Software Engineering Group 09 Project Design Specification

Author:	Runar Reve [rur7]
Config Ref:	SE.GP09.DESIGNSPEC
Date:	02th April 2019
Version:	1.1
Status:	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to describe and define the programs in our system, how those programs will be implemented and interact with each other. It is following the standards from the general Documentation Standards [1] and describing the system requirements for a pub tour system [2].

1.2 Scope

The document describes the programs in our system and breaks each program down in detail, specifying the names and purpose of the significant classes in each program, and how all of the components will work together.

1.3 Objectives

The objective of this document is to define how the system will be built, specify the names and purpose of significant classes and functions, the outcome of which should be a clear and concise design for our system that can be followed by all members of the team.

2. DECOMPOSITION DESCRIPTION

2.1 Programs in system

The Pub Tour system is built up of two programs and a database server:

- The desktop program
- The android program
- The database server

2.1.1 The desktop program

The desktop program provides administration over the data the android application will use. It can create a new pub list for a town and load in an existing pub list. It can also add pubs, delete pubs, and edit existing pubs in a list. Linking pictures of each pub to the details of the pubs is also present. Then when something is done it will export it into a database where the android program will be able to receive and use this information.

2.1.2 The android program

The android program collects the latest list of pubs in a selected town. The program can display all the pubs as a list and every pub can be selected to display further details about itself. It will display between one and five photos, the GPS location of the pub, opening and closing times, and a short description of the pub. It can also filter out pubs depending on some specific criteria (allows dogs, loud music, Welsh, etc), these criteria will also be displayed in further details.

The program will also have a random pub tour feature where it will create a random list of pubs, this list can be filtered or unfiltered, and then guide the user from pub to pub through the list. There is also a more planed pub tour where times are assigned to each pub and the program will tell the user where they should be at a specific time.

2.1.3 The database server

The database server will take care of storing all the pubs and towns that will be in the system. It can only be modified by the desktop program and viewed by both the desktop and the android program. It will be using PostgreSQL.

2.2 Significant Classes

2.2.1 Significant classes in the desktop program

Pub - A class to represent a pub, will be able to convert its attributes to be understood as SQL

Town - A class to represent a town, will be able to convert its attributes to be understood as SQL

PubEngin - This will be the main stage for the program and will set up the desktop program and link all the classes together.

DbCommunicator - This class will communicate with the database and send commands to receive, add, and edit the data and pubs in it.

2.2.2 Significant classes in the android program

DbReceiver - This class will retrieve the pubs and towns from the database server.

Pub - A class to represent a pub, will be able to convert its attributes to be understood as SQL

2.2.3 Significant tables in the database server

Pubs - This will store all the different pubs.

Towns - This will store all the different towns that will be allowed for pubs to be in.

2.3 Mapping from requirement to classes

2.3.1 Mapping requirement for the desktop system

Requirement	*Classes providing requirement*
FRD1	PubEngin, Town
FRD2	PubEngin, Town, Pub
FRD3	PubEngin, DbCommunicator
FRD4	PubEngin, DbCommunicator
FRD5	PubEngin, DbCommunicator, Pub
FRD6	PubEngin, DbCommunicator, Pub
FRD7	PubEngin, DbCommunicator
FRD8	PubEngin, DbCommunicator, Pub

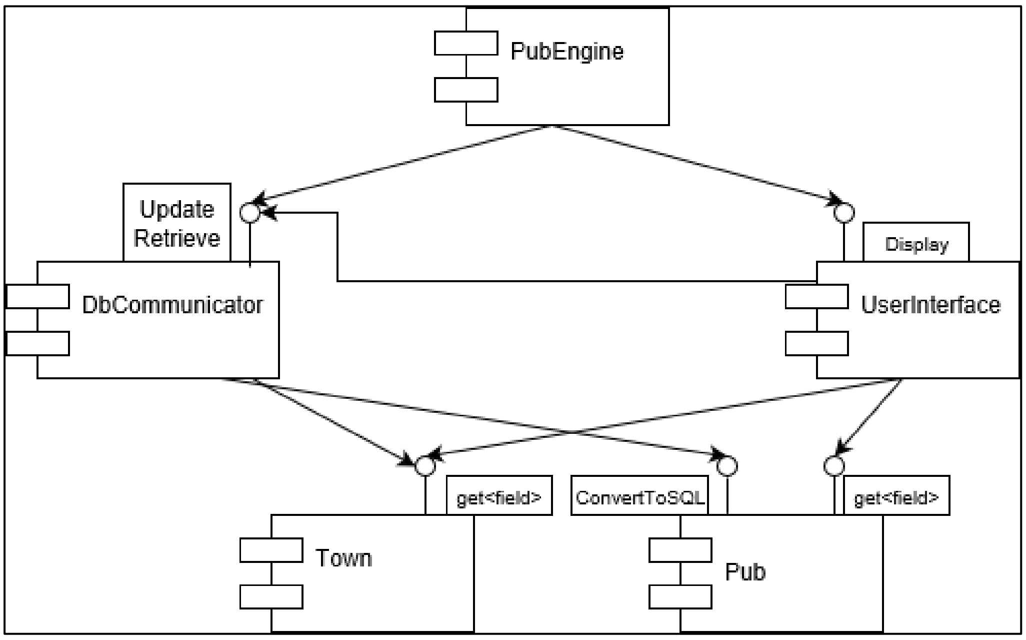
2.3.2 Mapping requirement for the mobile system

Requirement	*Classes providing requirement*
FRD1	DbReceiever, Pub
FRD2	Pub
FRD3	Pub
FRD4	Pub
FRD5	Pub
FRD6	N/A

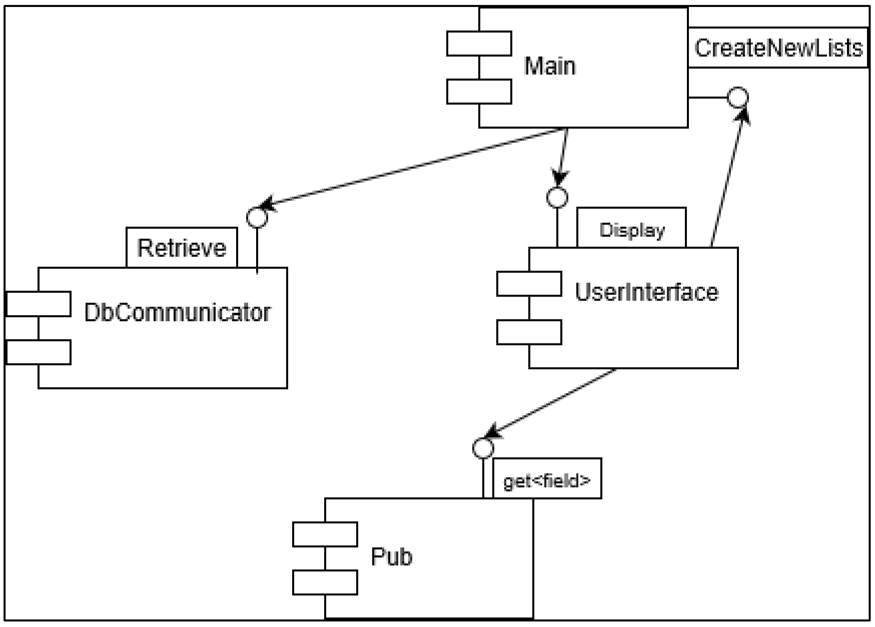
3. DEPENDENCY DESCRIPTION

3.1 Component Diagrams

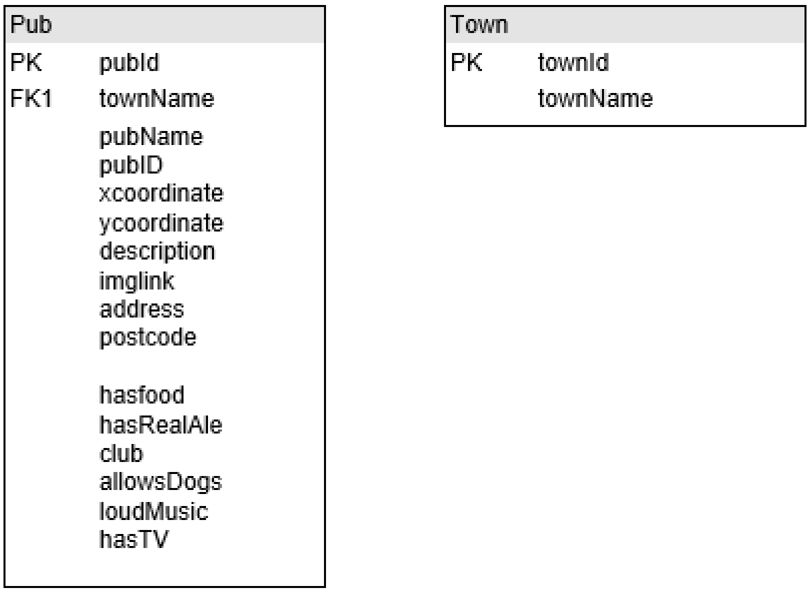
3.1.1 Component Diagram for Desktop Program



3.1.2 Component Diagram for Android Program



3.2 Table Diagram



Pub	
PK	pubId
FK1	townName
	pubName
	pubID
	xcoordinate
	ycoordinate
	description
	imglink
	address
	postcode
	hasfood
	hasRealAle
	club
	allowsDogs
	loudMusic
	hasTV

Town	
PK	townId
	townName

4. INTERFACE DESCRIPTION

4.1 PubTour interface specification

```
import java.util.ArrayList;

  /**
   * uk.ac.aber.cs221.group09.PubTourInterface - A interface to modify the data for in ways the user wants.
   * <p>
   * Some classes to sort to filter pubs, "filterPubs", that you give it following
   * an example pub that has the attributes the user wants.
   * There is a randomize pub where it will extract the amount of pubs you want randomly and
   * put them in another list
   *
   * @author (Runar Reve, rur7)
   * @version 0.1 (Draft)
   */
public class PubTourInterface {

    /**
     * Will filter out pubs to a new ArrayList that only has wanted attributes
     * @param MainList the Main list that is going to be filtered
     * @param samplePub example pub that has the wanted attributes, null where N/A
     * @return a new ArrayList with pubs with wanted attributes
     */
    ArrayList<Pub> filterPubs(ArrayList<Pub> mainList, Pub samplePub);

    /**
     * Create a random pub tour
     * @param mainPubList is the list that is going to be extracting pubs out from
     * @param amountOfPubs the amount of pubs wanted in the list
     * @return a new ArrayList with pubs that is going to be in the tour
     */
    ArrayList<Pub> randomPubs(ArrayList<Pub> mainPubList, int amountOfPubs);
}
```

4.2 Town interface specification

```
/**
   * uk.ac.aber.cs221.group09.object.TownInterface - An interface for a town class.
```

```
  * <p>
  * Will be used to create town objects and manipulate the data of it.
  * Use the getters and setters to retrieve or edit the towns data.
  * townToDatabase is used to convert the object into a string that can be used to communicate with a database
  *
  * @author (Runar Reve, rur7)
  * @version 0.1 (Draft)
  * @see Town
  * @see DbCommunicator
  */
public class TownInterface {

   /**
    * Converts the information of a town into something the database can read
    * @param town the town that is being converted
    * @return string that can be used in a SQL command
    */
   String townToDatabase(Town town);

  //Getters and setters for all variables
}
```

4.3 Pub interface specification

```
package uk.ac.aber.cs221.group09.object;
import uk.ac.aber.cs221.group09.database.DbCommunicator;

/**
 * uk.ac.aber.cs221.group09.object.PubInterface - An interface for a pub class.
 * <p>
 * Will be used to create a pub object and manipulate the data of it.
 * Use the getters and setters to retrieve or edit the pubs data.
 * pubToDatabase is used to convert the object into a string that can be used to communicate with a database
 *
 * @author (Runar Reve, rur7)
 * @version 0.1 (Draft)
 * @see Pub
 * @see DbCommunicator
 */
public class PubInterface {


   /**
    * Converts the information of the pub into something the database can read
    * @param pub the pub that being converted
    * @return string that can be used in a SQL command
    */
   String pubToDatabase(Pub pub);
  //Getters and setters for all variables
}
```

4.4 DbCommunicator interface specification

4.4.1 Receiver Side

```
package uk.ac.aber.cs221.group09.database;

  import uk.ac.aber.cs221.group09.object.Pub;
  import uk.ac.aber.cs221.group09.object.Town;

  import java.sql.SQLException;
  import java.sql.Statement;
  import java.util.ArrayList;

  /**
  * uk.ac.aber.cs221.group09.database.DbReceiverInterface - A interface for how to receive
  * towns and pubs from the database.
  * <p>
  * Both town and pubs have their separate methods for retrieving data from the database.
  * In every method it has to take in a Statement that is the connection with the database.
  *
  * For updating the database see DbUpdateInterface
  *
  * @author (Runar Rve, rur7)
  * @version 0.1  (draft)
  * @see (DbUpdaterInterface)
  */
public class DbReceiverInterface {

  //-----PUB-----
  /**
   * Will retrieve specific pubs depending on the filters that is given to it
   * @param town name of town that is being retrieved
   * @param statement the connection with the database
   * @return an ArrayList of pubs following the filters
   * @throws SQLException if an SQL error occurs
   */
  ArrayList<Pub> retrievePubs(Statement statement, String town)
  throws SQLException;

  //-----TOWN-----
  /**
   * Retrieves all the Towns in the database
   * @param statement the connection with the database
   * @return an ArrayList of all the towns in the database
   * @throws SQLException if an SQL error occurs
   */
  ArrayList<Town> retrieveTowns(Statement statement)
  throws SQLException;
}
```

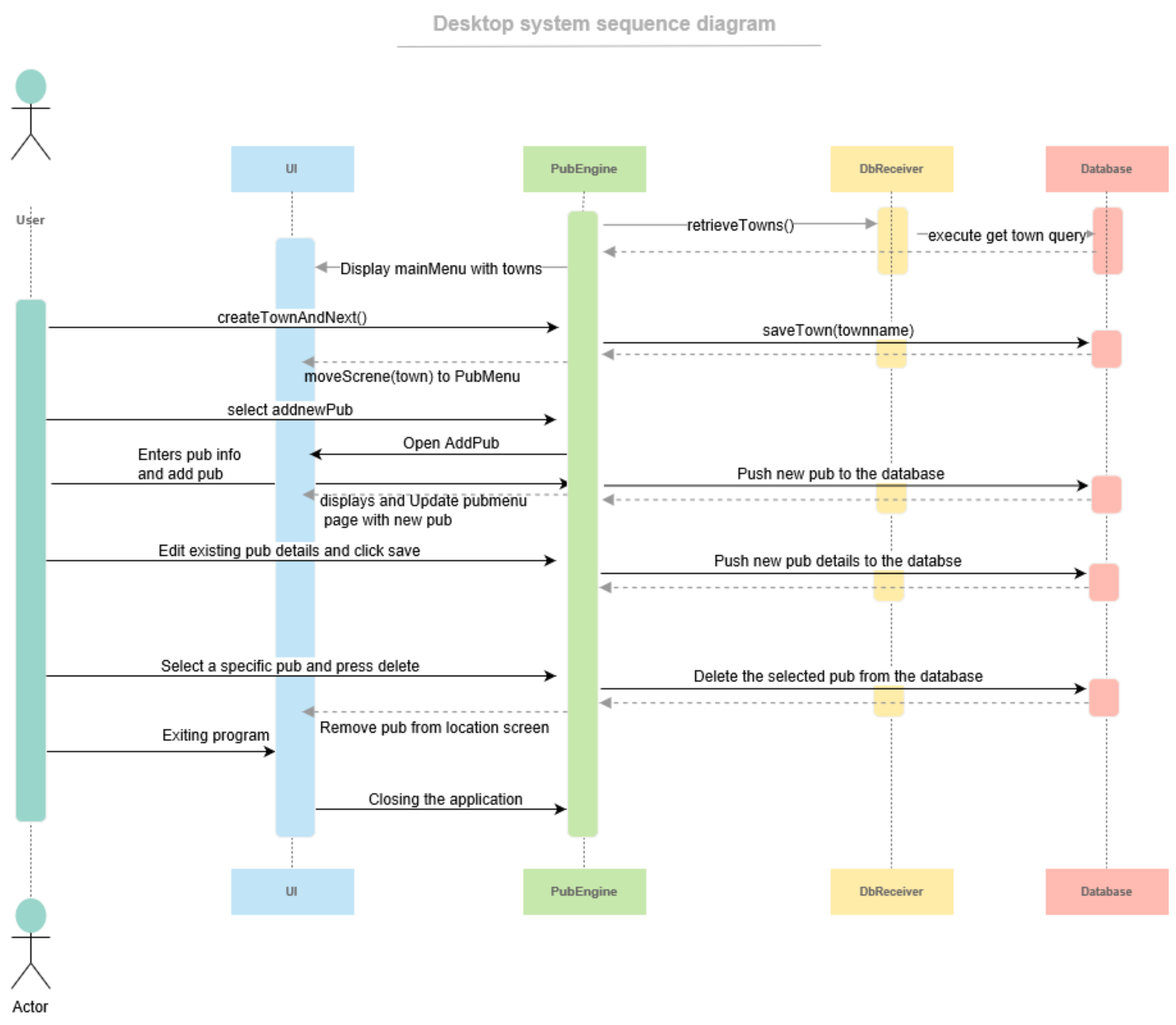
4.4.2 Update Side

```
package uk.ac.aber.cs221.group09.database;

import uk.ac.aber.cs221.group09.object.Pub;
import uk.ac.aber.cs221.group09.object.Town;
import java.sql.SQLException;

/**
 * DbUpdaterInterface - An interface for how to update the database.
 * <p>
 * Both town and pubs have their separate methods for updating the database.
 * In every method it has to take in a Statement that is the connection with the database.
 * For receiving data from the database see DBReceiverInterface.
 *
 * @author Runar Reve (rur7)
 * @version 0.1 Draft
 * @see DbReceiverInterface
 */
public class DbUpdaterInterface {

   //-----PUB-----
   /**
    * Adds a new pub to the database.
    *
    * @param newPub the new pub that is going to be added.
    * @throws SQLException if an SQL error occurs.
    */
   void addPub(Pub newPub)
          throws SQLException;
   /**
    * Remove an existing pub from the database.
    *
    * @param pub the pub that is being removed.
    * @return true if pub successfully removed the pub.
    * @throws SQLException if an SQL error occurs.
    */
   boolean removePub(Pub pub)
          throws SQLException;

   //-----TOWN-----
   /**
    * Adds a new town in the database.
    *
    * @param newTown the new town that is being pushed.
    * @throws SQLException if an SQL error occurs.
    */
   void addTown(Town newTown)
          throws SQLException;

   /**
    * Remove an existing town from the database.
    *
    * @param town the town that is being removed.
    * @return true if town was successfully removed.
    * @throws SQLException if an SQL error occurs.
    */
   boolean removeTown(Town town)
          throws SQLException;

} throws SQLException;
```

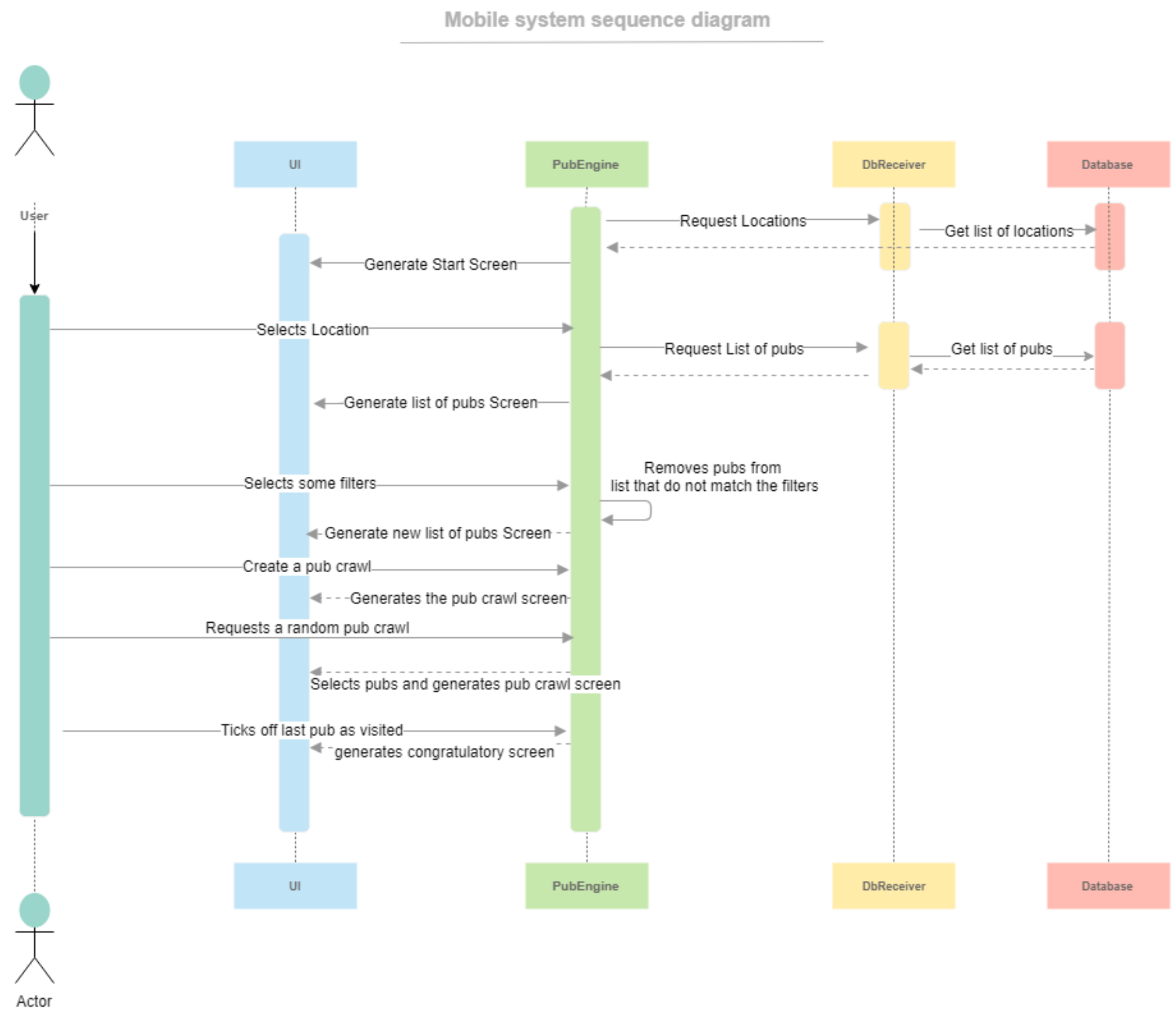
5. DETAILED DESIGN

5.1 Sequence diagrams

5.1.1 Desktop program


Desktop system sequence diagram
User
UI
PubEngine
DbReceiver
Database
retrieveTowns()
execute get town query
Display mainMenu with towns
createTownAndNext()
saveTown(townname)
moveScrene(town) to PubMenu
select addnewPub
Open AddPub
Enters pub info and add pub
Push new pub to the database
displays and Update pubmenu page with new pub
Edit existing pub details and click save
Push new pub details to the databse
Select a specific pub and press delete
Delete the selected pub from the database
Remove pub from location screen
Exiting program
Closing the application
Actor

5.1.2 Android Program


Mobile system sequence diagram
User
UI
PubEngine
DbReceiver
Database
Request Locations
Get list of locations
Generate Start Screen
Selects Location
Request List of pubs
Get list of pubs
Generate list of pubs Screen
Selects some filters
Removes pubs from list that do not match the filters
Generate new list of pubs Screen
Create a pub crawl
Generates the pub crawl screen
Requests a random pub crawl
Selects pubs and generates pub crawl screen
Ticks off last pub as visited
generates congratulatory screen
UI
PubEngine
DbReceiver
Database
Actor

5.2 Significant algorithms

5.2.1 Filtering pubs

For filtering pubs, we have a function that runs through getting the values of the sliders where the user picks filters. There are 3 stages for on the sliders: “No”, No Preference”, and “Yes”. If on “No Preference” the function will not check for that filter on the pub, but if it’s “Yes” and “No”, then it will test if it violates that characterises. If it at any point it violated any of the filters, then it will not add that pub to the filtered list and will move on to the next pub.

5.2.2 Communicating with database

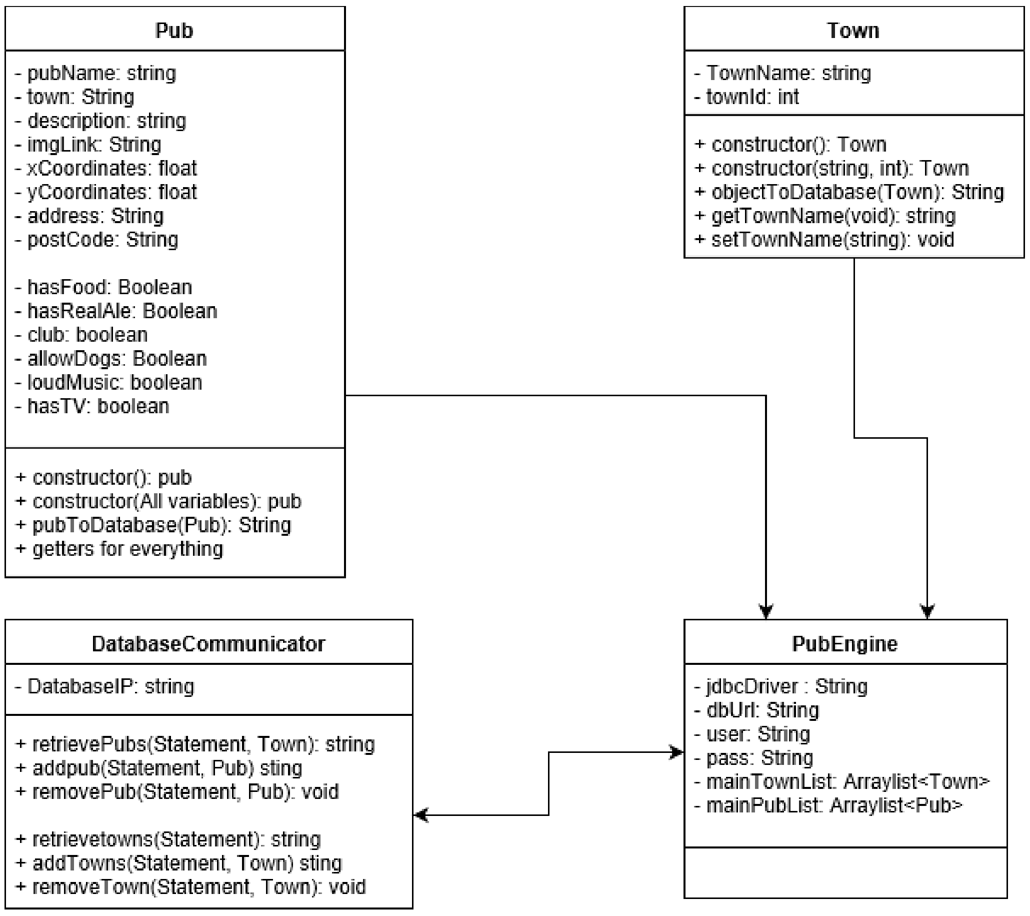
For communicating with the database, there will be established a new connection with the database every time the program is going to retrieve/update the database. This might reduce the speed of the retrieving/updating but will ensure a more robust connection. When it is creating the command that will be sent to the database will have the standard command for that function without of the information it going to retrieve/send (e.g. the name of the town where we are retrieving from or the pubs information to be added). The functions will add this information to the SQL command and send it off to the database. To See the information being stored in the database, see 3.2.

5.2.3 Creating Lists

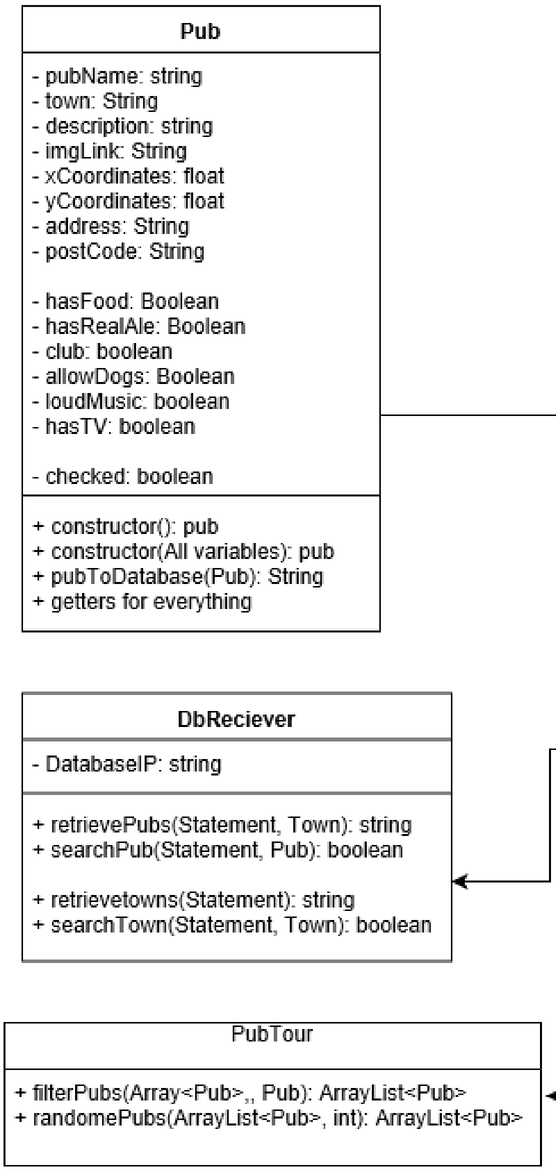
When creating a new list from the database it will extract every pub into an arraylist and pass that onwards to the PubEngine program as a master list of pubs from a specific town. When the user wants to manipulate this to another pub list it will extract all the wanted pubs, either filtering, randomising, or choosing the pubs to go into a new arraylist that will be used, but still saving the original list.

5.3 Significant data structures

5.3.1 Desktop UML diagram



5.3.2 Android program UML diagram



REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price. SE.QA.02. 2.1 Release
[2] Software Engineering Group Projects: Pub Tour Requirements Specification. C. J. Price. SE.QA.CSRS. 1.1 Release
[3] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy, B.P. SE.QA.03. 1.8 Release
[4] Software Engineering Group Projects: Design Specification Standards. C. J. Price. SE.QA.05. 2.1 Release
[5] Software Engineering Group Projects: Design Specification. C. J. Price. 1.01 Draft
[6] Software Engineering Group Projects: Java Coding Standard. C. J. Price, A. McManus. SE.QA.09. 2.0 Release

DOCUMENT HISTORY

Versio n	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	02/04/19	Original released document	RUR7
1.1	N/A	08/05/19	Improved diagrams and se	