Distributed systems project

Members

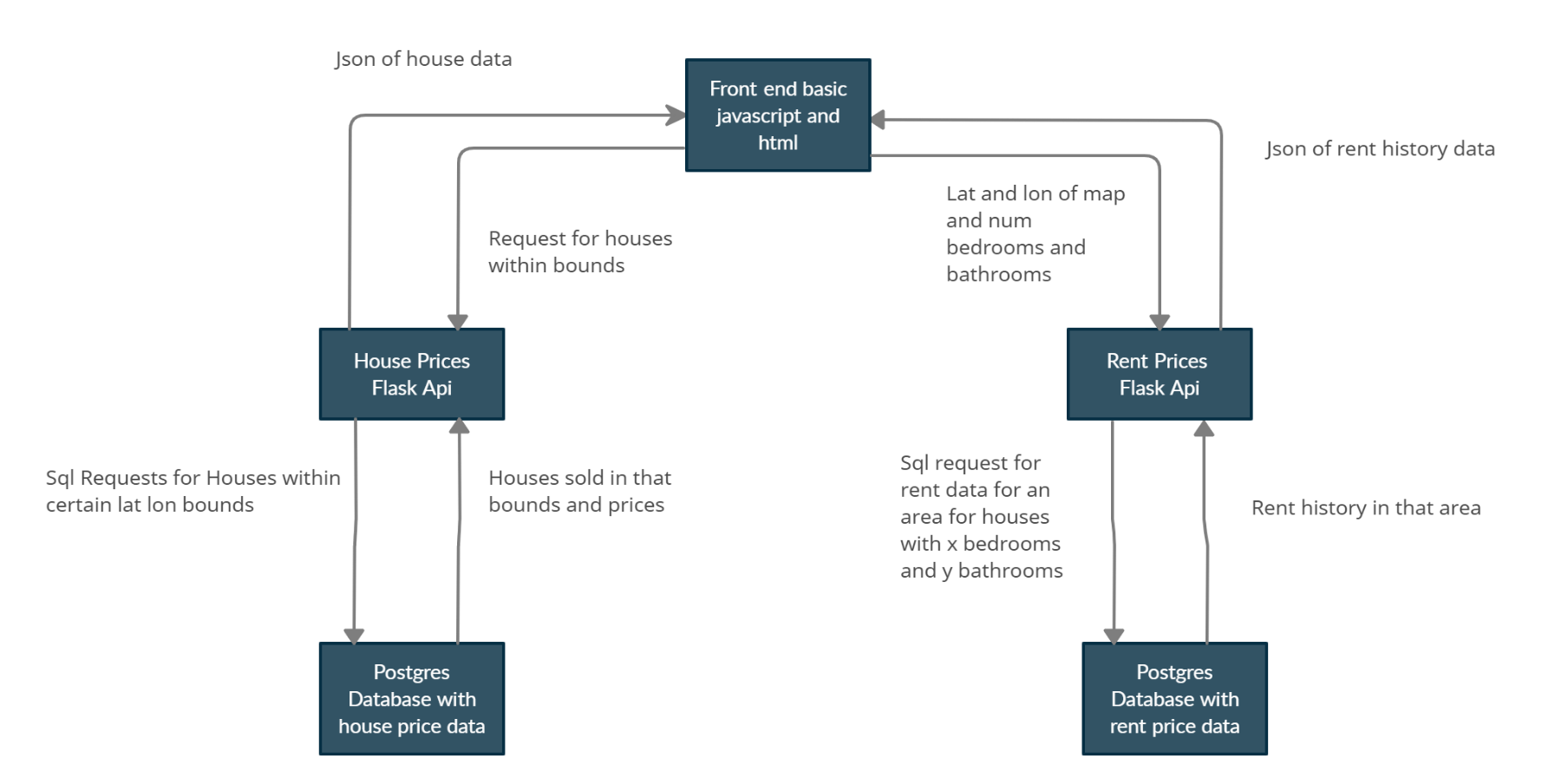
Martin Conlon 16403502

Matthew dwyer

Michael Bradley

Robert Collins

Introduction

The application which we have built is an application which can take in an address and will return you an interactive map which will show all the houses from the property price registry around that location which have sold and how much for. Along with this you will be given the average rent price for that area based on the number of bathrooms and bedrooms a house has. The data for the property price registry and rent data was gotten from the government website so it is real data. This information can be used during the evaluation of how good a particular property would be as an investment. This was built using a distribute architecture which is in the form

Each of these services were made into a docker image and then added to separate pods in a Kubernetes cluster and connected using ingress.

Front end:

This was developed using leaflet js for the mapping along with axios and html for the api requests and layout. I also used the google autocomplete api for searching for addresses as I couldn’t find any free versions which were close to the quality of googles.

House price api:

This was developed using a basic flask api which used sql alchemy to communicate with the databases. The data for this was gotten from the property price registry. This data only contains the addresses of sold houses so this needed to be converted to a lat and lon to display it on the map by reverse geocoding. Luckily I found a csv containing this online which was used to map the houses. This was then converted to a python docker image and added to a Kubernetes pod.

Rent price api:

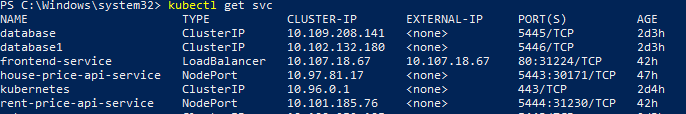
The data for this was gotten from the irish government website which gives us a breakdown of the average rents in a particular area based on the number of bedrooms and bathrooms. To do this I had to convert a latitude and longitude into the associated area field in the database. This caused a lot of trouble since there wasn’t any shapefile associated with the areas in the database so I had to use a imperfect system of geocoding the lat and lon . Using the address returned I would extract the county and location and search for that in the database. It is’nt a perfect system since it cant always be found since the address returned by our geocoding api doesn’t always match what we have in our database. This could be solved by creating a shapefile but this wasn’t feasible for this project. This flask api was then converted to a docker image using the python base image. We then added this to a pod and changed the ip to that of the internal Kubernetes ip for our rent price postgres database.

Databases:

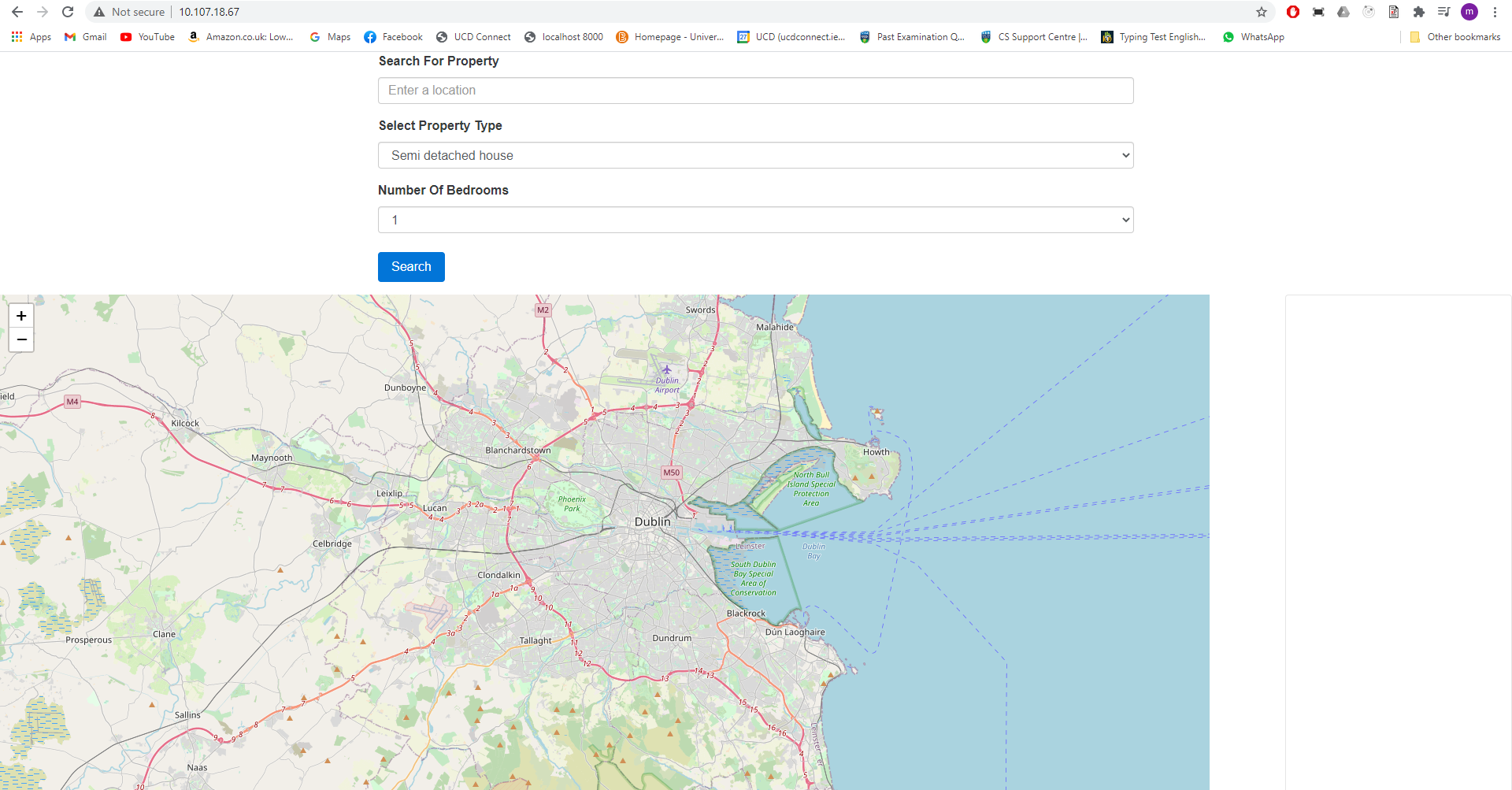
For my database I chose to use postgres for both our house and rent services. The reason I chose this is because we don’t plan on needing scaling of the amount of data in our database since we have a fixed amount of data which isnt large enough to require a noSql database. I did this using a postgres docker image. I then added this to our Kubernetes pod and gave it a persistent volume claim to store our data. To migrate over my data I went onto the bash command line of my house and rent price api which I have written a python script to add all the data to our databases from csv files.

Docker and Kubernetes:

When creating this I initially made the application run using a docker-compose file. Using this I then used a tool called Kompose which converts docker compose files into Kubernetes cluseters however it was only successful in creating the databases. For the house price and rent price services I simply added the docker image of them to the cluster docker directory and then used a yaml file to attach that docker image to a Kubernetes pod. For the front end I used a nginx base image to make a docker image out of our javascript and html. I then added this to the Kubernetes cluster similar to above and then used a yaml file with type loadbalancer so that the pod would be assigned an external ip which we can access the website from. To run the Kubernetes cluster locally I used minikube. I also attached this to the Kubernetes dashboard so that I could easily see the cluster was running as expected.



As seen here we have the frontend accessible to us on the ip 10.107.18.67. If we go to this ip in a browser we get our application



Which when we enter the an address works as expected.

