Software Engineering Group 11
SE_11_DS
Design Specification

Author:	Tom Raikes (tor10), Alan Spence (als48), Aled Davies (add20), Theo Goree (tcg2), Richard Chowne (rhc15), Qiaoyang Zheng (qiz), Jack Skitt (jas78), Gavin Reynolds (gar18), Elliot Oram (elo9), Aloysius Fernandes (alf33), Kieran Dunbar (kid10)
Configuration Ref:	SE_11_DS
Date:	2014-12-05
Version:	1.0
Status:	Release

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Table of Contents

1. Introduction

1.1. Purpose of Document

Outline the design philosophy behind our application from a high level perspective down to the low level perspective.

1.2. Scope

This document aims to be a concise yet detailed overview of our proposed system.

1.3. Objectives

This document covers:

- A breakdown of the programs individual sections
- How each module communications with another
- Our proposed implementation of significant classes
- A detailed design of our system

2. Decomposition Description

2.1. Programs in System – Qiz

2.1.1. Activities

It displays the user interface for users and different interactive interfaces. They are Login, register, main menu and recording, etc. When it receive the message by user operation, it will send the message to the Data processing to check its validity. These functions will be implemented by the significant class. This package services requirements FR1, FR2, FR9, EIR1 and PR1.

2.1.2. Data processing

It is processing the data from the user operation such as if the login information and format of the message conform to the requirement. After this, it will send the valid data or information to the Database. This package services requirements FR5, FR6, FR7 and FR8.

2.1.3. Database

It is the space to save the data and the program to exchange the data and information with the Android applications and Websites. It divides into two parts: Local and Sever. And it provides these functions: connection and Sync. This package services requirements FR6 and FR7.

2.1.4. Data

It is the package of completed information about plants. It mainly includes recording, author and other information. It is the medium between the other different programs. This package services requirements FR4, FR5, FR6, FR7 and FR8.

2.1.5. Filter

It is a program that alters the frequency information and data passing through it. It mainly provide these functions: Alphabetic filter, Date filter, Species filter. It is used for other programs to select useful and necessary information and data for them. This package services requirement FR9.

2.1.6. Website

This Program will be the entire website and run all needed functions for the server and database. It will contain the needed pages to supply the ability to view records and create edit and delete reserves and alter user data.

This program will fulfil and help fulfil the following requirements FR7, FR8 and FR9. It also helps towards the requirements of EIR1,PR1 and PR2

2.2. Significant Classes in Each Program

2.2.1. Android Classes

2.2.1.1. Significant classes in the Screen Program

- Login – This allows the user to enter their username and password to login into the application
- Register – This will allow the user to create an account with them providing a username, password and email
- Main Menu – This is where once the user logins in they can upload, manage and create the plants on the app
- Recording – This allows the user to record and submit data for a plant such as Name and GPS location. It has an image of the plant and also allows the user to search the database

2.2.1.2. Significant classes in the Data Processing Program

- Format Checker – This will make sure that the user will enter the correct data such as the right format for an email address
- Login Checker – This will make sure that the users trying to login has a valid account to login

2.2.1.3. Significant classes in the Database Program

- Connection – This allow the application to connect to the database
- Server – This is where the database will be held
- Local – this will keep a local version of the database for use by the application
- Sync – This will make sure that the application and the database are synced together so that the information gets updated on the database on the server as well as the android application

2.2.1.4. Significant classes in the Data Program

- Recordings (Data Store) – This is class gets all the users information setting information such as internet access and gets all recordings from the queue
- Recording – This will check the status, author and any records for that author
- Record – This will create the records with all the needed fields
- Author – This will be where the name of user who recorded the plant will be

2.2.1.5. Significant classes in the Filter Program

- Alphabetic Filter – This will be a filter where it will sort the Plants in Alphabetic order
- Date Filter – This will be a filter where it will sort the Plants in the order of which they were recorded
- Species Filter – This will be a filter where it will sort the Plants in the order of Species

2.2.2. Web Classes

2.2.2.1. Significant classes in the Login Program

- JS validation – to client side validate the inputs in the form for email and password
- Open DB connection – open up the users table in the database
- Send data to DB ($_POST) – send the entered form inputs to the database
- Return Boolean true/false if the user exists – if the details entered matches an entry in the database return a welcome message if not display an error message
- Redirect – If details correct redirect to the You.php page

2.2.2.2. Significant classes in the Logout Program

- Session destroy – destroys the sessions started by logging in
- Redirect – once destroyed redirects you to the login (index) page

2.2.2.3. Significant classes in the Register Program

- JS validation – to client side validate the fields in the form
- Open DB connection – open up the users table in the database
- Send data to DB ($_POST) – send the entered form inputs to the database
- Return Boolean if email already exists – check if email exists if not continue if so error message
- Write and save data into user table – save the data entered into the appropriate fields in the user table
- Redirect – if accepted direct to the login page so user can log in

2.2.2.4. Significant classes in the You Program

- Open DB connection – open the user table
- Print registered details – Display the users details kept in the database
- Print last login time and date – display last time the user logged in
- Edit details – allow the user to edit all the details kept on the database
- JS Validate – to client side validate the form on the page
- Write and save details – save the updated details to the database

2.2.2.5. Significant classes in the View Program

- Open DB Connection– open the database pages needed – reserves, species occurrence, species
- Print all data – print out all data in a table
- Filter to each reserve – enable a drop down menu filter to see only one reserve at a time
- Order by Latin name alphabetical – the table will automatically be ordered alphabetically by Latin name
- Re order by date, recorder, abundance – filters using radio buttons to re order by date, recorder name or abundance

2.2.2.6. Significant classes in the New Program

- Open DB connection– open the reserve table
- Name, location (OS grid), text description form – a form to enter the needed information

- Delete – drop down menu of reserves to choose and delete from
- JS Validate – to client side validate the input from the form
- Send data to DB($_POST) – send the entered form inputs to the database
- Write to DB and save to reserve table – write the inserted data into the necessary fields and save the new or updated records
- Boolean true/false if any reserve with same name/ OS grid exists – if already exists throw error message if not show reserve created/updated/deleted message

2.2.2.7. Significant support classes

- Main.css – This will be the main Cascading Style Sheet for the design of the website.
- Validation.js – This will be the file with all the JavaScript validation functions which will apply to any form field entered by the user.
- Filterfunctions.php – this will be the file with the filtering functions in which will be called and used by the view.php file.

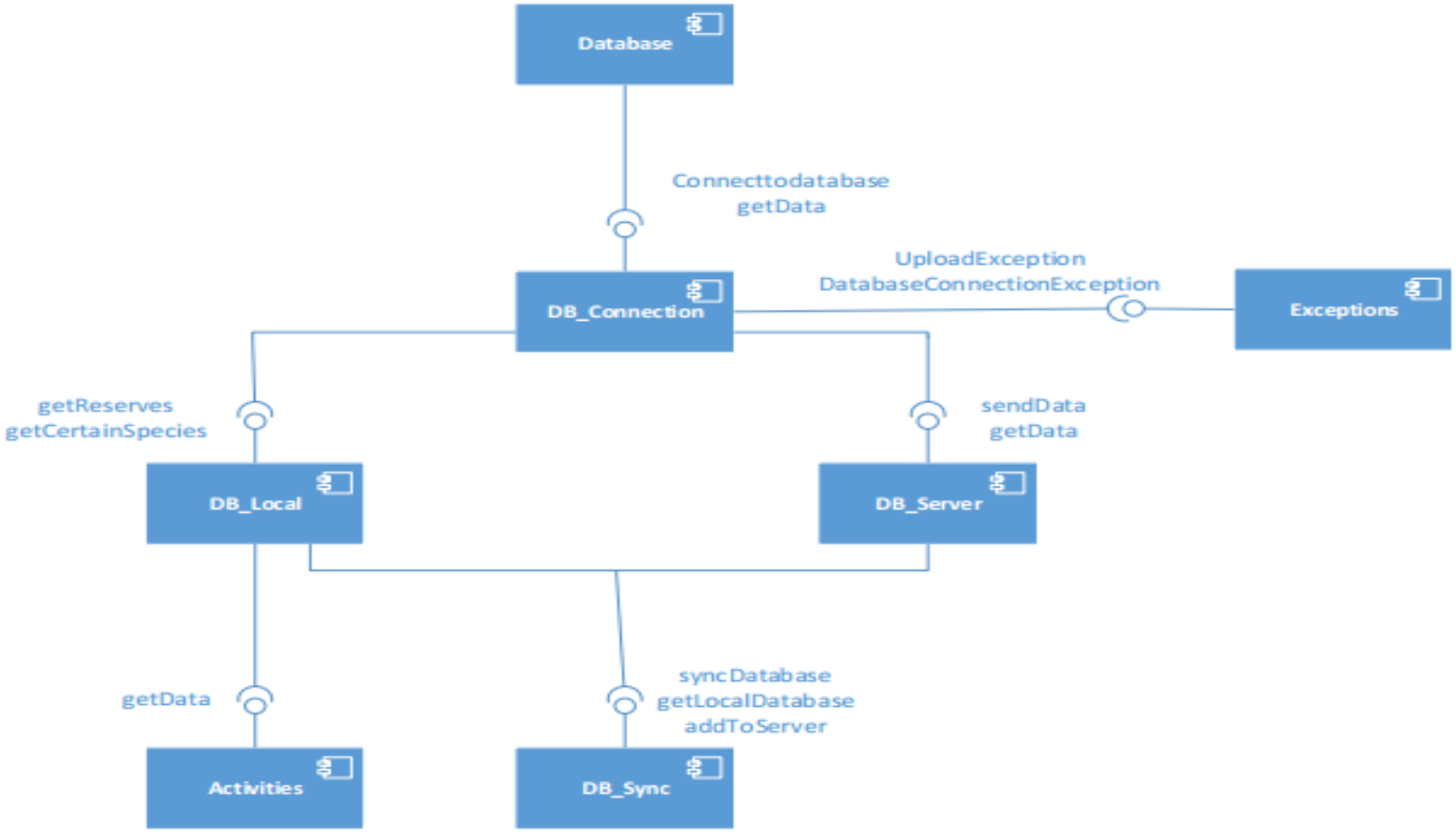
2.3. Mapping from Requirements to Classes

Requirements	Class Providing Requirements
FR1 and FR2	Recording, login Checker
FR3 and FR4	Record, Recording, Author
FR5	Record, Recording, Author
FR6	Connection, Server, Sync, local
FR7	Sync, Connection, Server, local
FR8	Open DB connection, Name, location (OS grid), Delete, Write to DB
FR9	Open DB Connection, Print all Data, Filter to reserve

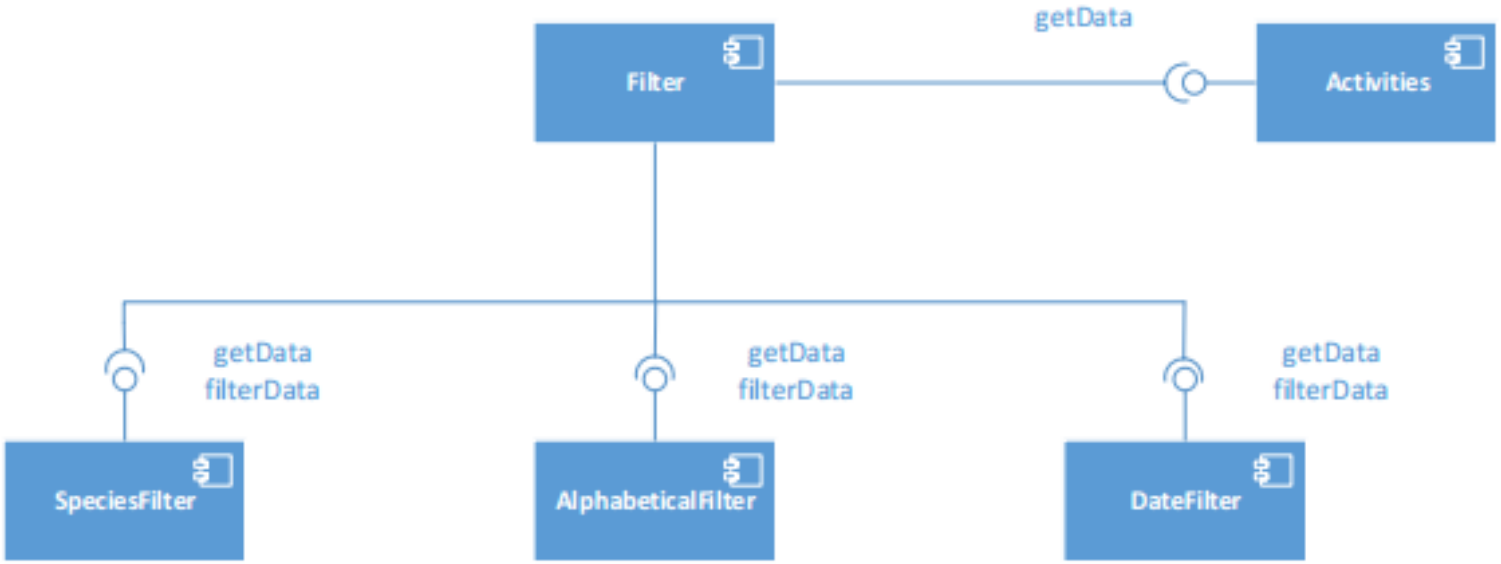
3. Dependency Description

3.1. Component Diagrams

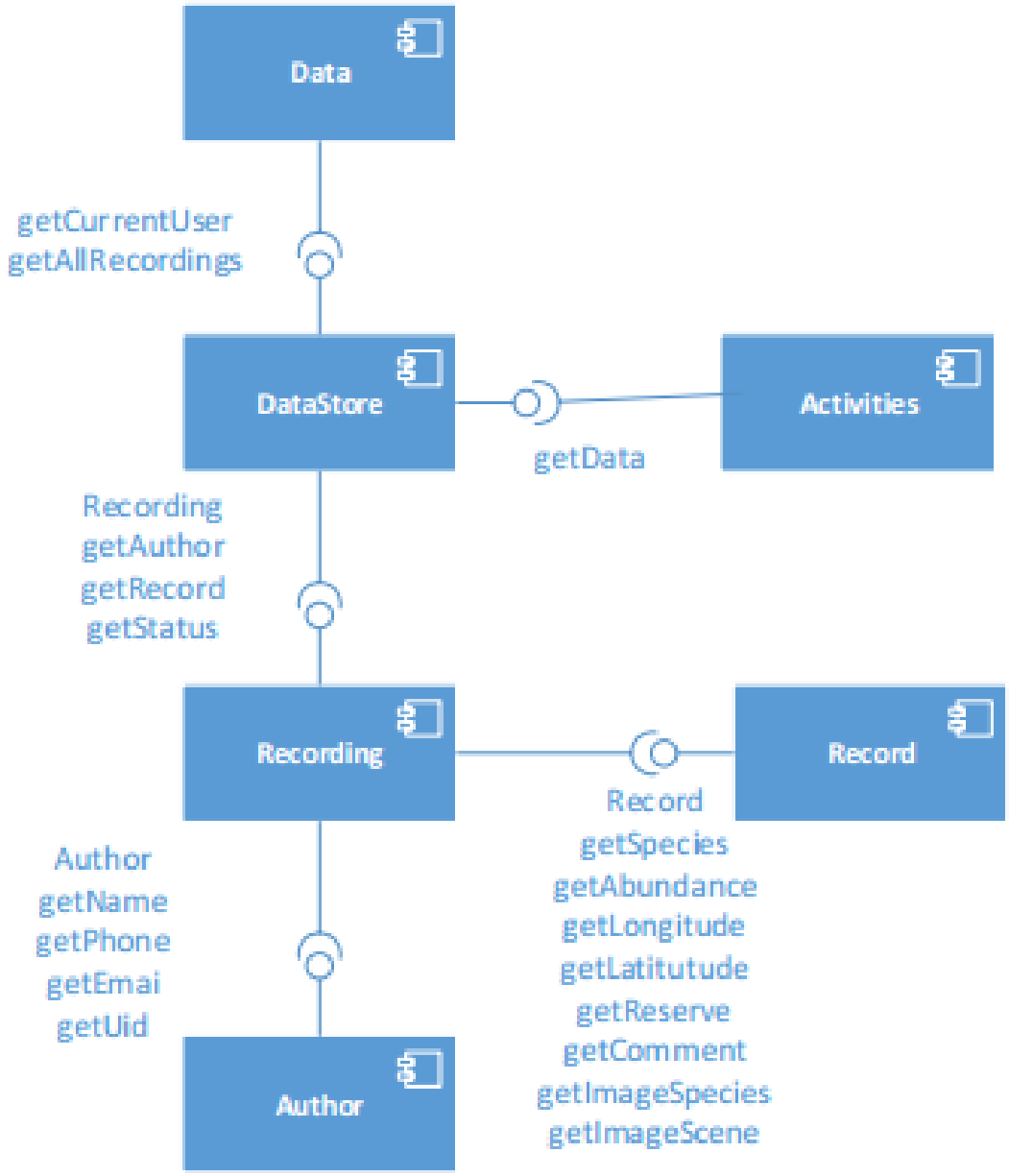
3.1.1. Database



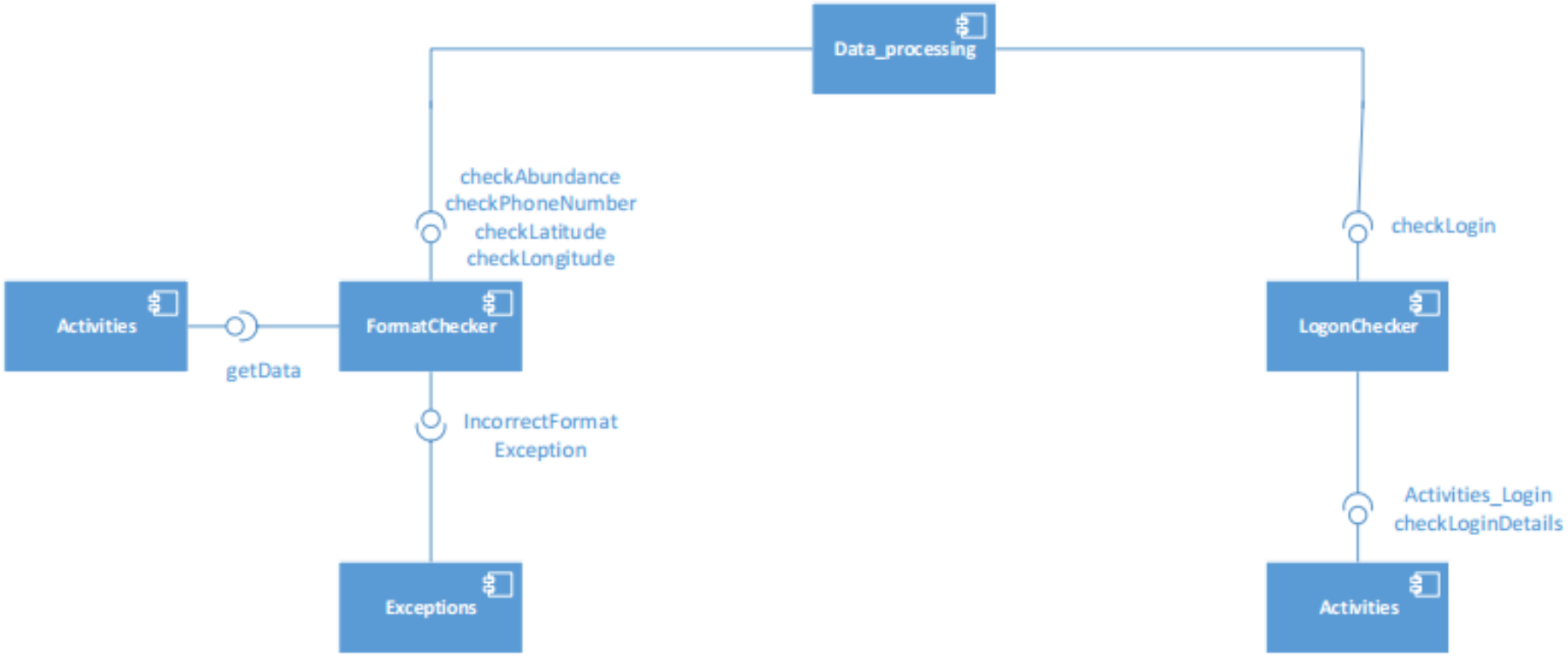
3.1.2. Filter


Filter
getData
Activities
getData
filterData
SpeciesFilter
getData
filterData
AlphabeticalFilter
getData
filterData
DateFilter

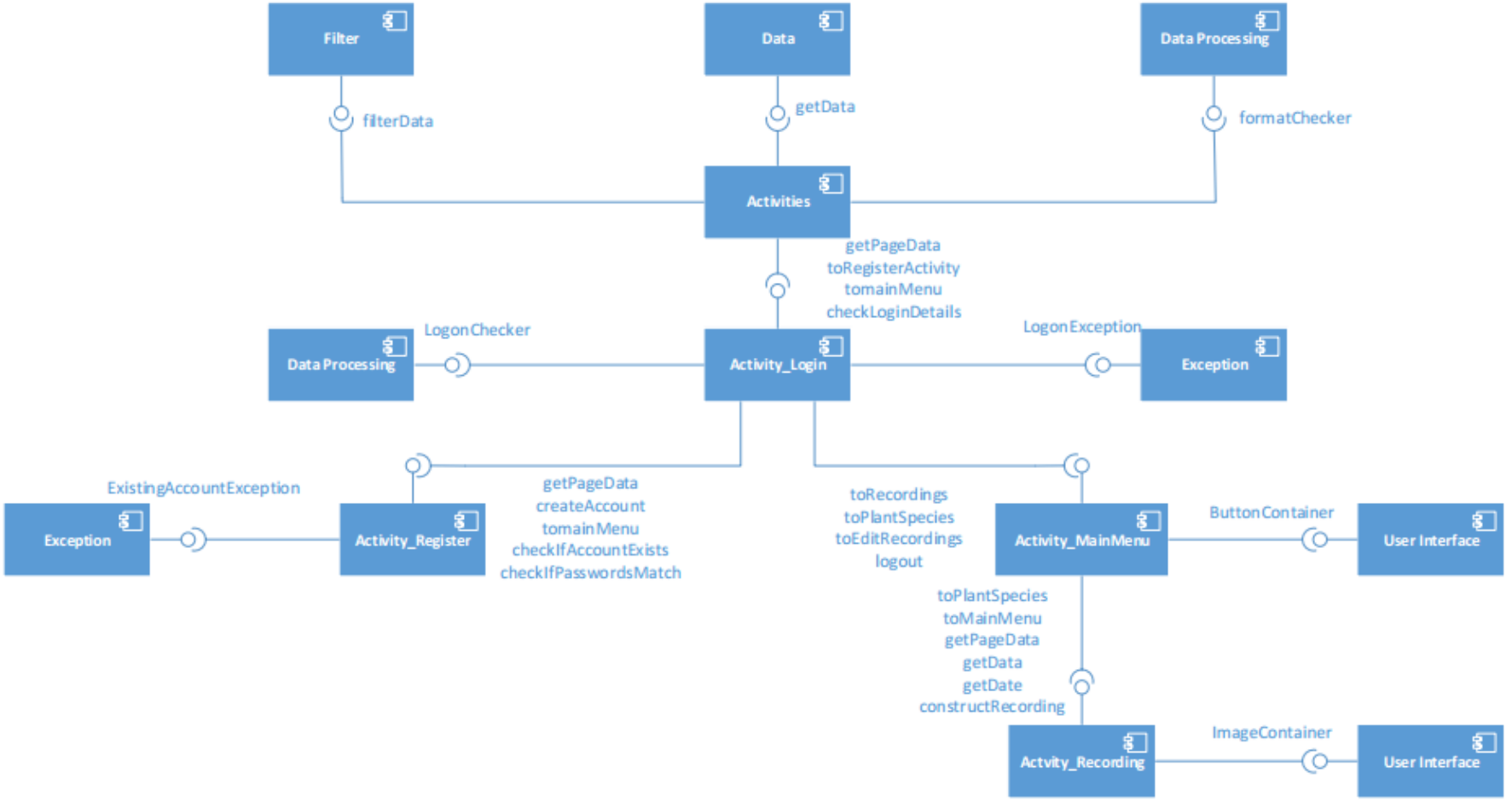
3.1.3. Data


Data
getCurrentUser
getAllRecordings
DataStore
Activities
getData
Recording
getAuthor
getRecord
getStatus
Recording
Record
Record
getSpecies
getAbundance
getLongitude
getLatitutude
getReserve
getComment
getImageSpecies
getImageScene
Author
getName
getPhone
getEmai
getUid
Author

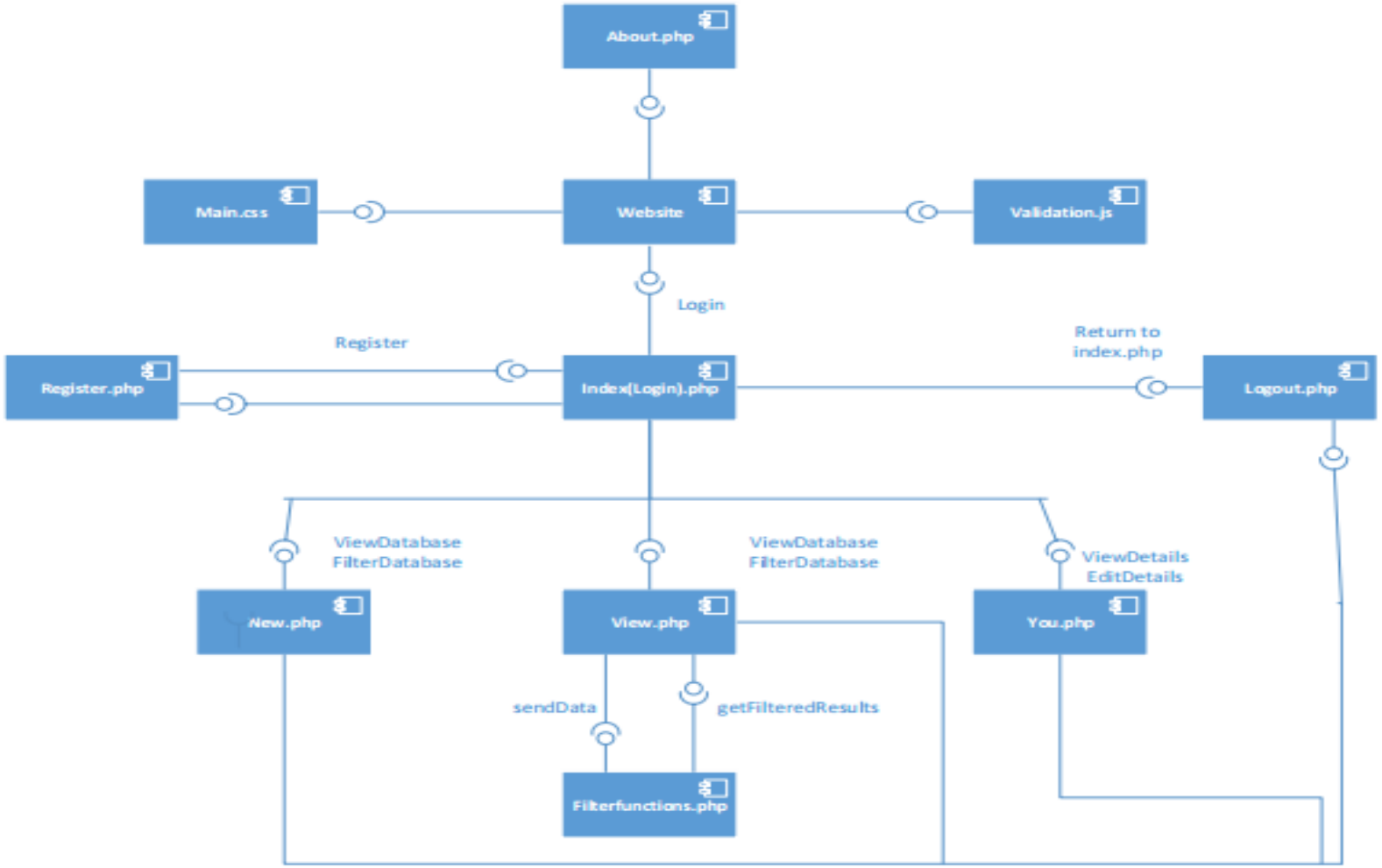
3.1.4. Data Processing



3.1.5. Activities



3.1.6. Website


About.php
Main.css
Website
Validation.js
Login
Register
Return to
index.php
Register.php
Index(Login).php
Logout.php
ViewDatabase
FilterDatabase
ViewDatabase
FilterDatabase
ViewDetails
EditDetails
New.php
View.php
You.php
sendData
getFilteredResults
Filterfunctions.php

4. Interface Description

4.1. Data Package Interface

```
package uk.ac.aber.cs221.group11.data;

import uk.ac.aber.cs211.group11.settings;

import uk.ac.aber.cs211.group11.ui;


/**

 * Outline for Datastore class this class holds the details of a session

 * i.e. It's user, settings, queue

 */

public class DataStore {


  private Author currentUser; //The person currently using the app, used to identify them.

  private RecordingQueue allRecordings; //Used to manipulate the recordings to be uploaded.

  private Settings currentSettings; //Holds the users settings, allows manipulation of these. And remembers them.


  /**

   * Returns the current user.

   *

   * @return Author the current user

   */

  public Author getCurrentUser();
```

```
/**
 * Returns the queue of recordings to be uploaded.
 *
 * @return RecordingQueue the queue of recordings to be uploaded
 */
public RecordingQueue getAllRecordings();

/**
 * Returns the store of users settings.
 *
 * @return Settings the store of the users settings
 */
public Settings getCurrentSettings();

/**
 * Overrides the current user with new one.
 *
 * @param a the new author to replace the existing one.
 */
public void setCurrentUser(Author a);

/**
 * Overrides the current queue of recordings and replaces it with another.
 *
 * @param rq the queue which will replace the existing one.
 */
public void setAllRecordings(RecordingQueue rq);
```

```
  /**
   * Overrides the current settings with new ones.
   *
   * @param s the new settings to replace the existing ones.
   */
  public void setCurrentSettings(Settings s);
}

/**
 * Outline for a class representing a single recording, not to be confused
 * with the record itself. This contains the records status and author.
 */
public class Recording {

  private Author author; //Represents the person responsible for the recording.
  private Record record; //A pointer to the object which represents a recording, this is where its data is (e.g. species, reserve etc.)
  private Status status; //A value determining whether the recording has been uploaded or is in a queue to be uploaded.

  /**
   * The constructor method.
   *
   * @param a The author responsible for the recording.
   * @param r The record associated with this recording.
   * @param s The status of the recording (in the queue or not).
   */
  public Recording(Author a, Record r, Status s);
```

```
/**
 * Returns the author of this recording.
 *
 * @return Author the author of the recording
 */
public Author getAuthor();

/**
 * Returns the record associated with this recording.
 *
 * @return Record the record associated with this recording
 */
public Record getRecord();

/**
 * Returns the status of the recording.
 *
 * @return Status the status of this recording
 */
public Status getStatus();

/**
 * Overrides the current author with new one.
 *
 * @param a the new author to replace the existing one.
 */
public void setAuthor(Author a);
```

```
  /**
   * Overrides the current record associated with this recording.
   *
   * @param r the record which will replace the existing one.
   */
  public void setRecord(Record r);

  /**
   * Overrides the current status with a new one.
   *
   * @param s the new status to replace the existing one.
   */
  public void setStatus(Status s);
}
```

```
/**
 * Outline for a class representing the data contained in a record.
 */
public class Record{

  private String species; //The name of the species of the specimen.
  private char abundance; //A character defining the species abundance (D,A,F,O,R).
  private int longitude; //Position of specimen east/west in degrees.
  private int latitude; //Position of specimen north/south in degrees.
  private Reserve reserve; //The reserve where the specimen was found.
  private String comment; //A comment about the specimen.
  private Image imageSpecies; //An image of the species.
  private Image imageScene; //An image of where the specimen was found.

  /**
   * The constructor method.
   *
   * @param species The name of the species of the specimen.
   * @param abundance A character defining the species abundance (D,A,F,O,R).
   * @param long Position of specimen east/west in degrees.
   * @param lat Position of specimen north/south in degrees.
   * @param iSpecies An image of the species.
   * @param iScene An image of where the species was found.
   */
  public Record(String species, char abundance, int long, int lat, Image iSpecies, Image
iScene);
```

```
/**
 * Returns the name of the species.
 *
 * @return String species name of the specimen
 */
public String getSpecies();

/**
 * Returns the abundance of the specimen.
 * D - "Dominant"
 * A - "Abundant"
 * F - "Frequent"
 * O - "Occasional"
 * R - "Rare"
 *
 * @return Char the abundance of the species
 */
public char getAbundance();

/**
 * Returns the longitudinal position of the specimen.
 *
 * @return int the longitudinal position of the specimen
 */
public int getLongitude();
```

```
/**
 * Returns the latitudinal position of the specimen.
 *
 * @return int the latitudinal position of the specimen
 */
public int getLatitude();

/**
 * Returns the reserve where the specimen was found.
 *
 * @return String the reserve where the specimen was found
 */
public String getReserve();

/**
 * Returns the comment associated with the record.
 *
 * @return String a comment about the record
 */
public String getComment();

/**
 * Returns the image of the species.
 *
 * @return Image an image of the species
 */
public Image getImageSpecies();
```

```
/**
 * Returns the image of the scene.
 *
 * @return Image an image of the scene
 */
public Image getImageScene();

/**
 * Overrides the current species with a new one.
 *
 * @param s The new species to replace the existing one.
 */
public void setSpecies(String s);

/**
 * Overrides the current abundance with a new one.
 *
 * @param a The new abundance to replace the existing one.
 */
public void setAbundance(char a);

/**
 * Overrides the current longitude with a new one.
 *
 * @param long The new longitude to replace the existing one.
 */
public void setLongitude(int long);
```

```
/**
 * Overrides the current latitude with a new one.
 *
 * @param lat The new latitude to replace the existing one.
 */
public void setLatitude(int lat);

/**
 * Overrides the current reserve with a new one.
 *
 * @param r The new reserve to replace the existing one.
 */
public void setReserve(Reserve r);

/**
 * Overrides the current comment with a new one.
 *
 * @param c The new comment to replace the existing one.
 */
public void setComment(String c);

/**
 * Overrides the current species image with a new one.
 *
 * @param iSpecies The new species image to replace the existing one.
 */
public void setImageSpecies(Image iSpecies);
```

```
  /**
   * Overrides the current scene image with a new one.
   *
   * @param iScene The new scene image to replace the existing one.
   */
  public void setImageScene(Image iScene);

}

/**
 * Outline for a class representing an Author of a recording.
 */
public class Author{

  private String name; //The name of the author
  private int phone; //The authors telephone number
  private String email; //The authors email address
  private String uid; //The authors user ID

  /**
   * The constructor method.
   *
   * @param name The authors name
   * @param phone The authors telephone number
   * @param email The authors email address
   */
  public Author(String name, int phone, String email);
```

```
/**
 * Returns the name of the author.
 *
 * @return String name of the author
 */
public String getName();

/**
 * Returns the phone number of the author.
 *
 * @return int the authors phone number
 */
public int getPhone();

/**
 * Returns the email address of the author.
 *
 * @return String the authors email address
 */
public String getEmail();

/**
 * Returns the user ID of the author.
 *
 * @return String the authors User ID
 */
public String getUid();
```

```
  /**
   * Overrides the current authors name with a new one.
   *
   * @param n The new name to replace the existing one.
   */
  public void setName(String n);

  /**
   * Overrides the current authors phone number with a new one.
   *
   * @param p The new number to replace the existing one.
   */
  public void  setPhone(int p);

  /**
   * Overrides the current authors email address with a new one.
   *
   * @param e The new email address to replace the existing one.
   */
  public void setEmail(String e);

  /**
   * Overrides the current authors UID with a new one.
   *
   * @param u The new user ID to replace the existing one.
   */
  public void setUid(String u);

}
```

4.2 Filter Package Interface

```
package uk.ac.aber.cs221.group11.filter;

/**
 * Outline for Filter.
 */

public class AlphabeticFilter {

	private Object[] data; //Holds the data to be filtered

	/**
	 * Returns the data held by the filter
	 *
	 * @return Object[] filtered data
	 */
	public Object[] getData();

	/**
	 * Filters the data given to it alphabetically
	 *
	 * @param dataToFilter The provided data which is to be filtered
	 * @return Object[] The now filtered data
	 */
	public Object[] filterData(Object[] dataToFilter);

}
```

```
public class DateFilter {

	private Object[] data; //Holds the data to be filtered

	/**
	 * Returns the data held by the filter
	 *
	 * @return Object[] filtered data
	 */
	public Object[] getData();

	/**
	 * Filters the data given to it by date
	 *
	 * @param dataToFilter The provided data which is to be filtered
	 * @return Object[] The now filtered data
	 */
	public Object[] filterData(Object[] dataToFilter);

}

public class SpeciesFilter {

	private Object[] data; //Holds the data to be filtered

	/**
	 * Returns the data held by the filter
	 *
	 * @return Object[] filtered data
	 */
	public Object[] getData();
```

```
        /**
         * Filters the data given to it by species
         *
         * @param dataToFilter The provided data which is to be filtered
         * @return Object[] The now filtered data
         */
        public Object[] filterData(Object[] dataToFilter);

}
```

4.3 Data Processing Package

```
package uk.ac.aber.cs221.group11.data_processing;

import uk.ac.aber.cs211.group11.activities.Activity_Login;


/**
 * Outline for Data Processing.
 */


public class FormatChecker {


    /**
     * Check that abundance representation has been entered.
     * D - "Dominant"
     * A - "Abundant"
     * F - "Frequent"
     * O - "Occasional"
     * R - "Rare"
     *
     * @param abundance The users input to be checked against valid char values in
validAbundance[]
     * @return char Relevant abundance character
     */
    public char checkAbundance(String abundance);


    /**
     * Check that phone number is the length of validPhoneLength variable.
     *
     * @param number The users input to be checked against validPhoneLength
     * @return int The users phone number in int format
     */
    public int checkPhoneNumber(String number);
```

```
    /**
     * Checks to ensure that latitude entered is of correct length.
     *
     * @param latitude The users input to be checked against the latitudes Max and Min
values
     * @return int The latitude the user checked
     */
    public int checkLatitude(int latitude);


    /**
     * Checks to ensure that longitude entered is of correct length.
     *
     * @param longitude The users input to be checked against the longitudes Max and Min
values
     * @return int The longitude the user checked
     */
    public int CheckLongitude(int longitude);


}
```

```
public class LogonChecker{

    /**
     * Check to see if the user has entered a valid user ID and password combination.
     *
     * @param uid The users UID input to be checked
     * @param password The users password input to be checked
     */
    public boolean checkLogin(String uid, String password);

}

public class ConnectivityChecker{

    /**
     * Checks if user has WiFi on.
     *
     * @return true if WiFI is on, false if WiFi is off
     */
    public boolean checkWifi();

    /**
     * Checks if user has GPS enabled.
     *
     * @return true if GPS is on, false if GPS is off
     */
    public boolean checkGPS();

}
```

4.4 Database Package

```
package uk.ac.aber.cs221.group11.db;

/**
 * Outline for the Database Connection Class
 */
public class DB_Connection {
	private String sql_query; //Holds an SQL query to be sent to the database

	/**
	 * Method to connect to the database
	 *
	 * @param name The name of the database
	 * @param location The location or address of the database
	 * @param port The port to connect to
	 */
	public static void connectToDatabase(String name, String location, String port);

	/**
	 * Method to retrieve data from the database
	 *
	 * @return An array of objects from the database
	 */
	public static Object[] getData();
}
```

```
/**
 * Outline for the Local Database Class
 */
public class DB_Local {
	private Object[] species; //List of local species
	private Reserves[] reserves; //List of local reserves

	/**
	 * Method to get the Reserves from the Local DB
	 *
	 * @param sqlStatement The SQL statement to execute
	 * @return An array of reserves
	 */
	public static Object[] getReserves(String sqlStatement);

	/**
	 * Method to get certain Species from the Local DB
	 *
	 * @param sqlStatement The SQL statement to execute
	 * @return An array of species
	 */
	public static Object[] getCertainSpecies(String sqlStatement);
}
```

```
/**
 * Outline for the Database Server Class
 */
public class DB_Server {
	/**
	 * Method to send data to the DB server
	 *
	 * @param sqlStatement The SQL statement to execute
	 */
	public static void sendData(String sqlStatement);

	/**
	 * Method to get data from the DB server
	 *
	 * @param sqlStatement The SQL statement to execute
	 */
	public static void getData(String sqlStatement);
}
```

```
/**
 * Outline for the Database Synchronisation Class
 */
public class DB_Sync {
	/**
	 * Method to synchronise the local and remote databases
	 *
	 * @param databaseName The database to synchronise with
	 * @return boolean indicating whether the synchronisation was successful
	 */
	public static boolean syncDatabase(String databaseName);

	/**
	 * Method to get the contents of the local database
	 *
	 * @return An array of objects with the contents of the local database
	 */
	public static Object[] getLocalDatabase();

	/**
	 * Method to add local data to the server
	 *
	 * @param localData The data to add to the server
	 */
	public static void addToServer(Object[] localData);
}
```

4.5 Activities Package

```
package uk.ac.aber.cs221.group11.activities;
import uk.ac.aber.cs211.group11.data;

/**
 * Outline for the class responsible for the login page
 * /
public class Activity_Login {
        private TextField username; //Takes username input
        private TextField password; //Takes password input
        private Button login; //Submits user data to login
        private Button register; //Navigates user to registration activity
        private Button workOffline; //Work without internet connection

        /**
         * Takes the user to the Register screen
         * @param v the current view
         */
        public void toRegisterActivity(View v);

        /**
         * Takes the user to the main menu screen
         * @param v the current view
         */
        public void toMainMenu(View v);

        /**
         * Gets the data from form input on the page
         * @param v the current view
         */
        public void getPageData(View v);
```

```
        /**
         * Checks the login details against those already in the database
         * @return true if login details are correct, false if they are not
         */
        public boolean checkLoginDetails();
}
```

```
/**
 * Outline for the class responsible for the registration screen
 */
public class Activity_Register {
        private Textfield username; //Takes username input
        private TextField password; //Takes password input
        private TextField confirmPassword; //A second password input
        private Button createAccount; //Submits data from user input

        /**
         * Takes the user to the createAccount screen
         * @param v the current view
         */
        public void createAccount(View v);

        /**
         * Takes the user to the MainMenu screen
         * @param v the current view
         */
        public void toMainMenu(View v);

        /**
         * Gets the data from form input on the page
         * @param v the current view
         */
        public void getPageData(View v);

        /**
         * Checks if the account already exists in the system
         * @return true if the account exists, false if not
         */
        public boolean checkIfAccountExists();
```

```
        /**
         * Checks if the password and re-enter password fields match
         * @return true if the passwords match, false if not
         */
        public boolean checkIfPasswordsMatch();

}
```

```
/**
 * Outline for class responsible for the main menu
 */
public class Activity_MainMenu {
	private ButtonContainer smallButtons; //Holds small buttons on activity
	private ButtonContainer largeButtons; //Holds large buttons on activity
	private RecordingsQueue recordingQueue; //Recordings to be uploaded

	/**
	 * Takes the user to the Recordings screen
	 * @param v the current view
	 */
	public void toRecordings(View v);

	/**
	 * Takes the user to the PlantSpecies screen
	 * @param v the current view
	 */
	public void toPlantSpecies(View v);

	/**
	 * Takes the user to the EditRecordings screen
	 * @param v the current view
	 */
	public void toEditRecordings(View v);

	/**
	 * Takes the user to the logout screen
	 * @param v the current view
	 */
	public void logout(View v);
}
```

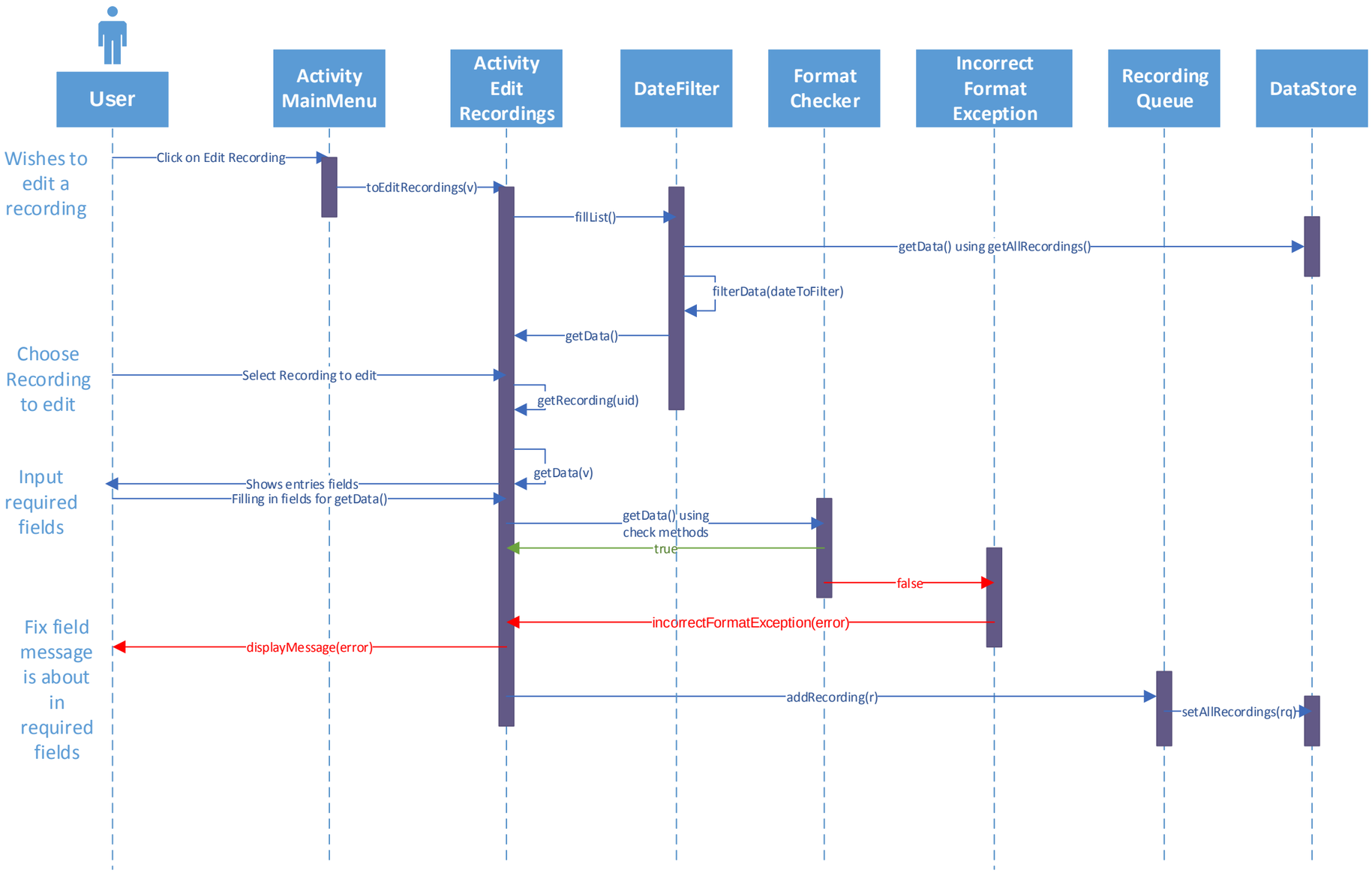
```
/**
 * Outline for class responsible for viewing recordings
 */
public class Activity_Recordings {
        private ImageContainer imageContainer; //Displays image of recording
        private TextField name; //Recording name
        private TextField reserve; //Recording reserve
        private TextField plant; //Plant associated with recording
        private Textfield longitude; //Recording longitude
        private Textfield latitude; //Recording latitude
        private TextArea notes; //Notes on recording

        /**
         * Takes the user to the PlantSpecies screen
         * @param v the current view
         */
        public void toPlantSpecies(View v);

        /**
         * Takes the user to the MainMenu screen
         * @param v the current view
         */
        public void toMainMenu(View v);

        /**
         * Gets the data from form input on the page
         * @param v the current view
         */
        public void getPageData(View v);
```

```
        /**
         * Takes data from the current view
         * @param v the current view
         */
        public void getData(View v);

        /**
         * Gets the current date to be added to the database
         */
        public void getDate();

        /**
         * Sets up the system ready to take a recording
         * @return the recording
         */
        public Recording constructRecordings();

}
```

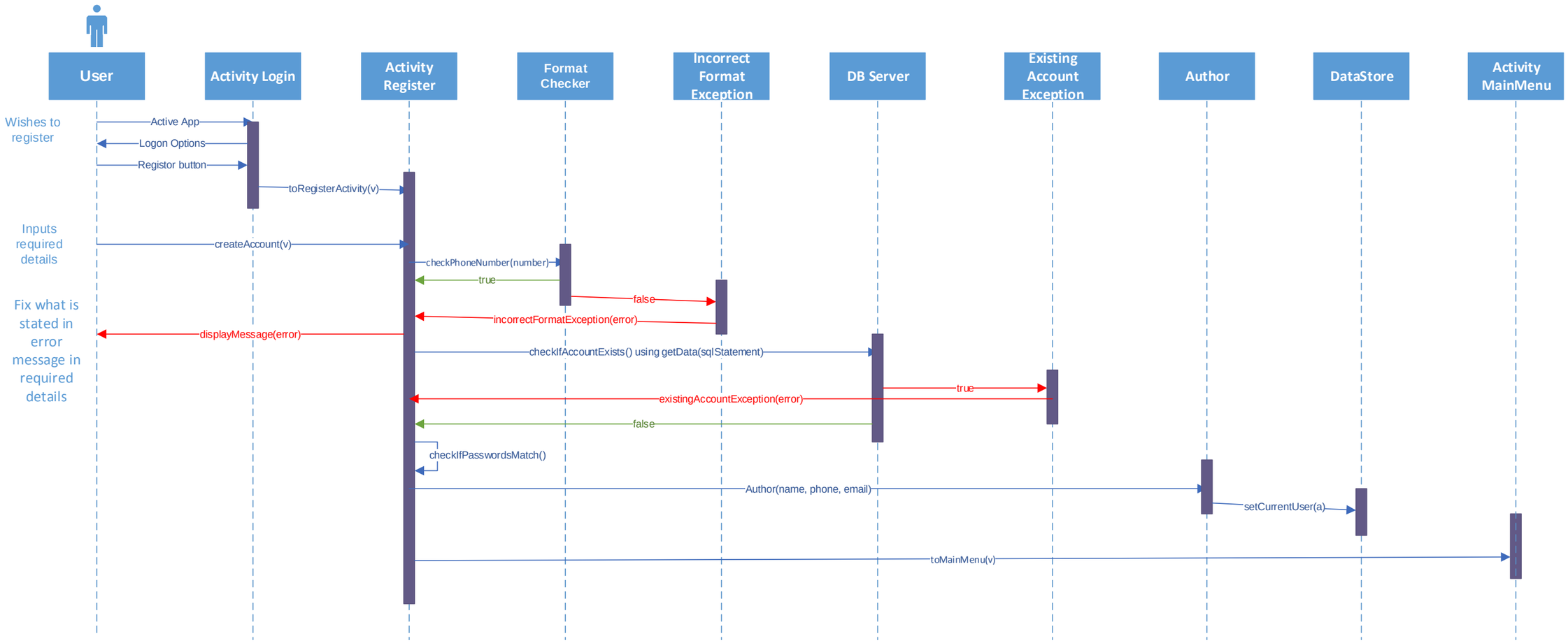
5. Detailed Design

5.1. Sequence Diagrams

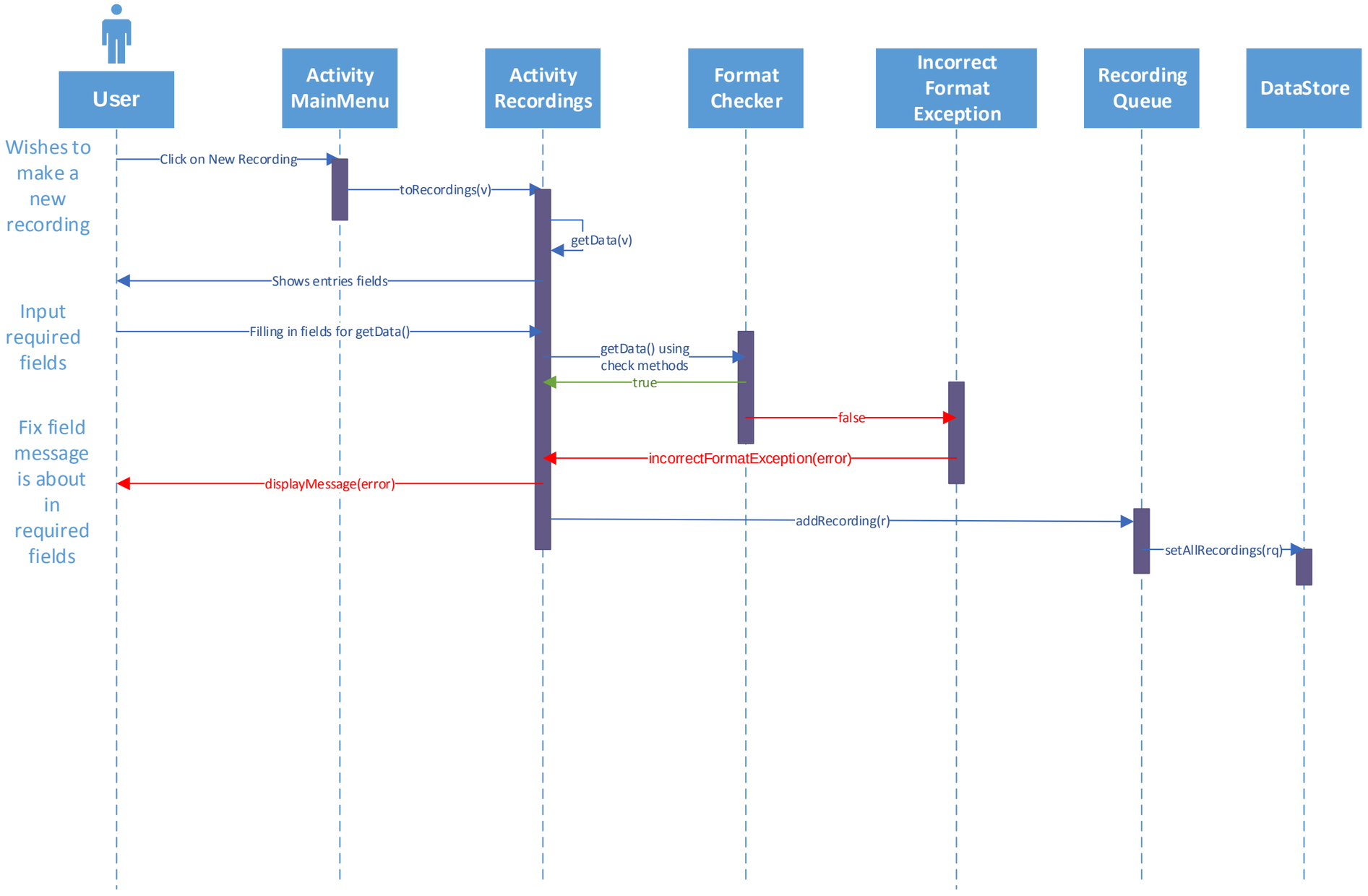
5.1.1. Logon Sequence Diagram



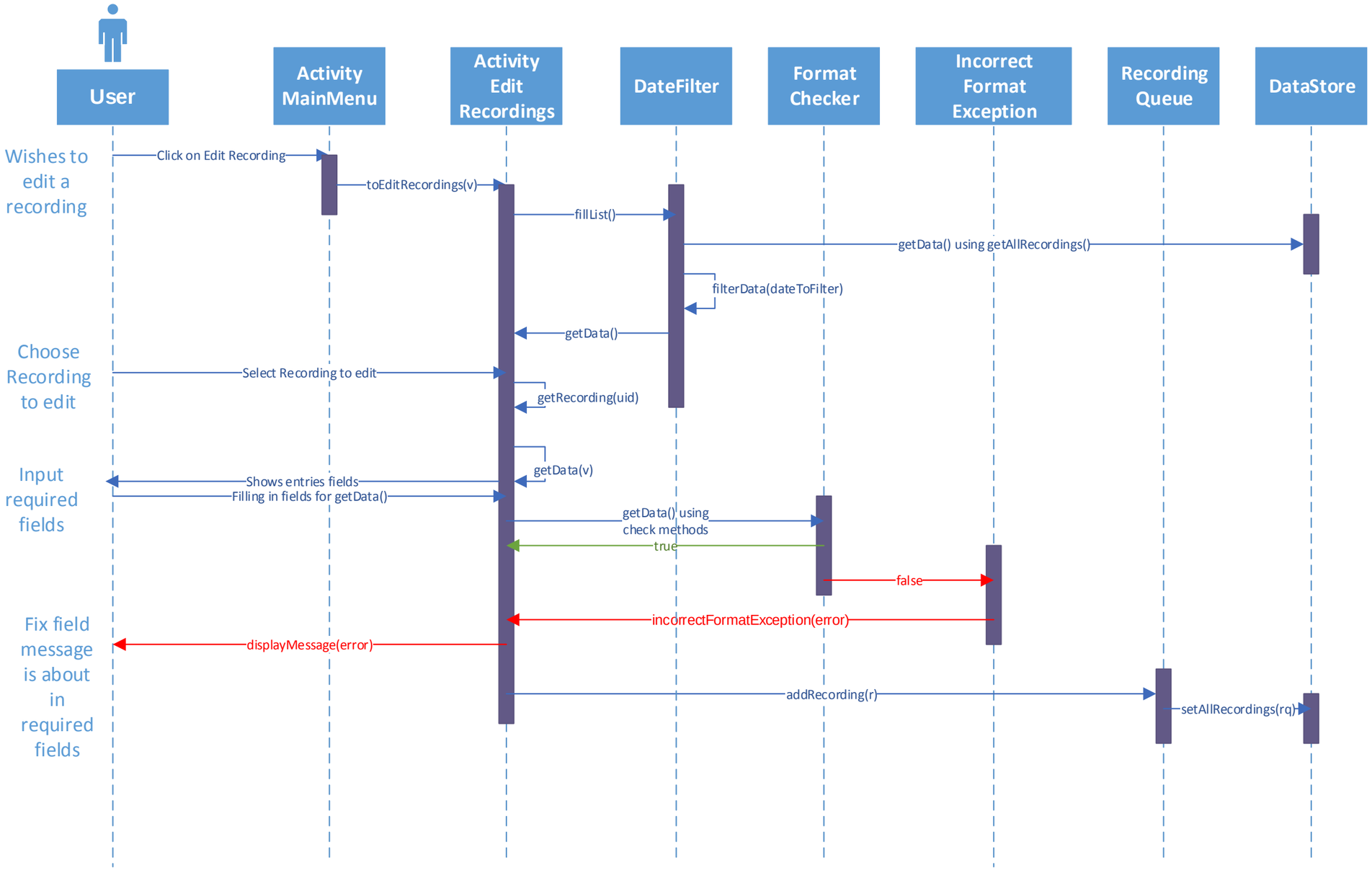
5.1.2. Register Sequence Diagram



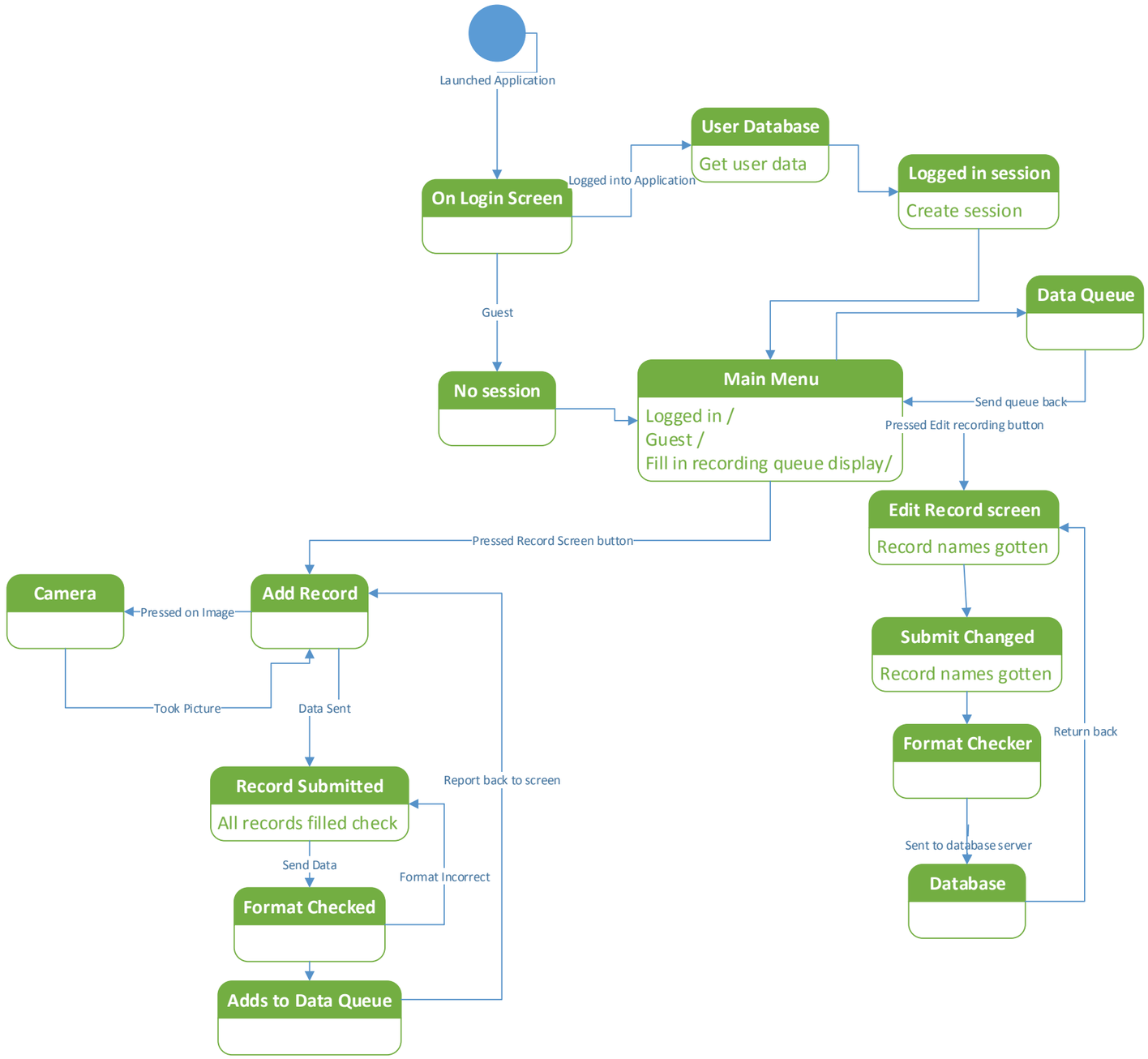
5.1.3. Adding Record Sequence Diagram



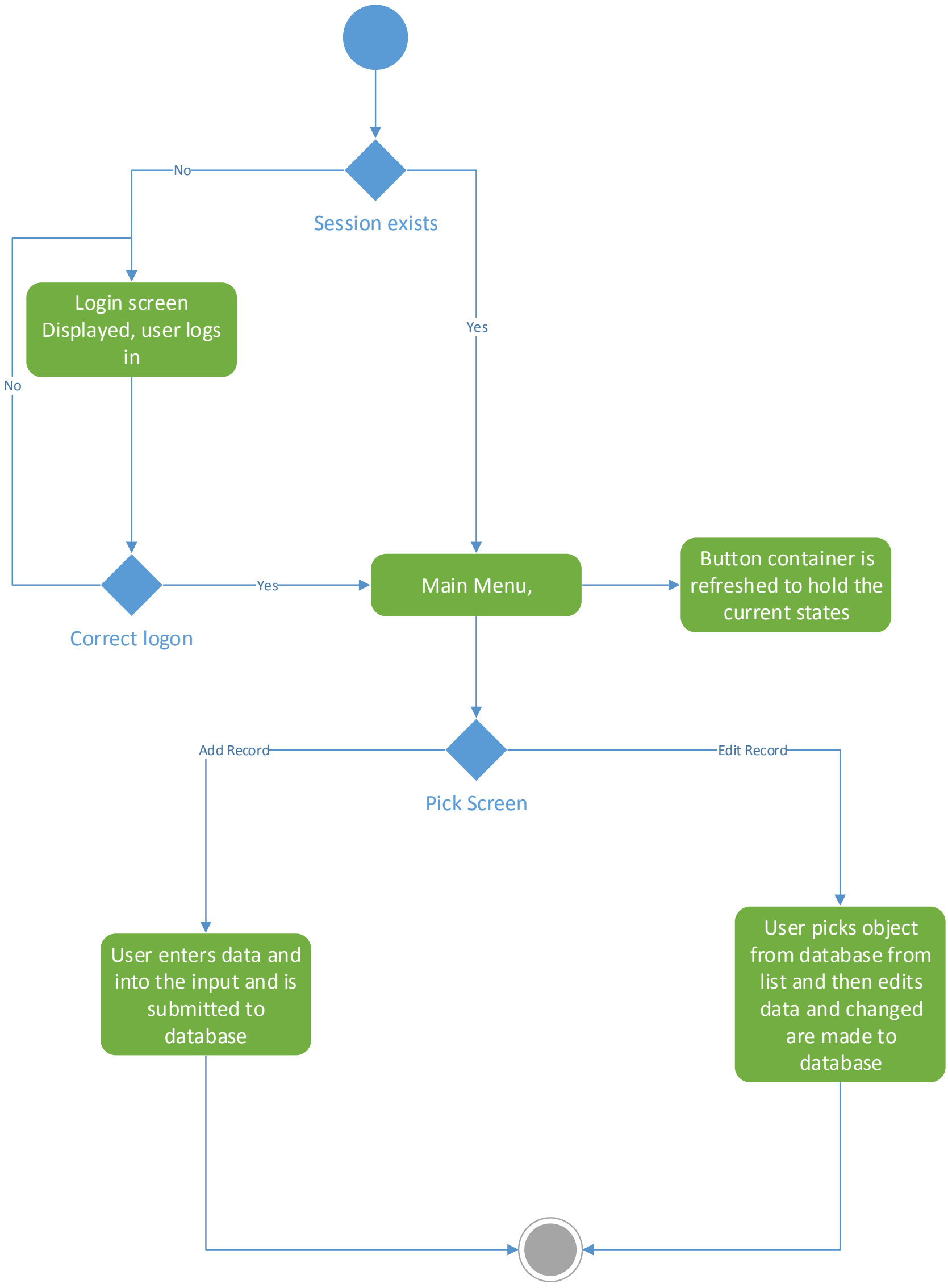
5.1.4. Updating Record Sequence Diagram



5.1.5. State Diagram



5.1.6. Activity Diagram



5.2. Significant Algorithms –

5.2.1. Filters

The filters work by restricting data to the search term , this can be useful to allow the user to optimise their search.

```
LinkedList<Records> dataCollected
LinkedList<Records> output
int matchScore = 0


foreach entry in dataCollected
        if entry matches filter condition
            add to output
        else
            ignore the data


For date filter
Date dateToFilter
Date range
   foreach entry in dataCollected
   get entry date and
        if entry date is in the range of dateToFilter+-range
            add to output
        else
            ignore the data
```

The search algorithm gets the data from the filter algorithm, it then compares the hamming distance of the binary input and search term to give a closest match the closest 10 are then stored in an array to be printed out at the end of the search. For example, if the user was searching for “Rose” but entered the search term “Ros” there would still be a result returned. This means there will almost always be a result.

```
int maximumHammingDistance
Record[] databaseRecord;
Record[] results;
String searchTerm
foreach record x in databaseRecord
   get hamming distance between x and searchTerm
   if x hamming distance is less than maximumHammingDistance
   foreach record y in results
      if x distance is less than y distance
```

```
        continue
    else if x distance is greater than or equal to y distance
        insert x behind y
else
continue
return results
```

5.2.2. ADT Pseudo code

Recording queue

int number of queue elements
int pointer to the front of the queue
int pointer to the back of the queue

add item to queue
    insert item at the back of the queue
    update pointer to the back of the queue using circular update method
    update number of elements in the queue

remove item from the queue
    take item off the head of the queue
    update pointer to the front of the queue using circular update method
    update number of elements in the queue

examine front
    return the element at the head of the queue

clear
    for each element in the queue
        make current element equal to null
    set number of elements to 0

length

return the number of elements in the queue

isEmpty

if the number of elements in the queue is 0

return true

else

return false

circular update

if a pointer is at the end of an array and the array NOT full

set the pointer to the beginning of the array (position 0)

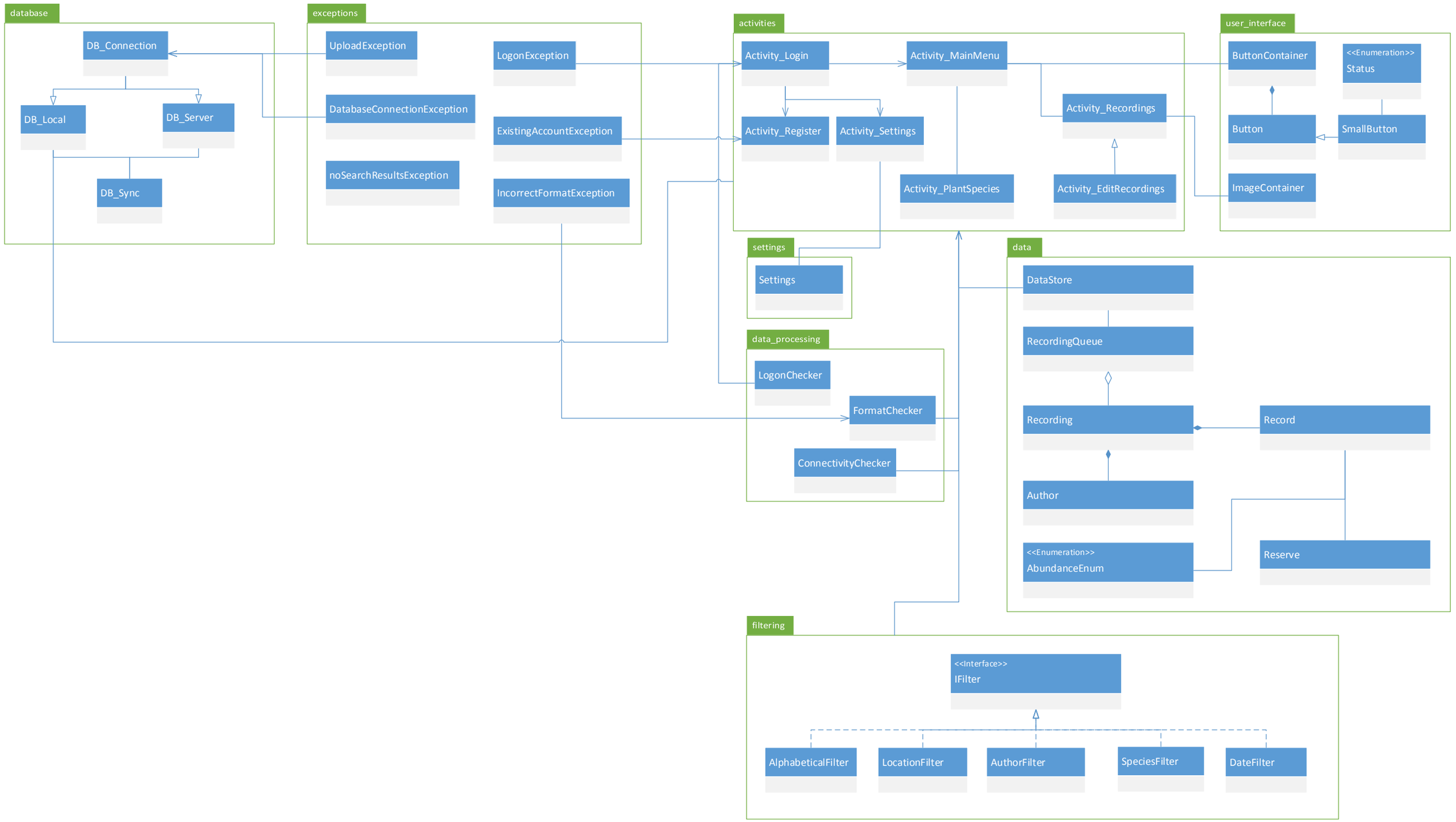
else

increment pointer

return pointer

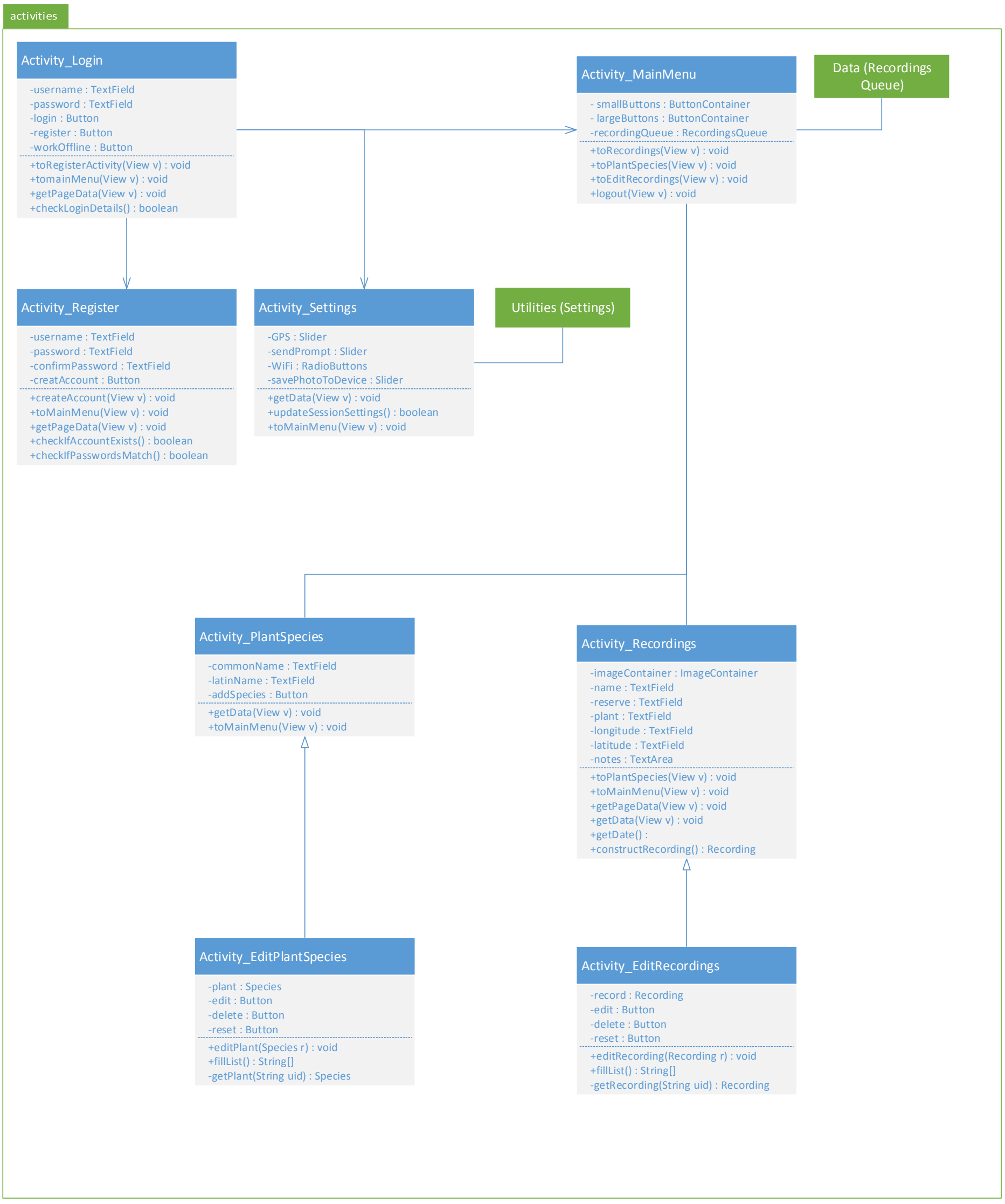
5.3. Significant Data Structure

5.3.1. Object Diagram

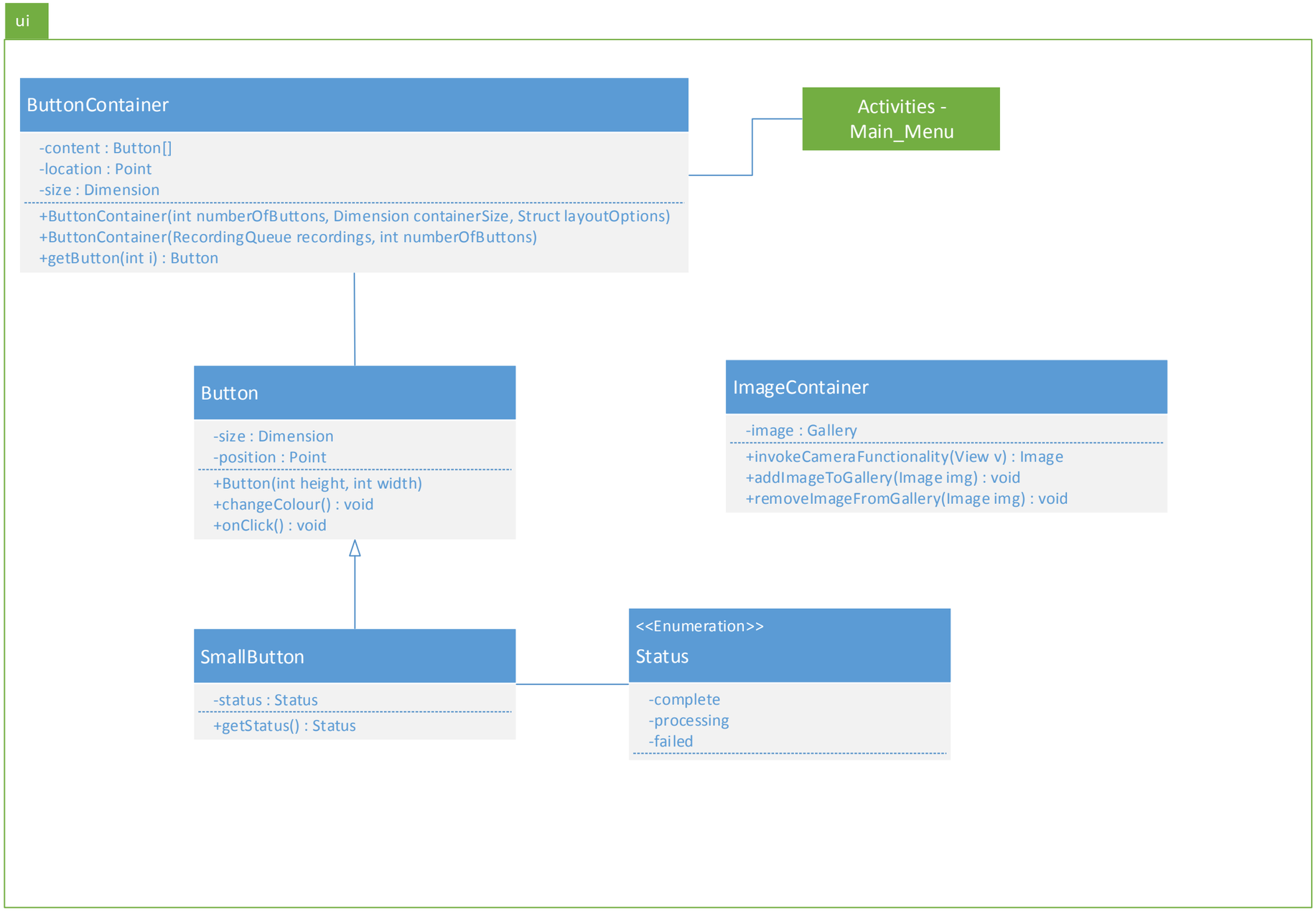


5.3.2. Class Diagrams

5.3.2.1. Activities Class Diagram

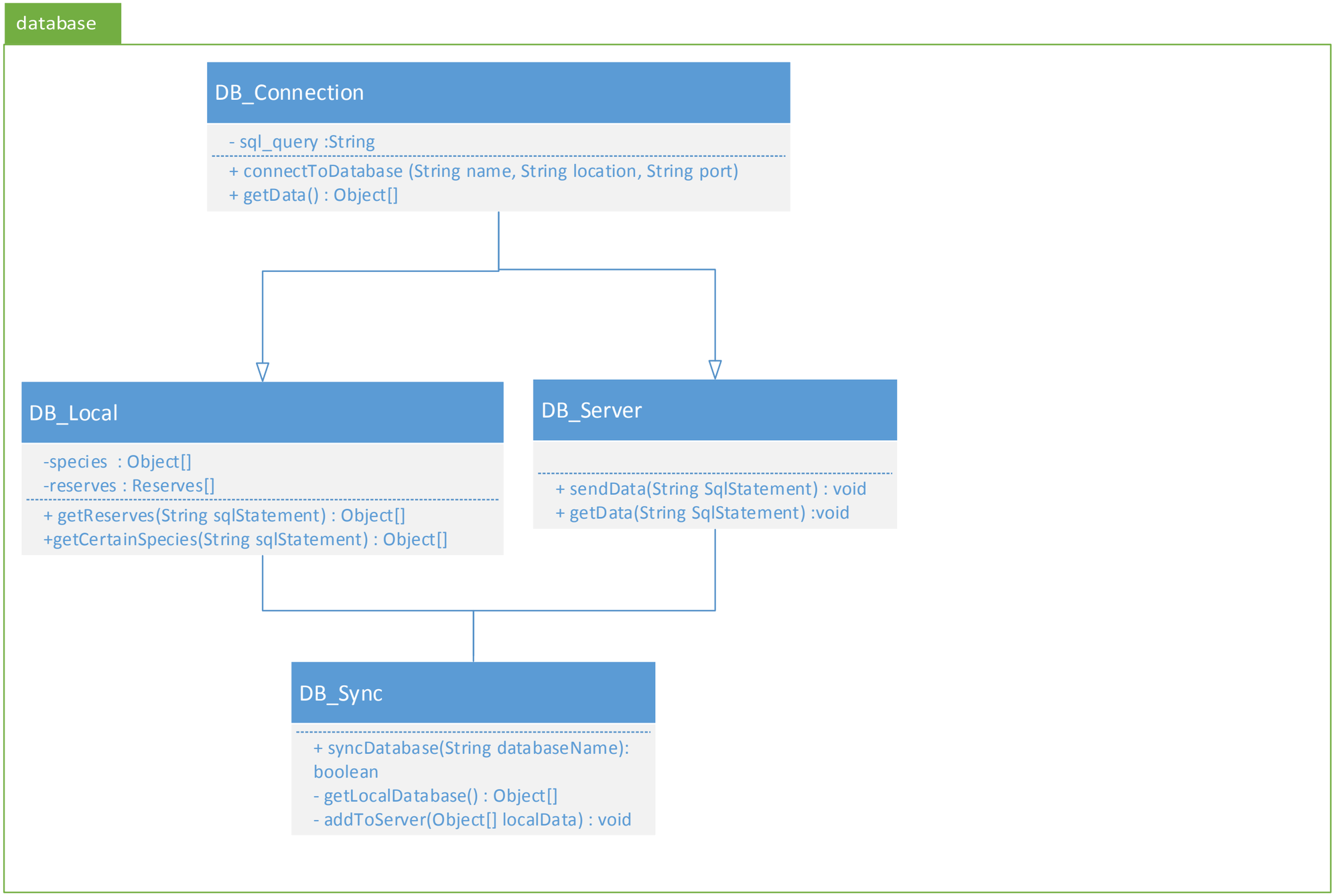


5.3.2.2. UI Class Diagram

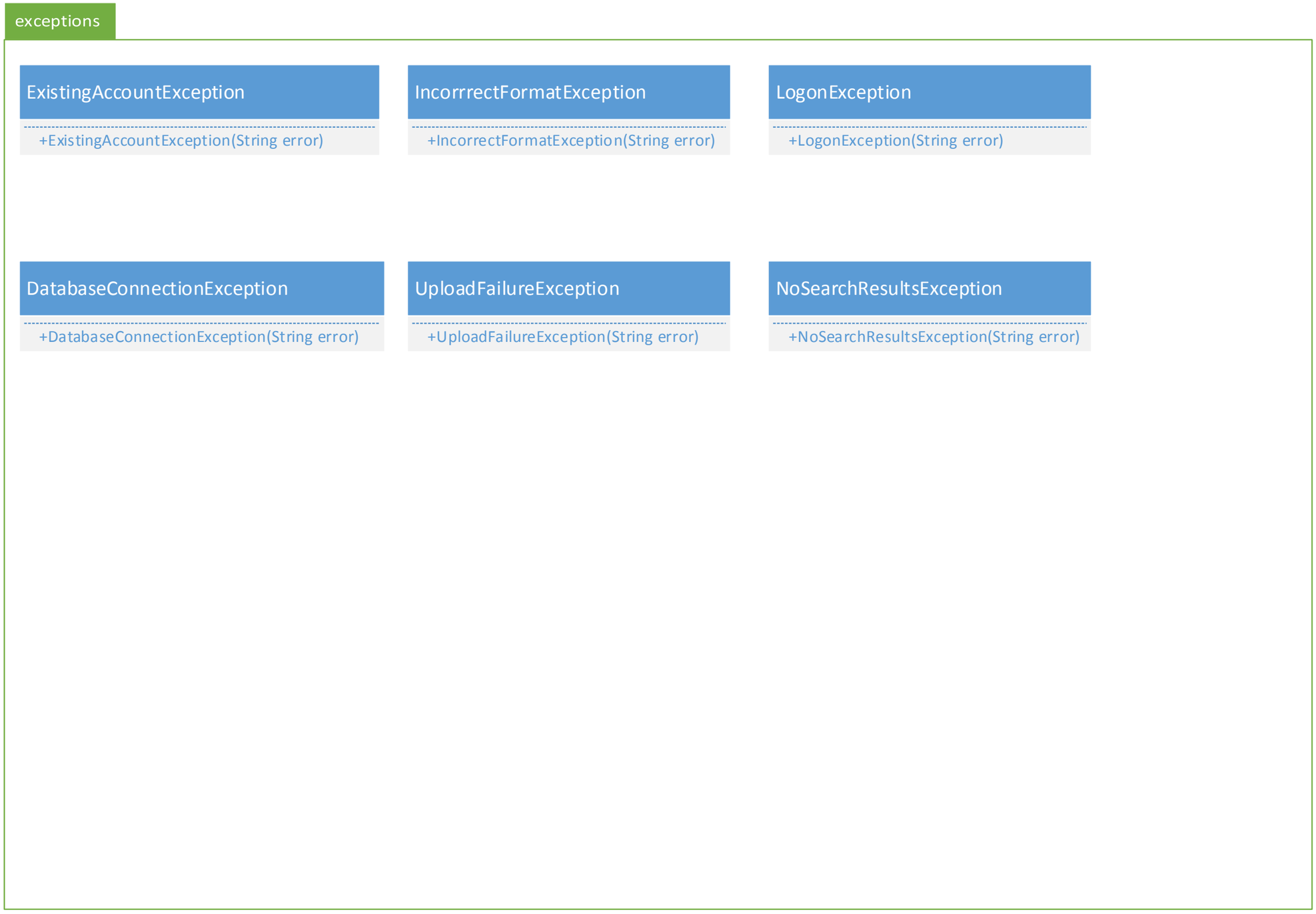


5.3.2.3. Data Processing Class Diagram

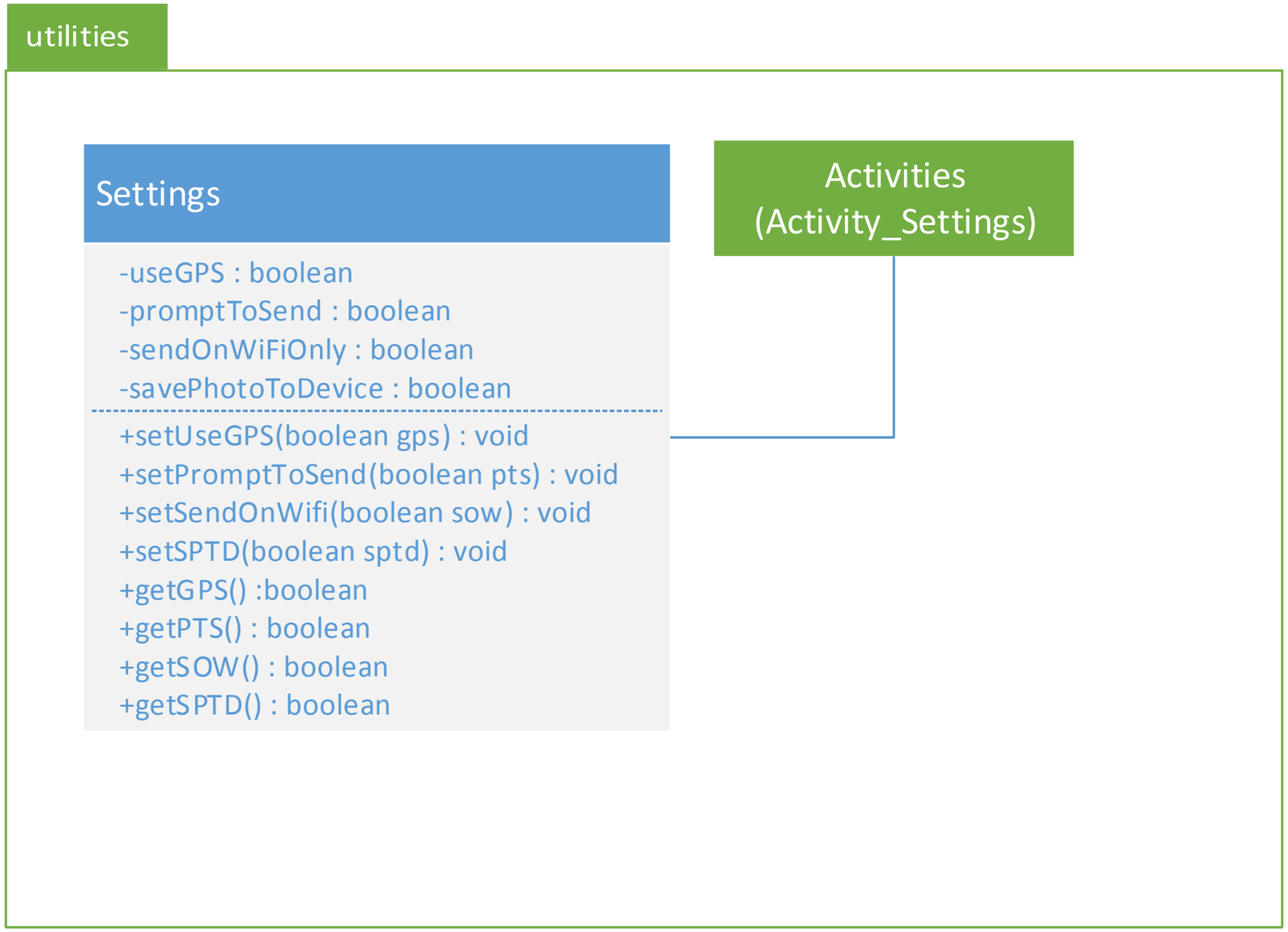
5.3.2.4. Database Class Diagram



5.3.2.5. Exceptions Class Diagram

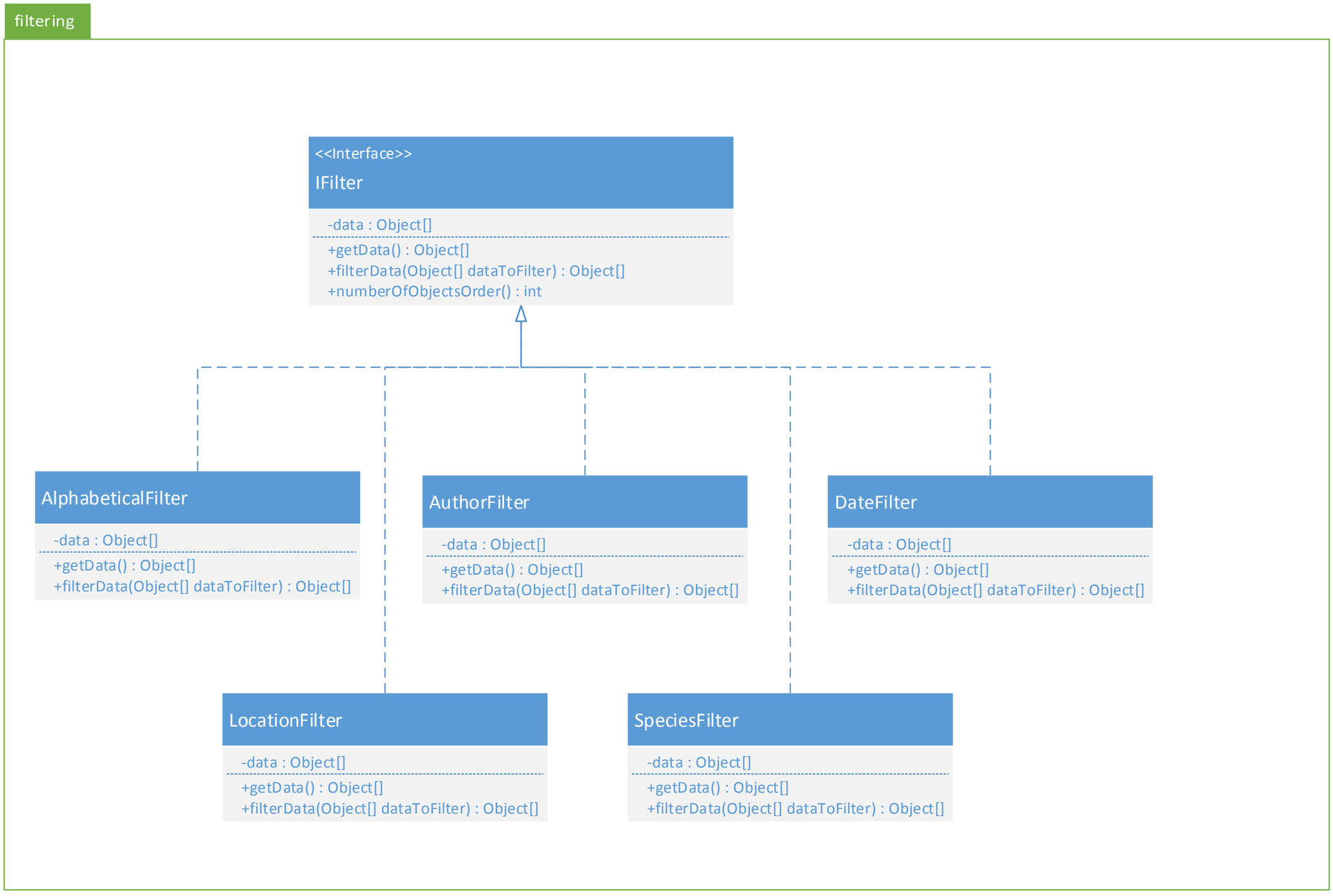


5.3.2.6. Utilities Class Diagram



5.3.2.7. Data Class Diagram

5.3.2.8. Filtering Class Diagram



6. REFERENCES

Software Engineering Group Projects Reserve Plant Species Recording Requirements Specification – N.W.Hardy – 06/10/2014

Software Engineering Group 11 Project Plan – T.Goree, T.Raikes, A.Spence, A.Davies et al

7. DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to document	Changed by
1.0	N/A	04/12/14	Document Created and Structured	Tcg2 & Add20