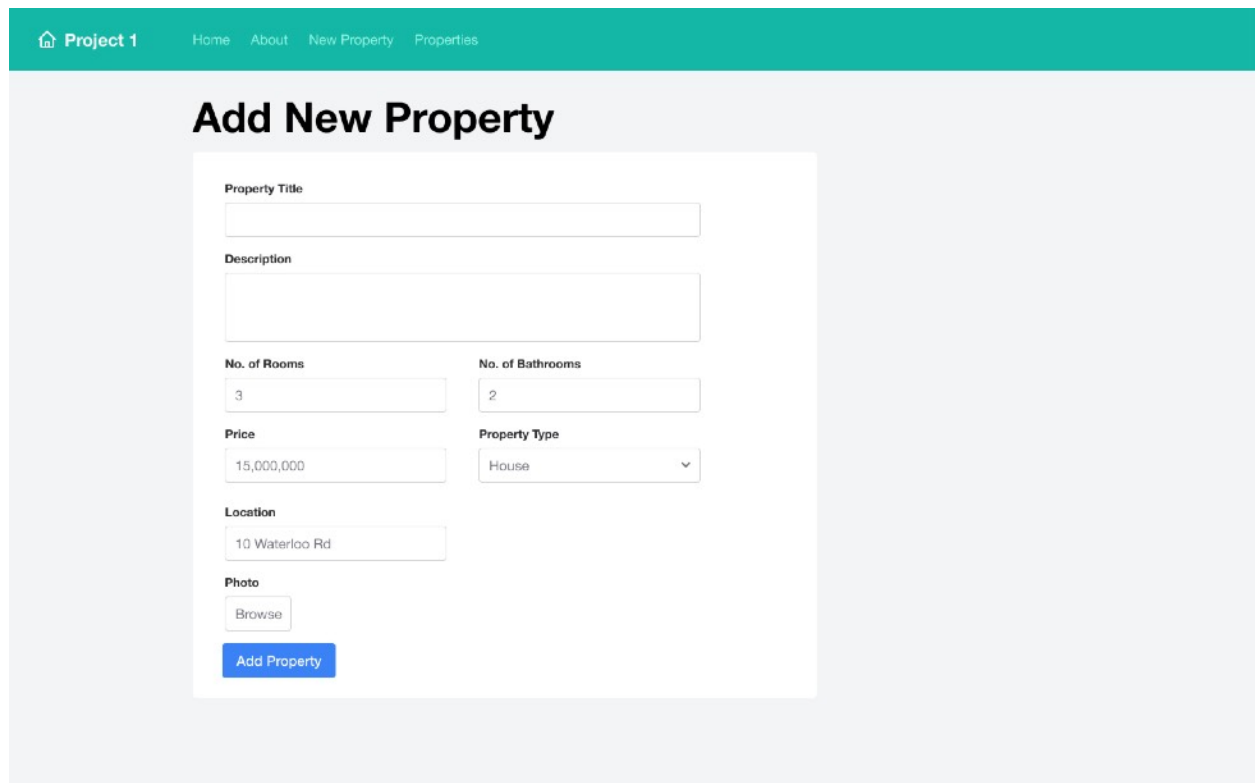


# INF03180 Project 1 (30 marks)

**Due: March 21, 2021 at 11:55 PM**

At the end of this project you will have a Flask based application that can accept and display information on properties available for rent/sale. The property information will also be stored in a PostgreSQL database.



The screenshot shows a web application interface for adding a new property. At the top, there is a teal navigation bar with a home icon and the text 'Project 1', followed by links for 'Home', 'About', 'New Property', and 'Properties'. Below the navigation bar, the main heading 'Add New Property' is displayed in bold. The form itself is a white box with a light gray border, containing several input fields and a submit button. The fields are labeled 'Property Title', 'Description', 'No. of Rooms', 'No. of Bathrooms', 'Price', 'Property Type', 'Location', and 'Photo'. The 'Property Type' field is a dropdown menu currently showing 'House'. The 'Photo' field has a 'Browse' button. At the bottom of the form is a blue 'Add Property' button.

**Project 1** Home About New Property Properties

## Add New Property

Property Title

Description

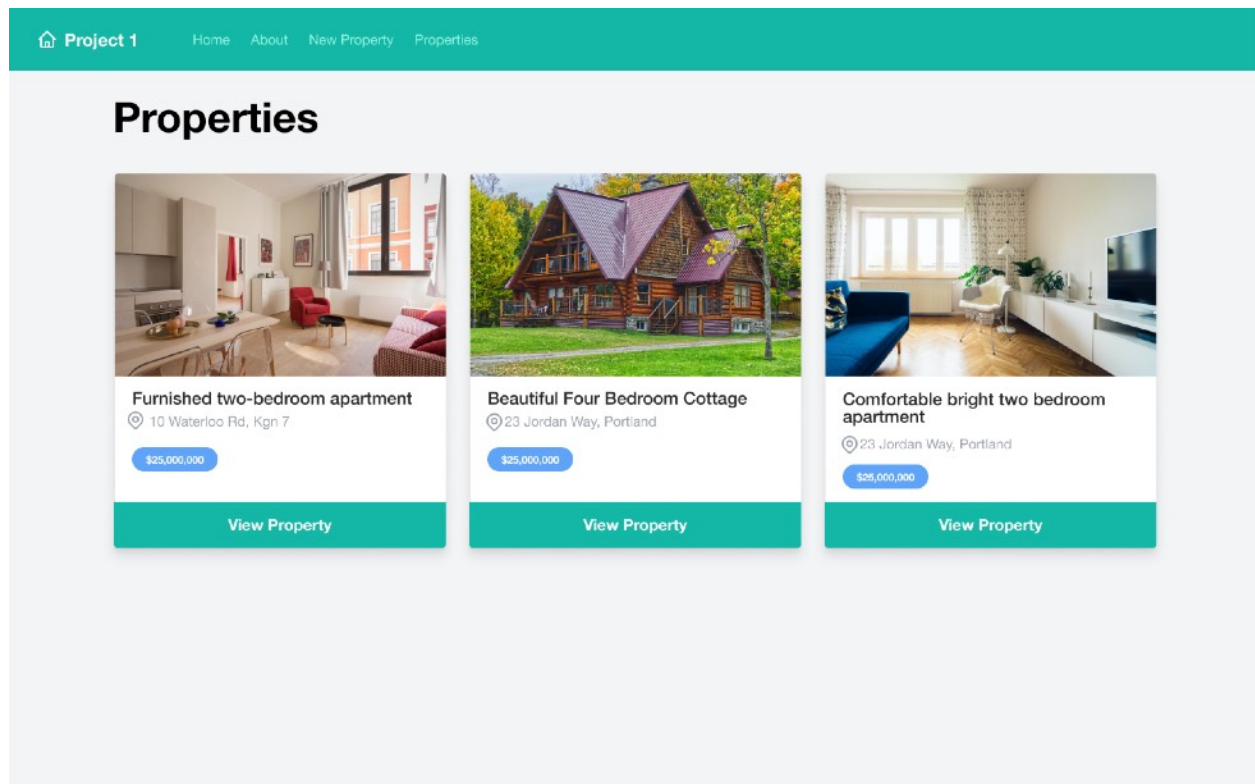
No. of Rooms  No. of Bathrooms

Price  Property Type

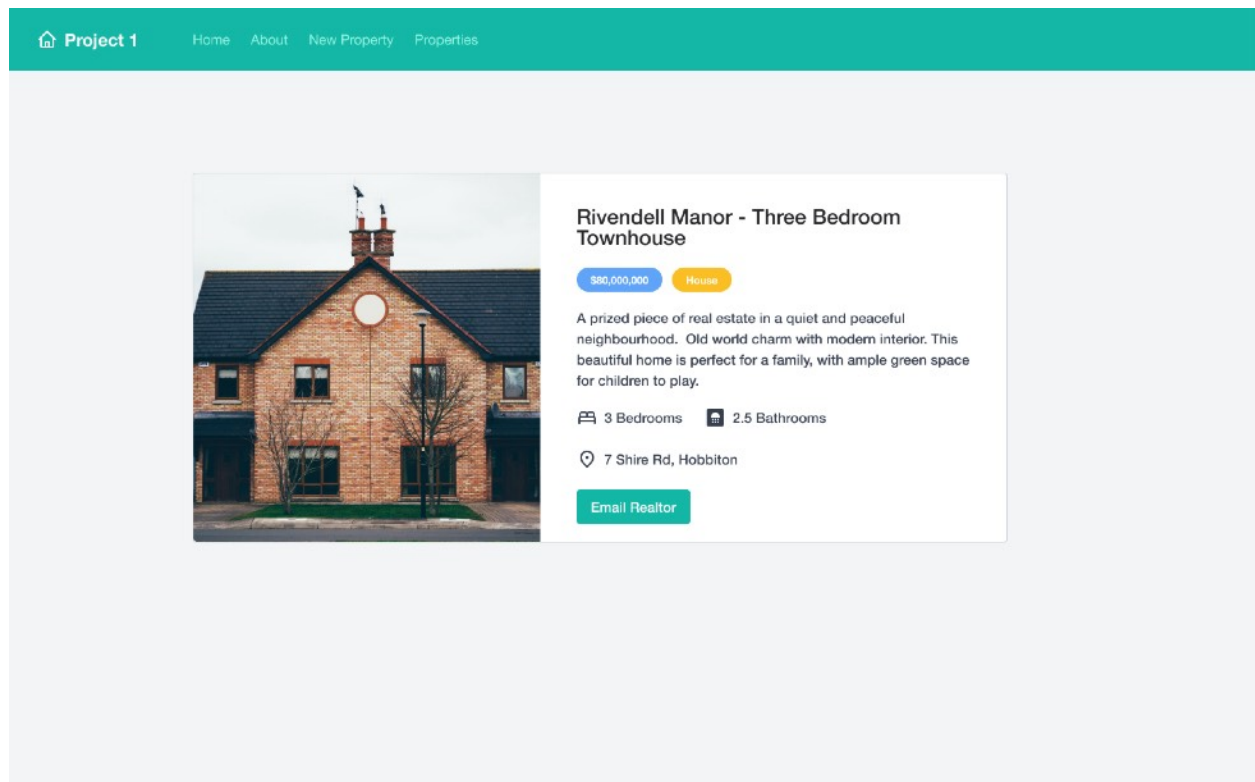
Location

Photo

*Figure 1: Add New Property Form*



*Figure 2: List of Properties available*



*Figure 3: Individual Property Information*

## Specifications

### Part 1

Use the knowledge you have gained in your Lectures, Tutorials and Labs to create a Flask App that accepts input for a user to create a new property and also display both a list of those properties as well as an individual property.

Feel free to use the Flask Starter code at [https://github.com/uwi-info3180/flask\\_starter](https://github.com/uwi-info3180/flask_starter) as a basis to start your application. Remember though you will also need to update your **requirements.txt** file to include any additional libraries you need for your project.

The following routes will need to be created and appropriate templates rendered:

1. **"/property"** For displaying the form to add a new property. (See *Figure 1*)
2. **"/properties"** For displaying a list of all properties in the database. (See *Figure 2*)
3. **"/property/<propertyid>"** For viewing an individual property by the specific property id. (See *Figure 3*)

The add new property form must be created using Flask-WTF and contain the following fields:

1. Text fields for **title**, **number of bedrooms**, **number of bathrooms**, **location** and **price**.
2. Select (option) field for **type** (whether House or Apartment)
3. Textarea field for a short **description**.
4. File upload field called **photo** which accepts the image of the Property.

Upon submission, the form should make a **POST** request to the **"/property"** route and validate the user input to prevent bad data. A unique **id** should be generated (e.g. an auto incrementing id field in your model) and also the filename of the **photo** for the new property should be saved in the database. All of this input must be stored in a PostgreSQL database.

Once a property is successfully added the user should be redirected to the

**"/properties"** route and a flash message should be displayed notifying the user that the property was successfully added.

**Note:** You **MUST** generate a migration file for your *Property* model so that the database can be recreated.

For the list of properties page, you are to display a list of *all* properties. Each property should display the *photo*, *title*, *location*, price and a *button/link* that when clicked should carry you to the individual property page where you can view more details. (See Figure 2).

On the Individual property page, you should have the property's *photo*, *title*, description, *no of bedrooms*, *no of bathrooms*, *location* and *price*, along with *whether or not it is a house or apartment*. You should also have a *button* to "Email Realtor" (See Figure 3). Please note the button does not need to do anything for this project.

**Note:** You must also add navigation links to the **"/property"** and **"/properties"** routes to your **header.html** file.

## Part 2

The finished application should be deployed to Heroku. If you haven't already done so, ensure that you sign up for an account on the Heroku website (<https://heroku.com>) and then do the following.

**Note:** You will also need the Heroku CLI. This will need to be installed on your local machine if you haven't already done so. To install the Heroku CLI, see instructions at <https://devcenter.heroku.com/articles/heroku-cli>.

```
heroku login
heroku apps:create
git push heroku master
```

**Note:** If you need to set any environment variables on Heroku (e.g. for an uploads folder or secret key or database url) then you can do so with the following command at the command line:

```
heroku config:set SECRET_KEY="my-super-secret-key"
```

## Creating a Postgres database on Heroku

You will also need to ensure that you have a Database setup on Heroku for your application. To create and provision the Postgres database add-on and create a PostgreSQL database on Heroku, follow the instructions at the following link:

<https://devcenter.heroku.com/articles/heroku-postgresql#provisioning-the-add-on>

**Note:** You will be using the **hobby-dev** plan when creating your database.

```
heroku addons:create heroku-postgresql:hobby-dev
heroku config -s
```

You should then see something like the following:

```
postgres://
yecsapnzl1ttgcb:8860d3512549e3ebba04aa68ede74496e30041ff
```

**458ea1d46d7c4ed7f5402af0@ec2-54-221-244-196.compute-1.amazonaws.com:5432/d9d6gbbcjoc71g**

This is the URI that you will use in your **SQLALCHEMY\_DATABASE\_URI** config option in your Flask Application. Ensure that you change driver (at the start of the URI) from **postgres** to **postgresql** for Flask SQLAlchemy.

**Note:** If you are using the **config.py** file that we have been using in your previous labs and already have **SQLALCHEMY\_DATABASE\_URI = os.environ.get('DATABASE\_URL')** then this should work automatically once you push to Heroku as the DATABASE\_URL environment variable will already be set on the Heroku server.

You will also need to ensure you make a modification to your Heroku **Procfile**. This will allow you to run the migration you created earlier (and any other future migrations) to create/update your Heroku Database. The updated **Procfile** will look similar to this:

```
release: python manage.py db upgrade --directory migrations
web: gunicorn -w 4 -b "0.0.0.0:$PORT" app:app
```

## Submission

Submit your code via the "**Project 1 Submission**" link on OurVLE. You should submit the following links:

1. Your Github repository URL for your Flask app e.g. <https://github.com/{yourusername}/info3180-project1>

2. Your URL for your Heroku app e.g. <https://{yourappname}.herokuapp.com>

## Grading

1. 3 routes /property, /properties, /property/<propertyid> should be defined along with their respective view functions. (3 marks)
2. /property has form with *title*, *description*, no of bedrooms, no of bathrooms, location, file upload field called photo and select property type field. (4 marks)
3. The Form successfully submits and property added to database. (2 mark)
4. File successfully uploads and filename stored in database. (2 marks)
5. Ensure id is automatically generated for property. (1 marks)
6. You should be able to view the specific user property at /property/<propertyid> (5 marks)
7. You should be able to view a list of properties added at /properties (2 marks)
8. A Property database model should exist in models.py. (2 marks)
9. Database Migration(s) should be created and the database should be able to be recreated from that migration. (2 marks)
10. Navigation links for "New Property" and "Properties" are in the navigation bar at the top. (2 marks)
11. Deployed to Heroku and the application works there. If you are able to create a property and view it then it also means the database was created and connected to properly on Heroku. (2 marks)
12. 'Property', 'Properties' and 'Add Property Form' should look similar to screenshots. (3 marks)