HELLENIC REPUBLIC

National and Kapodistrian
University of Athens

Τεχνολογίες Ηλεκτρονικού Εμπορίου

Ganoti Kyriaki
M1465

For the implementation of this project, Spring Framework has been used. The **EbayController** annotated with the @RestController annotation is responsible to map the requested resources to specific methods and to return the available data to the client.

EbayController holds dependency on the various services this app offers, in order to retrieve the requested data. Each of these services is being annotated with the Spring's @Service annotation.

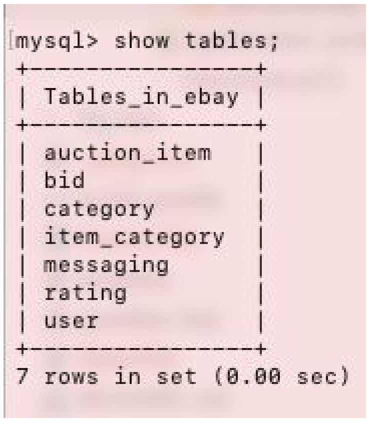
Each service, retrieves the data from the DAO level, holding dependency on classes that are annotated with the @Repository annotation. Spring helps us with the Dependency Injection for the resources each component holds dependency on, so we don't need to worry about creating and injecting new objects into other objects (or "dependencies") .

1. Database Schema

On the DAO level, we perform the queries we need in order to retrieve the data we need.

The database schema lives inside the **schema.sql** file, under ebay_db folder, inside the resources folder. The database we create its called ebay.

On the ebay database you will find the following tables:



```
mysql> show tables;
+-----------------+
| Tables_in_ebay |
+-----------------+
| auction_item    |
| bid             |
| category        |
| item_category   |
| messaging       |
| rating          |
| user            |
+-----------------+
7 rows in set (0.00 sec)
```

Indexes, constraints and foreign keys have been used where I thought its was needed.

2. Business Logic

Inside the EbayController you can find all the available endpoints the Android app uses, in order to retrieve and to post data.

- Auction:

An item its connected to an auction. A user is able by using the android app to create an auction by submitting a POST request. After the successful creation of an Auction for a specific item, the user is able to activate the auction. An auction that is not activated, cannot accept bids.

- Bid:

A bid is an action that can be performed by any user that has signed up to the application. The application supports a user to be able to bid for the same auction more than one time, as long as the amount of the bid is different from any other bids that this user has already submitted for the same auction.

- Messaging

There are no screens on the application that support this functionality, but the service is completed in everything that the client could need for the messaging process. Specifically, the available endpoints allow a user:

To retrieve information for new messages that user has not read yet
To send message/messages to any other user that has a winning bid to an auction that this user created

To send message/messages to any other user-creator of auction, with the user to has a winning bid.

To delete messages that the user owns.

- Category

Any item available to an auction belongs to one or more categories. There is a N:N relationship between AuctionItem and Category, thats why there is an intermediate table called item_category on the db.

- User

When a user has performed a signup/login is able to post and get data from the service. Specific endpoints like search auctions by category, fetch all categories, fetch all auctions don't require a user to has already login to the application. For all the other endpoints though and basically everything that interacts with the application, requires the user to have logged in to the application.

Every request to an endpoint that needs a user to have logged, accepts the userId of the user on the header. The value of this header (**X-User-Id**) will be encrypted since the data are transferred over TLS, so we are sure that the application is secure and another malicious user cannot obtain a userId that shouldn't know.

Every request that needs the user to be logged in, make a check in order to identify if this user exists indeed on the db.

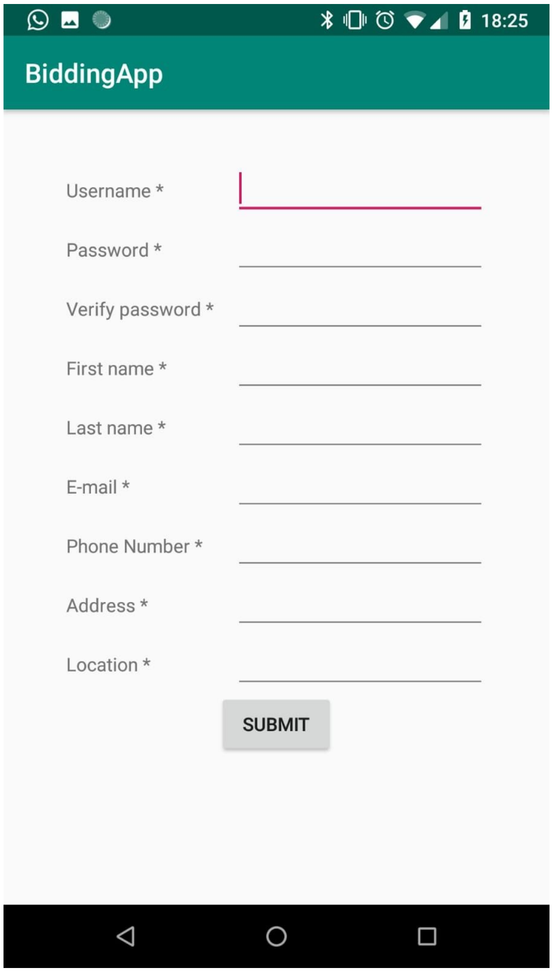
** Passwords are stored to the database after they have been hashed!

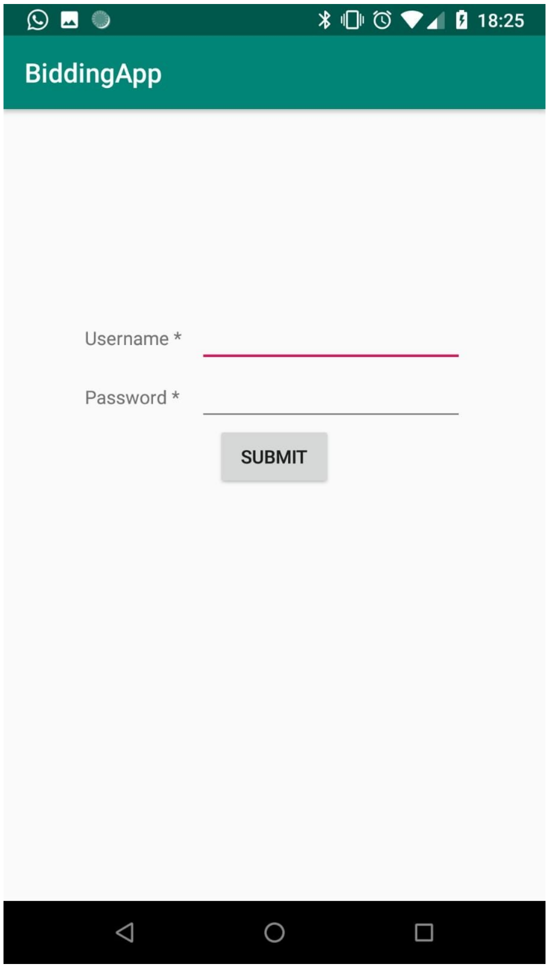
** Please set your db contact details on application.properties file

I used the following project: https://github.com/AlexDisler/mysql-vagrant for running a database instance inside a virtual machine

3. Application

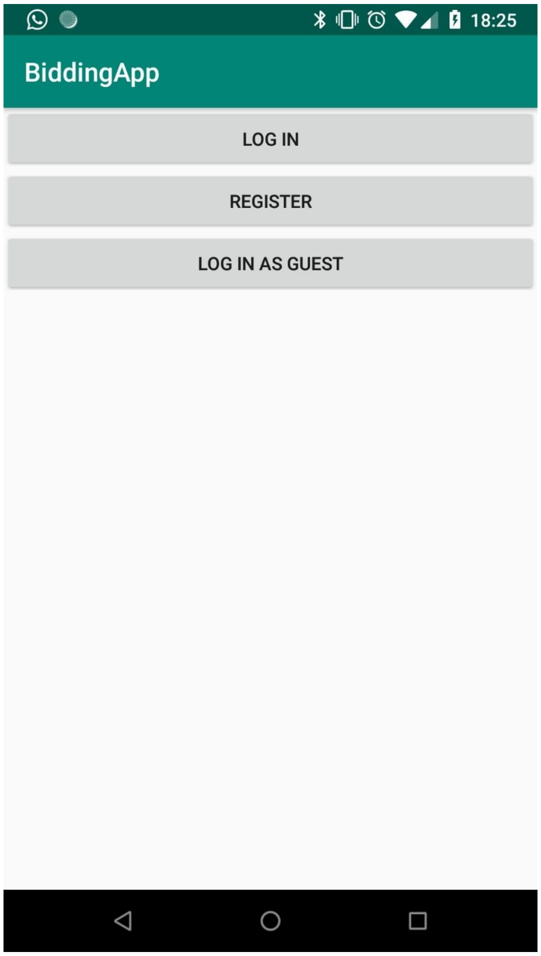
Some screens of the app follow:

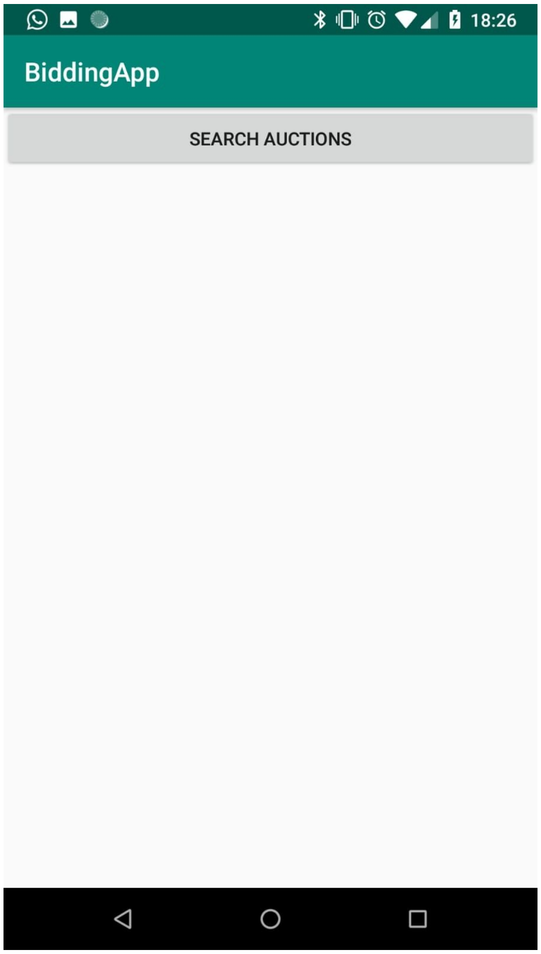




User registration

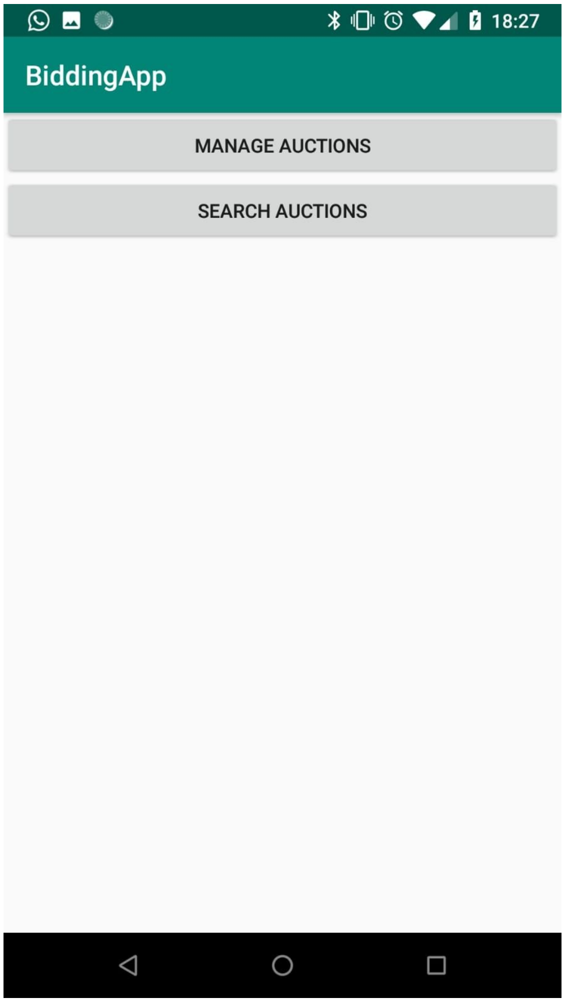
User login



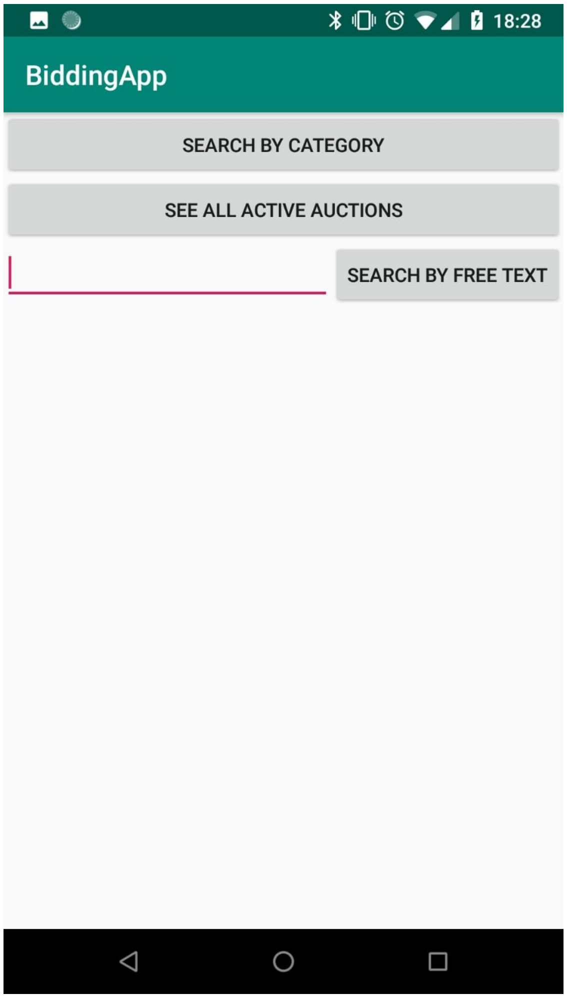


Starting screen

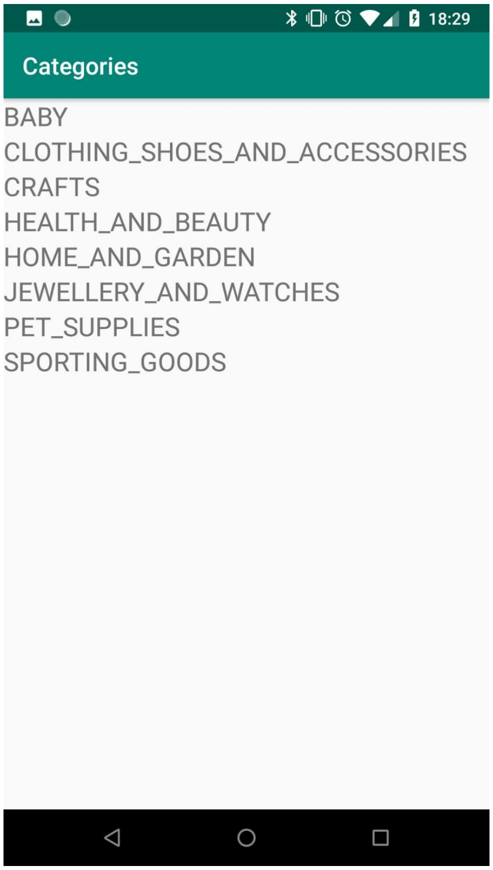
Guest user



Login as user

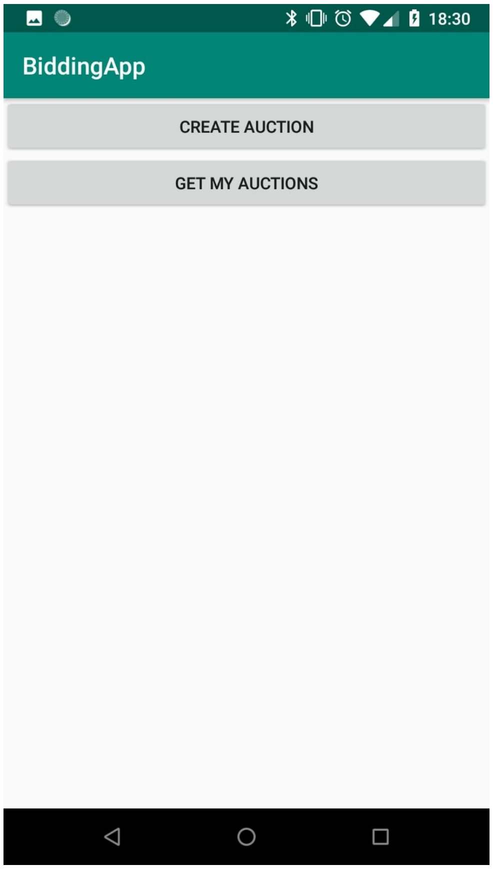


Search options screen

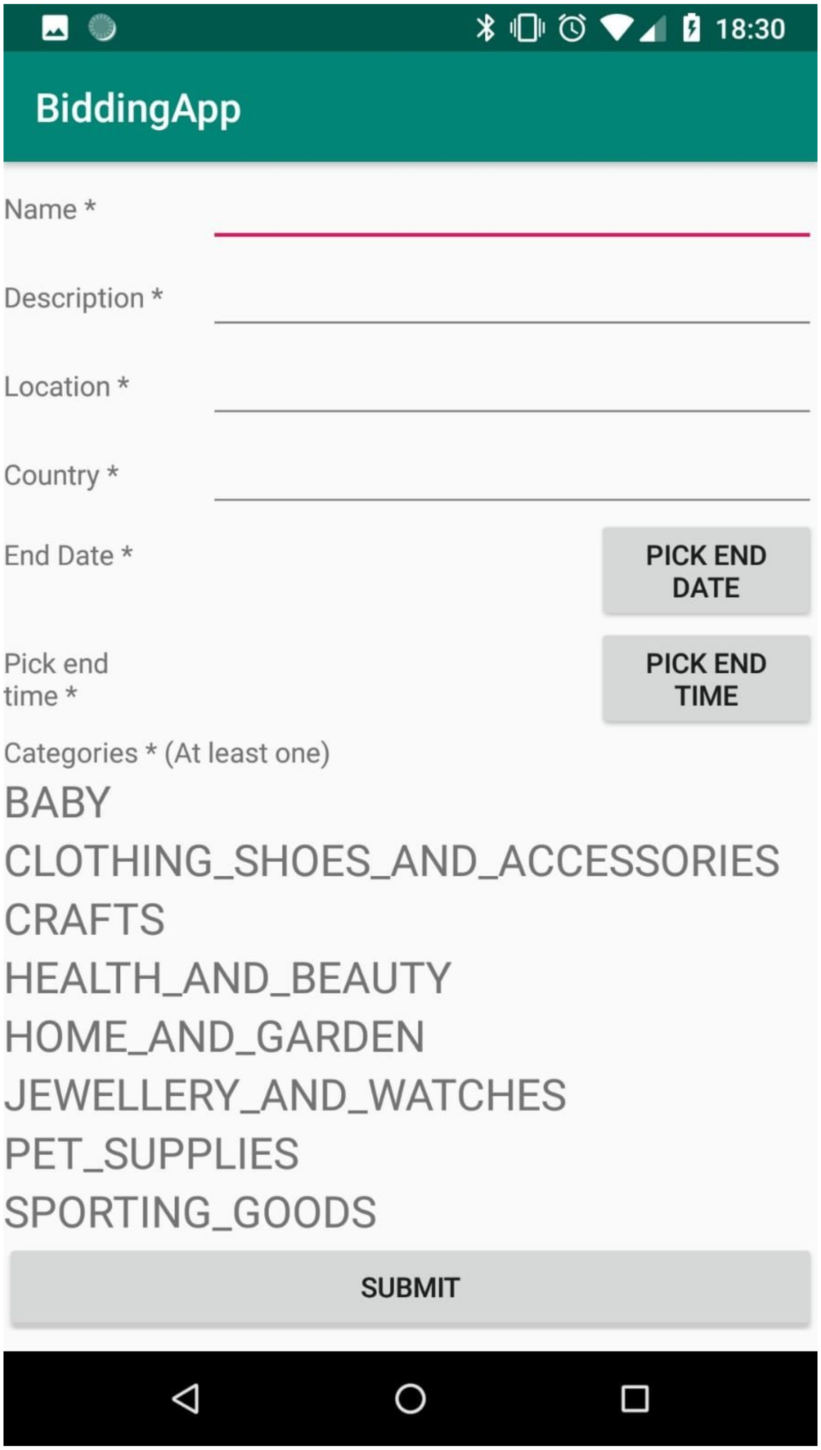


BABY
CLOTHING_SHOES_AND_ACCESSORIES
CRAFTS
HEALTH_AND_BEAUTY
HOME_AND_GARDEN
JEWELLERY_AND_WATCHES
PET_SUPPLIES
SPORTING_GOODS

Search by Category



Manage auctions screen



Create Auction Screen