COMP 7700 Project 3 Design Document

Online Shopping System team:
Members:
Donna Jackson
Leshan Zhao
(Ravindra Joshi
Mehnaz Tabassum)

Catalogue

1. Introduction: User cases

In this project, we designed and implemented a working (user acceptable) system for two use cases we provided in Assignment 3. This project is an extension of our project 2, Mr. Jackson implemented the Android App for the first user case, and I implemented the Web Server using RESTFul Design and Web Programming Language. The web server is listening at port 7701 of a free AWS EC2 web server, it's address is:

18.217.155.95:7701

The root page will direct the user to the service he needs. Currently the customer and guest (both are buyers) services are available. The user can also use url like the following format to access the service:

18.217.155.95:7701/UserLogin?UserRole=Guest
18.217.155.95:7701/UserLogin?UserRole=Customer
18.217.155.95:7701/search?username=lzhao&password=password
18.217.155.95:7701/search
18.217.155.95:7701/search?category=Household&description=broom
18.217.155.95:7701/productDetail?id=1002

The first user case is when a buyer who wants to search for a broom he wants, he will type in the description or keyword of the product, e.g., "broom", selected a category (optional), and the system will show him a list of products that fit the description. He can view the more detail information of the product by clicking the button "View Detail", and then the app will get into a page with more details of the selected product.

The second user case is a seller who wants to sell his idle broom (brand new), he will upload it to the system.

2. Architecture

In this project, we are using core Java to implement the system.

The system is designed using multi-tier, model-view-controller, client/server architecture.

a. Client-Server Implementation

The server component is running independently in a cloud-based service. In this case, we utilized the free Amazon Elastic Compute Cloud（Amazon EC2） platform to set up a windows server remotely, so the client app can run locally independent of the server.

We have set the server running, listening to the request from port "7700" at public IP address "**18.219.81.142**" (the address is dynamically allocated, will vary each time we reboot the remote EC2 instance.)

The cloud server access is provided in the attached file "OnlineShopping.rdp", where the password for the administrator is "TNJVNIK18IvGtxYym9INeHNxeAlpx)(z"

The client component could run independently in any desktop computer with java. (The server component could also be set up and run on any desktop computer with java and will be able to listen to request if there's network connection.)

The detail design is shown in the following figures together with MVC and Multi-tier pattern.

b. MVC architecture implementation

Our system implements the following models, views and controllers and had them cooperate:

Models:
(consistent between client and server)
Models for Remote Data Access:
RequestModel
ResponseModel
ProductModel
ProductImageModel

Models for Information Storage:
Product
ProductDetails

Order
OrderDetails
Shipper
ShipperMethod
Types
Supplier
Shipper

Users
UserRole
ContactInfo
Authorization
AuthorizationLevels

Views:

WelcomeView (start screen)
LoginScreenView

For Buyer:
SearchView
- SearchInputBoxPanel
- SearchResultPanel
- ItemDetailPanel
- CheckoutView (being implemented)

ShipperView (Not included, To be implemented)
ShoppintCartView (Not included, To be implemented)

For Seller: (Being Implemented)
SellerView
ProductMaintenanceView

For SuperUser: (Not included, To be implemented)
SuperUserView
RegisterViewScreen

Controllers:

WelcomeController
LoginController

For Buyer screen:
SearchController
(no controller for search input box panel because it's too simple)
SearchResultController
ItemDetailController
CheckoutController (being implemented)
ShipperController (Not included, To be implemented)
ShoppintCartController (Not included, To be implemented)

For Seller screen: (Being Implemented)
SellerController
ProductController

For SuperUser Screen: (Not included, To be implemented)
SuperUserController
RegisterController

Application.java:
starting point of the app

Presentation Layer

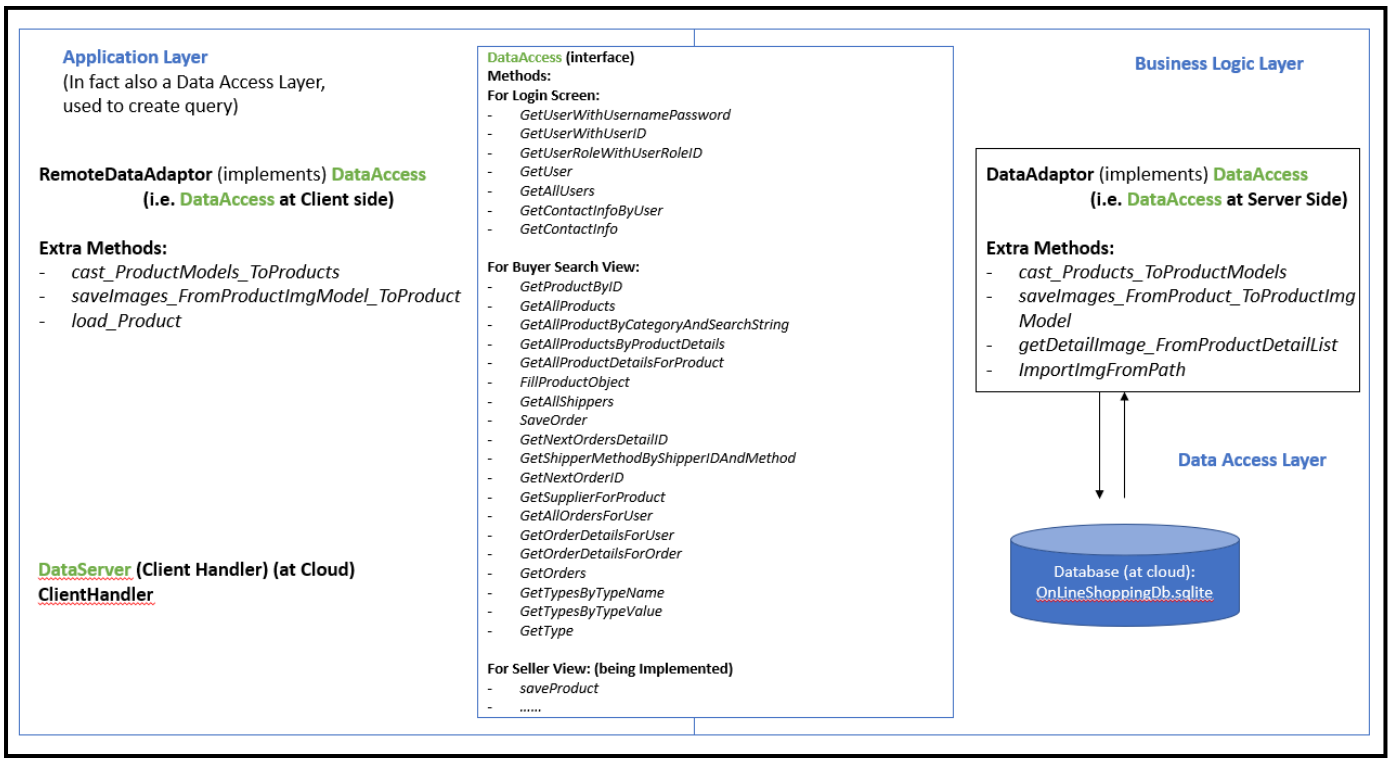
Ms. Jackson also provided a very elaborated version of all the classes and methods in the attached file "ProjectClassDiagram.png" (and .jpg).

The detail design of how the architecture works and cooperate with other architectures is shown in the following figures together with Client-Server and Multi-tier pattern.

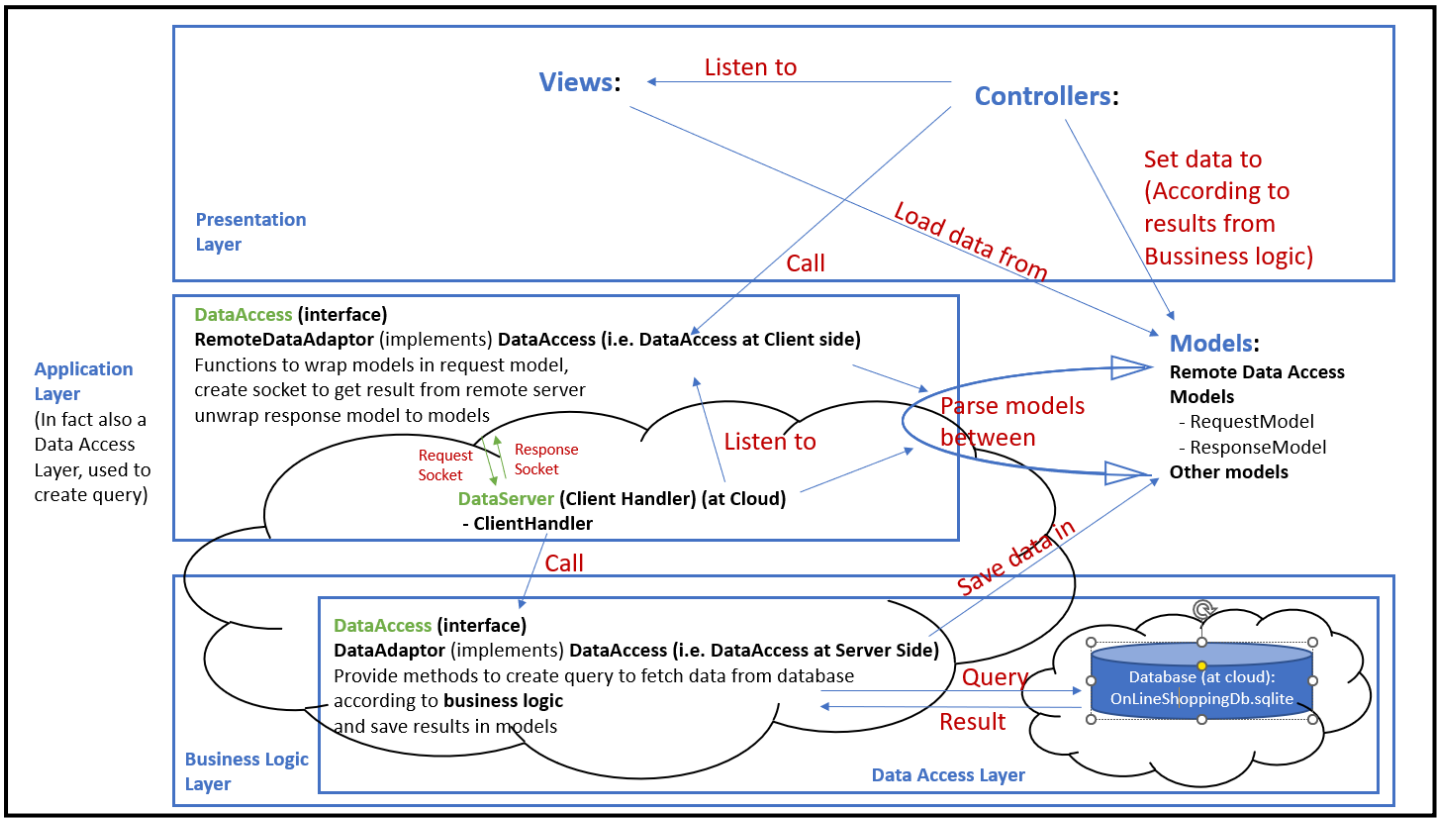
c. Multi-tier architecture implementation
   i. Presentation Layer (User Interface)
   ii. Application Layer
   iii. Business Logic Layer
   iv. Data Access Layer

The presentation layer of our system includes the components as shown in the above figure right side.

The application layer, business logic layer and data access layer of our system is a bit more complex:



The detail design of how the architecture works and cooperate is shown in the following figure:

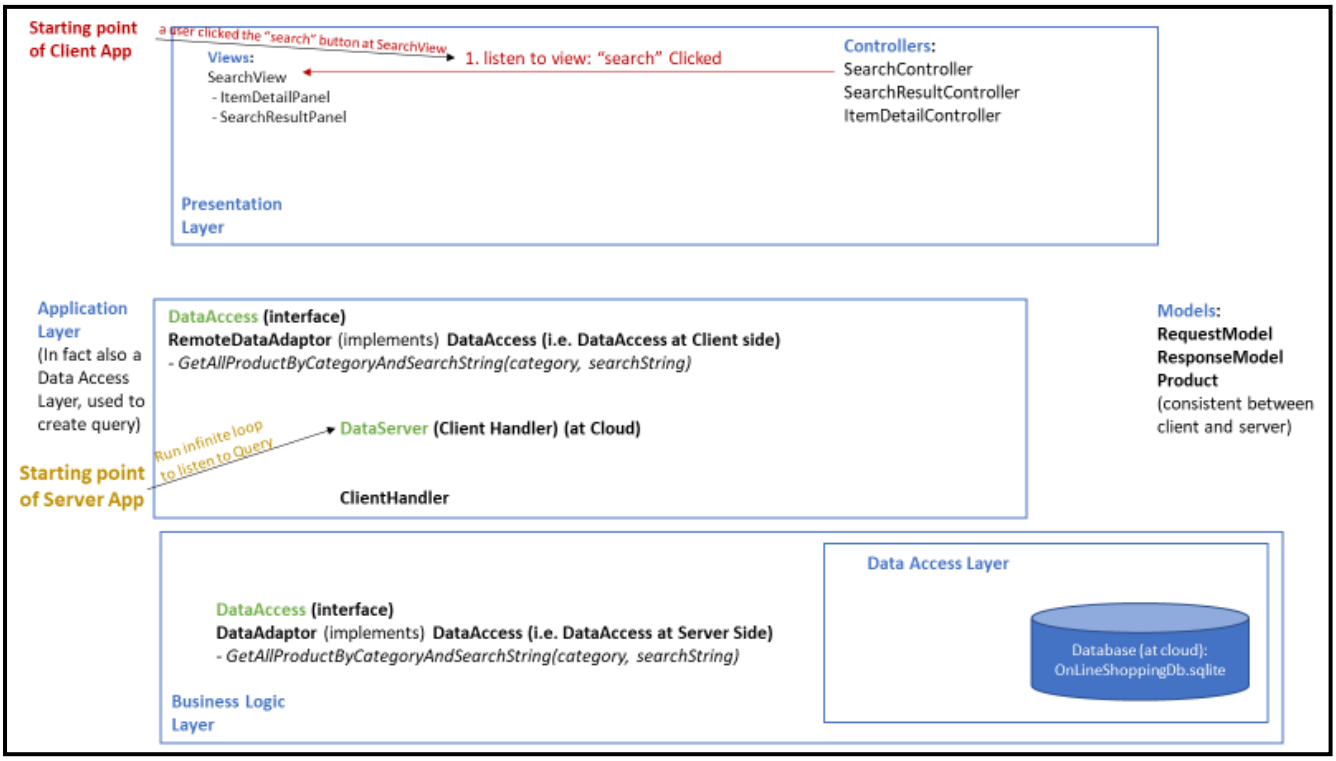


As this may be too abstract to understand, we would like to use an example to elaborate the architecture.

All the figures can be zoomed as they are provided in the attached file "*DesignDocument_pptFormat.pptx*".

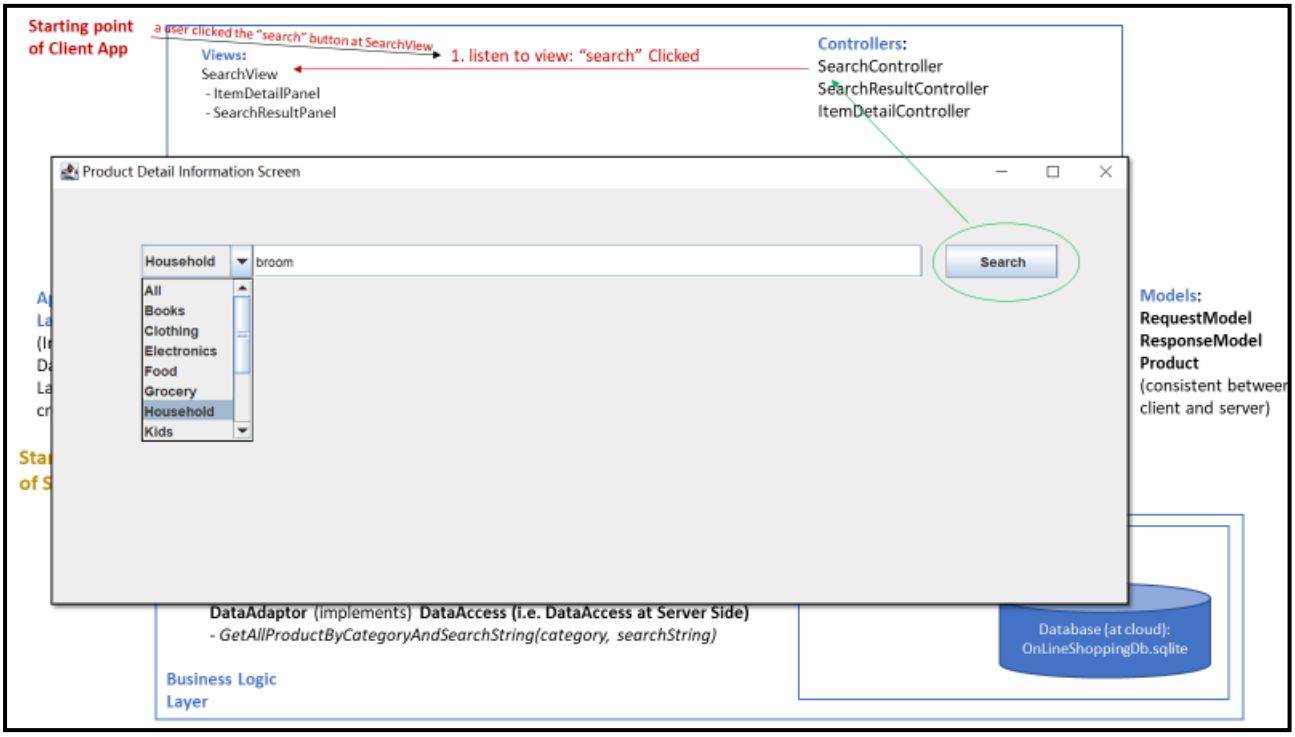
We are now going to use an example to elaborate how the architecture is working: (will also explain in the video)

First, given that the server application has been running on the cloud, assume that a user typed in "broom" in the search input box and clicked the "search" button at the search screen on the client side.

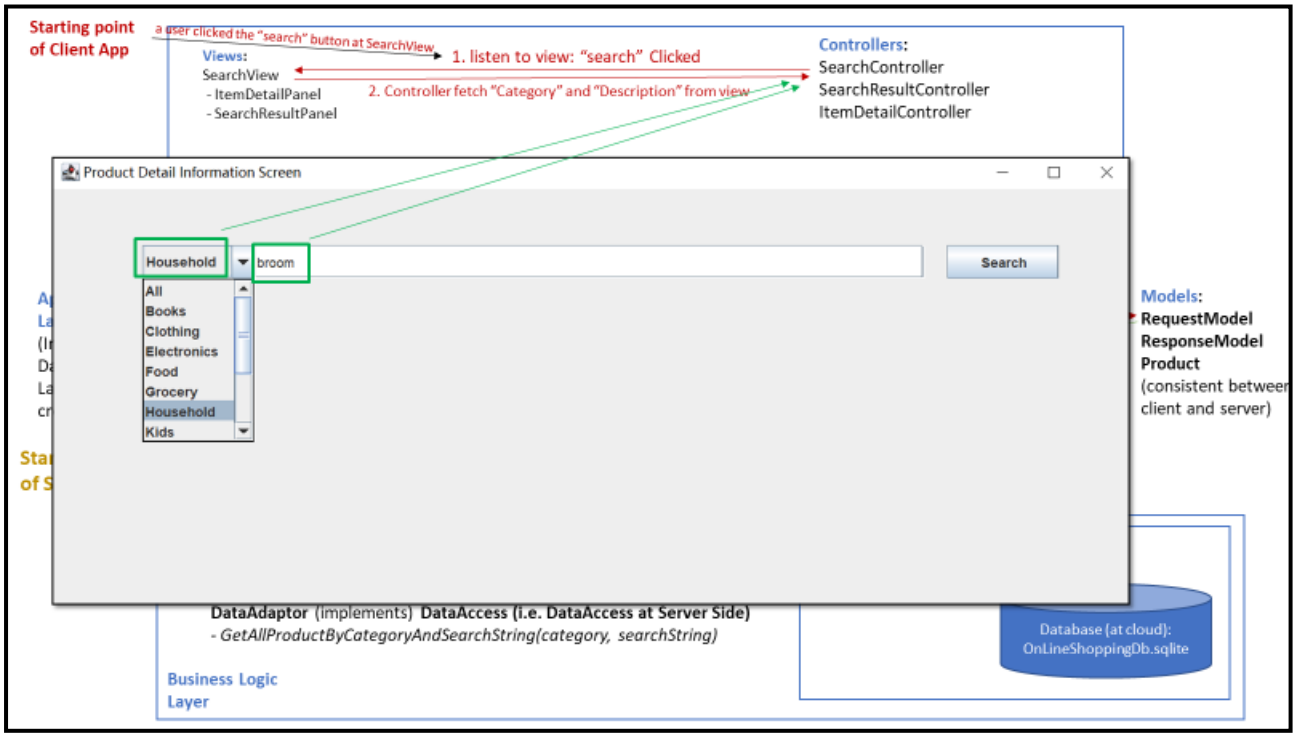


This action will initiate a series of behaviors of the system:

1. **As the "SearchController" is listening to the button "Search" of the "SearchView", it observed the click event when the user clicked it.**



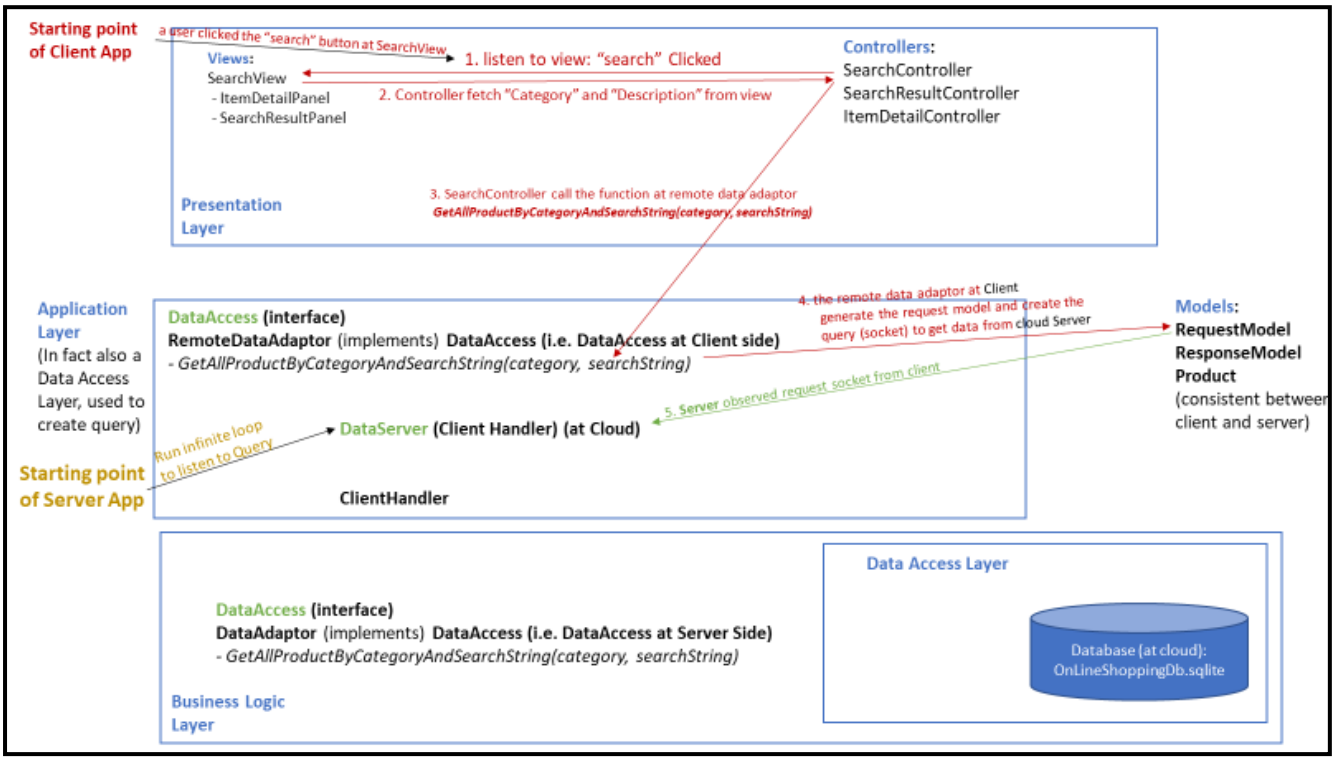
2. **The Controller will then fetch the "Category" and "Description" from the input box, i.e., "category" = "Household" and "description" = "broom"**



3. **The SearchController will call the method**

   "***GetAllProductByCategoryAndSearchString(category, searchString)***

   Provided by the application's remote data adaptor
4. **The remote data adaptor at Client side will put the "Category" and "Description" message into a new RequestModel, cast it to a JSON string and create a socket to write the string into the DataInputStream; then it will send the request to the remote Server host (i.e., at IP address** 18.219.81.142, **port** 7700**).**
5. **Since the Server is running an infinite loop to listen to TCP sockets at port 7700, it will observe the request socket from client immediately.**

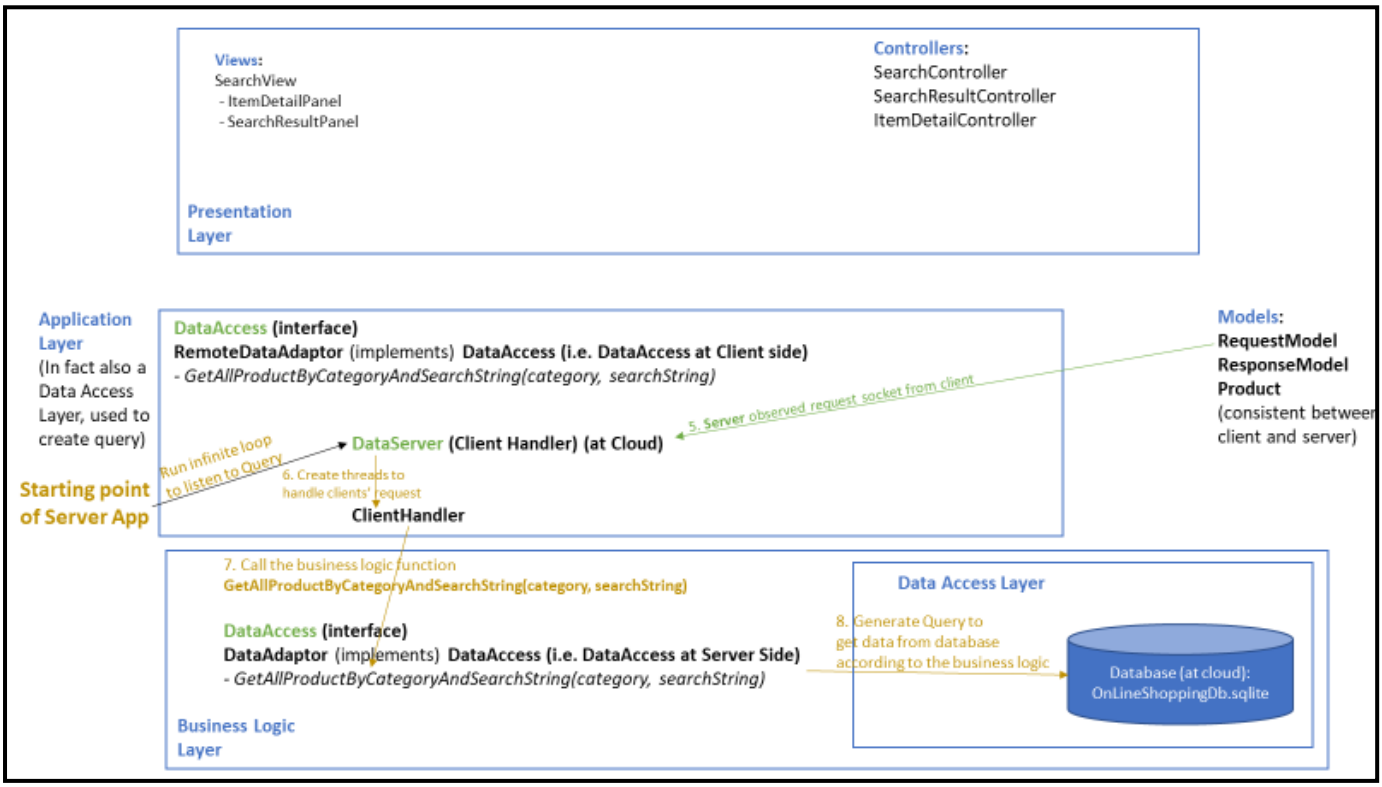


6. **Once the server received the TCP request, it accept it immediately, and create a new thread to handle it.**
7. **The new thread, i.e., the new "ClientHandler", will read the data from the DataInputStream and pass it to a RequestModel at the server side. And it will read the request code from the request model, and based on the code it will now that this request asks for a list of product with designated *category* and *description*, so it will read the category and description from the received request model, and call a corresponding method in Server's DataAdaptor, the**

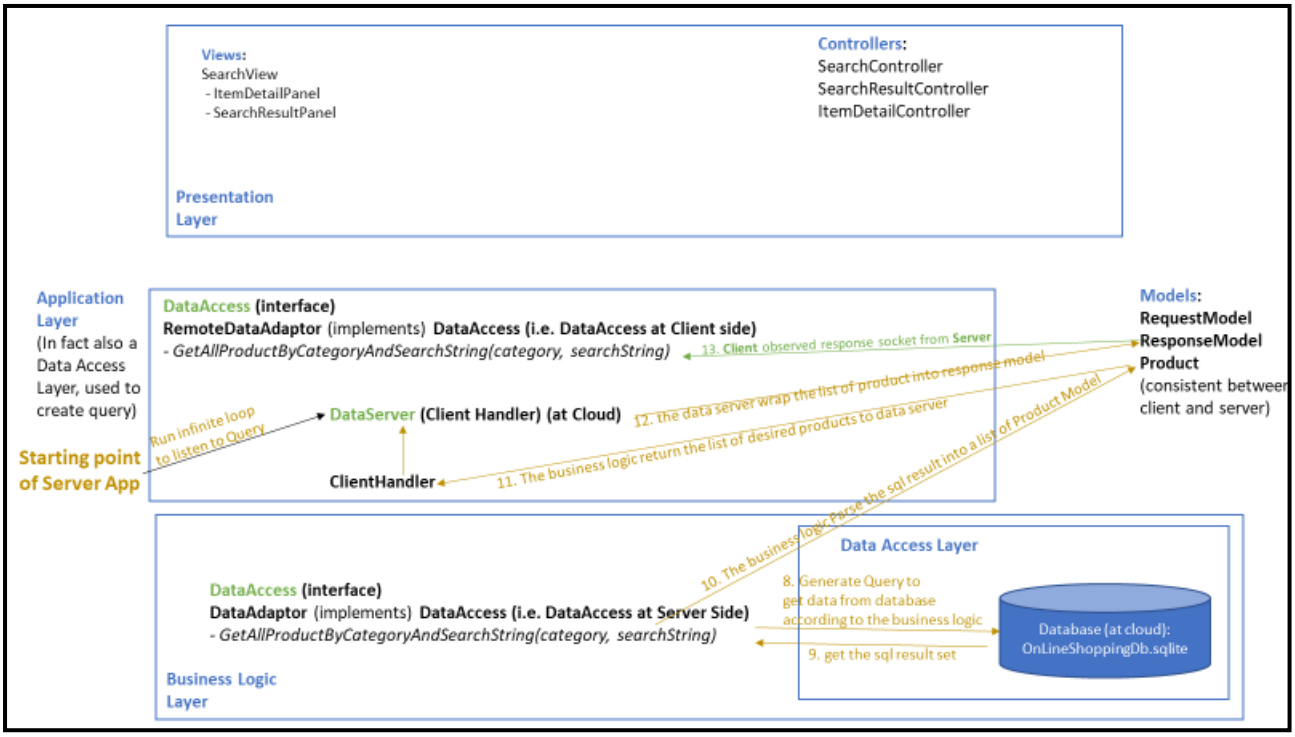
   GetAllProductByCategoryAndSearchString(category, searchString)

   function.

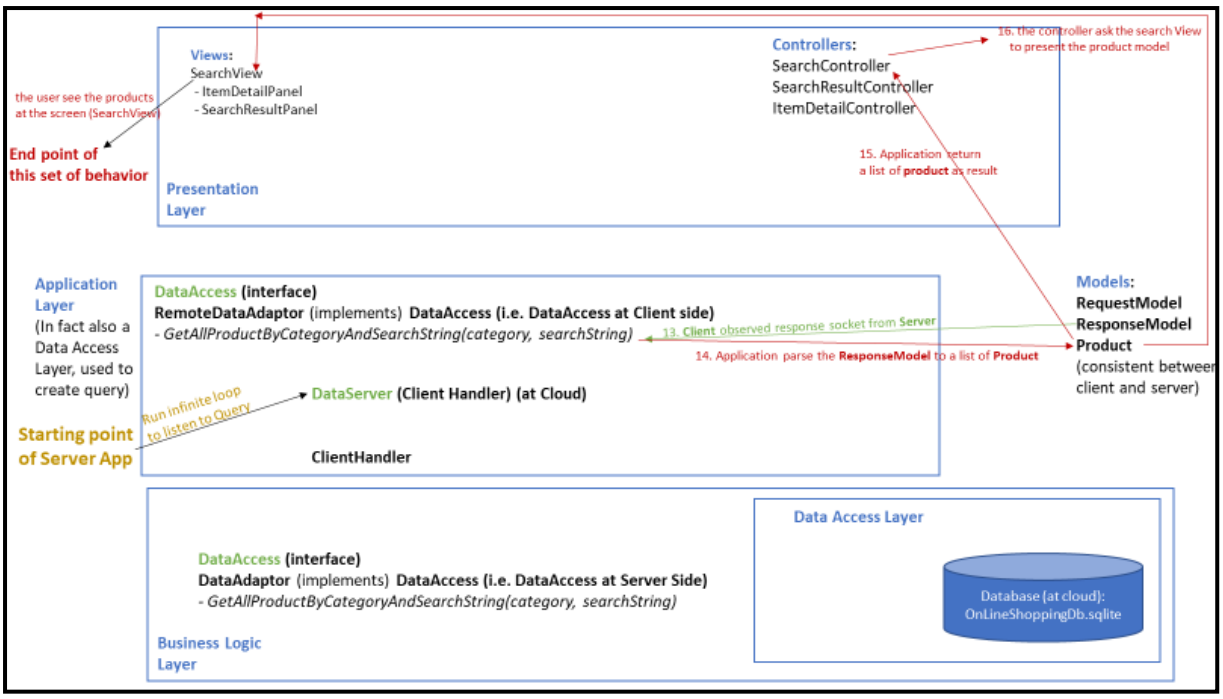
8. **This method will further use the category and searchString to generate the SQL query that obtain a set of result from the database saved at the server side.**



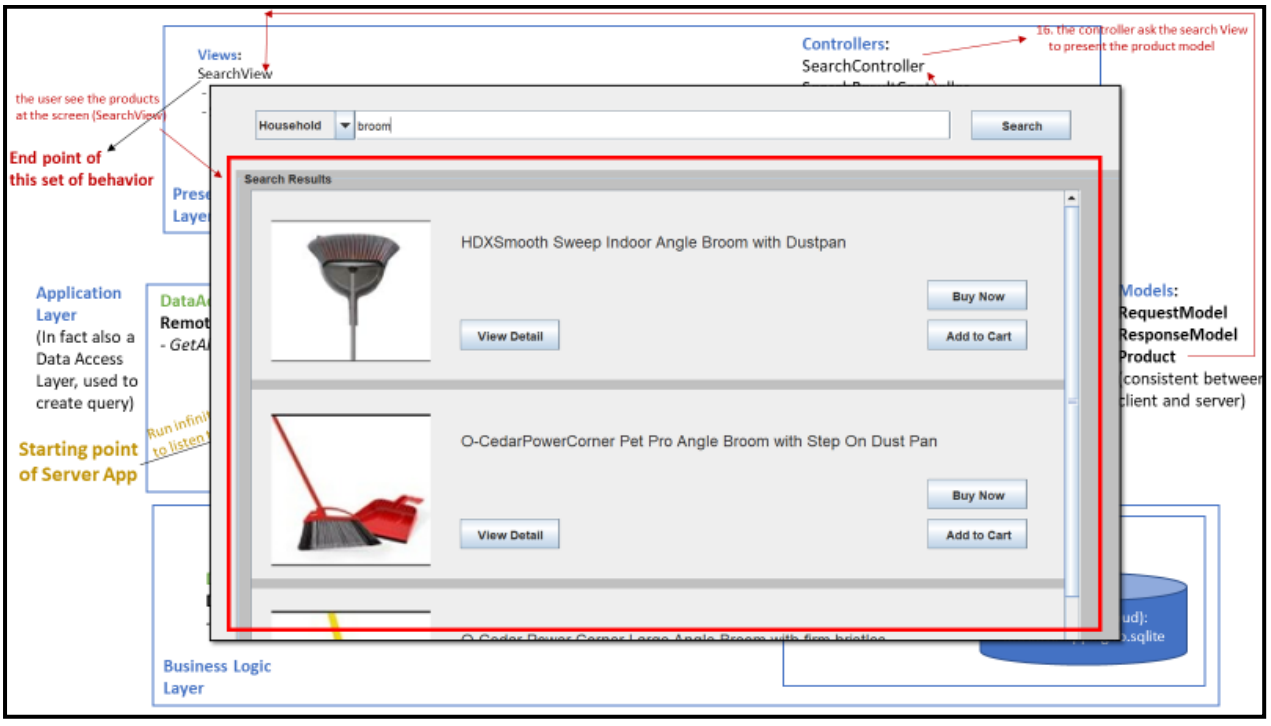
9. The database will execute the SQL query and return a set of result to the method.
10. The method will parse the result set into a list of Product Models.
11. And the method will return the list of Product Models to the client handler.
12. The client handler will wrap the list of products into a response model, set the response code being "OK", and cast the response model to a JSON string and write it into the DataOutputStream.
13. Since the client is waiting to read something from the socket, now it observed data from the DataOutputStream, read the data from the dos into a JSON string and parse it into a ResponseModel.



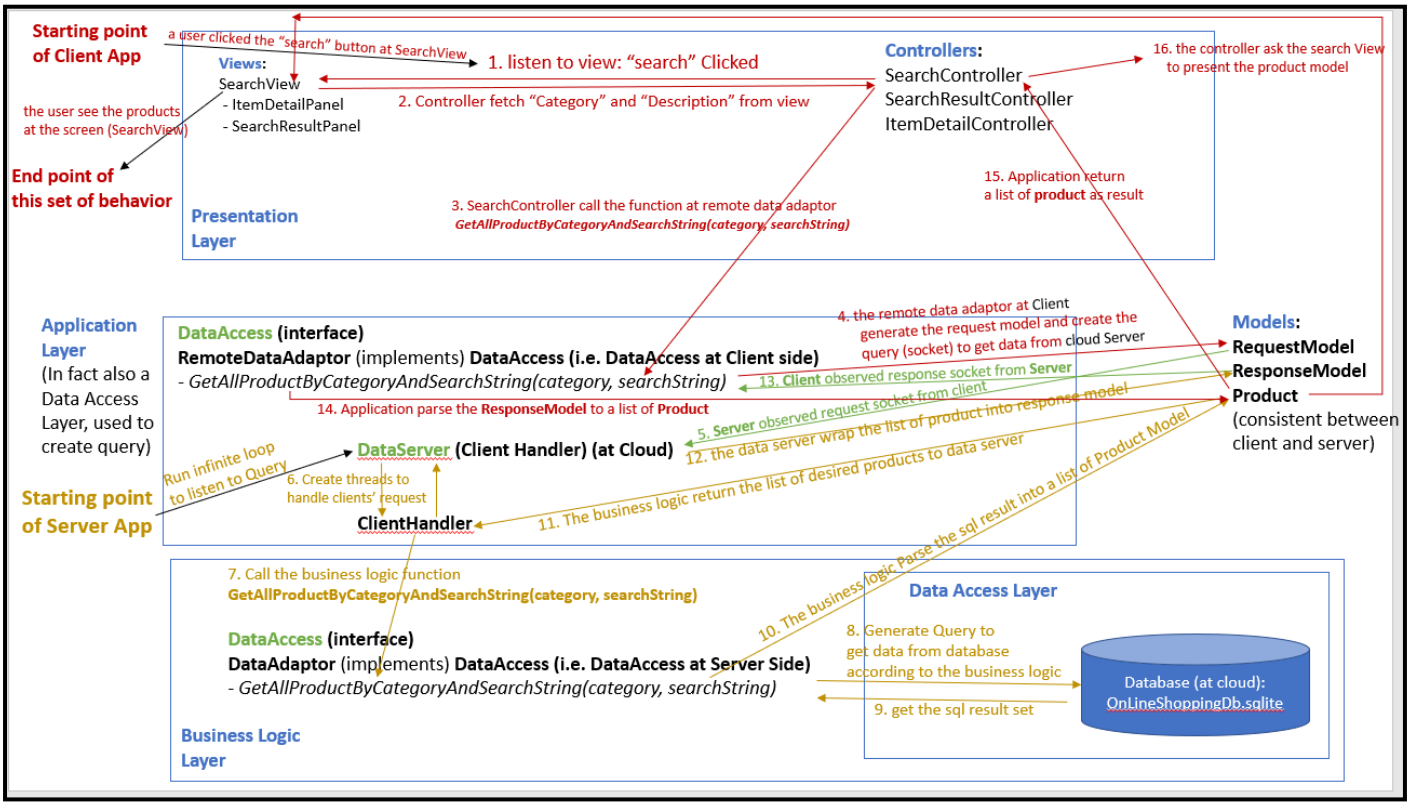
14. Then, the "*GetAllProductByCategoryAndSearchString*" method will read the response code from the response model, i.e., "OK", so it knows that it got the data it wants.
    And since the request is made by itself, it knows that the content of the response should be a list of product models. So it parse the ResponseModel to a list of Product.
15. Then, the application layer returns this list of Product to the SearchController.
16. The SearchController than ask the SearchView to load the Product in the list one by one, for each product in the list the SearchView will create a new sub-panel to present it.



Finally, the user got what he wants: a list of product that contains the keyword "broom". This leads to the end of this series of system's behavior, and the user can continue to action, the next controller, i.e., the "SearchResultController", will probably be working since the SearchView is now presenting the "SearchResultPanel".



The entire flow of actions of this example is provided as below:



(Red arrows represent action flows on the client side, yellow represents action flows on the server side, and the green arrows represents the data stream between client and server)

For other user actions, e.g., user login, user click "View Detail" button, etc., the architecture works mostly the same way.

d. Android Client

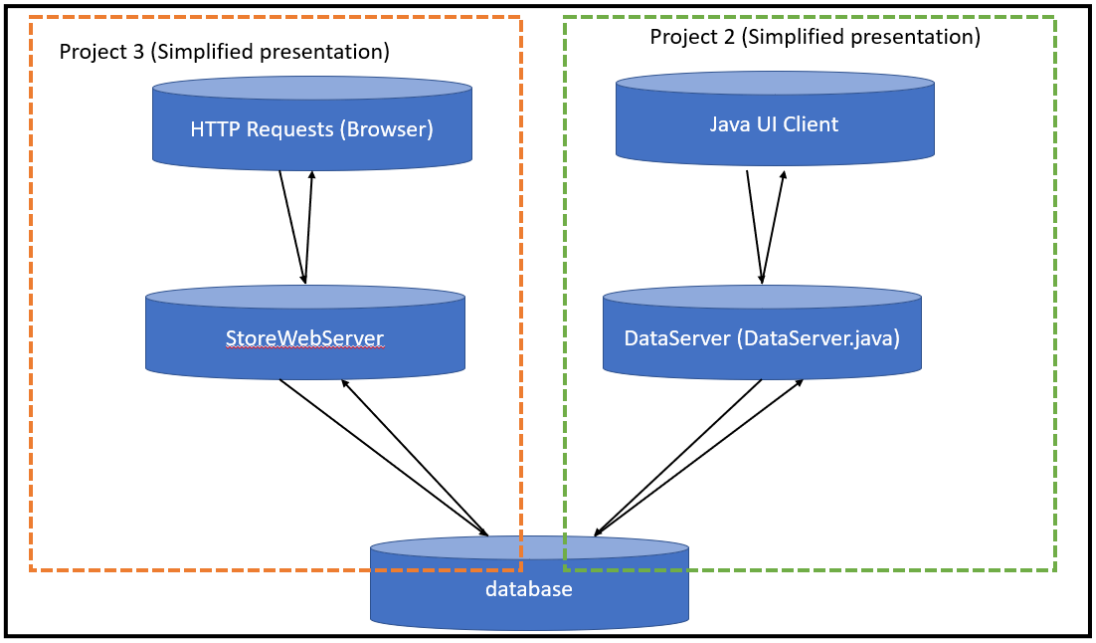
In this project, we also implemented an Android UI Client. The detailed documentation is provided by Ms. Jackson, Donna.

e. Web Server

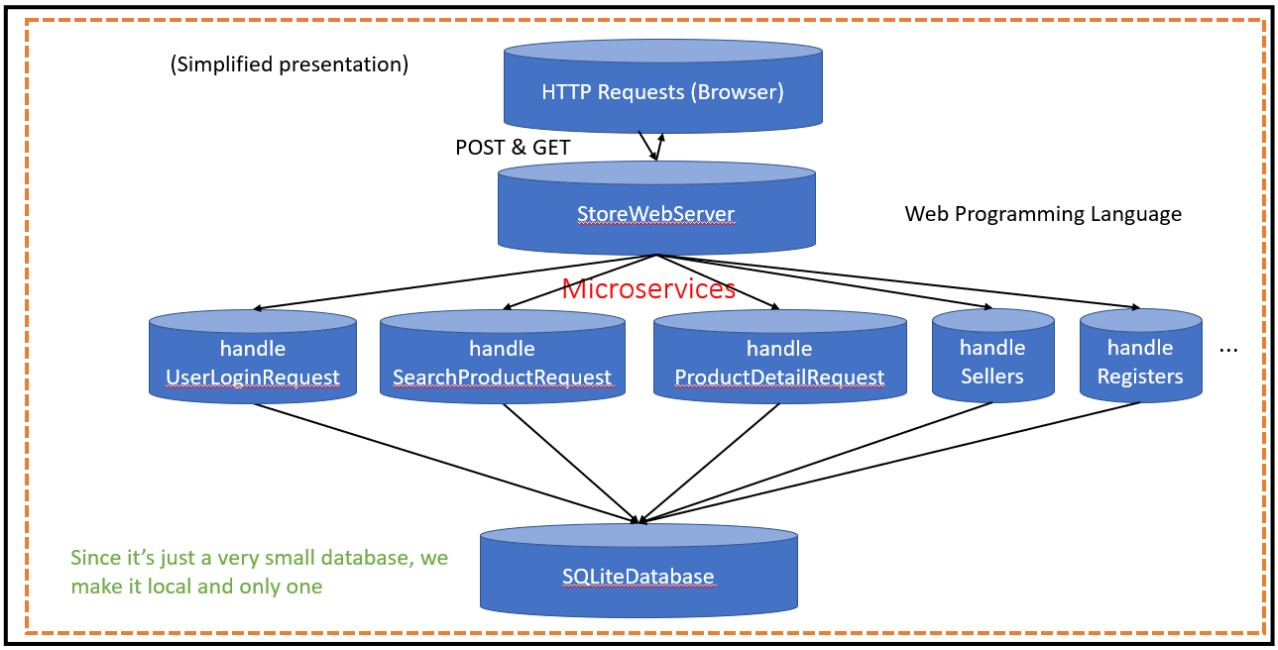
In this project, we had a new server component works as a web server, using RESTful design and implemented in Java (e.g., most actions and parameters passing via URI). Any web browser can work as a client. The web server is listening at port 7701 of a free AWS EC2 web server, it's address is:

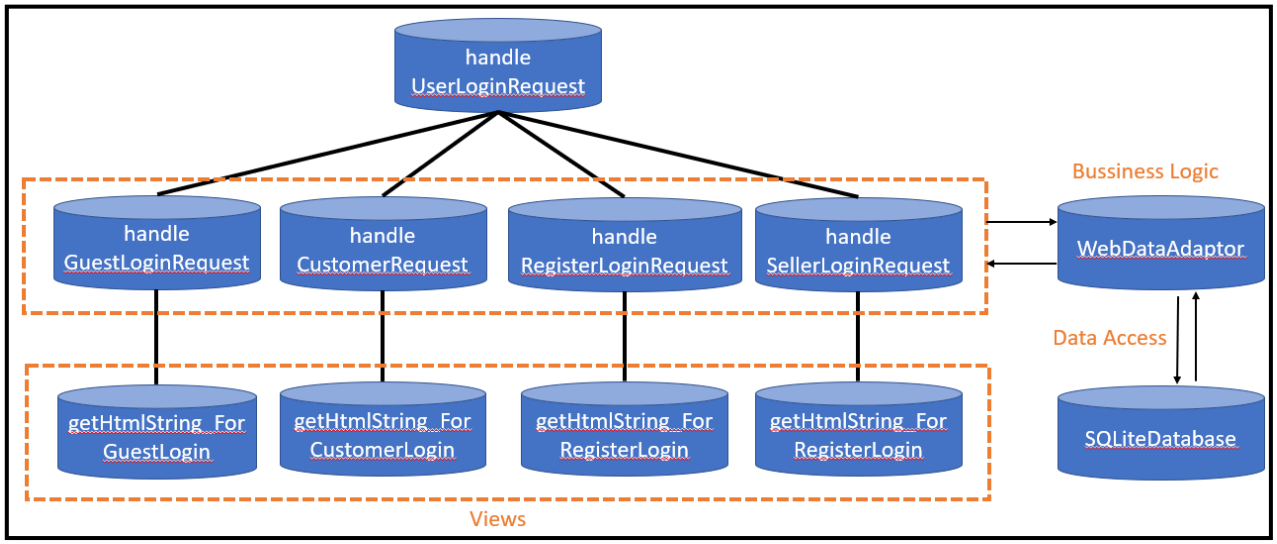
18.217.155.95:7701

As a new component, the web server access the database in parallel to the data server for the java UI in previous project.



The server will further assign the incoming requests to several microservices, which access a shared database.





3. Database

We are using sqlite database to store and manage our users, products, and orders information.

The database is also provided in the attached file "*OnLineShoppingDb.sqlite*".

Since Ms. Jackson and I are responsible for the Buyer user case (and the MVC and client-server architecture implementation), we don't have the seller part yet, so the data in the database are loaded by tying or executing queries directly (instead of a "saveProduct" method in the seller part).

We created 5 super users (UserRoleID is 1001) in the database currently:

The usernames are "donnapjackson2020", "lzhao", "rjoshi", "mtabassum", and we provided a user access for Prof. Nguyen, with user name "tnguyen".

The passwords for all of us are "password".