Ashkon Zariv

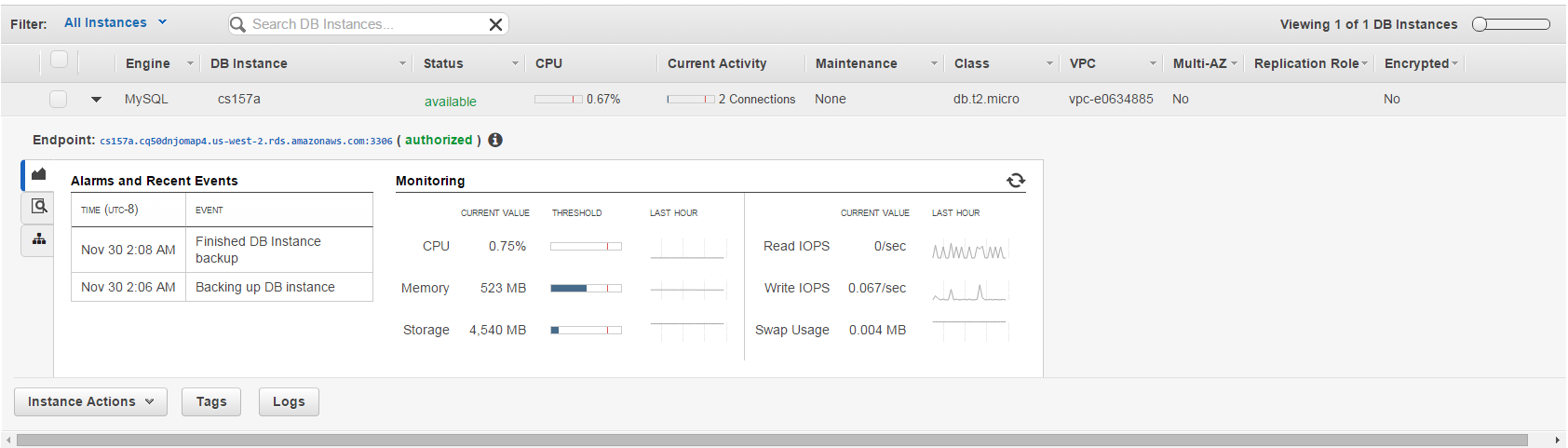
Brendan Kao

11/30/2015

Kao-Zariv Team Project Report

Team Goal: Our goal was to provide an intuitive data access and manipulation system for users who wanted to find an apartment that they can rent in the San Francisco Bay Area. In order to do this, we implemented a database management system that would be able to properly organize user listings in an easily accessible, user-friendly way.

Tools: We used a cloud-based database running on the Amazon Web Service system, MySQL Workbench for database management, Java.swing source code for GUI development, and JDBC for querying and updating the database’s tables.

AWS Console:

Amazon Web Service offers a free remote relational database with a Single-core CPU and 1GB Ram. This database can have an uptime of 750 hours a month.

URL: **cs157a.cq50dnjomap4.us-west-2.rds.amazonaws.com:3306**

Username: guest

Password: password

This database can be accessed remotely through MySQL workbench by opening a new connection using these credentials. The JDBC driver utilizes the above credentials to connect and query the database during runtime.

Database model:

The database consists of three different tables that interact in different way in order to properly organize our data and provide data to display to our GUI. Partitions were implemented on the apartments table in order to optimize access to the data, as it is very densely populated.

Apartments(apartment\_id, name, host\_name, address, zip\_code, accommodates, bathrooms, bedrooms, price, host\_id)

CREATE TABLE `apartments` (

`apartment\_id` int(11) NOT NULL,

`name` varchar(200) NOT NULL,

`host\_name` varchar(45) DEFAULT NULL,

`address` varchar(300) NOT NULL,

`zip\_code` varchar(10) NOT NULL,

`accommodates` int(11) DEFAULT NULL,

`bathrooms` varchar(10) DEFAULT NULL,

`bedrooms` varchar(10) DEFAULT NULL,

`price` varchar(45) DEFAULT NULL,

`host\_id` int(11) NOT NULL,

PRIMARY KEY (`apartment\_id`),

UNIQUE KEY `apartment\_id\_UNIQUE` (`apartment\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1

/\*!50100 PARTITION BY KEY (apartment\_id)

PARTITIONS 15 \*/;

Customers(customer\_id, email, password, name, address, city, state, zip\_code)

CREATE TABLE `customers` (

`customer\_id` int(11) NOT NULL AUTO\_INCREMENT,

`email` varchar(45) NOT NULL,

`password` varchar(45) DEFAULT NULL,

`name` varchar(45) DEFAULT NULL,

`address` varchar(65) DEFAULT NULL,

`city` varchar(65) DEFAULT NULL,

`state` varchar(20) DEFAULT NULL,

`zip\_code` varchar(10) DEFAULT NULL,

PRIMARY KEY (`customer\_id`,`email`),

UNIQUE KEY `customer\_id\_UNIQUE` (`customer\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=latin1;

Requests(request\_id, customer\_id, location, check\_in, check\_out, guests)

CREATE TABLE `requests` (

`request\_id` int(11) NOT NULL AUTO\_INCREMENT,

`customer\_id` int(11) DEFAULT NULL,

`location` varchar(45) DEFAULT NULL,

`check\_in` varchar(45) DEFAULT NULL,

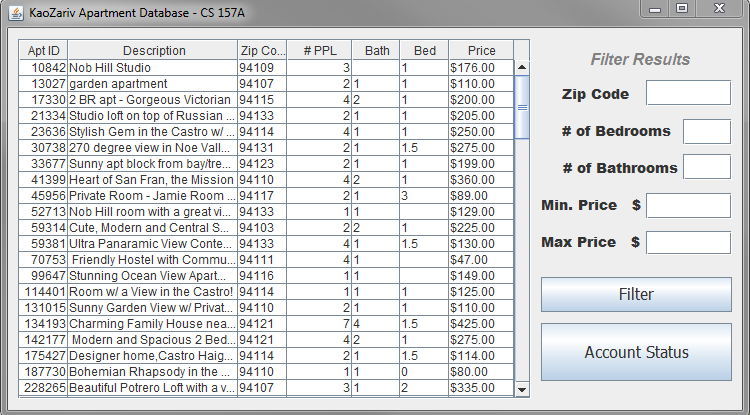
`check\_out` varchar(45) DEFAULT NULL,

`guests` int(11) DEFAULT NULL,

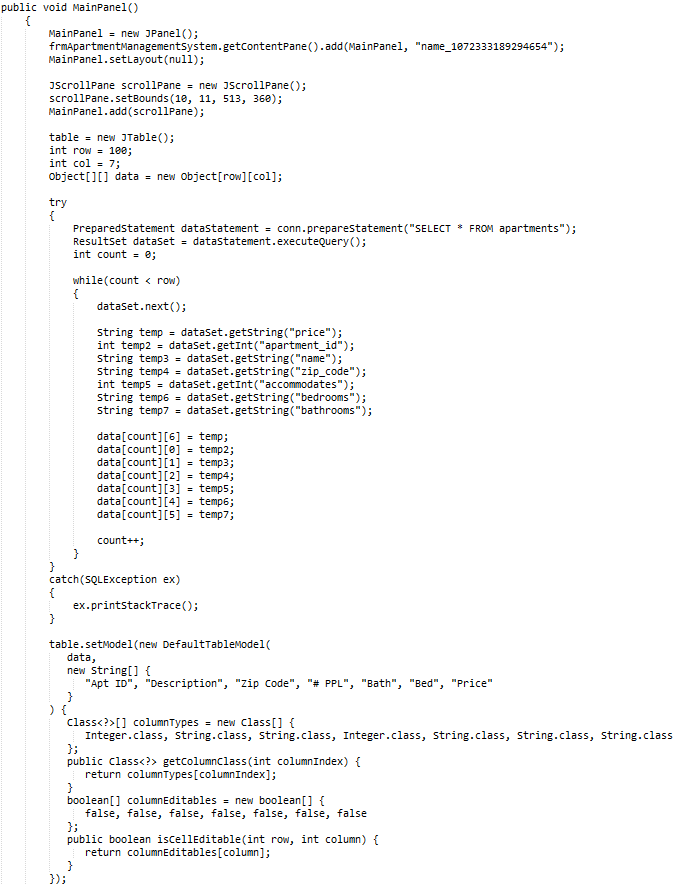
PRIMARY KEY (`request\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=61 DEFAULT CHARSET=latin1;

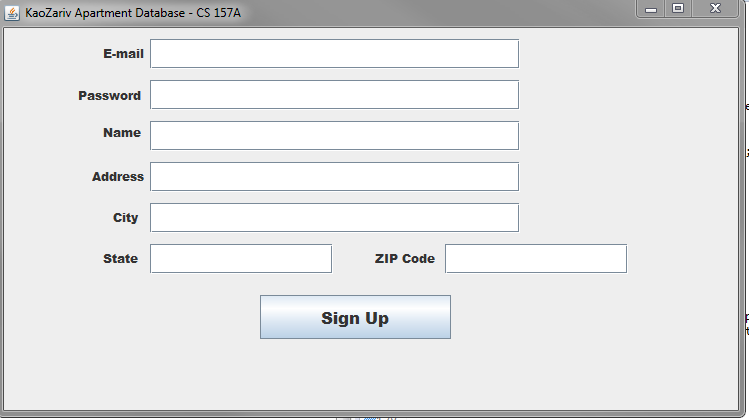
Data Storage and Analitics: The data we used for the apartment listings was scraped off of the AirBnB websites, and imported into the apartments table using a CSV file (attached as listings-new.csv). This CSV file has over 7500 entries and includes only the most recent listings for apartment rentals in the San Francisco area.

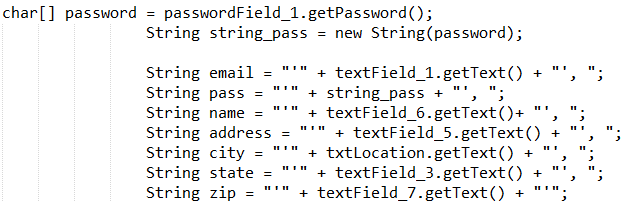
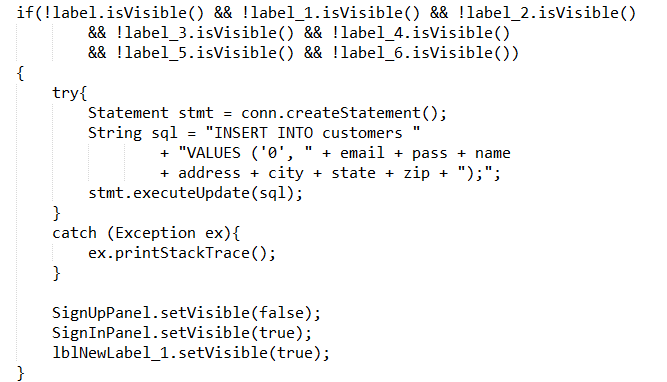
This pane utilizes JDBC and the JTable swing construct in order to display different listings based off of filtering parameters listed on the side.

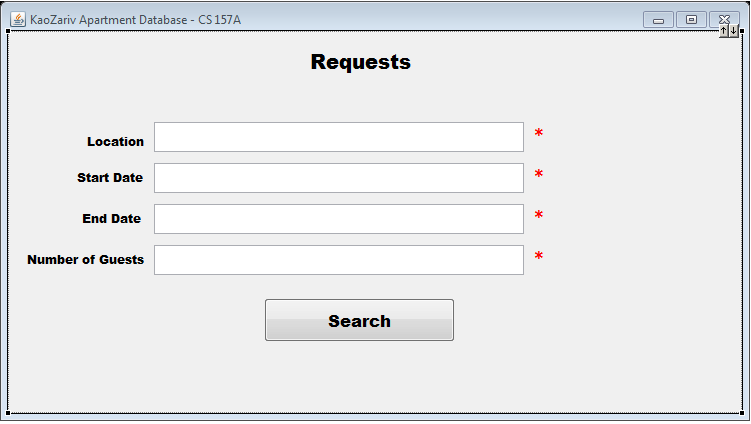
The following source code shows how the JTable is populated when this panel is initialized. On initialization, the first 100 elements of the table are displayed in the JTable by querying the database and writing the data into each individual cell systematically. The table can later be manipulated by inputting values in the text boxes on the side, which will query the database with the given parameters and rewrite the table.



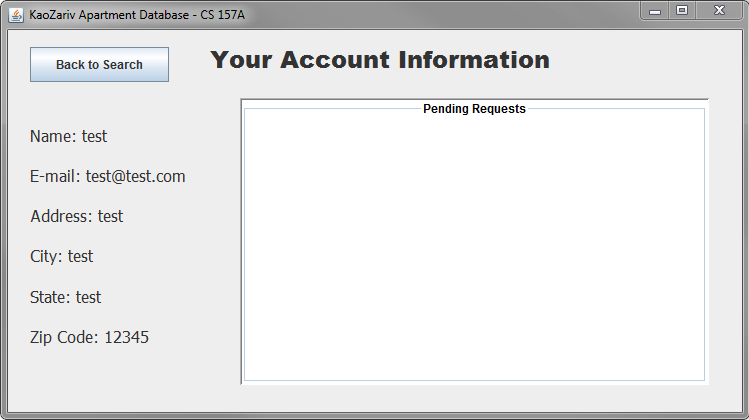
Data for the other two tables are inserted at runtime, either by creating a new user in the case of ‘customers’ or creating a new apartment request in the case of ‘requests’. These data entries are submitted on button click in the GUI panels.

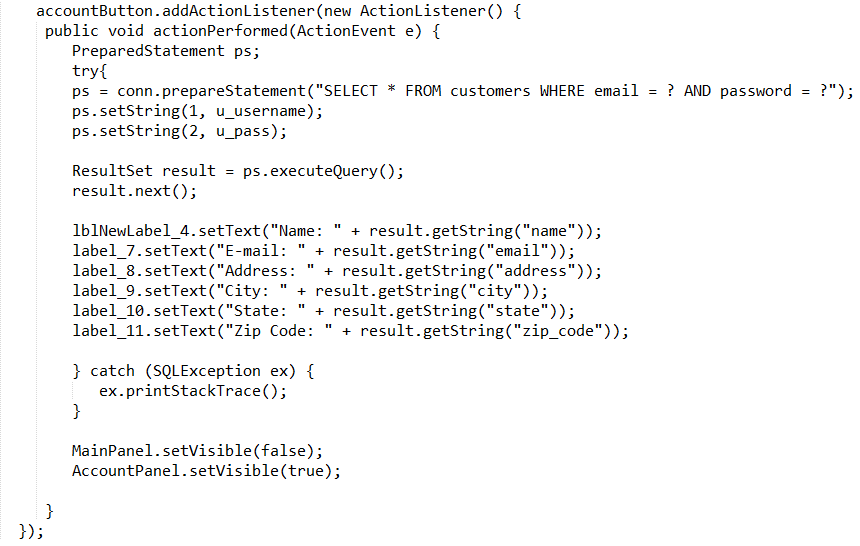


This SignUp Pane allows for a new user to be added to the customers table after filling out a form and submitting it using the ‘Sign Up’ button. These are two snippets of the code which handle the writing of data to the customers table.

On double-click of an element in the JTable, a request form is displayed where the user will be able to put in a request for an apartment based on the listing selected.

These listings will be displayed on the user’s account page, and will be associated to their id in the requests table. This will then be displayed along with a user’s account information in the customers table on the account pane.



This source code snippet writes the data from the customers table on a user’s information onto the account panel.

What were the biggest problems the team had to solve and how were they solved?

Getting everything set up initially was our biggest challenge. Since we had never used JDBC and MySQL workbench before, we ran into some issues connecting the two with Eclipse. By doing some research and by referencing the sample code given to us, we were able to resolve our problems and proceed with coding the GUI. Also, setting up a remote database server was challenging, as we wanted to make a platform that we could both reference and edit instead of having local copies of our database on our own respective computers.

How was performance considered?

Performance was considered through the partitioning of our apartments table in order optimize table access. We also attempted to optimize our code when initializing our swing elements by limiting the amount of data we display at any one time.

How was it tested?

We ran a stress test to see how many entries we wanted to display in our JTable from the database. We found that having a limit of one hundred entries at any one time gave us enough to work with while also allowing for the swing components to stay sharp. We tested it by having other people use our program to make sure all the components were intuitive and user-friendly.

What was each member’s contribution?

Brendan Kao

I contributed by creating some of the panes in the GUI, populating the table containing the data, and writing queries to modify the table.

Ashkon Zariv

Wrote panes for the GUI, setup remote database, imported data into tables from CSV files, initialized and maintained github repository, setup JDBC connections in source code, wrote JDBC queries to modify rows in tables.