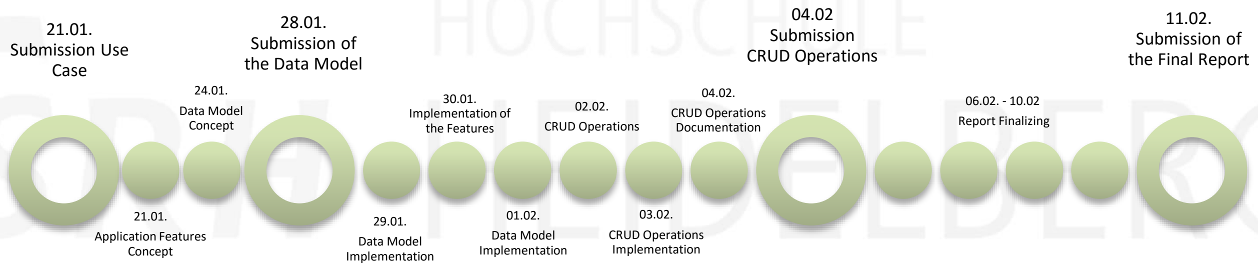
6 The Redis Database

The whole chapter and its content is created and provided by the Redis subgroup, including Mukhtar Hussain, Alpana Chaphalkar, Ankit Dixit and Jaydeep Galani.

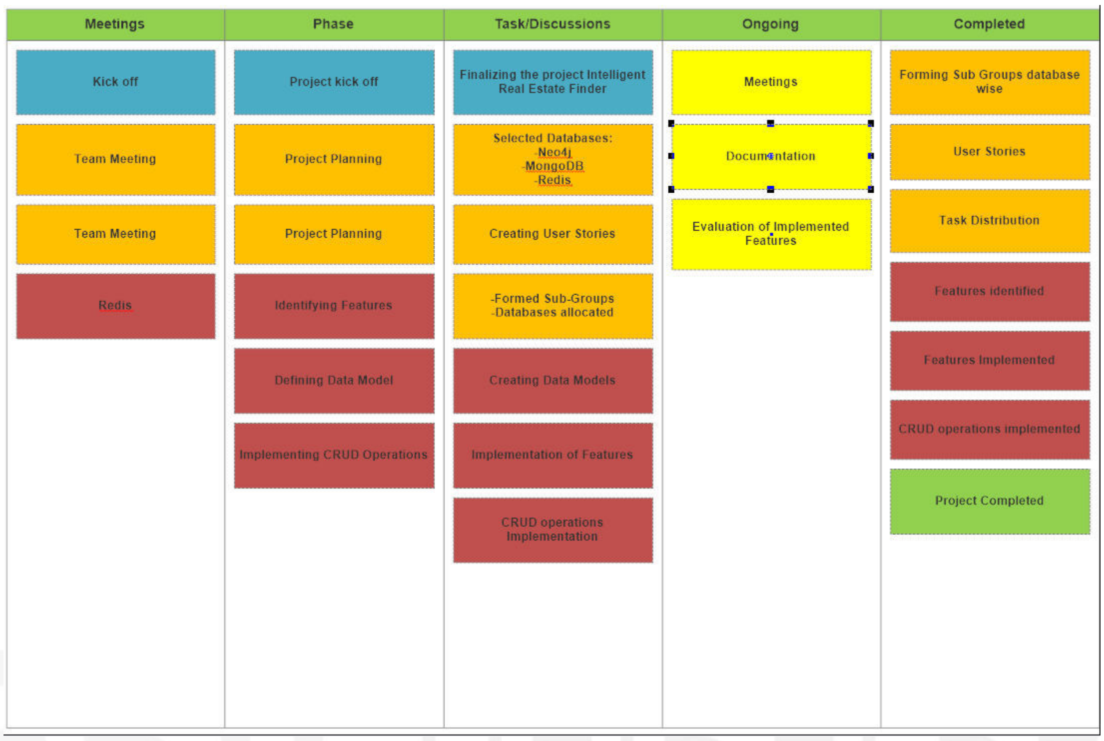
6.1 Redis - Project Management Aspects

This chapters gives an overview of work progress done by the Redis subgroup. During the timespan of 21st January to 10th February the Redis team had overall 11 meetings. Meetings held from 21st January to 24th January focused on the features to be implemented and designing data model. Meetings held from 29th January to 4th January were purely focused on implementation of the features and performing CRUD operations. All the meeting after that were dedicated to finalizing and documenting the project work.


21.01. Submission Use Case
21.01. Application Features Concept
24.01. Data Model Concept
28.01. Submission of the Data Model
29.01. Data Model Implementation
30.01. Implementation of the Features
01.02. Data Model Implementation
02.02. CRUD Operations
03.02. CRUD Operations Implementation
04.02. CRUD Operations Documentation
04.02 Submission CRUD Operations
06.02. - 10.02 Report Finalizing
11.02. Submission of the Final Report

Graphic 6.1-1: Timeline within the Redis database

The above mentioned timeline shows a pictorial view of the tasks completed by the Redis subgroup. With the help of proper planning and allocation of tasks within the Redis team, all the submission milestones related to data model, the CRUD operations were hence successfully achieved.



Meetings	Phase	Task/Discussions	Ongoing	Completed
Kick off	Project kick off	Finalizing the project Intelligent Real Estate Finder	Meetings	Forming Sub Groups database wise
Team Meeting	Project Planning	Selected Databases: -Neo4j -MongoDB -Redis	Documentation	User Stories
Team Meeting	Project Planning	Creating User Stories	Evaluation of Implemented Features	Task Distribution
Redis	Identifying Features	-Formed Sub-Groups -Databases allocated		Features identified
	Defining Data Model	Creating Data Models		Features Implemented
	Implementing CRUD Operations	Implementation of Features		CRUD operations implemented
		CRUD operations Implementation		Project Completed

Graphic 6.1-2: KanBan Board Redis

To keep track of the tasks allocation and task completion status within the Redis subgroup, a KanBan board was also prepared for better understanding about the progress of the project. Further detailed information about the meetings of the Redis subgroup can be found in the attached meeting protocols (attachments section XXXIX-XLL).

6.2 Redis - Introduction

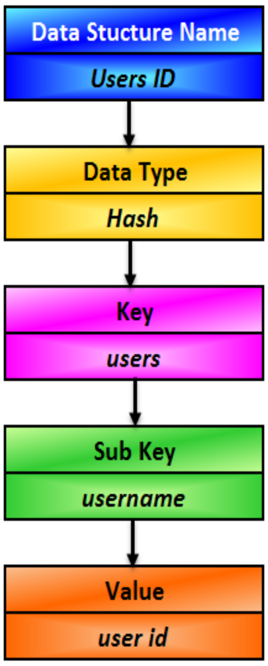
Redis is the a Key-value memory database: when Redis Server runs first time, it loads data from the disk into the memory. Redis is in-memory data store. All operations within Redis are in-memory operations. For data persistence, Redis periodically save the data to the disk asynchronously. The characteristics of pure memory operation make it very good performance, it can handle more than 100,000 read or write operation per second. [Jing H, Haihong E & Guan Le, 2011]. Redis supports advanced data structures, though not to the degree that a document-oriented database would. It supports set-based query operations but not with the granularity or type support you'd find in a relational database. [Eric R, Jim W, 2012].Data structures like strings, lists, sets, sorted sets, hashes are supported by redis database. As Redis is a nosql database, It supports two key principles of the CAP theorem. CAP theorem states that a distributed system cannot meet the three district needs simultaneously, but can only meet two of them. Such a database system stores data in the distributed nodes, and also ensure the consistency of these data, but not good enough support for the availability [Jing H, Haihong E & Guan Le, 2011]. Hence redis database is concerned about consistency and partition tolerance(CP). Riak is a key value store database where data is distributed over multiple servers and the reason for this distribution of data because it is designed to store large amount of data. Although riak provides features like bulk data storage and fault tolerance, riak does not provide faster access, whereas redis is a key value store database specifically designed for ultra-fast access.

6.3 Redis - Data model

This chapter gives a detailed information about the data model in REDIS database for the project - Intelligent Real Estate Finder. The proposed data structure is designed by keeping in mind all the use cases that were submitted before. In this document, each and every feature is explained by defining its purpose and illustrated by providing pictorial view of data types which indicates its key-value relationships.Proposed data model has been designed in a way that will help in exploring functionalities of key-value store database like increasing performance and enhancing the user experience with the intelligent features.

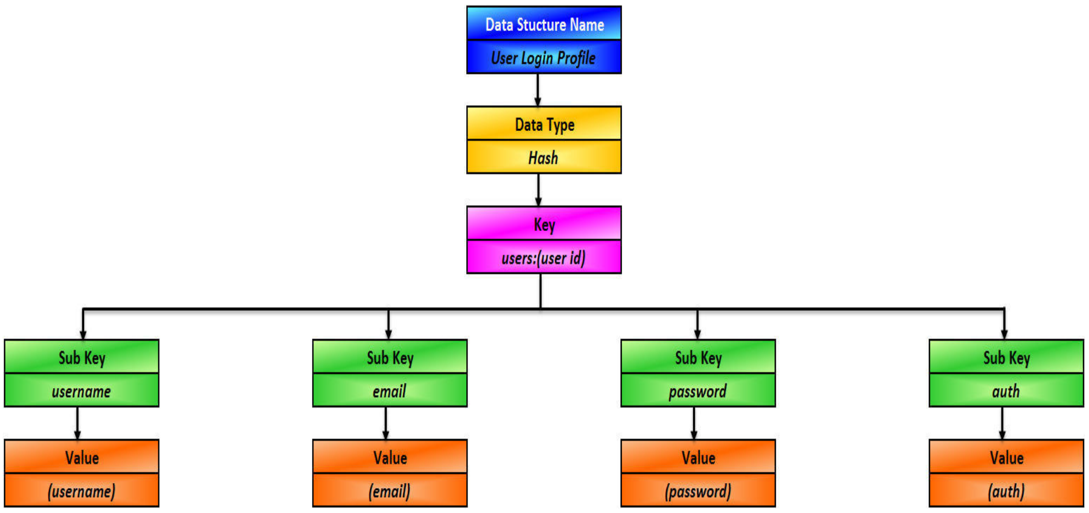
6.3.1 Application Feature: User Login

For intelligent real estate management application, the user login credentials will be stored in REDIS database. Two HASH data types will be used as shown in following Graphics. For authentication and security purpose an extra field is added in the hash set named as 'Auth' which will contain a random generated string value. On every user login, this will help in authenticating user and replacing this string value as a cookie instead of password itself. User Session management can be managed by configuring web server to use redis database as session handler.



Graphic 6.3.1-1: Data Structure- User ID

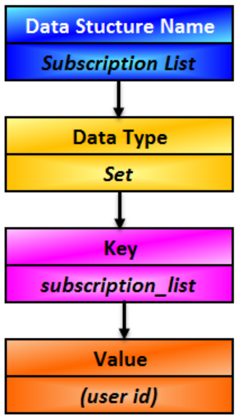
Following graphic shows a data structure of user login profile which will be used for user to log into the application.



Graphic 6.3.1-2: Data Structure- User Login Profile

6.3.2 Application Feature subscription List

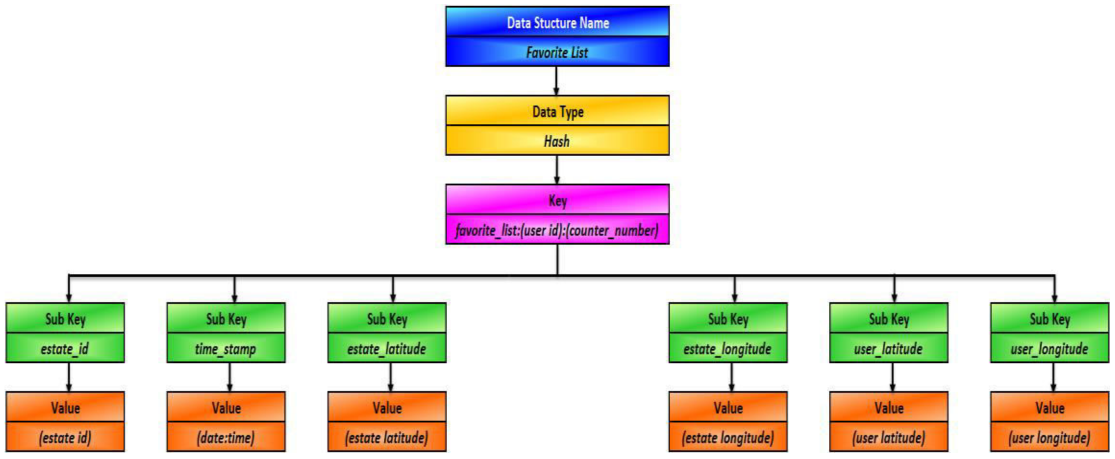
In order to send the news letters to the users, subscription status of the users need to be maintained in the database. For this purpose, a SET data type is used to store all the user ids as values. If any user is subscribed for reception of newsletters, user id of that user will be added in the Set with key subscription_list.



Graphic 6.3.2-1:: Data Structure- Subscription List

6.3.3 Application Feature: User Favorite List

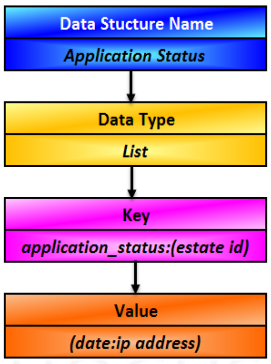
For managing the favourite lists of user, necessary information like estate id, date and time, estate location in a HASH data type will be stored which will help the application to suggest better options to the user when he/she logs in. The suggestions will be based on location of the estates that are in the user's favourite list.As showned in Graphic , the random number in key will help in storing list of favourite items that will help application layer to filter out the items in the list and give suggestions accordingly.



Graphic 6.3.3-1: Data Structure- Favourite List

6.3.4 Application Feature: Application Status

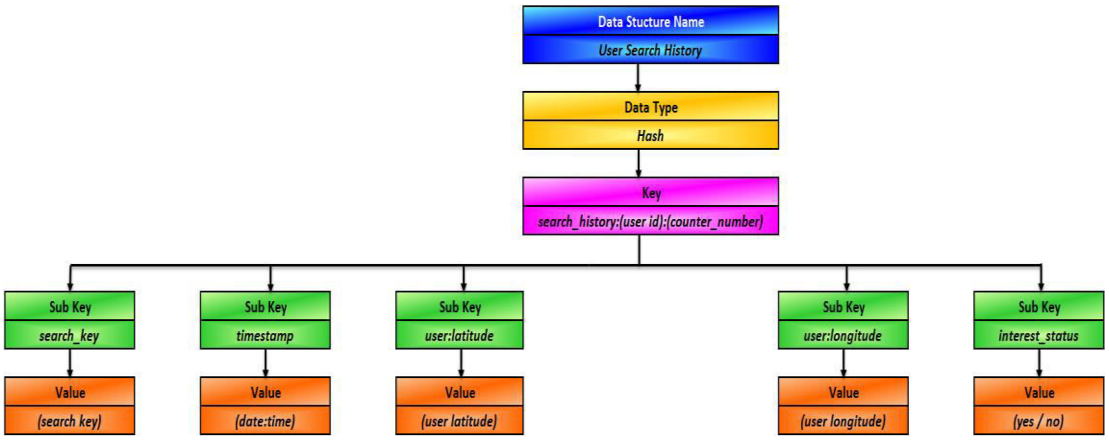
To judge the number of users visiting the application, the IP address of users visiting the application in a LIST data type will be stored. For better reporting on the application layer, details like date, number of counts will be captured which will help in generating various types of reports with the help of which popularity of the application can be known.



Graphic 6.3.4-1*:* Data Structure- Application Status

6.3.5 Application Feature: User Search History

There might be certain cases in which user might not get offers in which user is interested in. For better user experience, all the search helps in which user was interested in a HASH data type will be captures that will help to provide automatic notifications or push messages to users when the offers of his/her interest is available later on.

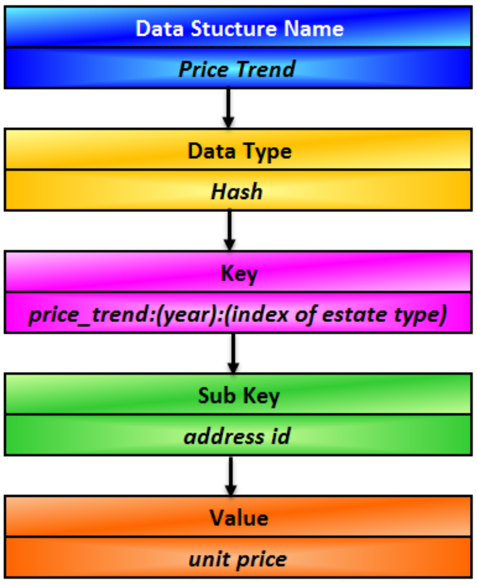


Graphic 6.3.5-1*:* Data Structure- User Search History

6.3.6 Application Feature: Price Trend

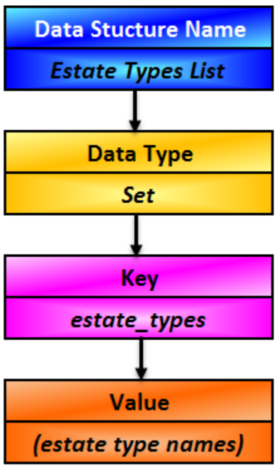
Price trend data based on the year,location and estate type is also stored in the database as

HASH data type.This will help application layer to perform various logic on the data and represent a price trend over a period of time in a pictorial view like graphs, charts.



Graphic 6.3.6-1: Data Structure- Price Trend

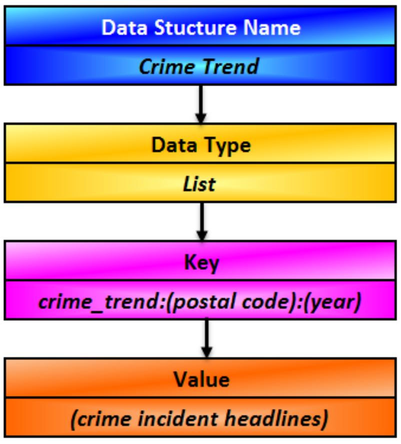
Following graphic shows the data structure of estate types list which will be used for building price trend based on these estate types in a particular location.



Graphic 6.3.6-2: Estate Types List

6.3.7 Application Feature: Crime Trend

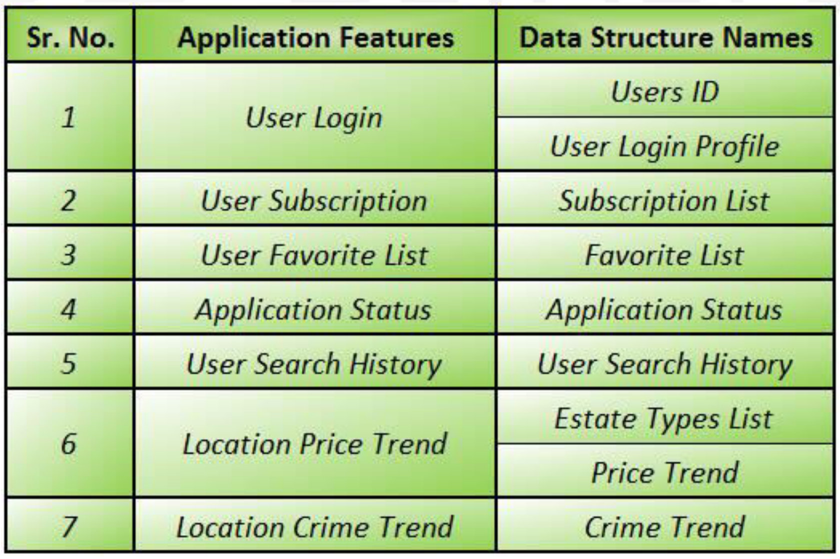
Area wise crime trend over a period of time is one of the important information which the application should able to provide. For this, information like date and crime headline in LIST data type in redis and associate it with zip code as a key will be stored.



Graphic 6.3.7-1: Crime Trend

6.3.8 Summary

Following table provides the list of proposed data structures names based on the application feature. For data models based on each data structure, please refer previous sections.



Sr. No.	Application Features	Data Structure Names
1	*User Login*	*Users ID*
		User Login Profile
2	*User Subscription*	*Subscription List*
3	*User Favorite List*	*Favorite List*
4	*Application Status*	*Application Status*
5	*User Search History*	*User Search History*
6	*Location Price Trend*	*Estate Types List*
		Price Trend
7	*Location Crime Trend*	*Crime Trend*

Graphic 6.3.8-1:*:* List of data structures based on application features

6.4 Redis - Implementation and CRUD Operations

In this document, Redis CRUD operations based on supported application features are explained and these application features relate to the user stories proposed in previous documentation. Also, proposed data structures mentioned in previous sections are implemented with the help of Redis desktop manager.

Conventions used in this document:

The commands for performing any of the CRUD operations are written in following format: Command: ***HGETALL users:US0001*** Here, the word with all capital words indicate command keyword while rest are the fields associated with the command. US0001 used in the command always indicate the user id of the user.

Please refer previous submitted documentation for following:

- Supported User Stories: User story 2, 3

6.4.1 Application Feature: User Login

Supported User Stories: User story 2, 3

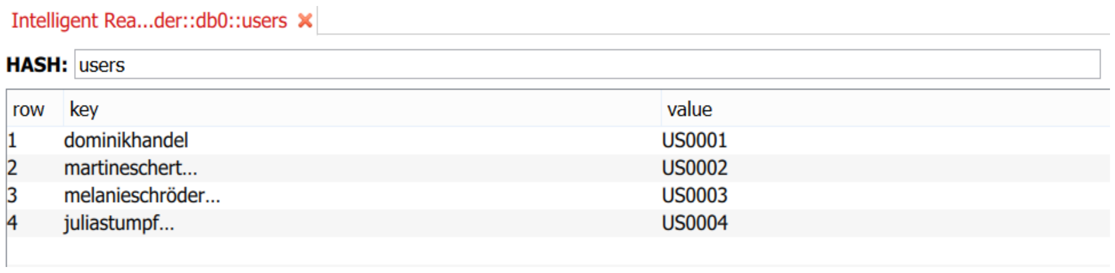
Description: When user logs into the application, first step of the application is to check whether the user is registered or not. In order to achieve this step, **read** operation needs to be performed based on hash set where key is **users.** If the user is registered application will fetch the user id. Command: ***HGETALL users***

In order to authenticate the user, the hash of the password against the fetched user id will be returned through the application and for this **read** operation needs to be performed on hash set with key users:(user_id). Command: ***HGETALL users:US0001***

In case user logs in from third part application (like Facebook, twitter, etc.), the authentication token provide by these applications needs to be stored in redis database. For this **create** operation is required. Command: ***HMSET users:US0001 auth 4bd2b007716888ed6bf6c2399a6d7305***

In case the user wants to change the password, **update** operation will be performed. Command: ***HMSET users:US0001 password 3df948067000799fbd0eac5687df571f86f18d38d8c9e146ba3af128f8e3ec04***

Please refer following graphic which is a implementation of data structure related to user login feature.

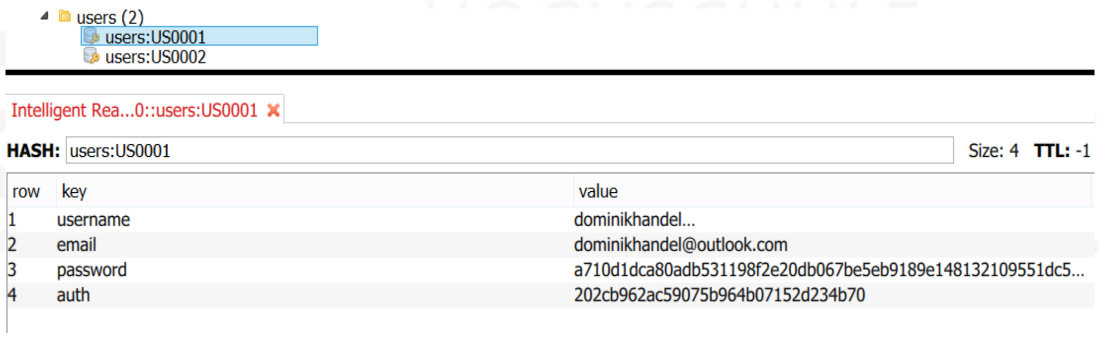

Intelligent Rea...der::db0::users

HASH: users

row	key	value
1	dominikhandel	US0001
2	martineschert...	US0002
3	melanieschröder...	US0003
4	juliastumpf...	US0004

Graphic 6.4.1-1: Data Structutre- Users ID

Please refer following graphic which is a implementation of data structure related to user login feature.


users (2)
users:US0001
users:US0002

Intelligent Rea...0::users:US0001

HASH: users:US0001 Size: 4 **TTL:** -1

row	key	value
1	username	dominikhandel...
2	email	dominikhandel@outlook.com
3	password	a710d1dca80adb531198f2e20db067be5eb9189e148132109551dc5...
4	auth	202cb962ac59075b964b07152d234b70

Graphic 6.4.1-2: Data Structutre- User Login Profile

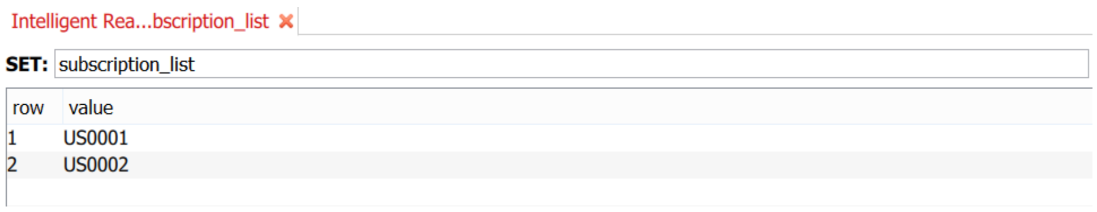
6.4.2 Application Feature: User Subscription

Supported User Stories: User story 1, 2, 3, 4, 5, 6

Description: When user subscribes for the newsletters, user id will be stored in a set with key **subscription_list**. For this, **create** operation is required. Command: ***SADD subscription_list US0001***

If the user unsubscribes for newsletters, its entry in the database will be **deleted**. Command: ***SREM subscription_list US0001***

Please refer following graphic which is a implementation of data structure related to user subscription feature.


Intelligent Rea...bscription_list

SET: subscription_list

row	value
1	US0001
2	US0002

Graphic 6.4.2-1: Data Structutre- Subscription List

6.4.3 Application Feature: User Favourite List

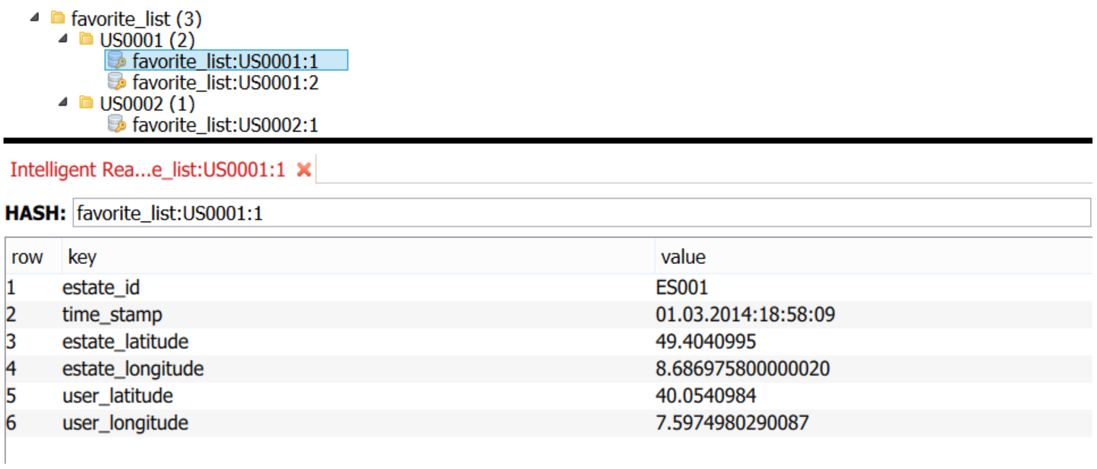
Supported User Stories: User story 1, 5

Description: In order to store all the offers in the favourite list of the user, the details like estate id, timestamp, user location, estate location will be added in the hash set with key favourite_list:(user_id):(random_number). For this **create** operation is required. Command: ***HMSET favourite_list:US0001:1 estate_id ES001 time_stamp 01.03.2014:18:58:09 estate_latitude 49.4040995 estate_longitude 8.686975800000020 user_latitude 40.0540984 user_longitude 7.5974980290087***

In order to retrieve all the favourite items in user's favourite list, **read** operation is required on the same hash set. Command: ***HGETALL favourite_list:US0001:1***

When user no more requires offers in favourite list, **delete** operation will be performed on the same hash set. Command: ***DEL favourite_list:US0001:1***

Please refer following graphic which is a implementation of data structure related to user favorite list feature.



favorite_list (3)
- US0001 (2)
  - favorite_list:US0001:1
  - favorite_list:US0001:2
- US0002 (1)
  - favorite_list:US0002:1

Intelligent Rea...e_list:US0001:1

HASH: favorite_list:US0001:1

row	key	value
1	estate_id	ES001
2	time_stamp	01.03.2014:18:58:09
3	estate_latitude	49.4040995
4	estate_longitude	8.686975800000020
5	user_latitude	40.0540984
6	user_longitude	7.5974980290087

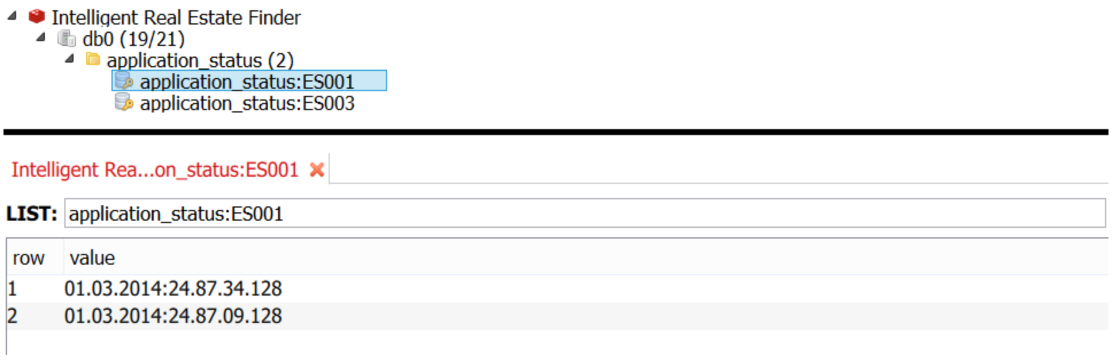
Graphic 6.4.3-1: Data Structutre- Favorite List

6.4.4 Application Feature: Application Status

Supported User Stories: User story 1, 3, 4, 5, 6

Description: For all the users viewing the estate offers, user's IP addresses along with the date will be stored in a database and for that **create** operation is required on the list with key application_status:(estate_id). Command: ***LPUSH application_status:ES001 01.03.2014:24.87.09.128***

Please refer following graphic which is a implementation of data structure related to application status feature.



Intelligent Real Estate Finder
- db0 (19/21)
  - application_status (2)
    - application_status:ES001
    - application_status:ES003

Intelligent Rea...on_status:ES001

LIST: application_status:ES001

row	value
1	01.03.2014:24.87.34.128
2	01.03.2014:24.87.09.128

Graphic 6.4.4-1: Data Structutre- Application Status

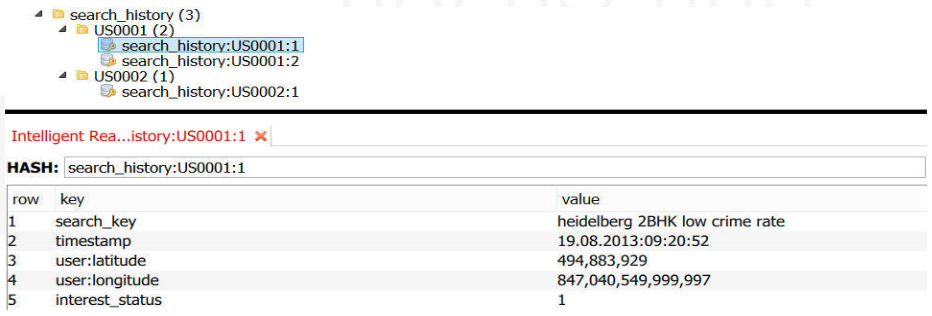
6.4.5 Application Feature: User Search history

Supported User Stories: User story 1, 3

Description: For storing search history of user, details like search keyword, timestamp, user location, interested status will be added into the database. For this **create** operation is required on hash set with key search_history:(user_id):(counter_value).

Command: ***HMSET search_history:US0001:1 search_key "heidelberg 2BHK low crime rate" timestamp 19.08.2015:09:20:52 user:latitude 494,883,929 user:longitude 847,040,549,999,997 interest_status 1***

Please refer following graphic which is a implementation of data structure related to user search history feature.


```
search_history (3)
  US0001 (2)
    search_history:US0001:1
    search_history:US0001:2
  US0002 (1)
    search_history:US0002:1
```

Intelligent Rea...istory:US0001:1

HASH: search_history:US0001:1

row	key	value
1	search_key	heidelberg 2BHK low crime rate
2	timestamp	19.08.2013:09:20:52
3	user:latitude	494,883,929
4	user:longitude	847,040,549,999,997
5	interest_status	1

Graphic 6.4.5-1: Data Structutre- User Search History

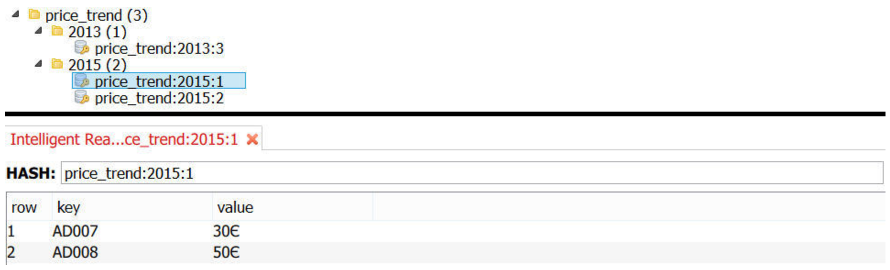
6.4.6 Application Feature: Location Price Trend

Supported User Stories: User story 6

Description: Based on the yearly survey of the unit price of estate type in various locations, **create** operation will be performed on hash set with key price_trend:(year):(estate_type_index).Command: ***LPUSH estate_types flat land house*** Command: ***HMSET price_trend :2015:3 AD007 40***

In order to generate price trend over period of time, **read** operation will be performed on the same hash set. Command: ***LRANGE estate_types 0 -1*** Command: ***HGETALL price_trend:2015:3***

Please refer following graphics which is a implementation of data structures related to location price trend feature.



```
price_trend (3)
  2013 (1)
    price_trend:2013:3
  2015 (2)
    price_trend:2015:1
    price_trend:2015:2
```

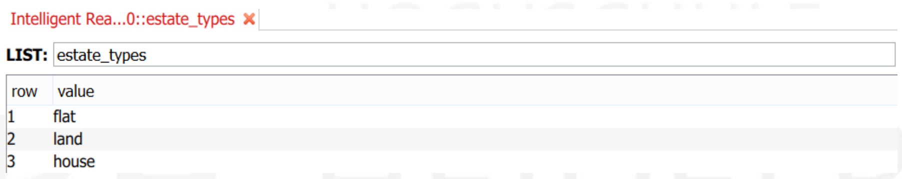
Intelligent Rea...ce_trend:2015:1

HASH: price_trend:2015:1

row	key	value
1	AD007	30€
2	AD008	50€

Graphic 6.4.6-1: Data Structutre- Price Trend

Please refer following graphics which is a implementation of data structures related to location price trend feature.



Intelligent Rea...0::estate_types

LIST: estate_types

row	value
1	flat
2	land
3	house

Graphic 6.4.6-2: Data Structutre- Estate Types List

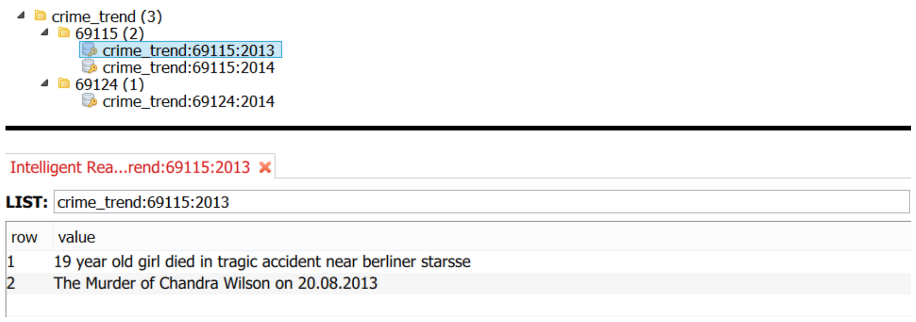
6.4.7 Application Feature: Location Crime Trend

Supported User Stories: User story 6

Description: Based on the yearly crime report, headlines of crime incidents will be stored into the database. For this **create** operation will be performed on the list with key crime_trend:(postal_code):(year). Command: ***LPUSH crime_trend:69115:2015 “the murder of Chandra Wilson on 28.08.2015”***

In order to generate the crime trend over period of time, **read** operation will be performed on the same list. Command: ***LRANGE crime_trend:69115:2015 0 -1***

Please refer following graphic which is a implementation of data structure related to location crime trend feature.


crime_trend (3)
69115 (2)
crime_trend:69115:2013
crime_trend:69115:2014
69124 (1)
crime_trend:69124:2014

Intelligent Rea...rend:69115:2013

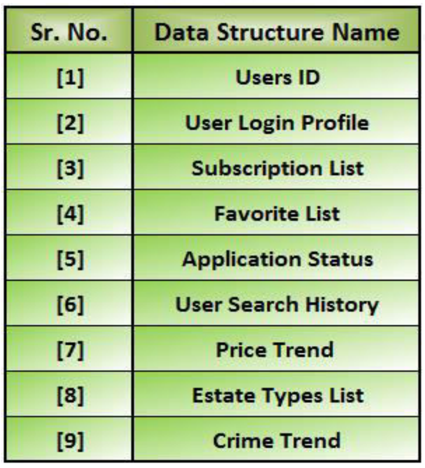
LIST: crime_trend:69115:2013

row	value
1	19 year old girl died in tragic accident near berliner starsse
2	The Murder of Chandra Wilson on 20.08.2013

Graphic 6.4.7-1: Data Structutre- Crime Trend

6.4.8 Summary

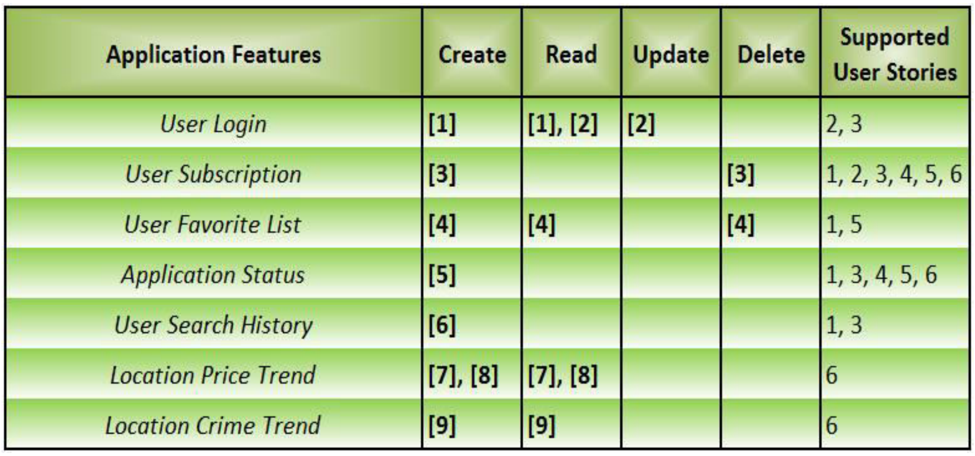
Following table shows the list of data structures implemented in the redis database and based on these data structures application features can be supported.



Sr. No.	Data Structure Name
[1]	Users ID
[2]	User Login Profile
[3]	Subscription List
[4]	Favorite List
[5]	Application Status
[6]	User Search History
[7]	Price Trend
[8]	Estate Types List
[9]	Crime Trend

Graphic 6.4.8-1: List of Data Structures

In the following table **[1]**, **[2]**, etc. indicate the data structure name listed in the previous table of list of data structures.



Application Features	Create	Read	Update	Delete	Supported User Stories
User Login	[1]	[1], [2]	[2]		2, 3
User Subscription	[3]			[3]	1, 2, 3, 4, 5, 6
User Favorite List	[4]	[4]		[4]	1, 5
Application Status	[5]				1, 3, 4, 5, 6
User Search History	[6]				1, 3
Location Price Trend	[7], [8]	[7], [8]			6
Location Crime Trend	[9]	[9]			6

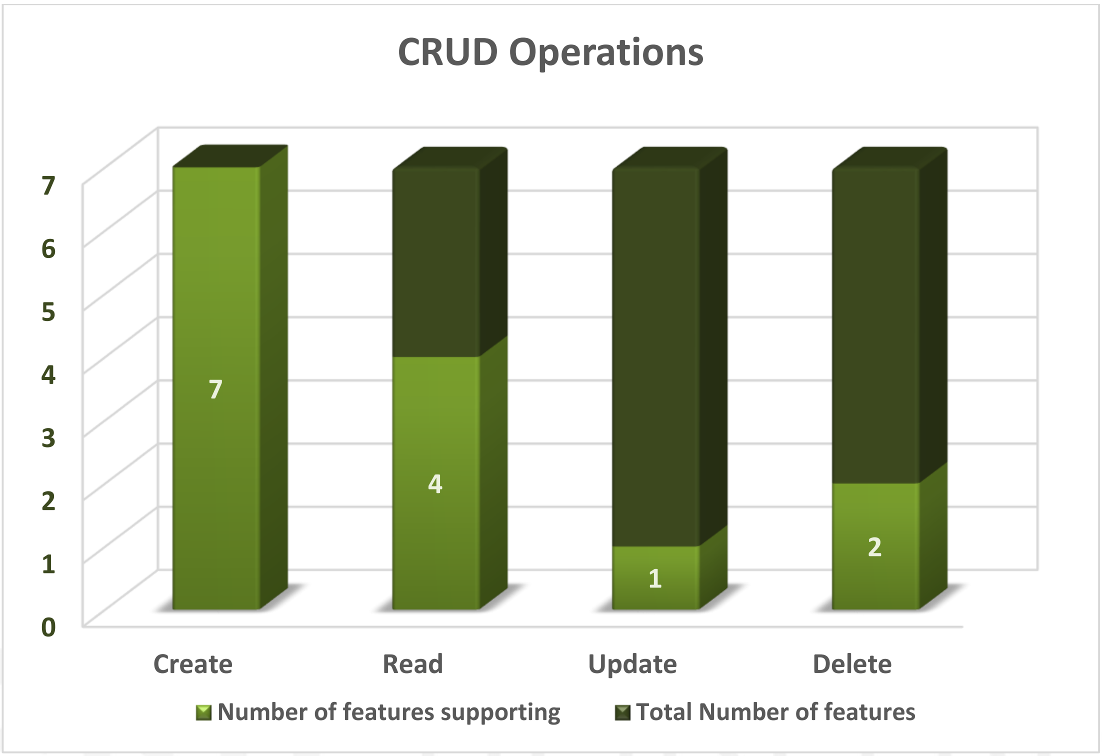
Graphic 6.4.8-2: CRUD relationship between application features and data structures

The following table and graph indicates the number of application features supported for each of the CRUD operation, so that all the supported features of Redis database are implemented. From the below graph, it can be concluded that proposed Redis Data Model is performing insert(create) and read operations most of the times.

CRUD Operations	**Create**	**Read**	**Update**	**Delete**
Number of features supporting	7	4	1	2
Total Number of features	7			

Graphic 6.4.8-3: Number of features supporting CRUD operations

The code segments used in Redis database can be found in the attachments (attachment section XLII-LII).



Graphic 6.4.8-4: Number of features supporting CRUD operations 2

10.4 Redis code

Application Features supported by redis database are implemented with Node.js and are tested in console.

1. Following code is for user login feature:

1.1 Filename : <usersModule.js>
Comments: This file contains all functions necessary to run userLoginInput.js, registerUser.js

```
exports.check_userLogin = function(username, password, client){
  client.hget("users", username, function (err, user_id) {
    if (err) {
      console.log(err);
    }
    if (user_id === null) {
      console.log("User with username '"+ username + "' does not exist!");
    } else {
      console.log("-----------------------------------------------");
      console.log('Found User ID: ' + user_id);
      client.hmget("users:" + user_id, "email", "password",function (err, user_array) {
          console.log("-----------------------------------------------");
          if (err) {
            console.log("Error: " + err);
          } else {
            console.log("Email: " + user_array[0]);
            console.log("password: " + password);
            //console.log("Password: " + user_array[1]);
            if (password === user_array[1]) {
              console.log(username + " successfully logged in!");
            } else {
              console.log("username or password is incorrect!");
            }
          }
      });
    }
    client.quit();
  });
}

var createUserId = function (num, size) {
    var s = num+"";
    while (s.length < size) s = "0" + s;
    return s;
}

exports.create_userLogin = function (username, email, password, client) {
  client.hkeys("users", function (err, keys) {
    if (err) {
      console.log("Error: " + err);
    } else {
      console.log('------------------------------------------');
      var user_id = "US" + createUserId(keys.length + 1, 4);
      //console.log("User ID: " + user_id);
      client.hset("users", username, user_id, function (err) {
        if (err) {
          console.log(err);
        } else {
          console.log(username + " with User ID " + user_id + " is created!");
        }
      });
      client.hmset("users:" + user_id, "username", username, "email", email, "password", password, "auth", "", function (err) {
        if (err) {
          console.log(err);
        } else {
          console.log(username + " is registered!");
        }
      });
    }
    client.quit();
  });
}
```

1.2 Filename: registerUser.js

Comments: When user wants to register, user details like username,password,email will be stored in database and user id will be created. Validations like proper email format, username format are been taken care of. Inbuild library 'crypto' is used to generate and store password in 'sha256' hash format.

```
var prompt = require('prompt');
var redis = require('redis');
var client = redis.createClient();
var crypto = require('crypto');
var um = require('./usersModule');

var inputFields = {
  properties: {
    username: {
      pattern: /^[a-z0-9\-\_]+$/i,
      message: 'Username must be only letters, underscores, or dashes',
      required: true
    }, email: {
      pattern:                    /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/i,
      message: 'Please enter correct email format',
      required: true
    },password: {
      hidden: true
    }
  }
};

prompt.start();

prompt.get(inputFields, function (err, result) {
  console.log('Command-line input received:');
  console.log('  username: ' + result.username);
  console.log('  email: ' + result.email);
  console.log('  password: ' + result.password);
  var pwdHash = crypto.createHash('sha256').update(result.password).digest('base64');
  um.create_userLogin(result.username, result.email, pwdHash, client);
});
```

1.3 Filename: userLoginInput.js

Comments: This code will be executed when a registered user wants to log in into the application. Providing access to only registered user is also handled.

```
var prompt = require('prompt');
var redis = require('redis');
var client = redis.createClient();
var crypto = require('crypto');
var um = require('./usersModule');

var inputFields = {
  properties: {
    username: {
      pattern: /^[a-z0-9\-\_]+$/i,
      message: 'Username must be only letters, underscores, or dashes',
      required: true
    }, password: {
      hidden: true
    }
  }
};

prompt.start();

prompt.get(inputFields, function (err, result) {
  console.log('Command-line input received:');
  console.log('  username: ' + result.username);
  console.log('  password: ' + result.password);
  var pwdHash = crypto.createHash('sha256').update(result.password).digest('base64');
  //console.log('  hash: ' + pwdHash);
  um.check_userLogin(result.username, pwdHash, client);
});
```

2. Following code is for Subscription List Feature:

2.1 Filename: userSubscriptionModules.js
Comments: This file contains all functions required to run userSubscription.js file.

```
exports.create_subscription = function (user_id, client) {
  client.sadd("subscription_list", user_id, function (err) {
    if (err) {
      console.log(err);
    } else {
      console.log(user_id + " added in subscription list");
    }
    client.quit();
  });
}

exports.delete_subscription = function (user_id, client) {
  client.srem("subscription_list", user_id, function (err) {
    if (err) {
      console.log(err);
    } else {
      console.log(user_id + " removed from subscription list");
    }
    client.quit();
  });
}

exports.read_subscription_list = function (client) {
  client.smembers("subscription_list", function (err, user_ids) {
    if (err) {
      console.log(err);
    } else {
      console.log("----------------------------------------");
      for (var i = 0; i < user_ids.length; i++) {
        console.log(user_ids[i]);
      }
      console.log("----------------------------------------");
    }
    client.quit();
  });
}
```

2.2 Filename: userSubscription.js
Comments: All subscribed users will be maintained with the help of this code.

```
var redis = require('redis');
var client = redis.createClient();
var prompt = require('prompt');
var usm = require('./userSubscriptionModules');

console.log("Press 1 to Subscribe");
console.log("Press 0 to Unsubscribe");

prompt.start();

prompt.get("option", function (err, result) {
  if (err) {
    console.log(err);
  } else {
    switch (result.option) {
      case "1": usm.create_subscription("US0001", client);
      break;
      case "0": usm.delete_subscription("US0006", client);
      break;
      default: console.log("Please enter valid option!");
    }
  }
  usm.read_subscription_list(client);
});
```

3. Following code is for User Favourite List Feature:

3.1 Filename: favoriteListModules.js
Comments: This file contains all the functions which are required to run the file favouriteList.js

```
exports.create_favoriteList = function (user_id, counter, estate_id, time_stamp, estate_latitude, estate_longitude,
user_latitude, user_longitude, client) {
      client.hmset("favourite_list:" + user_id +":" + counter, "estate_id", estate_id, "time_stamp", time_stamp, "estate_latitude",
estate_latitude,
        "estate_longitude", estate_longitude, "user_latitude", user_latitude, "user_longitude", user_longitude, function (err) {
          if (err) {
            console.log(err);
          } else {
            console.log("favourite_list:" + user_id +":" + counter + " is created!");
          }
          client.quit();
        });
}

exports.read_favoriteList = function (user_id, client) {
  client.keys("favourite_list:" + user_id +":*", function (err, keys_array) {
    if (err) {
      console.log(err);
    } else {
      for (var i = 0; i < keys_array.length; i++) {
        client.hmget(keys_array[i], "estate_id", "time_stamp", "estate_latitude", "estate_longitude", "user_latitude",
"user_longitude", function (err, favourite_arr) {
          if (err) {
            console.log(err);
          } else {
            console.log('---------------------------------------------------');
            console.log("User Id: " + user_id);
            console.log("Estate ID: " + favourite_arr[0]);
            console.log("Time Stamp: " + favourite_arr[1]);
            console.log("Estate Latitude: " + favourite_arr[2]);
            console.log("Estate Longitude: " + favourite_arr[3]);
            console.log("User Latitude: " + favourite_arr[4]);
          console.log("User Longitude: " + favourite_arr[5]);
        }
        client.quit();
      });
    }
  }
 });
}

exports.delete_favoriteList = function (user_id, client) {
  client.keys("favourite_list:" + user_id +":*", function (err, list_array) {
    if (err) {
      console.log(err);
    } else {
      for (var i = 0; i < list_array.length; i++) {
        client.del(list_array[i], function (err) {
          if (err) {
            console.log(err);
          } else {
            //console.log("Favourite List of user with ID " + user_id + " deleted!");
          }
        });
      }
      console.log("Favourite List of user with ID " + user_id + " deleted!");
    }
    client.quit();
  });
}


exports.delete_favoriteListItem = function (user_id, item_no, client) {
  client.del("favourite_list:" + user_id +":" + item_no, function (err) {
    if (err) {
      console.log(err);
    } else {
      console.log("Favourite List Item " + item_no +" of user with ID " + user_id + " deleted!");
    }
    client.quit();
  });
}
```

3.2 Filename: favouriteList.js
Comments: With the help of this code all the offers in the user's favourite list will be captured and stored in the database. Similary, retrieval and deletion of items is also handled.

```
var redis = require('redis');
var client =  redis.createClient();
var timestamp = require('console-timestamp');
var prompt = require('prompt');
var flm = require('./favoriteListModules');

var ts = timestamp('DD.MM.YYYY:hh:mm:ss');
var key_counter = timestamp('DDMMYYYYhhmmss');

console.log("----------------------Favourite List Menu-------------------------");
console.log("1 - Mark Favourite Item");
console.log("2 - Show Favourite List");
console.log("3 - Delete Favourite List Item");
console.log("4 - Delete Favourite Complete List");

prompt.start();

prompt.get("option", function (err, result) {
  if (err) {
    console.log(err);
  } else {
    switch (result.option) {
      case "1": flm.create_favoriteList("US0005", key_counter, "ES008", ts, 49.4040995, 8.68697580000002, 40.0540984,
7.5974980290087, client);
      break;
      case "2": flm.read_favoriteList("US0001", client);
      break;
      case "3": flm.delete_favoriteListItem("US0005", 2, client);
      break;
      case "4": flm.delete_favoriteList("US0005", client);
      break;
      default: console.log("Please enter valid option!");
    }
  }
});
```

4. Following code is for Application Status Feature:

4.1 Filename: applicationStatus.js
Comments: Following code is used for storing the user's ip address with the timestamp who have viewed an estate offer. Thus, helping to maintain a record of number of views on particular estate.

```
var redis = require('redis');
var client = redis.createClient();
var timestamp = require('console-timestamp');

var date = timestamp('DD.MM.YYYY');

var mark_application_status = function (estate_id, visit_date, ip_addr) {
  client.lpush("application_status:" + estate_id, visit_date + ":" + ip_addr, function (err) {
    if (err) {
      console.log(err);
    } else {
      console.log("application_status:" + estate_id + " is inserted!");
    }
    client.quit();
  });
}

mark_application_status("ES001", date, "123.41.120.22");
```

5. Following code is for User Search History Feature:

5.1 Filename: searchHistory.js
Comments: This code stores all the details related to the user's search history like search keywords, user location, timestamp and interest status.

```
var redis = require('redis');
var client = redis.createClient();
var time_stamp = require('console-timestamp');

var ts = time_stamp('DD.MM.YYYY:hh:mm:ss');
var key_counter = time_stamp('DDMMYYYYhhmmss');

var mark_searchHistory = function (user_id, counter, search_key, timestamp, user_latitude, user_longitude,interest_status) {
    client.hmset("search_history:" + user_id + ":" + counter, "search_key", search_key, "timestamp", timestamp,
     "user_latitude", user_latitude, "user_longitude", user_longitude, "interest_status", interest_status,
      function (err) {
        if (err) {
          console.log(err);
        } else {
          console.log("search_history:" + user_id + ":" + key_counter + " is inserted!");
        }
        client.quit();
      });
}

mark_searchHistory("US0003", key_counter, "2 BHK in Aachen semi-furnished", ts, "504,883,920", "657,040,549,999,120", 1);
```

6. Following code is for Location Price Trend Feature:

6.1 Filename: priceTrendModules.js
Comments: This file contains all functions required to run the priceTend.js file.

```
exports.insertEstateType = function (client, prompt) {
  var inputFields = {
    properties: {
      estate_type: {
        message: 'Estate Type',
        required: true
      }
    }
  };
  prompt.start();
  prompt.get(inputFields, function (err, res) {
    if (err) {
      console.log(err);
    } else {
      client.rpush("estate_types", res.estate_type, function (err) {
        if (err) {
          console.log(err);
        } else {
          console.log("Estate type " + res.estate_type + " is inserted!");
          process.exit(0);
        }
        client.end();
      });
    }
  });
}

exports.get_estate_types = function (client) {
  client.lrange("estate_types", 0, -1, function (err, list_array) {
    if (err) {
      console.log(err);
    } else {
      console.log('-------------Estate Types ------------');
      for (var i = 0; i < list_array.length; i++) {
        j = i + 1;
        console.log(j + ". " + list_array[i]);
      }
    }
    client.quit();
  });
}

exports.insert_priceTrend = function (client, prompt) {
  var inputFields = {
    properties: {
      Year: {
        message: 'Year',
        required: true
      }, Address_ID: {
        message: 'Address ID',
        required: true
      }, Unit_Price: {
```

```
      message: 'Unit Price',
      required: true
    }
  }
};
client.llen("estate_types", function (err, size) {
  if (err) {
      console.log(err);
  } else {
    prompt.start();
    prompt.get("estate_type_option", function (err, res) {
      if (err) {
        console.log(err);
      } else {
        if (res.estate_type_option < size + 1) {
          prompt.get(inputFields, function (err, reply) {
            client.hmset("price_trend:" + reply.Year + ":" + res.estate_type_option, reply.Address_ID, reply.Unit_Price, function
(err) {
              if (err) {
                console.log(err);
              } else {
                console.log("price_trend:" + reply.Year + ":" + res.estate_type_option + " is inserted!");
              }
              client.quit();
            });
          });
        } else {
          console.log("Please enter valid option!");
        }
      }
    });
  }
});
}


var getPriceTrend = function (estate_type_index, client) {
  client.keys("price_trend:*:" + estate_type_index, function (err, year_trend) {
    if (err) {
      console.log(err);
    } else {
      for (var i = 0; i < year_trend.length; i++) {
        (function (j) {
          //client.hmget(year_trend[j], "address_id", "unit_price", function (err, listItems) {
          client.hgetall(year_trend[j], function (err, listItems) {
            if (err) {
              console.log(err);
            } else {
              console.log('---------------------------------------------');
              var year = year_trend[j].split(":");
              console.log("Year: " + year[1]);
              var sub_key = [];
              for (var i in listItems) {
                sub_key.push(JSON.stringify(i));
                sub_key.push(JSON.stringify(listItems[i]));
              }
              console.log("Address ID: " + sub_key[0].replace(/^"(.*)"$/, '$1'));
              console.log("Unit Price: " + sub_key[1].replace(/^"(.*)"$/, '$1'));
            }
            client.quit();
          });
        })(i);
      }
    }
  });
}

exports.readPriceTend = function (client, prompt) {
  client.llen("estate_types", function (err, size) {
    if (err) {
      console.log(err);
    } else {
      prompt.start();
      prompt.get("estate_type_option", function (err, res) {
        if (err) {
          console.log(err);
        } else {
          if (res.estate_type_option < size + 1) {
            getPriceTrend(res.estate_type_option, client);
          } else {
            console.log("Please enter valid option!");
          }
        }
      });
    }
  });
}
```

6.2 Filename: priceTrend.js
Comments: This code is used for storing the unit price of an estate type based on location and the year. Also, yearly price trend based on estate type can be viewed using this code.

```
var redis = require('redis');
var client = redis.createClient();
var cli = redis.createClient();
var prompt = require('prompt');
var pr = require('prompt');
var ptm = require('./priceTrendModules');



console.log("---------------------- Price Trend Menu -------------------------");
console.log("1 - Insert Estate Type");
console.log("2 - Insert Price based on estate type");
console.log("3 - Read Price based on estate type");

prompt.start();

prompt.get("option", function (err, result) {
  if (err) {
    console.log(err);
  } else {
    switch (result.option) {
      case "1": ptm.insertEstateType(client, prompt);
      break;
      case "2": ptm.get_estate_types(cli);
      ptm.insert_priceTrend(client, prompt);
      break;
      case "3": ptm.get_estate_types(cli);
      ptm.readPriceTend(client, prompt);
      break;
      default: console.log("Please enter valid option!");
      client.quit();
      cli.quit();
    }
  }
});
```

7. Following code is for Location Crime Trend Feature:

7.1 Filename: crimeTrendModules.js
Comments: This file contains all functions required to run the crimeTend.js file.

```
exports.create_crime_trend = function (zipcode, year, headline, client) {
  client.lpush("crime_trend:" + zipcode + ":" + year, headline, function (err) {
    if (err) {
      console.log(err);
    } else {
      console.log("crime_trend:" + zipcode + ":" + year + " is inserted!");
    }
    client.quit();
  });
}

exports.read_crime_trend = function (zipcode, client) {
  client.keys("crime_trend:" + zipcode + ":*", function (err, keys_array) {
    if (err) {
      console.log(err);
    } else {
      for (var i = 0; i < keys_array.length; i++) {
        (function (j) {
          client.lrange(keys_array[j], 0, -1, function (err, result) {
            if (err) {
              console.log(err);
            } else {
              console.log('------------------------------------------');
              var year = keys_array[j].split(":");
              console.log("Year: " + year[2]);
              for (var k = 0; k < result.length; k++) {
                console.log(JSON.stringify(result[k]).replace(/^"(.*)"$/, '$1'));
              }
            }
            client.quit();
          });
        })(i);

      }
    }
  });

}
```

7.2 Filename: crimeTrend.js
Comments: This code is used for storing the crime headlines based on the location and year. Also, yearly crime trend can be viewed using this code.

```
var redis = require('redis');
var client = redis.createClient();
var prompt = require('prompt');
var  ctm = require("./crimeTrendModules");

var inputFields = {
  properties: {
    Zipcode: {
      message: 'Zipcode',
      required: true
    }, Year: {
      message: 'Year',
      required: true
    }, Headline: {
      message: 'Headline',
      required: true
    }
  }
};

var postalcode = {
  properties: {
    zipcode: {
      message: 'Zipcode',
      required: true
    }
  }
};

console.log("---------------------- Crime Trend Menu ------------------------");
console.log("1 - Insert Crime Trend");
console.log("2 - Read Crime Trend");

prompt.start();


prompt.get("option", function (err, reply) {
  if (err) {
    console.log(err);
  } else {
    switch (reply.option) {
      case "1":
      prompt.get(inputFields, function (err, res) {
        ctm.create_crime_trend(res.Zipcode, res.Year, res.Headline, client);
      })
      break;
      case "2":
      prompt.get(postalcode, function (err, rep) {
        ctm.read_crime_trend(rep.zipcode, client);
      })
      break;
      default: console.log("Please enter valid option!");
    }
  }
});
```