National College of Ireland

Final year Software project

Smartmove

Anthony Bloomer
BSc in Computing (Data Analytics)
Student Number: x13114271
Email: anthony.bloomer@student.ncirl.ie

Declaration Cover Sheet for Project Submission

SECTION 1 Student to complete

Name:

Student ID:

Supervisor:

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Name:

Date:

Abstract

Smartmove is a data platform which aims to provide businesses and consumers with insight into the property market in Ireland and the United Kingdom. Smartmove is made up of several software components. The main features include a Data Warehouse, REST API and in-depth data analysis using R. The platform retrieves data from multiple sources including Daft.ie, Zoopla and the Property Price Register. ETL tools for retrieving and processing data from these sources have been made as part of this project. The REST API allows third-party developers to use Smartmove data in their own applications. The endpoints include features such as the ability to retrieve JSON data that can easily be consumed by the Google Charts API. Smartmove is an open source project and all of the software components are available on Github.

The potential users of Smartmove include:

1. Businesses who wish to gain intelligence on how to better target areas.
2. Consumers who intend to research property prices before purchasing.
3. Developers who wish to use Smartmove data in their own applications.

Contents

0.1 Introduction

0.1.1 Background

Smartmove was initially known as Incomestack. The initial idea was to provide businesses with housing price data such that they can better target areas. Overtime, more use cases have been found including a REST API that allows third-party developers to include Smartmove data in their own applications. Smartmove fills a niche in the market with a unique idea with very little competitors in this space. The Central Statistics Office (CSO) does provide property sale information but it is not easy to use and is not as granular in the results it gives back. For example, the CSO provides sale pricing for County Dublin but not cities within Dublin such as Blackrock, Blanchardstown, Tallaght, etc. Smartmove solves this problem by breaking down property sale statistics by town, county and country. This data can then be retrieved by third party developers using the Smartmove REST API. Smartmove also provides real-time data using the Zoopla API as well as scraping data from Daft.ie since they do not provide a public API.

0.1.2 Aims

The aim of this project is to provide businesses and consumers with intelligence enabling them to gain insight into the property market in Ireland and the UK. Smartmove not only aims to be a successful software project but also a viable product that consumers find valuable. Smartmove aims to provide third party developers with the ability to retrieve data using the Smartmove REST API. The API includes several endpoints including the ability to retrieve JSON data that can easily be consumed by the Google Charts API.

Use cases for Smartmove

- Targeted Advertising
- Investing
- House Price Prediction
- Mobile Applications

0.1.3 Technologies

The following technologies are used in the Smartmove platform.

Python 2.7

Python is a general purpose programming language that is widely used in software development as it is mature, easy to use and has a wide range of third party libraries available for it. For the REST API, Smartmove uses Flask. Flask is a micro-framework written in Python. Flask includes many extensions that extend the functionality of Flask. Flask-restplus is an extension that provides useful features that help in API development such as the ability to automatically generate API documentation.

MySQL

MySQL is an open source Relational Database Management System (RDBMS). MySQL is used for the operational database as well as the Smartmove Data Warehouse.

RStudio

R is an open source programming language for statistical computing. R provides R Studio which is a environment that allows users to write code which provides instant feedback. For example, users can write code to create a chart and the output is displayed automatically in the environment. R and the RStudio IDE was used in the data analysis component of this project.

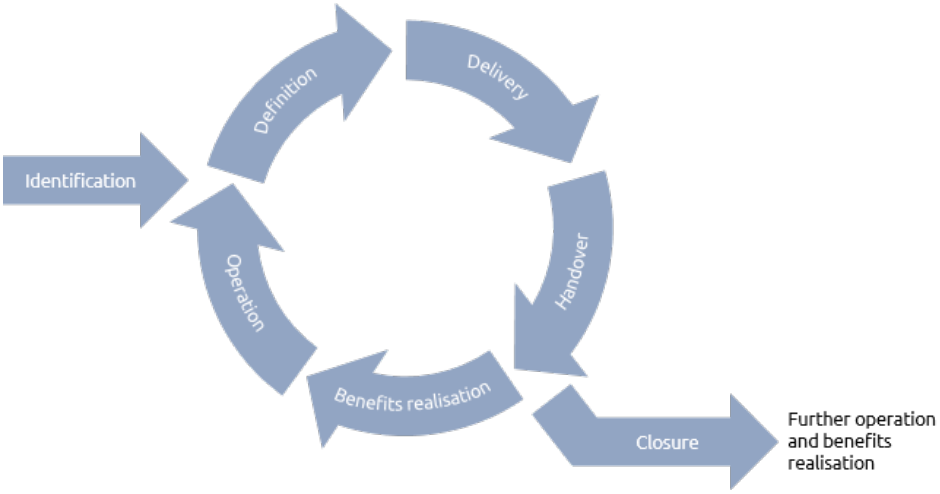
0.1.4 Structure

This section will briefly discuss the structure of this report. In section 0.2, the methodology employed will be introduced. Section 0.3 will briefly introduce the reader to the data sources used in the Smartmove platform. In Section 0.4, the requirement specification will be discussed including the functional and nonfunctional requirements. Section 0.5 discusses the data warehouse architecture. In section 0.6, the reader is introduced to the technical details of the REST API. Section 0.7 presents the Graphical User Interface of the Smartmove platform. In section 0.8, a technical overview of the implementation is discussed including the REST API, ETL process and the Data Analysis and Visualization component of this project. Section 0.9 discusses the testing methodologies used. Section 0.10 provides the installation and usage manual for the components of the Smartmove platform including the ETL scripts for the Smartmove databases, the REST API,

and the Daftlistings and Zoopla libraries built as part of this of this project.

0.2 Methodology

The Smartmove platform has been developed using an iterative model. "The iterative model does not start with a full specification of requirements. It begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model." (Level et al., 2017) The following figure illustrates the iterative approach.



0.3 Data Sources

This section will briefly discuss the data sources used as well as the ETL tools built to assist in retrieving data from those data sources.

0.3.1 Daft.ie

Daft.ie is Ireland's largest property website. Unfortunately, Daft do not provide a public API. This problem led to the development of Daftlistings. Daftlistings is a python library Smartmove has created. It enables programmatic interaction with Daft. Users can filter properties by location, sale type, price and property type. The library is open source and has gained considerable developer traction. Please refer to section 0.10.4 for the libraries usage manual.

0.3.2 Zoopla.co.uk

Zoopla is a leading property price website in the United Kingdom. Zoopla provide a public API that allow developers to create applications using hyper local data on 27m homes, over 1m sale and rental listings, and 15 years of sold price data. (Developer.zoopla.com, 2017)

Smartmove has developed a Python wrapper for the Zoopla API as it enables an easy way to interact with the API. The library includes helpful methods including the ability to search property listings, retrieve average area sold prices, area zed indices, etc. This library has been made open source and is available on Github. Please refer to section 0.10.5 for the libraries usage manual.

0.3.3 Property Price Register

The Residential Property Price Register is produced by the Property Services Regulatory Authority (PSRA) pursuant to section 86 of the Property Services (Regulation) Act 2011. The property price register includes the date of sale, price and address of all residential properties purchased in Ireland since the 1st January 2010, as declared to the Revenue Commissioners for stamp duty purposes.(Property Price Register, 2016). The Property Price Register frequently update their website with new property sales. The user can download this data in CSV format. ETL scripts have been written to extract the PPR data and load into the Smartmove operational database.

0.4 Requirement Specification

0.4.1 Functional Requirements

0.4.1.1 REST API Requirements

This section will present the functional requirements for the REST API.

Properties

The system shall provide a properties endpoint enabling the user to retrieve data on properties and filter by sale type, location and price.

Charts

The system shall provide a charts endpoint enabling the user to retrieve data that can easily be consumed by the Google Charts API.

Towns

The system shall provide a towns endpoint enabling the user to retrieve town sale statistics.

Counties

The system shall provide a counties endpoint enabling the user to retrieve county sale statistics.

Countries

The system shall provide a counties endpoint enabling the user to retrieve country sale statistics.

0.4.1.2 Display Descriptive Charts

The system shall allow the user to view charts using a KPI dashboard.

0.4.1.3 Predict Future Prices

The system shall allow the user to view predicted sale prices.

0.4.1.4 Retrieve data from the Property Price Register

The system shall retrieve and process the PPR data once it is publicly available.

0.4.1.5 Retrieve data from Zoopla.co.uk

The system shall regularly retrieve and process property sale information from Zoopla.co.uk.

0.4.1.6 Scrape data from Daft.ie

The system shall regularly scrape and process property sale information from Daft.ie.

0.4.1.7 Persist Data

The system shall store the data collected from the data sources in an operational database.

0.4.1.8 Sign-Up

The user shall have the ability to register at Smartmove.

Sign Up Use Case

Sign up

Scope

The scope of this use case is to allow a user to sign up at Smartmove.

Description

This use case describes the sign up process.

Flow Description

Precondition
None

Activation
This use case starts when the user visits the sign up page.

Main flow

- The user enters their desired their username and password.
- The system validates the users details.
- The user is redirected to the homepage.

Alternate flow
-

Exceptional Flow
-

Termination
-

Post condition
The system goes to a wait state.

0.4.1.9 Login

The user shall have the ability to login at Smartmove.

Login Use Case

Scope

The scope of this use case is to allow a user to login

Description

This use case describes the login process.

Flow Description

Precondition
None

Activation
This use case starts when the user visits the login page.

Main flow

- The user enters their their user name and password.
- The system validates the users details.
- The user is redirected to the homepage.

Alternate flow
-

Exceptional Flow
-

Termination
-

Post condition
The system goes to a wait state.

0.4.1.10 Create Application

The user shall have the ability to create a new application. Creating an application provides the user with an API Key that can be used authorizing requests to the API.

Create Application Use Case

Scope

The scope of this use case is to allow a user to create a new application.

Description

This use case describes the process of creating a new application.

Flow Description

Precondition
The user is logged in.

Activation
This use case starts when the user visits the application creation page.

Main flow

- The user enters their their applications name and description.
- The system creates an API key using UUID.
- The system stores the users application details.
- The user is redirected to their applications page.

Alternate flow
-

Exceptional Flow
-

Termination
-

Post condition
The system goes to a wait state.

0.4.2 Nonfunctional Requirements

0.4.2.1 API Rate Limiting

The Smartmove API should including rate limiting. The API should allow 2,000 requests per day and 100 requests per hour.

0.4.2.2 API Authentication

The Smartmove API should require authentication. To use the Smartmove API, you must include an API key when loading the API.

0.4.2.3 Unit Testing

The software components should be tested to guarantee stability.

0.4.2.4 Data Recovery

The system should perform daily backups of the Smartmove data.

0.4.2.5 Software Documentation

The software components should include documentation.

0.4.2.6 Open Source

The software components should be made open source.

0.4.2.7 Portability

The software components should be portable meaning the components run across different operating systems and hardware.

0.4.2.8 Availability

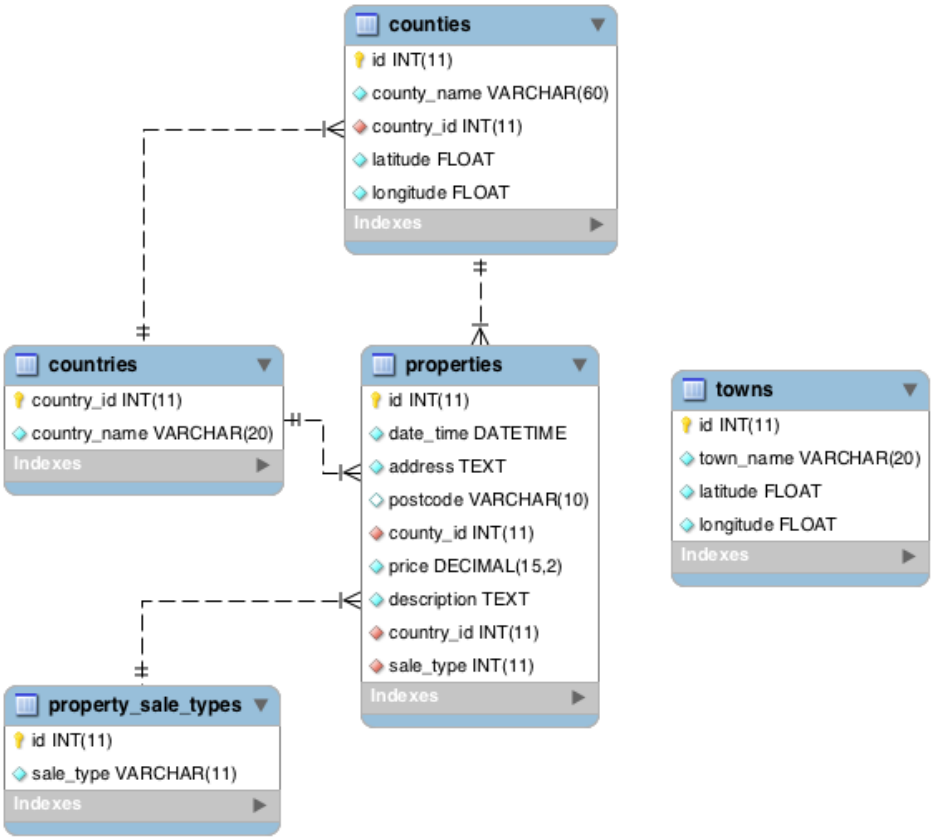
The REST API should experience very little downtime.

0.5 Data Warehouse

As defined by Bill Inmon, a Data Warehouse is a "subject-oriented, integrated, time-variant, non-updatable collection of data used in support of management decision-making processes." Data is extracted from multiple data sources and loaded into the Smartmove Data Warehouse. The data warehouse breaks down sales figures by counties, towns and dates.

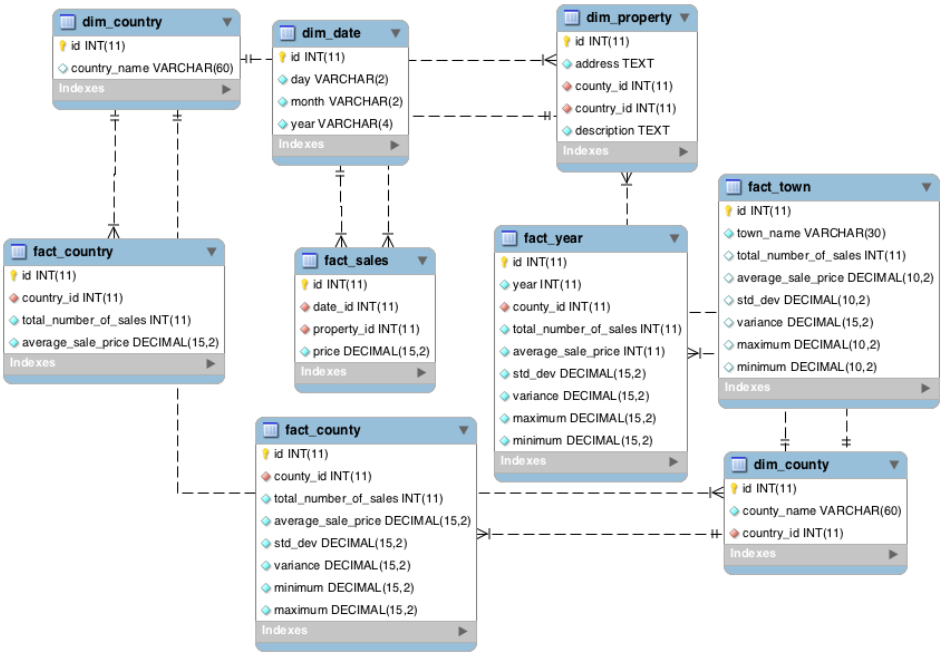
0.5.1 Operational Database Design

The operational database stores data from multiple sources including Daft.ie, Zoopla and the Property Price Register. The Operational database schema is presented below.



0.5.2 Data Warehouse Schema

Below you can find the schema for the data warehouse. The data warehouse stores aggregated house price information and breaks data down by town, county and year.



0.5.3 Data Transformation

This section will briefly discuss the data transformations and SQL procedures required in order to build and load the data warehouse. Several SQL procedures have been written to compute statistics from the data in the operational database and then load into the data warehouse. For example, the following code snippet computes descriptives for each county and stores the aggregated data in the data warehouse.

```
DELIMITER //

CREATE PROCEDURE counties()
BEGIN
  DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE _id VARCHAR(40);
  DECLARE cur CURSOR FOR SELECT id FROM smartmove.counties;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done := TRUE;
  OPEN cur;
  testLoop: LOOP
    FETCH cur INTO _id;
    IF done THEN
      LEAVE testLoop;
    END IF;
    CALL insert_county(_id);
  END LOOP testLoop;

  CLOSE cur;
```

```
END

//

DELIMITER //

CREATE PROCEDURE insert_county(IN _county_id INT(11))
BEGIN
  INSERT INTO smartmove_data_warehouse.fact_current_sales
      (insert_datetime, county_id, total_number_of_sales,
      average_sale_price)
  SELECT CURDATE(), county_id, COUNT(*) AS total_number_of_sales,
      ROUND(AVG(price), 2) AS average_sale_price
  FROM smartmove.properties AS p
  WHERE county_id = _county_id
  AND sale_type = 2
  AND country_id = 1;
END

//

CALL counties()
```

0.6 REST API

Web Services is software designed to support machine-to-machine communication and interaction over a network.(W3, 2017) The advantages of web services include:

- Services offer an interface to access functionality.
- Open protocols are used for communication.
- Services can be distributed and are reachable over a network.
- Machine-to-machine interoperability.

The Smartmove API is built using the Flask micro-framework. The API makes use of the Flask-restplus extension. Flask-restplus extends Flask with functionality for quickly building REST APIs. The extension includes features such as the ability to automatically generate API documentation using Swagger.

0.6.1 Rate Limiting

The Smartmove API includes rate limiting. The API allows 2,000 requests per day and 100 requests per hour. This feature is common in most web

APIs. It is useful from a security and business perspective as it prevents users from sending too many requests to the API.

0.6.2 Authentication

The Smartmove API requires authentication. To use the Smartmove API, you must include an API key when loading the API. This adds an extra layer of security to the API and will allow Smartmove to monitor developer usage. Please note this feature is disabled by default when running the API locally.

0.6.3 Endpoints

In this section, the endpoints for the Smartmove API will be presented.

0.6.3.1 Properties

This endpoint provides the endpoints that allow the user to retrieve properties throughout Ireland and the UK. Users can filter by sale type, location and date. The table below shows the endpoint information for the Properties endpoint.

Method	**URL**	**Description**
GET	/properties/	Get all properties
GET	/properties/{id}	Get a property by ID.
GET	/properties/search/{term}	Search properties

0.6.3.2 Towns

This endpoint provides endpoints to retrieve sale statistics for towns. The table below shows the endpoint information for the Towns endpoint.

Method	**URL**	**Description**
GET	/towns/	Get all towns
GET	/towns/compare/	Compare sale statistics between two towns.
GET	/towns/{id}	Get a town by ID.

0.6.3.3 Countries

This endpoint provides endpoints to retrieve sale statistics for countries. The table below shows the endpoint information for the Countries endpoint.

Method	**URL**	**Description**
GET	/countries/	Get all countries
GET	/countries/{id}	Get a country by ID.

0.6.3.4 Counties

This endpoint provides endpoints to retrieve sale statistics for counties. The table below shows the endpoint information for the Counties endpoint.

Method	URL	Description
GET	/counties/	Get all counties
GET	/counties/compare/	Compare sale statistics between two counties.
GET	/counties/{id}	Get a county by ID.
GET	/counties/{name}/{year}	Get yearly sale statistics for a given county.

0.6.3.5 Charts

This endpoint allows the user to retrieve JSON data that can easily be consumed by the Google Charts API. The table below shows the endpoint information for the Charts endpoint.

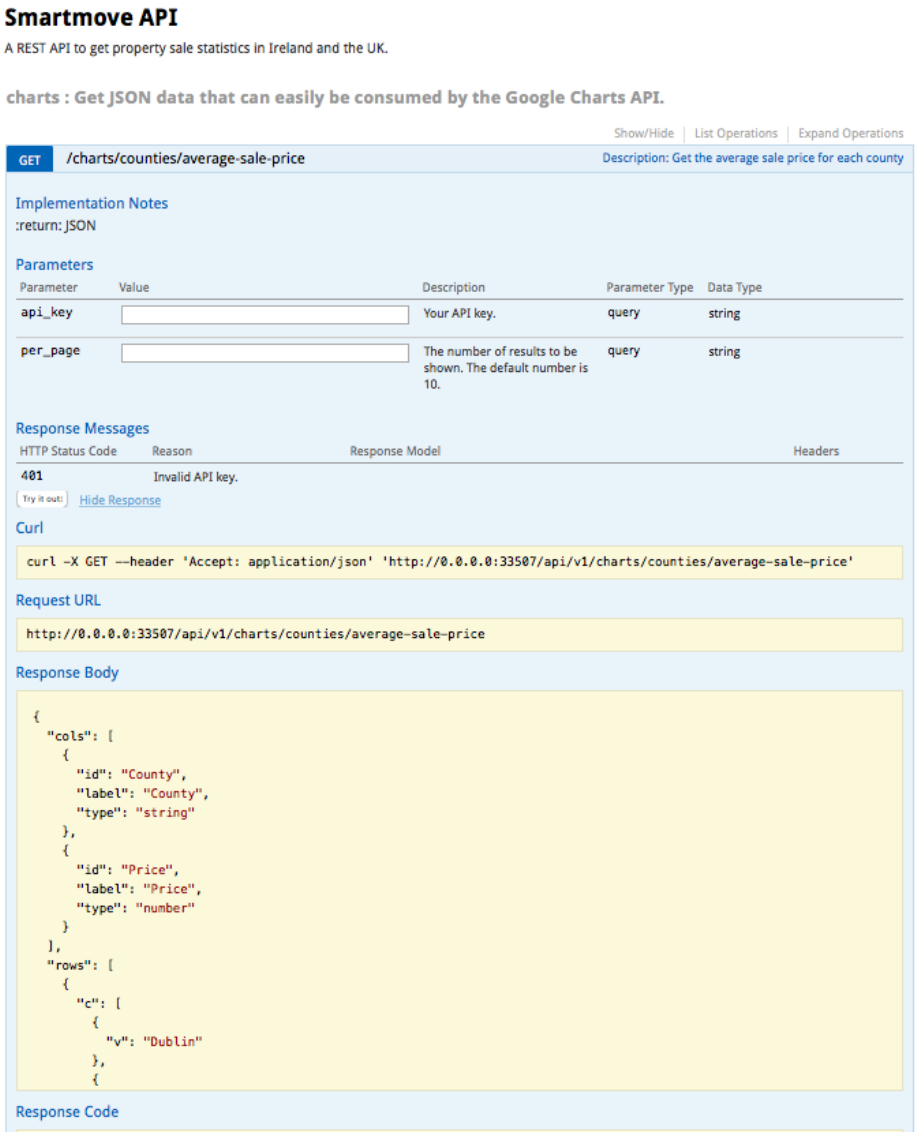
Method	URL	Description
GET	/charts/counties/average-sale-price	Get the average sale price for each county
GET	/charts/new-dwellings/average-sale-price	Get the average sale price of new dwellings between 2010-2016.
GET	/charts/new-dwellings/number-of-sales	Get the number of sales of new dwellings between 2010-2016.
GET	/charts/table/	Get the average sale price and total number of sales for each town.
GET	/charts/{name}	Get the average sale price for each year for a given county.

0.7 Graphical User Interface (GUI) Layout

0.7.1 REST API

The REST API is built using Python with the Flask-restplus extension which provides useful features such as automatically generating Swagger documentation. The Swagger user interface allows users to view and test the API. The screenshot below presents the Swagger initial screen.

Users can expand endpoints and enter query parameters. Below a screenshot is presented showing how a user can test the Charts endpoint.

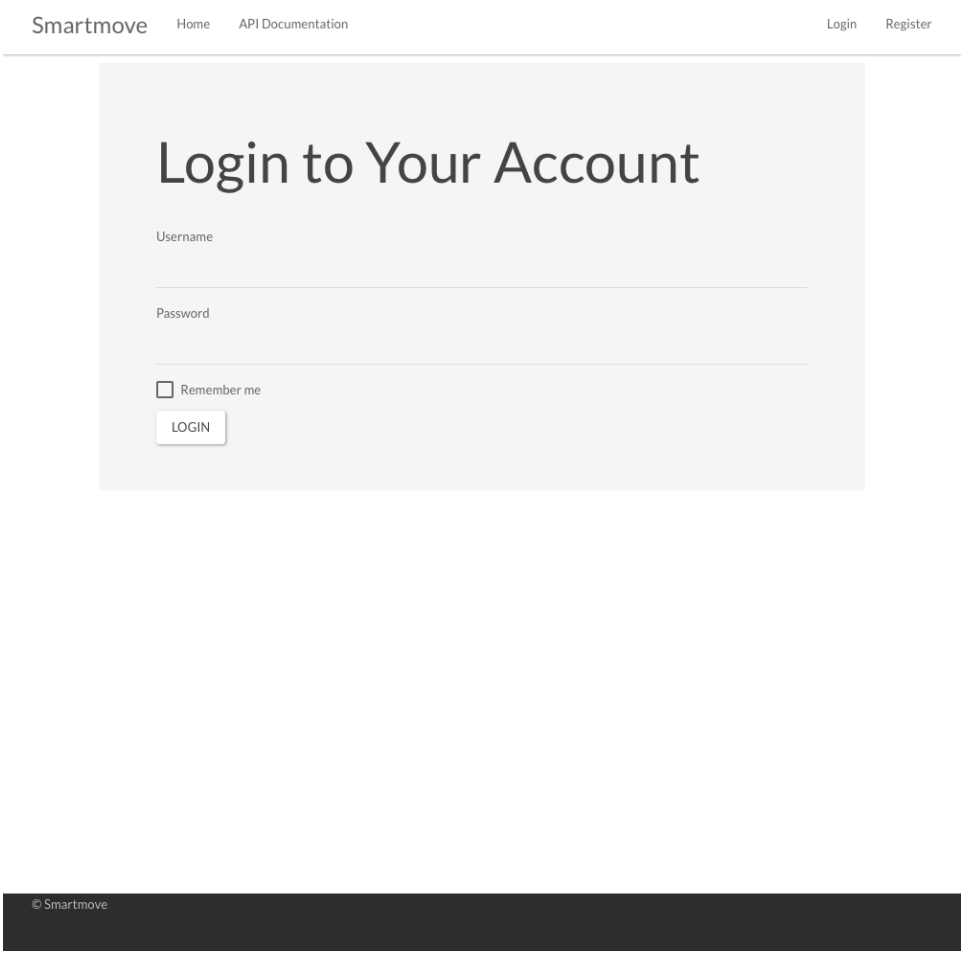


0.7.2 Homepage

The Smartmove homepage is intended to promote the Smartmove platform as well as allow users to register an application to retrieve an API key for the Smartmove API.

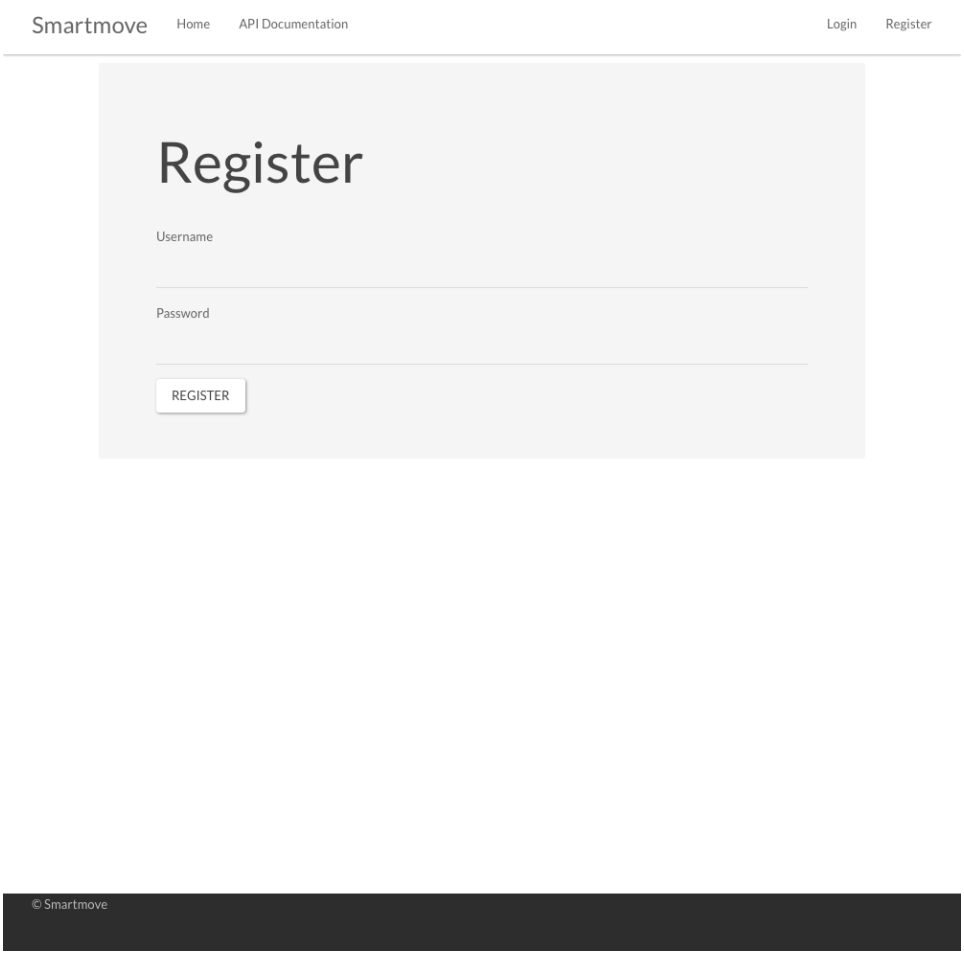
0.7.2.1 Login

This user interface allows the user to login at Smartmove.



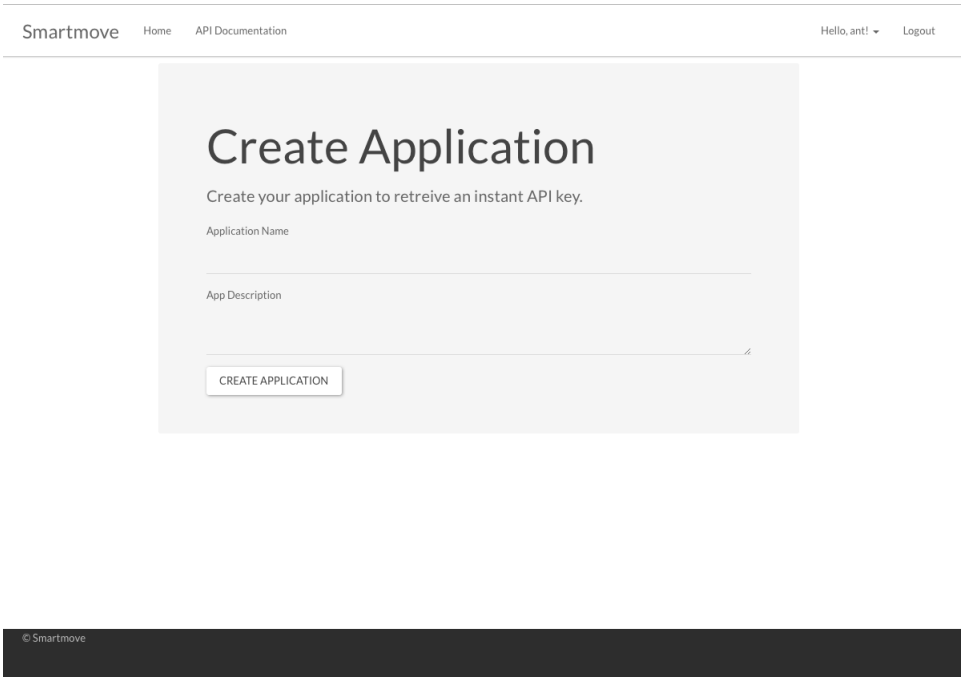
0.7.2.2 Register

This user interface allows the user to register an account at Smartmove.



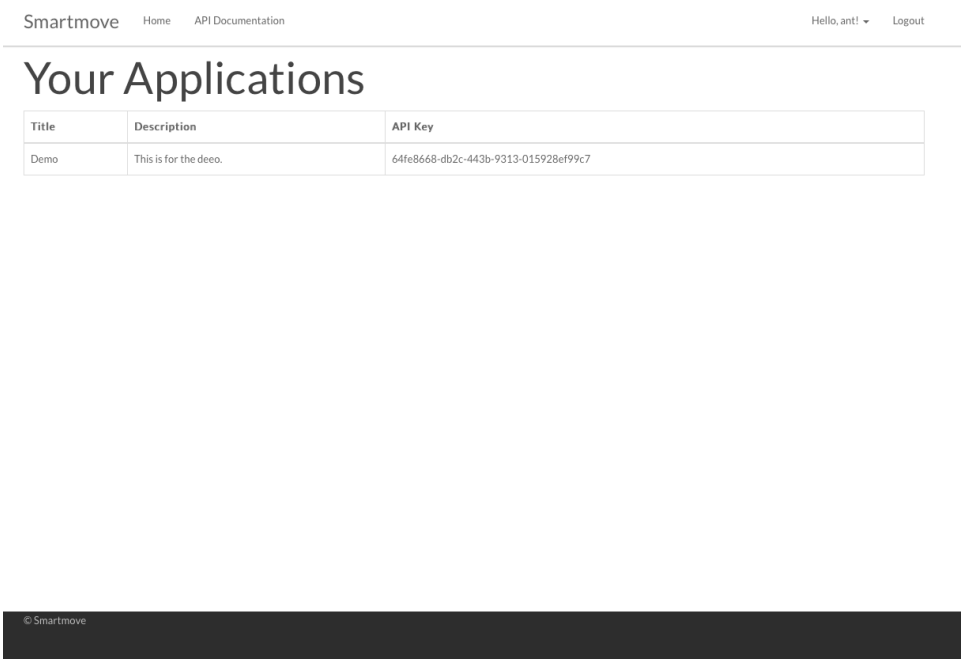
0.7.2.3 Create Application

This user interface allows the users to create an application.



0.7.2.4 List Applications

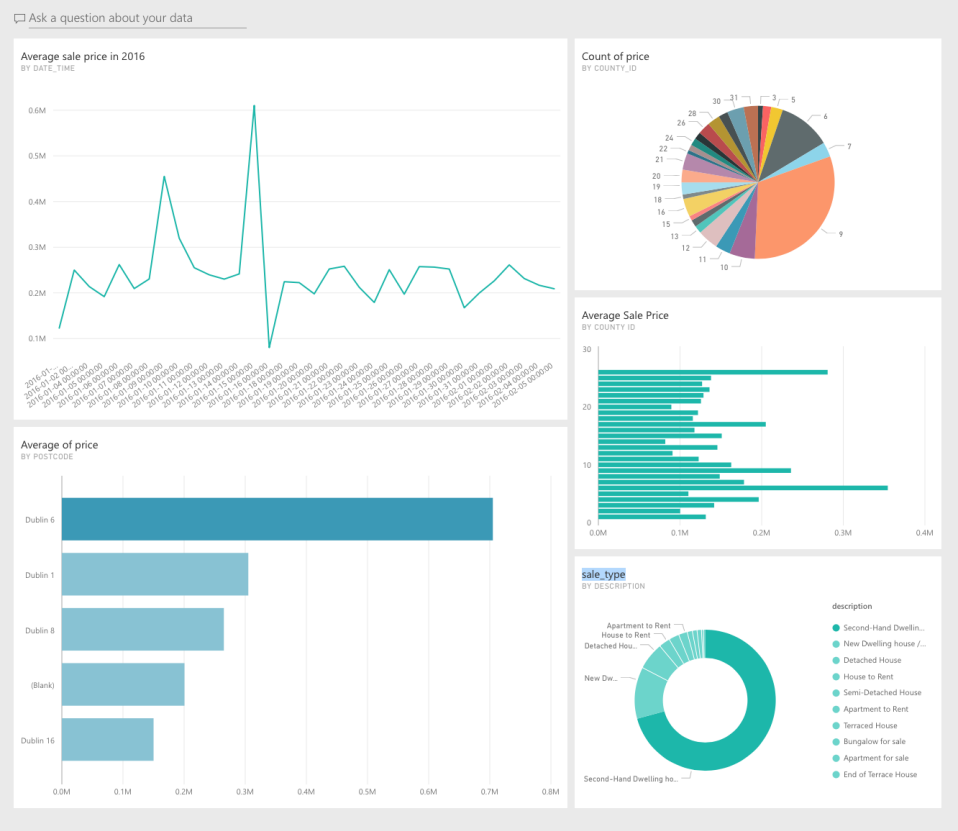
This user interface allows the users to view the applications they have created.

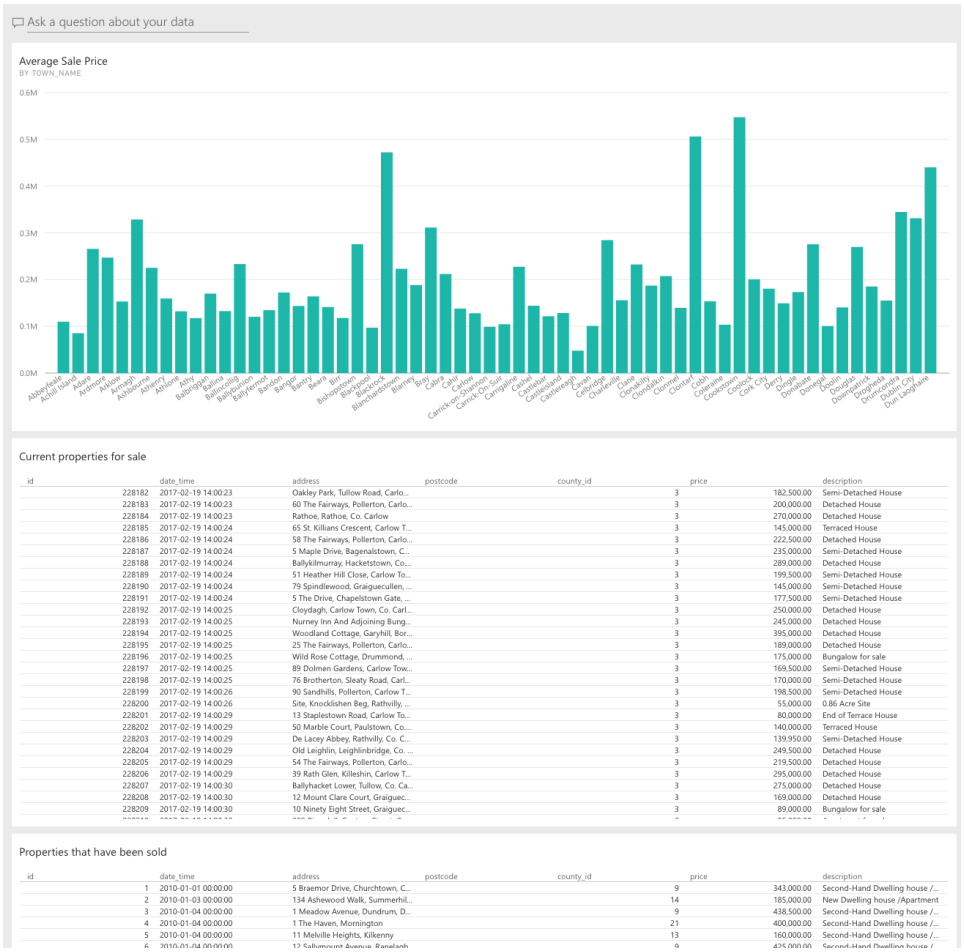


0.7.3 KPI Dashboard

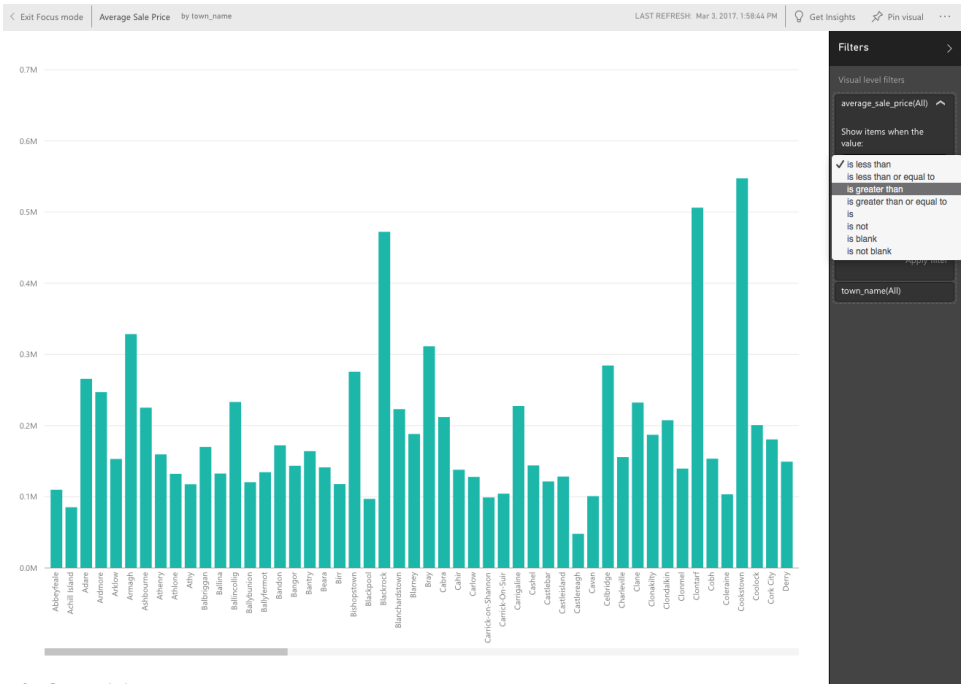
The KPI dashboard has been created using PowerBI. PowerBI is a tool released by Microsoft that allows users to update their dataset to the service and create interactive charts. It is a similar tool to Tableau. Users can also gain insights where the program presents interesting facts about the data. Below is a screen of the Smartmove KPI dashboard.

0.7.3.1 Dashboard User Interface





Users of the dashboard can focus on one of the charts and add filters such as filtering a sale price less than a particular value. A demonstration of the user interface is shown below.



0.8 Implementation

This section will discuss the implementation details of this project including the ETL process and the main features of the REST API.

0.8.1 ETL Process

Smartmove uses data from multiple sources. This data is firstly staged in the operational database. Then, data is transformed and loaded into the Smartmove data warehouse. Several scripts have been written using Python to perform the extraction of data. For example, Smartmove uses the daftlistings library created as part of this project to scrape data from Daft.ie and store the data in the database.

```
if __name__ == '__main__':
    with connection.cursor() as cursor:
        sql = "SELECT id, county_name, country_id " \
              "FROM counties where country_id = 1"
        cursor.execute(sql)
        results = cursor.fetchall()
        if results:
            for result in results:
                id = result['id']
                cn = result['county_name']
                ci = result['country_id']
```

```
            scrape(id, cn, ci, options['sale'])
        print('Done!')
```

The script above loops through each county in the Smartmove database and passes it to the scrape method. The scrape method takes four parameters: the county id, the county name, the country name and a dictionary of parameters to be passed to the get_listing function. For example, to retrieve current sale information, we would pass the following dictionary.

```
{
   'sale': {
        'listing_type': 'properties',
        'sale_type': 'sale',
        'sale_type_id': 2,
        'sale_agreed': False
    }
}
```

The scrape method then loops through each page of listings and stores the house information in the Smartmove database. The algorithm described is presented below.

```
def scrape(county_id, county, country_id, params):
    d = Daft()
    pages = True
    offset = 0
    while pages:
        listings = d.get_listings(
            county=county,
            area='',
            offset=offset,
            listing_type=params['listing_type'],
            sale_type=params['sale_type'],
            sale_agreed=params['sale_agreed']
        )
        if not listings:
            pages = False
        for listing in listings:
            address = listing.get_formalised_address()
            price = listing.get_price()
            price = calculate_price(country_id, price)
            if not price or not address:
                continue
            address = is_sale_agreed(address)
            address = address.lower().title().strip()
            if not exists(address):
                nbedrooms = listing.get_num_bedrooms()
                nbathrooms = listing.get_num_bathrooms()
                try:
                    nbedrooms = int(nbedrooms.split()[0])
                except:
                    pass
                try:
                    nbathrooms = int(nbathrooms.split()[0])
                except:
                    pass
                date_time = strftime("%Y-%m-%d %H:%M:%S", gmtime())
                description = listing.get_dwelling_type()
                insert([
                    date_time,
                    address,
                    county_id,
                    price,
                    description,
                    country_id,
                    params['sale_type_id'],
                    nbedrooms,
                    nbathrooms
                ])

        offset += 10
```

Once this data is loaded into the operational database, we can then aggregate the data and load it into the data warehouse. Several SQL procedures have been written to retrieve data from the operational database, aggregate it then store in the data warehouse. For example, the following SQL procedure loops through each county and computes the average sale price and number of sales from 2010 to 2017.

```
DELIMITER //

CREATE PROCEDURE county()
BEGIN
  DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE _id VARCHAR(40);
  DECLARE _year INT(11);
  DECLARE cur CURSOR FOR SELECT id FROM smartmove.counties;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done := TRUE;
  SET _year = 2010;
  OPEN cur;
  testLoop: LOOP
    FETCH cur INTO _id;
    IF done THEN
      LEAVE testLoop;
    END IF;
    WHILE _year <= 2017 DO
      CALL insert_year_stats(_id, _year);
      SET _year = _year + 1;
    END WHILE;
    SET _year = 2010;
  END LOOP testLoop;

  CLOSE cur;
END

//

DELIMITER //

CREATE PROCEDURE insert_year_stats(IN id VARCHAR(20), IN year
    INT(11))
BEGIN
  INSERT INTO smartmove_data_warehouse.fact_year (year, county_id,
      total_number_of_sales, average_sale_price)
  SELECT year(date_time),
         county_id,
         COUNT(price),
         AVG(price)
  FROM smartmove.properties
```

```
  WHERE sale_type = 1
  AND county_id = id
  AND year(date_time) = year
  GROUP BY year(date_time);
END

//

CALL county()
```

0.8.2 REST API

As discussed, the Smartmove API uses the Flask micro-framework. The Flask-restplus extension extends the Flask framework with additional functionality such as decorators that allow us to define the API endpoint, query parameters and error responses. For example, consider the following class which gets a property by ID.

```
@api.route('/<id>')
@api.param('id', 'The property identifier')
@api.param('api_key', 'Your API key.')
@api.response(404, 'Property not found')
@api.response(401, 'Invalid API key.')
class GetPropertyById(Resource):
    @api.doc('get_property_by_id')
    @api.marshal_with(property_model)
    def get(self, id):
        """
        Description: Get a property by ID.
        :return: JSON
        """
        if request.args.get('api_key') and
            validate_key(request.args.get('api_key')) or
            settings.ENV == 'TESTING':

            sql = "select * from smartmove.properties as p " \
                  "join smartmove.counties as c " \
                  "on p.county_id = c.id " \
                  "where p.id = %s"
            with conn.cursor() as cursor:
                cursor.execute(sql, id)
            data = cursor.fetchone()
            return data if data else api.abort(404)

        else:
            api.abort(401)
```

The @api.route decorator specifies the endpoints URL. The @api.param

decorator specifies the query parameters and @api.response adds HTTP response information if any error occurred. The Flask-restplus extension also allows us to automatically generate API documentation by using the @api.doc and @api.marshal_with decorators within the class.

When a user makes a request to the API, the user must pass their API key as a query parameter. When the system receives a request, it checks to see if the API exists before returning any data. If the API key does not exist it will return a 401 response.

Another requirement for the Smartmove REST API is rate limiting. Rate limiting is useful as it allows us to limit how many requests a user can make to the API. This is beneficial from a business and security perspective as it prevents users from abusing the API. Rate limiting is implemented easily using the Flask-limiter extension. With the extension, we can specify rate limits for the application. For example:

```
limiter = Limiter(
    app,
    key_func=get_remote_address,
    global_limits=["2000 per day", "100 per hour"]
)
```

The API includes many useful features such as the ability to retreive data that can easily be consumed by the Google Charts API. For example the following CURL command:

```
curl -X GET --header 'Accept: application/json'
    'http://0.0.0.0:33507/api/v1/charts/<county_name>'
```

Returns the JSON below that can be consumed by the Google Charts API to display a line chart of the average sale price for each year for a given county.

```
{
  "cols": [
    {
      "id": "Year",
      "label": "Year",
      "type": "number"
    },
    {
      "id": "Price",
      "label": "Price",
      "type": "number"
    }
  ],
```

```
  "rows": [
    {
      "c": [
        {
          "v": 2010
        },
        {
          "v": 333401
        }
      ]
    }
  ]
}
```

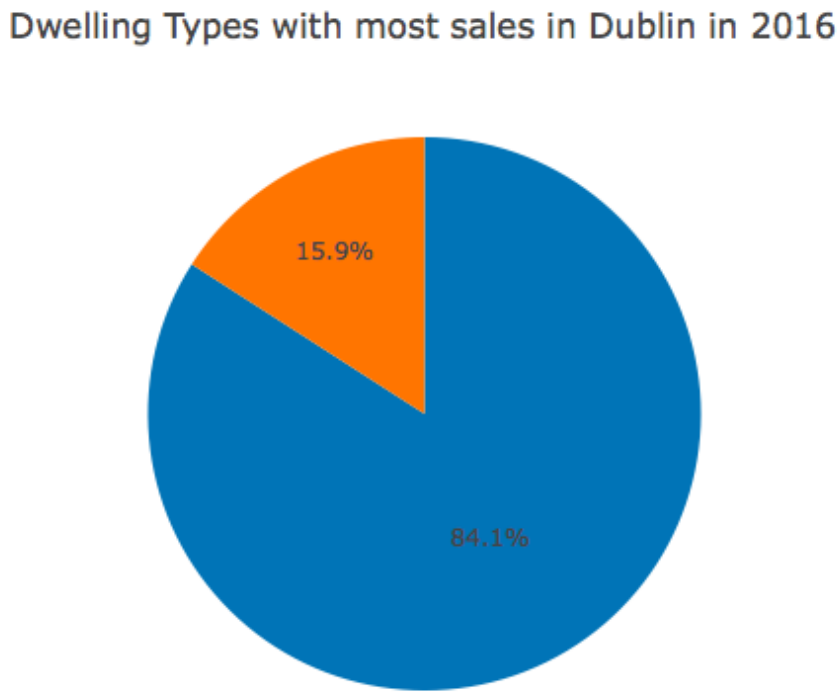
0.8.3 Data Analysis / Visualization

In this section, the data analysis and visualization implementation details will be briefly discussed. One of the main goals of this project is to provide information that people find valuable. The analysis was carried out using R and MySQL. Consider the following code snippet that finds which type of properties had the most sales in Dublin in 2016.

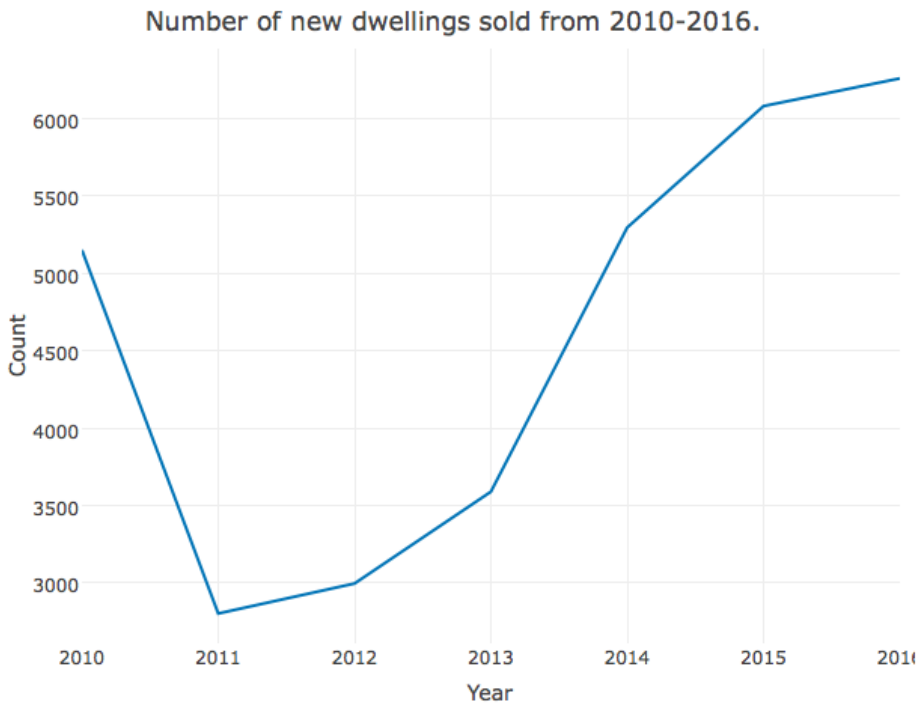
```
data <- get_data(
  'select distinct(description) as d, count(*) as count
  from properties
  where sale_type = 1
  and county_id = 9
  and year(date_time) = 2016
  group BY d'
)
summary(data)
p <- plot_ly(data,
             labels = ~ d,
             values = ~ count,
             type = 'pie') %>%
  layout(
    title = 'Dwelling Types with most sales in Dublin in 2016',
    xaxis = list(
      showgrid = FALSE,
      zeroline = FALSE,
      showticklabels = FALSE
    ),
    yaxis = list(
      showgrid = FALSE,
      zeroline = FALSE,
      showticklabels = FALSE
    )
  )
```

```
print(p)
```

The get_data function takes a SQL query as an argument, queries the database and returns the data. The data returned from the query is then passed into the plot_ly function. Plotly is a library that allows us to create charts using R. The above code outputs the following pie chart:



Another example of the analysis carried out is a chart displaying the total number of new dwellings sold from 2010-2016.



The chart above is generated using the Plotly library. The code snippet for this is presented below.

```
data <- get_data(
  'select year(date_time) as py, count(*) as count
  from properties
  where description = "New Dwelling house /Apartment"
  group by py'
)
head(data)
exclude <- data[0:7, ]
exclude
plot_ly(
  data = exclude,
  x = ~ py,
  y = ~ count,
  type = 'scatter' ,
  mode = 'lines'
) %>%
  layout(
    title = 'Number of new dwellings sold from 2010-2016.',
    xaxis = list(title = 'Year'),
    yaxis = list(title = 'Count')
  )
```

A final example to be presented is a heatmap of how the total number of

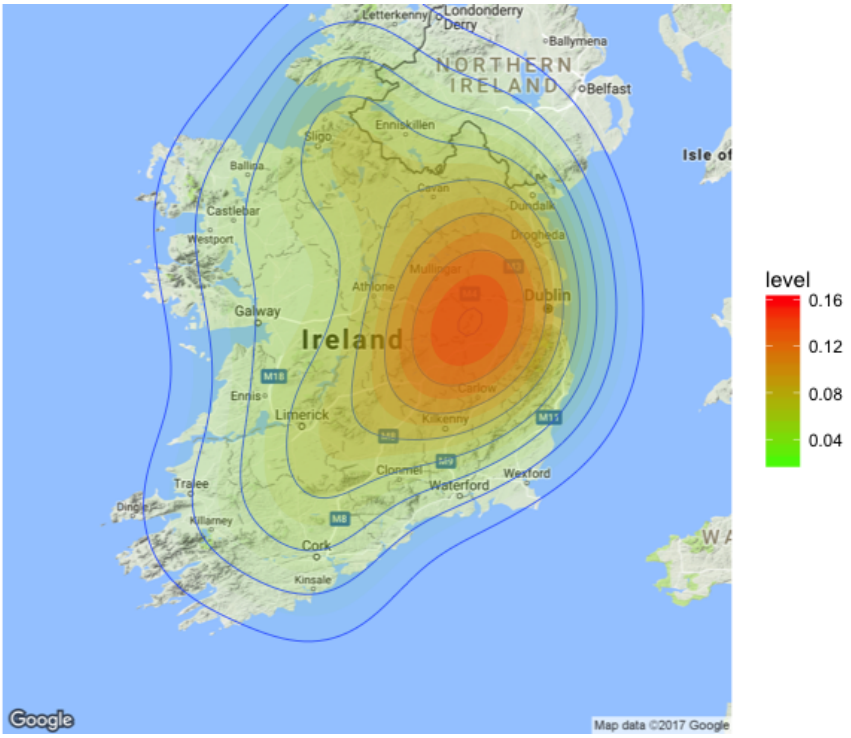
sales are distributed throughout Ireland.

```
data <-
  get_data(
    'select distinct c.county_name, c.latitude as lat, c.longitude
        as lon, count(p.id)
    from properties as p
    join counties as c
    on c.id = p.county_id
    where sale_type = 1
    and p.country_id = 1
    group by c.county_name, lat, lon'
  )

head(data)

# Code Reference:
    http://www.geo.ut.ee/aasa/LOOM02331/heatmap_in_R.html
map_location <-
  c(lon = -8, lat = 53)
m <- get_map(location = map_location, zoom = 7)
ggmap(m, extent = "device") + geom_density2d(data = data,
                                             aes(x = lon, y = lat), size
                                                 = 0.3) + stat_density2d(
                                               data = data,
                                               aes(
                                                 x = lon,
                                                 y = lat,
                                                 fill = ..level..,
                                                 alpha = ..level..
                                               ),
                                               size = 0.01,
                                               bins = 16,
                                               geom = "polygon"
                                             ) + scale_fill_gradient(low
                                                 = "green", high =
                                                 "red") +
  scale_alpha(range = c(0, 0.3), guide = FALSE)
```

The code above generates the heatmap presented below. It uses the ggmap library which enables developers to use the Google Maps API in R.



0.9 Testing

This section will briefly discuss the testing methodologies used in the Smartmove platform.

0.9.1 Blackbox Testing

The automatically generated Swagger documentation is ideal for testing the main endpoints. The blackbox approach was carried out on each of the API endpoints to ensure the API was sending the correct data.

0.9.2 Unit Testing

Unit tests have been written for each of the Smartmove software components. The Smartmove API includes unit tests as well as the Zoopla and Daftlistings library. This is made possible using the unittest library built into Python 2.7. Consider the following code which tests the properties endpoint.

```
class ApiTest(unittest.TestCase):
    base = 'http://0.0.0.0:33507/api/v1/'

    def call(self, method):
        req = requests.get(url=self.base + method)
        return req
```

```
class PropertyTest(ApiTest):
    def test_properties(self):
        self.assertTrue(self.call('properties/').status_code, 200)

    def test_property_by_id(self):
        r = self.call('properties/6').json()
        self.assertTrue(r['county_name'], 'Dublin')

    def test_property_search(self):
        r = self.call('properties/search/ballivor/')
        self.assertTrue(r.status_code, 200)
```

0.10 Installation / Usage Manual

The following technologies must be installed on the user's system:

- Python 2.7
- Virtualenv
- Git
- MySQL

0.10.1 Load Scripts

This section will discuss the installation instructions for the database load scripts.

1. First you'll need to create a virtualenv and install the requirements.

```
cd smartmove-load-scripts
# Create a virtual environment
virtualenv env
source env/bin/activate
# Install the requirements
pip install -r requirements.txt
```

2. Create the databases. Run the database creation scripts in /queries/schema to create the main Smartmove database and the data warehouse.
3. Update the configuration file with your database settings in config.py.
4. Execute the PPR script in **ppr/ppr.py** to load property price data into the smartmove database.

5. Run the procedures found in **/queries/procedures** to compute the statistics for the data warehouse.

0.10.1.1 Retrieving Real-time Data

To load current sale information into the Smartmove database run the Daft and Zoopla load scripts. The load scripts for Daft and Zoopla are found in **daft/** and **zoopla/**. If you are retrieving data from Zoopla, you will need to update the config file with your Zoopla API key.

0.10.1.2 Property Price Register Notifier

The load scripts includes a script to notify you when the Property Price Register update their website. If it has the script will send an email to notify you. Navigate to **ppr/notifier.py** and update the script with your email, password and the recipient email address. The script checks whether the site has been updated once a day. You will need to allow less secure applications in your Gmail account.

0.10.2 REST API

In this section, the installation details for the API is presented.

```
cd smartmove-api

# Create a new virtual environment
virtualenv env
source env/bin/activate

# Install the requirements
pip install -r requirements.txt

# Run the server
python app.py
```

0.10.3 Smartmove Website

The Smartmove website is intended to promote the platform and serve as a portal for third party developers to register their application to retrieve an API key. Below the installation process is presented.

```
cd smartmove

# Create a new virtual environment
virtualenv env
```

```
source env/bin/activate

# Install the requirements
pip install -r requirements.txt

# Run the server
python run.py
```

0.10.4 Daftlistings

Daftlistings is a web scraper that enables programmatic interaction with daft.ie. The library has been tested on Python 2.7 and Python 3.5.2. In this section, usage examples and the libraries usage information will be presented.

Installation

```
pip install daftlistings
```

Examples

Get the current properties for rent in Dublin that are between 1000 and 1500 per month.

```
from daftlistings import Daft

d = Daft()

listings = d.get_listings(
    county='Dublin City',
    area='Dublin 15',
    listing_type='apartments',
    min_price=1000,
    max_price=1500,
    sale_type='rent'
)

for listing in listings:
    print(listing.get_formalised_address())
    print(listing.get_daft_link())
```

Get the current sale agreed prices for properties in Dublin.

```
listings = d.get_listings(
    county='Dublin City',
    area='Dublin 15',
    listing_type='properties',
```

```
    sale_agreed=True,
    min_price=200000,
    max_price=250000
)

for listing in listings:
    print(listing.get_formalised_address())
    print(listing.get_daft_link())
```

The following script loops through each page of property listings for property sales in Dublin 15.

```
offset = 0
pages = True

while pages:

    listings = d.get_listings(
        county='Dublin City',
        area='Dublin 15',
        offset=offset,
        listing_type='properties'
    )

    if not listings:
        pages = False

    for listing in listings:
        print(listing.get_agent_url())
        print(listing.get_price())
        print(listing.get_formalised_address())
        print(listing.get_daft_link())
        print(' ')


    offset += 10
```

Methods

get_listings()

The get_listings method accepts the following parameters.

- max_beds: The maximum number of beds.
- min_beds: The minimum number of beds.
- max_price: The maximum value of the listing
- min_price: The minimum value of the listing

- county: The county to get listings for.
- area: The area in the county to get listings for. Optional.
- offset: The page number.
- listing_type: The listings you'd like to scrape i.e houses, properties, auction or apartments.
- sale_agreed: If set to True, we'll scrape listings that are sale agreed.
- sale_type: Retrieve listings of a certain sale type. Can be set to 'sale' or 'rent'.
- sort_by: Sorts the listing. Can be set to 'date', 'distance', 'price' or 'upcoming_viewing'.
- sort_order: 'd' for descending, 'a' for ascending.

get_address_line_1()
This method returns line 1 of the listing address.

get_address_line_2()
This method returns line 2 of the listing address.

get_town()
This method returns the town.

get_county()
This method returns the county.

get_formalised_address()
This method returns the full address.

get_listing_image()
This method returns the URL of the listing image.

get_agent()
This method returns the agent name.

get_agent_url()
This method returns the agent URL.

get_daft_link()
This method returns the URL of the listing.

get_dwelling_type()

This method returns the dwelling type.

get_posted_since()
This method returns the date the listing was posted.

get_num_bedrooms()
This method returns the number of bedrooms.

get_num_bathrooms()
This method returns the number of bathrooms.

get_price()
This method returns the price.

0.10.5 Zoopla

zoopla is a python wrapper for the Zoopla API. Below you will find usage examples for the library.

Examples

Retrieve property listings for a given area.

```
from zoopla import Zoopla
zoopla = Zoopla(api_key='your_api_key', debug=True,
    wait_on_rate_limit=True)

search = zoopla.search_property_listings(params={
    'maximum_beds': 2,
    'page_size': 100,
    'listing_status': 'sale',
    'area': 'Blackley, Greater Manchester'
})

for result in search:
    print result.price
    print result.description
    print result.image_url
```

Retrieve a list of house price estimates for the requested area.

```
zed_indices = zoopla.area_zed_indices({
    'area': 'Blackley, Greater Manchester',
    'output_type': 'area',
    'area_type': 'streets',
    'order': 'ascending',
```

```
    'page_number': 1,
    'page_size': 10
})

print zed_indices.town
print zed_indices.results_url
```

Generate a graph of values for an outcode over the previous 3 months and return the URL to the generated image.

```
area_graphs = zoopla.area_value_graphs('SW11')

print area_graphs.average_values_graph_url
print area_graphs.value_trend_graph_url
```

Retrieve the average sale price for houses in a particular area.

```
average = zoopla.get_average_area_sold_price('SW11')
print average.average_sold_price_7year
print average.average_sold_price_5year
```

0.11 Future Work

Future work would include incorporating other metrics to improve the predictive model as well as including pricing information from other countries. Smartmove uses data from Ireland and the UK.

0.12 Conclusion

In this paper, the system architecture and components of the Smartmove platform has been discussed including the Data Warehouse and the REST API. The opportunities of this project include:

- A platform that can fill a certain niche in the market and fill a business and consumer need.
- A REST API that allows developers to use Smartmove in their own applications regardless of the programming language or hardware device.
- A data warehouse with a scalable architecture to allow for further data sources to be incorporated.

Learning curve:
Rather than focusing on one research area, Smartmove implements a Data

Warehouse, REST API and a data analysis using R. Thus, researching several research areas have been difficult. If I was to complete this project again, I would have focused on one research area instead of focusing on several research areas.

Blockers encountered:
The main blockers encountered was trying to build a good regression model. There was little correlation found in the data set. Incorporating other data points including the number of businesses for a given area improved the accuracy of the model. Another blocker encountered was hosting. Heroku offer a free tier but you are limited to the amount of data you can store. This was the limitation as Smartmove processes large amounts of data.

0.13 Bibliography

Property Price Register. (2017). *Residential Property Price Register - Home Page.* [online] Available at: https://www.propertypriceregister.ie/ [Accessed 6 May 2017].

W3. (2017). Web Services Glossary. [online] Available at: https://www.w3.org/TR/2004/ [Accessed 7 May 2017].

Developer.zoopla.com. (2017). Zoopla Property API - Welcome to the Zoopla Developer Network. [online] Available at: http://developer.zoopla.com [Accessed 7 May 2017].

Level, I., Tutorial, A., Dates, 2., Tests, I., Us, C., Policy, P., Use, T. and Us, A. (2017). What is Iterative model- advantages, disadvantages and when to use it?.[online] Available at: http://istqbexamcertification.com/ [Accessed 7 May 2017].

Iterative Lifecycle. (2017).[image] Available at: https://praxisframework.org/ [Accessed 7 May 2017].

0.14 Appendix

0.14.1 Project Proposal

Objectives

The objective of this project is to build a real time dashboard that will display statistics on property prices in Ireland and the UK. The goal is to provide intelligence to businesses so they can better target areas. Data

will be extracted from various sources and will be loaded into the IncomeStack data warehouse where we can perform statistical computations for the dashboard. The project will feature an Application Programming Interface (API) so third party developers can utilize the data in their own applications.

Background

One way businesses can determine household income in a given area is to look at property prices. IncomeStack aims to provide intelligence to businesses so they can better target areas. The Central Statistics Office (CSO) does provide property sale information but it is not easy to use and is not as granular in the results it gives back. For example, CSO provide data for County Dublin but not cities within such as Blackrock, Blanchardstown, Tallaght, etc. IncomeStack will try solve this problem by displaying statistics in an easy to use format as well as aggregated property price information at a granular level. The project will feature a KPI dashboard, REST API, tools for extracting and loading data into the database and a data warehouse. The technical details of these components will be detailed in section three of this proposal. I will briefly discuss some use cases for a project such as this. One use case for a project such as this would be Pay-per-click (PPC) advertising. Ive managed the Google AdWords campaigns for my families company for two years. When we set up the AdWords campaign we were primarily targeting all areas within Dublin. I performed some calculations on the property price data I attained, grouping each town within Dublin by average sale price. I changed our campaign to only target areas with the highest sale average. The company seen an increase in 12% to 16.8% in conversion rates in just under six months. Another use case is to generate custom reports for a given area. In my previous internship, my employer was aware of the project I was working on and asked could I retrieve property price information for cities within Manchester. Using the Zoopla API and the Python wrapper I wrote for it, I was able to compute the statistics needed and export as .CSV. Monetization is an area being considered and details are outlined in section 8 of this proposal.

Technical Approach

Data Sources

Zoopla API wrapper

Zoopla is a leading resource for property sale prices in the United Kingdom. Zoopla provide a public API that allow developers to create applications using hyper local data on 27m homes, over 1m sale and rental listings, and 15 years of sold price data. (Developer.zoopla.com, 2016) I have developed

a wrapper around the Zoopla API which allows us to easily call the various API methods using Python. This wrapper is open source and is available on Github and the Python Package Index (PyPi). I aim to model some of IncomeStack REST API off of Zooplas API methods. For example, Zoopla have an API method called Average Area Sold Price which returns the average sale price for properties in a given area.

Daft.ie Web Scraper

Daft.ie is one of Irelands largest property websites. Unfortunately, Daft do not provide a public API so I have developed a web scraper that allows to scrape real time property price information from the Daft.ie website. The scraper has been written in Python and makes use of the Beautiful Soup library. Beautiful Soup is a Python library used for scraping data from webpages. The Daft.ie web scraper is open source and is available on Github and The Python Package Index (PyPi). This library has already gained developer traction.

Property Price Register

The Residential Property Price Register is produced by the Property Services Regulatory Authority (PSRA) pursuant to section 86 of the Property Services (Regulation) Act 2011. (Property Price Register, 2016) The data provides the date of sale, price and description of properties sold between 2010 and 2016. The Property Price Register provides downloadable access to this data in .csv format. Using Python, we can load each .csv file into the IncomeStack database for further data analysis.

KPI Dashboard

The KPI dashboard is the face of this project. It is where the business intelligence will be consumed by end users. The dashboard will use the JavaScript Google Charts library and consume data from the JSON REST API currently under development.

JSON RESTful API

IncomeStack aims to provide third party developers with a JSON API to utilize the data in their own applications. I propose to develop a REST API so that the data could be used in other applications and on hardware devices such as mobile and tablet devices. The REST API will be developed in Django.

Data Warehouse

Data will be extracted from multiple sources and will be sent to the data warehouse. I consider the data warehouse to be one of the most important components of this project. I aim to build a data warehouse that is scalable and future proof for any further data sources or models we wish to incorporate.

Special resources required

No special resources are required for this project.

Project Plan

Please find the project plan attached.

Technical Details

Languages: Python, JavaScript, MySQL
Libraries: Google Charts, Google Visualization API, Django, Bootstrap

Evaluation

The components of this project will be tested using the Unit Testing library built into Python. I have wrote test cases for the Zoopla API and Daft.ie web scraper. I hope to gain feedback from others and will be distributing surveys and user testing. The dashboard will feature internal analytics using the Mixpanel library so we can see how users use the system and ways in which to increase its usability.

Monetization

Monetization is still an area being actively considered at this point. The goal of IncomeStack is to provide businesses with intelligence that they find is valuable. One way in which monetization could work is a subscription model where users pay a monthly fee of 10 to access the dashboard as well as providing access to the private API.

0.14.2 Monthly Reflective Journals

0.14.2.1 September

My Achievements
This month, I attended the project pitch where I presented my fourth year project idea. Thankfully, the judges accepted my proposal. I am pleased

that the research I have done and the components I have built over the summer months has value and can be used. I can now dive deeper into further research and development of my project.

My Reflection
I feel the project pitch went well. Initially, I was nervous to some extent but once we started talking back and forth about the project idea I felt more confident and at ease. The judges suggested important areas in which to focus on such as the importance of a good data warehouse implementation, other data sources to explore and the user friendliness of the dashboard. I am really grateful of the advice the judges gave me. I plan to research and implement their suggestions over the coming months.

Intended Changes
This month I will continue to focus on researching data warehouse design. I have built a simple star schema for the data I have already collected but still have a lot to learn and implement. I plan to collect more data such as population and crime data and populate the data warehouse with that information. I plan to make a start on the initial dashboard user interface.

0.14.2.2 October

My Achievements
This month, I have made progress with the properties database schema. I have decided to collect data by sale type: sold, for sale and sale agreed. I think this this approach will allow me to categorise the data better. Before I was just collecting property prices without including a sale type. This is a problem because how would I know which sale type the property price refers to. I have continued my research into data warehouse design. I watched one course on data warehouse design on Pluralsight which gave me an understanding of warehousing basics. I have started the design of my data warehouse. There is some questions I still need answered so I plan to continue research and development over the following month. I have decided that I will use Django for the API and dashboard implementation. Django is a web application framework written in Python. This framework will suit me well as most of the tools I have built are written in Python. I have researched the JavaScript charts libraries and have decided to use the Google Charts API.

My Reflection
This month I have decided on how I plan to implement some of the components of my fourth year project and finalised on platforms and libraries I intend to use. There is still some aspects in regard to data warehouse I am unsure of so I look forward to speaking to my project supervisor to get

feedback.

Intended Changes
This month I will continue to focus on researching data warehouse design. I have built a simple star schema for the data I have already collected but still have a lot to learn and implement. I plan to collect more data such as population and crime data and populate the data warehouse with that information. I plan to make a start on the initial dashboard user interface.

0.14.2.3 November

My Achievements

This month, I started to work on the Application Programming Interface (API). I changed from my initial plan to develop the API in Django to Flask. Flask is a Python micro-framework that allows developers to add components as they are needed. It isnt as heavy as Django as there isnt as much needed to learn. Building the API in Flask allows me to develop quickly and in time for the prototype presentation on the 19th of December. One notable change this month is that I changed the name of the platform from Incomestack to Smartmove. As I have started developing my project, I have been able to think of new use cases for the platform. Initially, Smartmove was a platform for businesses to gain intelligence into the property market. Now, this platform could also appeal to first-time buyers and curious consumers who wish to gain insight into the property market. With the API under development, this platform also appeals to third party developers. I have continued my research into Data Warehouse design and have developed a prototype data warehouse built on the snowflake model. I have fact tables for county statistics, country statistics and overall sale statistics. Ive set up my load scripts to scrape data from the multiple sources on weekly intervals.
My Reflection

Im happy that my idea has evolved over time and is becoming a platform that appeals to not just businesses but developers and consumers wishing to gain insight into the property market. Switching from Django to Flask for the API was a good decision as it allows me to develop the API faster in preparation for the prototype presentation. One concern I have now is that as my dataset is becoming larger I am considering database hosting providers. Ive looked at Azure and Amazon AWS

Intended Changes

Next month, I intend to research further statistics I can use for the property data obtained. I plan to implement forecasting which I have learned about

in my Business Data Analysis course. I will continue to work on the data warehouse research and development, and continue work on the API in time for the prototype presentation.

0.14.2.4 December

My Achievements

I attended the midpoint presentation which went well. The judges advised me on features to research and implement. My Reflection I felt the presentation went well and I was really grateful of the helpful advice and the positive remarks the judges gave me. The judges pointed out things I was missing in the midpoint technical report which I will keep in mind for the final report. They gave me helpful suggestions on features to research and implement which I intend to do in the upcoming months.

Intended Changes

I will now focus on researching and implementing the features the judges advised me to. I am going to research forecasting algorithms so we can predict property prices in a given area and research market trends on topics such as what factors influence property prices. I intend to complete the REST API, including API methods to generate charts based on given query parameters and start working on the graphical user interface for the KPI dashboard. Next semester, I will be studying Advanced Business Data Analysis and Data and Web Mining. I intend to apply knowledge learned in these modules for my Software Project.

0.14.2.5 January

My Achievements

This month I continued to work on the Smartmove API. I made really good advancements with the API and work is almost complete. The previous version of the API worked but I wasnt happy with the structure of it. Everything was in one file app.py. I researched folder structure for large Flask applications and found some good documentation on developing scalable Flask applications. I found an extension for Flask called Flask-restplus which is a framework for developing REST APIs. The great thing about flask-restplus is that you can add decorators to your classes that will document your API and the API documentation is shown when you browse to your root API url. Another great achievement this month is that I got the API hosted on Heroku. Its not 100% complete as I had to take some code out for it to work but Im slowly adding the features back in. Heroku offer

free MySQL databases but they are only 5mb which is too small for what I need so I am using Microsoft Azure which offers 20mb databases for free which should suffice for now. I have started the groundwork for the KPI dashboard. Right now the backend is written using Laravel. I choose this PHP framework because it is the framework I am most experienced with and I also wanted to show the benefit of developing REST APIs is that it doesnt matter what technology the API is written in, it is compatible across programming languages and frameworks. However I do think Laravel is overkill for the dashboard and I am currently looking at other solutions.

Intended Changes

- Complete API.
- Start working on the KPI dashboard.
- Apply some knowledge learned in Business Data Analysis and Data Web Mining to my project.

0.14.2.6 February

My Achievements

This month I focused worked on fixing some issues with the smartmove rest API. I have fixed most of them but I am still having issues on the Heroku live app. I have decided to not build a dashboard from scratch but instead use PowerBI for data analysis and visualisation. Building the dashboard is yet another software component and if I spend the remainder of the semester doing that I won't have much time to focus on the analysis. PowerBI is an amazing tool released by Microsoft. It allows you to upload the datasets and then create visuals. It also has a gain insights feature where it will analyze the data and present facts about the data that I would not have considered beforehand. The insights it presented, I was able to incorporate into the Smartmove API. As well as that I focused on applying some knowledge learned in the Business Data Analysis module into an in-depth R analysis I am focusing on. I have basic descriptive and inferential statistics, regression, testing the dataset for normality and removing outliers.

My Reflection

I feel I have made the right decision focusing on the data analysis instead of working on more software development components of my project.This has allowed me to gain insights and gather interesting facts about the data instead of focusing on the software development process.

Intended Changes

I intend to focus on more data analysis this month. I plan to incorporate knowledge learned in the Data and Web Mining module. Right now I'm just doing simple multiple linear regression but I intend to use some machine learning techniques to gain better sales predictions for properties. I also intend to prepare for the project showcase. I am currently working on the poster and thinking about the best way to demonstrate the project to interested attendees at the showcase.

0.14.2.7 March

My Achievements

Wow, time flies! This is my last reflective journal! This month, I started to research building a predictive model to predict house prices. I applied knowledge learned in my Data and Web Mining module and looked at some advanced regression models built into Scikit-Learn including Random Forest Regression and Gradient Boosting Regression. Scikit-Learn is a Python library for Machine Learning and includes classification and regression algorithms that are useful in building predictive models. This month, I also completed the task of designing the poster for the showcase.

My Reflection

I carried out some pre-processing on the data to make the dataset more suitable for the task of building the predictive model. Unfortunately, the datasets features produce a very low accuracy score using the R2 score function. I could not find high correlations between the variables In my dataset. The data from the property market register does not include needed features so I will need to research other factors that contribute to predicting house prices.

Intended Changes

Next month, I intend to research new features to build a better predictive model. I used the Yelp API to find the total number of businesses operating in a given area. This feature has the highest correlation with the sale price. I will need to look at other features in building a predictive model.