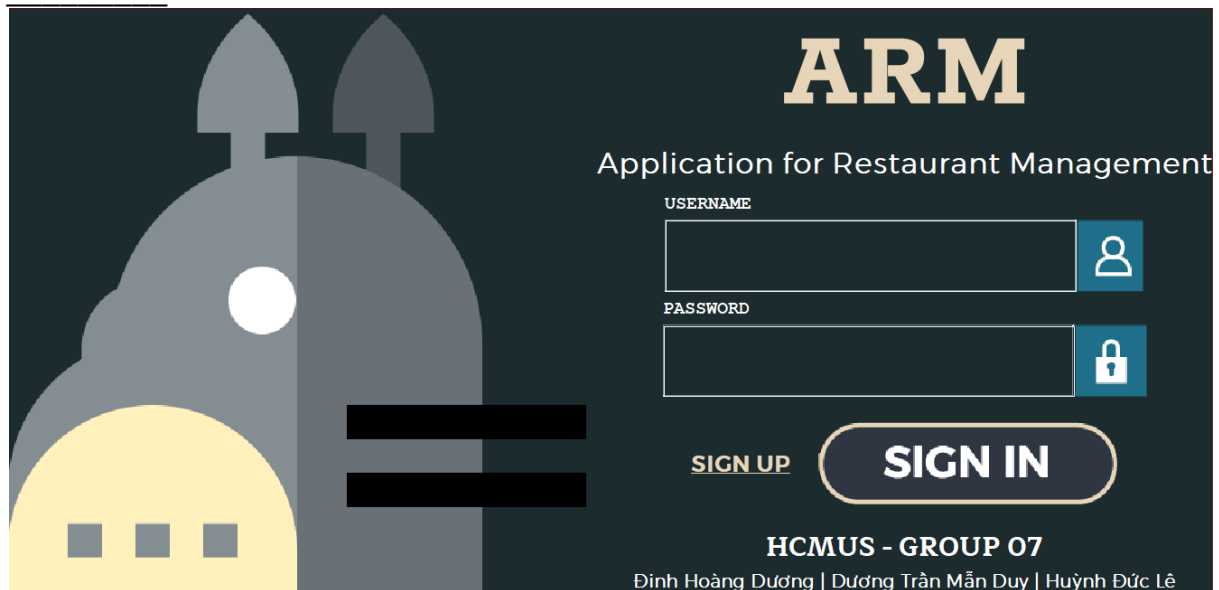
In source code, to avoid rapidly creating/destroying connection to the database, we made our database instance become singleton.

For database connection, we ship a database account, whose permission is under only the database owner, together with source code. This would cause serious security issues if byte-code were de-compile.

A screenshot of a code editor showing C# code for a MongoDB connection. The code is part of a class named 'ModelManager'. It defines a 'ConnectionString' variable 'connString' with a value for a MongoDB cluster. It then uses 'MongoClientSettings.builder()' to create settings.

```
private ModelManager() {  
    ConnectionString connString = new ConnectionString(  
        "mongodb+srv://ManDuy:ManDuy177013@rootcluster.7m3s7.mongodb  
        .net/RestatouilleDB?retryWrites=true&w=majority"  
    );  
    MongoClientSettings settings = MongoClientSettings.builder()
```

But this application was tended to be used by internal employees of a restaurant so we will live with that.



The image shows the 'ARM' Sign In screen. On the left is a stylized illustration of a restaurant building with a yellow dome and grey walls. The title 'ARM' is in large, bold, yellow letters. Below it, the text 'Application for Restaurant Management' is in white. The form has two input fields: 'USERNAME' and 'PASSWORD'. The 'PASSWORD' field has a lock icon on the right. Below the fields are two buttons: 'SIGN UP' and 'SIGN IN'. The 'SIGN IN' button is highlighted with a yellow border. At the bottom, the text 'HCMUS - GROUP 07' and 'Đinh Hoàng Dương | Dương Trần Mẫn Duyệt | Huỳnh Đức Lê' is displayed in white.

**ARM**

Application for Restaurant Management

USERNAME

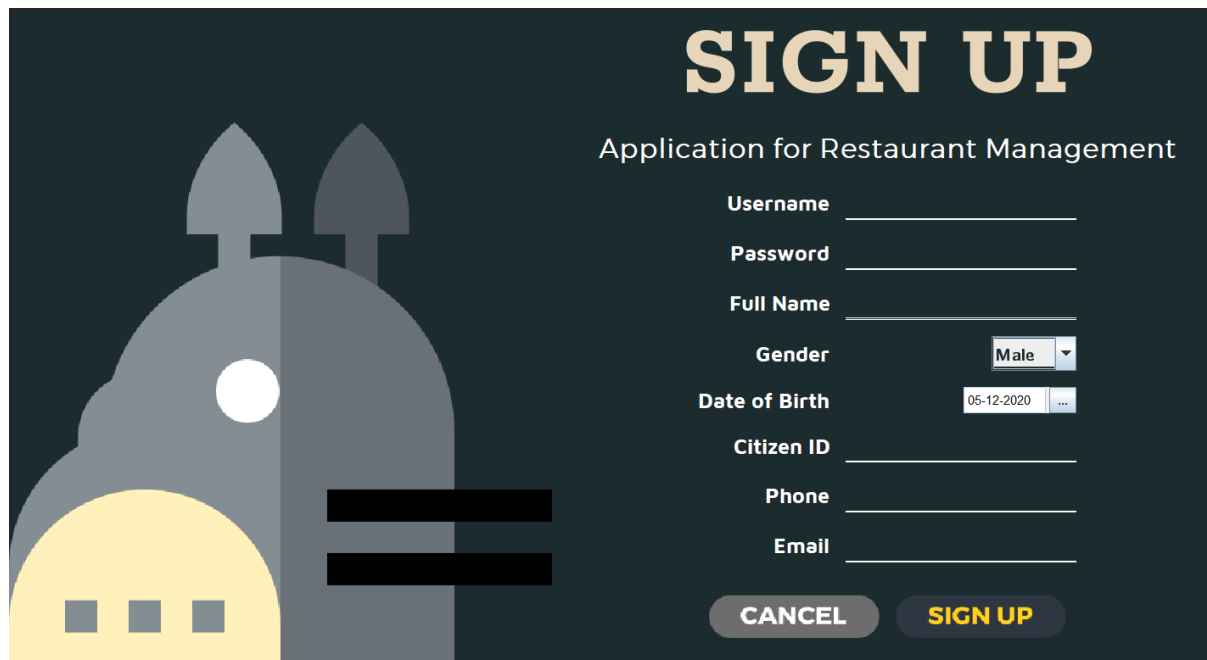
PASSWORD

**SIGN UP** **SIGN IN**

**HCMUS - GROUP 07**

Đinh Hoàng Dương | Dương Trần Mẫn Duyệt | Huỳnh Đức Lê

(Sign In screen)



The image shows the 'ARM' Sign Up screen. On the left is the same stylized illustration of a restaurant building. The title 'SIGN UP' is in large, bold, yellow letters. Below it, the text 'Application for Restaurant Management' is in white. The form has several input fields: 'Username', 'Password', 'Full Name', 'Gender' (with a dropdown menu showing 'Male'), 'Date of Birth' (with a date picker showing '05-12-2020'), 'Citizen ID', 'Phone', and 'Email'. At the bottom are two buttons: 'CANCEL' and 'SIGN UP'. The 'SIGN UP' button is highlighted with a yellow border.

**SIGN UP**

Application for Restaurant Management

Username

Password

Full Name

Gender Male

Date of Birth 05-12-2020

Citizen ID

Phone

Email

**CANCEL** **SIGN UP**

(Sign Up screen)

## Plan

Duration	Task	Author
11/11/2020 - 5/12/2020	Design Login & Sign Up screen	Dương
	Validate user input	
	Implement ViewModel for Login screen	Duy
	Write methods to communicate with database	
	Implement ViewModel for Sign Up screen	Lê
	Write new User after sign up to database	
6/12/2020 - 12/12/2020	Design Menu screen	Dương
	Implement Bill view and confirm order	
	Implement ViewModel for Menu screen	Duy, Lê
	Implement ViewModel for Bill	

13/12/2020 - 19/20/2020	Design User Info screen	Dương
	Implement ViewModel for User Info screen	Duy, Lê
20/12/2020 - 26/12/2020	Design Statistic screen	Dương
	Implement ViewModel for Statistic screen	Duy, Lê
27/12/2020 - 2/1/2021	Design Manage User screen for manager	Dương
	Implement ViewModel for Manage User screen	Duy, Lê
3/1/2021 - 9/1/2021	Design Membership screen	Dương
	Implement ViewModel for Membership screen	Duy, Lê

-

---

## References

[Stack Overflow - Where Developers Learn, Share, & Build Careers](#) - Solving all kinds of problem

[https://commons.apache.org/proper/commons-codec/download\\_codec.cgi](https://commons.apache.org/proper/commons-codec/download_codec.cgi) - For encryption

[CompletableFuture \(Java Platform SE 8 \)](#) - Asynchronous Code

<https://www.flaticon.com/> - For UI elements

[Free Vectors, Stock Photos & PSD Downloads | Freepik](#) - For UI elements

<https://github.com/LGoodDatePicker/LGoodDatePicker.git> - For date picker UI

[Maven Repository: com.google.code.gson » gson » 2.8.6 \(mvnrepository.com\)](#) - For working with JSON

<https://www.mongodb.com> - For database