Isabella Oledan

CSCI 233

Professor William Sakas

Real Estate App

This application is intended to act as a platform between buyers and sellers of real estate. The core feature is a marketplace of real estate listing in which users can create properties, find information make and offers on any property. The application uses python 2.7 for the main application logic. Specifically, we use the easy\_gui module for the graphical user interface and MySQLdb for interactions with the MySQL database.

The process begins with a user sign up or login. Users either create new credentials or enter their existing credentials which are then verified by a query to the database. From there, users enters the main menu where they can view a property, add a property, manage their properties or view existing offers on their properties. In the view property menu, users are shown a list of all existing properties. Once a property is selected, users are taken into the individual property menu with details about the property, view offer history, or make an offer on the current property. In the view property menu, users are shown all details of the selected property. In the view offer history menu, users are presented with a list of offers. In the make an offer menu, users may enter a bid. In the add property menu, users are presented with a form they use to create a property listing. In the manage properties menu, users are shown a list of all properties that they own. In the view existing offers menu, users are presented with any existing offers on the properties that they own.

This application has utilizes three tables: customer, properties, and offers. In the customer table, there are six fields: customer id, first name, last name, email, password, and phone number. In the properties table, there are twelve fields: property id, property type, number of bedrooms and number of bathrooms, sqft, lot size, minimum price, maximum price, address, city, zip code, and user id (owner of lot). In the offers table, there are six fields: offer id, customer id, property id, bid amount, amount sold, and decision (whether or not the property is sold). To prepopulate the database, there are 10 entries in each table.

A user in this database is represented by an email and a password combination as well as a unique id. Every user is able to author multiple properties, each of which have their own id and is related to the customer table by the customer’s id. Similarly, a user can make an offer on a specific property which is recorded with property id and a customer id. This creates a unique offer listing on the database. A property can have multiple offers.

In the manage properties tab, users are shown a list of all properties they own. An inner join is used to display the property address, city, and property type of the properties that belong to the current user. This selects the necessary fields from the customer and properties table.

In the view existing offers tab, users are presented with any existing offers on the properties that they own. An inner join is used to display the property which the user owns and all existing offers on the property. This selects the necessary fields from the properties and offers table.

To facilitate the login process, we created a user class which takes in a name and user id. This user object is instantiated once a user has either logged in or signed up. This object is called multiple times throughout the app for authentication/identification purposes.

This app is a collection of functions which are like individuals views that contain the application logic and structure of the current interface. For example, the offer history view (def offer\_history(prop\_id)) contains a query that dynamically changes based on the property that the user chooses to view. The query is the then sent to the database and returns a list of offers on that property. Finally, the data is organized and presented in the gui with the easy\_gui api. Each view can be connected to each other in order to make a coherent application.

In addition, the app has a collection of helper functions that are used to send queries to the database and to return the data in an organized format. For example, a helper function (def wrap(string)) is used to correctly format a string for database entry.