**System and Software Architecture Description (SSAD)**

**Fuppy**

**Team No.7**

Krupa Patel (Product Manager)

Adil Assouab (Requirement Engineer)

Yiyuan Chen (Software Architecture)

Praveen Chander (Designer/Prototyper)

Zhouyun Feng (Developer)

Fereshteh Khorzani (Quality Focal Point)

**11/28/2016**

**Version History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Changes made** | **Rationale** |
| 10/10/16 | YC | 1.0 | * Original template for use with Instructional ICM-Sw v1.0 | * Initial draft for use with Instructional ICM-Sw v1.0 |
| 10/16/16 | YC | 1.1 | * Modify the use-case diagram and tables name and index of tables | * Based on suggestions cited by professor on presentation |
| 11/18/16 | YC | 1.2 | * Add technical part diagram | * Based on current model of project |
| 11/28/16 | YC | 1.3 | * Modify specific model diagram | * Based on product of our project |

Table of Contents

1. Introduction

1.1 Purpose of the SSAD

1.2 Status of the SSAD

2. System Analysis

2.1 System Analysis Overview

2.1.1 System Context

2.1.2 Artifacts & Information

2.1.3 Behavior

2.1.3.1 Capability Authentication

2.1.3.2 Capability Search

2.1.3.3 Capability Appointment

2.1.3.4 Capability Receive Notification

2.1.4 Modes of Operation

2.2 System Analysis Rationale

3. Technology-Independent Model

3.1 Design Overview

3.1.1 System Structure

3.1.2 Design Classes

<Classes n>

3.1.3 Process Realization

3.2 Design Rationale

4.Technology-Specific System Design

4.1 Design Overview

4.1.1 System Structure

4.1.2 Design Classes

<Classes n>

4.1.3 Process Realization

4.2 Design Rationale

5. Architectural Styles, Patterns and Frameworks

Table of Tables

*Table 1: Actors Summary 8*

*Table 2: Artifacts and Information Summary 10*

*Table 3: Process Description 11*

*Table 4: Typical Course of Action - Login:Successful \_ 12*

*Table 5: Alternate Course of Action - Login:Failure 12*

*Table 6: Process Description 13*

*Table 7: Typical Course of Action - Register:Successful 13*

*Table 8: Alternate Course of Action - Register:Failure 13*

*Table 9: Process Description 14*

*Table 10: Typical Course of Action - Logout:Successful 14*

*Table 11: Alternate Course of Action - Logout:Failure 14*

*Table 12: Process Description 15*

*Table 13: Typical Course of Action - Search:Successful 15*

*Table 14: Alternate Course of Action - Search:Failure 16*

*Table 15: Process Description 16*

*Table 16: Typical Course of Action - Display:Successful 16*

*Table 17: Process Description 17*

*Table 18: Typical Course of Action - Make Appointment:Successful 17*

*Table 19: Alternate Course of Action - Make Appointment:Failure 18*

*Table 20: Process Description 18*

*Table 21: Typical Course of Action - Cancel Appointment:Successful 18*

*Table 22: Alternate Course of Action - Cancel Appointment:Failure 19*

*Table 23: Process Description 19*

*Table 24: Typical Course of Action - Receive Appointment Notification:Successful 20*

*Table 25: Alternate Course of Action - Receive Appointment Notification:Failure 20*

*Table 26: Hardware Component Description 23*

*Table 27: Software Component Description 24*

*Table 28: Design Class Description 25*

*Table 29: Hardware Component Description 30*

*Table 30: Software Component Description 30*

*Table 31: Design Class Description 32*

*Table 32: Architectural Styles, Patterns, and Frameworks 35*

Table of Figures

*Figure 1: System Context Diagram 9*

*Figure 2: Artifacts and Information Diagram 10*

*Figure 3: Process Diagram 11*

*Figure 4: Conceptual Domain Model 22*

*Figure 5: Hardware Component Class Diagram 22*

*Figure 6: Software Component Class Diagram 23*

*Figure 7: Deployment Diagram 23*

*Figure 8: Design Class Diagram 25*

*Figure 9: Robustness Diagram 27*

*Figure 10: Sequence Diagram 27*

*Figure 11: Hardware Component Class Diagram 29*

*Figure 12: Software Component Class Diagram 29*

*Figure 13: Deployment Diagram 30*

*Figure 14: Design Class Diagram 31*

*Figure 15: Process Realization Diagram 33*

**1. Introduction**

This document is mainly to provide the basic System and Software Architecture for customer, developer and architect, so that they can easily understand the architecture of this system which they want or they need to develop.

#### 1.1 Purpose of the SSAD

The purpose of the SSAD is to document the project analysis and design at the foundation phase. In addition, the SSAD shows the basic architecture of the project system, which can be reference for the developer to develop the system based on this architecture. And this document also includes all kinds of layers structure, which will provide great help for developer. Moreover, by looking at SSAD, maintainer and client will have a better understating for the whole system, therefore, they could clearly know how to maintain and whether the product delivered is adapted their objective.

The SSAD involves analyzing and designing of the system, which is used for communication among the architect, developer, and customer.

#### 1.2 Status of the SSAD

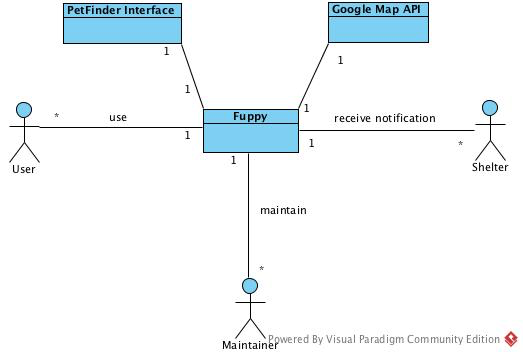
This document is the first version of the SSAD, so there is no key difference from the previous version. And this document has basically finished for the total structure of our team project.

**2. System Analysis**

#### 2.1 System Analysis Overview

The primary purpose of Fuppy project is to provide a platform for all kinds of Users and shelters in mobile android application so that they can adopt pets at Fuppy with shelters. In the Fuppy mobile android application, users could use Fuppy to search some types of pets which they are willing to adopt, and Fuppy will provide a filter for users to choose certain requirements, such as type, age, location and more. In addition, Fuppy will show the suitable results in mobile application including some information but not details. Furthermore, users could take another step to get details for the specific pet, and if they really like the pets, they can make an appointment with the shelter to see the pet and get more details. Finally, users will decide whether to adopt or not.

##### 2.1.1 System Context

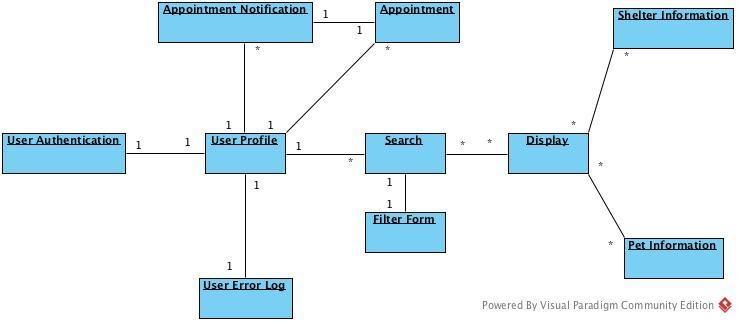


**Figure 1: System Context Diagram**

**Table 1: Actors Summary**

|  |  |  |
| --- | --- | --- |
| **Actor** | **Description** | **Responsibilities** |
| User | The user who uses Fuppy to adopt pets | 1. Perform Authentication to use Fuppy  2.Update their profile  3.Manage Appointments  4.Search for the pet |
| Shelter | A kind of user to provide pet and information in Fuppy | 1.upload the information of pets  2.manage and update profile |
| Maintainer | A person to maintain Fuppy after Fuppy being put into market | 1.keep the Fuppy run successfully  2.if some errors happen, cope with it |
| PetFinder Interface | An interface for app to fetch the specific data from PetFinder database | 1.fetch and give back the result to App when receive the request from App  2.keep the data correctness |
| Google Map API | An API for app to implement map functionality | 1.can integrate with the App successfully  2.provide precise map information |

##### 2.1.2 Artifacts & Information

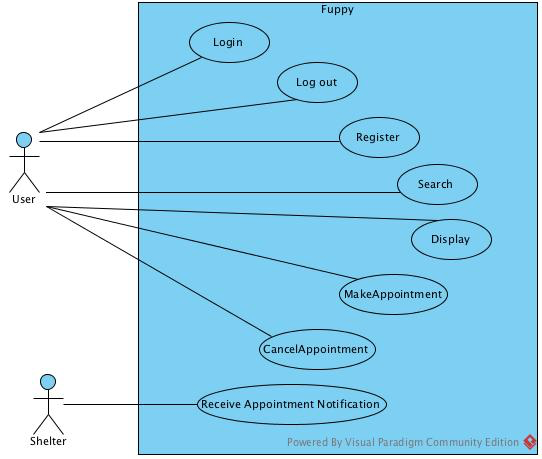


**Figure 2: Artifacts and Information Diagram**

**Table 2: Artifacts and Information Summary**

|  |  |
| --- | --- |
| **Artifact** | **Purpose** |
| ATF-1:User Authentication Form | Provide the form to the user to register or login the system |
| ATF-2:User Profile | Contains users’ basic information such as name, age, phone and so on. |
| ATF-3:User Error Log | Contains all the error reports of the errors generated on users side |
| ATF-4:Pet Information | Contains all the details of one pet, including the pets belongs to which shelter. |
| ATF-5:Shelter Information | Contains all the information of shelters, including all pets they have |
| ATF-6:Search Filter Form | Gives user the option to select the kind of pets they wish to look for, and based on the that,if fetches the data from database. |
| ATF-7:Appointment Form | Gives user the option to select the time and date on which they would like to visit the shelter and based on user selection a notification is send shelter. |
| ATF-8:Appointment Notification | Notifies the users when they make appointments successfully. |

##### 2.1.3 Behavior



**Figure 3: Process Diagram**

###### 2.1.3.1 Capability Authentication

**2.1.3.1.1 Process Login**

**Table 3: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Login |
| **Purpose** | Determine whether a user logging into system is authenticated, and if so, find out what privilege of this user. |
| **Requirements** | Authentication |
| **Development Risks** | NONE |
| **Pre-conditions** | User has connected to the Network.  The database server works properly. |
| **Post-conditions** | If user is authorized, give the appropriate role for the user to access system; otherwise, user is denied access to the system. |

**Table 4: Typical Course of Action - Login:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Enter username and password |  |
| **2** | Click login |  |
| **3** |  | Send the form to Authentication backend to check its valid |
| **4** | Receive verification and session data |  |
| **5** |  | Redirect to home page |

**Table 5: Alternate Course of Action - Login:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-4** | Refer to Typical Course of Action | |
| **5** |  | Display error message to users like invalid username or password |
| **6** | Click OK button |  |
| **7** |  | Redirect to Login page |

**2.1.3.1.2 Process Register**

**Table 6: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Register |
| **Purpose** | Determine whether the informations input by users have been registered, and whether the informations are valid, and finally register. |
| **Requirements** | Authentication |
| **Development Risks** | NONE |
| **Pre-conditions** | User has connected to the Network.  The database server works properly. |
| **Post-conditions** | If the information is invalid or has been registered, give the appropriate message to user to correct the input; otherwise, successfully register the informations and send feedback to users. |

**Table 7: Typical Course of Action - Register:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Enter username, password and other information |  |
| **2** | Click Register |  |
| **3** |  | Send the form to Authentication backend to check its valid |
| **4** | Receive verification and session data |  |
| **5** |  | Redirect to Home page |

**Table 8: Alternate Course of Action - Register:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-4** | Refer to Typical Course of Action | |
| **5** |  | Display error message to users like invalid username or password |
| **6** | Click OK button |  |
| **7** |  | Redirect to Register page |

**2.1.3.1.3 Process Logout**

**Table 9: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Logout |
| **Purpose** | Used for user log out safely to the application |
| **Requirements** | Authentication |
| **Development Risks** | It maybe difficult to shut off all the threads which related to users. |
| **Pre-conditions** | User has connected to the Network. |
| **Post-conditions** | Users log out successfully with all session done, and if they want to login again, they have to input username and password again. |

**Table 10: Typical Course of Action - Logout:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Click Logout Button |  |
| **2** |  | Call the function in application to logout and send result to user |
| **3** | Receive message that has log out successfully |  |

**Table 11: Alternate Course of Action - Logout:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-2** | Refer to Typical Course of Action | |
| **3** |  | Display error message to users |
| **4** | Click Logout Button again |  |
| **5** | Redo the Typical Course of Action | |

###### 2.1.3.2 Capability Search

**2.1.3.2.1 Process Search**

**Table 12: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Search |
| **Purpose** | Search the ideal pets by input some key information in filter, and give a list of pets which are accord with the requirements. |
| **Requirements** | Authentication, Login |
| **Development Risks** | Connection with database interface. |
| **Pre-conditions** | User has connected to the Network.  The database server works properly. |
| **Post-conditions** | If there are some probably results, store them into a model; otherwise, shows none of pets suitable. |

**Table 13: Typical Course of Action - Search:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Enter key requirements in filter |  |
| **2** | Click Search Button |  |
| **3** |  | Send the form to backend, and call the function to fetch the suitable results. |
| **4** | Receive results data |  |
| **5** |  | Call Display Process |

**Table 14: Alternate Course of Action - Search:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-4** | Refer to Typical Course of Action | |
| **5** |  | Display message none of pets suitable |
| **6** | Click OK button |  |
| **7** | Redo to Typical Course of Action |  |

**2.1.3.2.2 Process Display Result**

**Table 15: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Display |
| **Purpose** | Display the results which users use filter search before by showing pets information with a picture and short introduction |
| **Requirements** | Authentication, Login, Filter Search |
| **Development Risks** | The UI for displaying image view.  Connection between http and android cause can’t fetch picture through url address. |
| **Pre-conditions** | User has connected to the Network.  The database server works properly.  Successfully connection between http and android. |
| **Post-conditions** | If there are lots of results, show the first 8 results, when request more results, shows again; otherwise, show all results. |

**Table 16: Typical Course of Action - Display:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** |  | Call the function to analyze JSON list, and display the results from model list. |
| **2** | Click Load More button |  |
| **3** |  | Redo seq#1 |

###### 2.1.3.3 Capability Appointment

**2.1.3.3.1 Process Make An Appointment**

**Table 17: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Make Appointment |
| **Purpose** | Determine when and where to meet the appointment for user to shelters. |
| **Requirements** | Authentication, Login |
| **Development Risks** | NONE |
| **Pre-conditions** | User has connected to the Network.  The database server works properly.  Valid information in time and other information. |
| **Post-conditions** | If the appointment is valid, create a new appointment data and store it in database; Otherwise, users should re-appointment with shelters. |

**Table 18: Typical Course of Action - MakeAppointment:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Enter date, place with shelter |  |
| **2** | Click Appointment button |  |
| **3** |  | Send the form to backend to check its valid, if so create a new appointment in database |
| **4** | Receive feedback and successful message |  |
| **5** |  | Redirect to home page |

**Table 19: Alternate Course of Action - MakeAppointment:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-3** | Refer to Typical Course of Action | |
| **4** |  | Display error message to users like invalid data time, notice users to reschedule |
| **5** | Click OK button |  |
| **6** |  | Redirect to Appointment page |

**2.1.3.3.2 Process Cancel An Appointment**

**Table 20: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Cancel Appointment |
| **Purpose** | Cancel an appointment existed |
| **Requirements** | Authentication, Login, Make Appointment |
| **Development Risks** | NONE |
| **Pre-conditions** | User has connected to the Network.  The database server works properly.  There has been making appointment first. |
| **Post-conditions** | If there is an appointment which is not out of date, delete the data in the database, and notice both users and shelters of message that the appointment has been canceled; otherwise, report errors to user could not do this operation. |

**Table 21: Typical Course of Action - Cancel Appointment:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** | Choose one of appointment |  |
| **2** | Click Cancel button |  |
| **3** |  | Send the cancel form to backend to check its valid, if so delete the data in database |
| **4** | Receive feedback |  |
| **5** |  | Redirect to Appointment page |

**Table 22: Alternate Course of Action - Cancel Appointment:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-3** | Refer to Typical Course of Action | |
| **4** |  | Display error message to users |
| **5** | Click OK button |  |
| **6** |  | Redirect to Cancel Appointment page |

###### 2.1.3.4 Capability Receive Notification

**2.1.3.4.1 Process Receive Appointment Notification**

**Table 23: Process Description**

|  |  |
| --- | --- |
| **Identifier** | Receive Appointment Notification |
| **Purpose** | For shelter to receive appointment notification made successfully by user. |
| **Requirements** | User Make Appointment |
| **Development Risks** | Auto generate an email and send to shelter when user make an appointment successfully |
| **Pre-conditions** | Shelters’ email address correct.  User make an appointment successfully. |
| **Post-conditions** | If the email sent to shelter successfully, show feedback to user; otherwise, resend email to shelter again. |

**Table 24: Typical Course of Action - Receive Appointment Notification:Successful**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1** |  | Receive the response of users’ successful appointments |
| **2** |  | Auto-generated an email and then send email to shelter |
| **3** | Receive the email successfully |  |

**Table 25: Alternate Course of Action - Receive Appointment Notification:Failure**

|  |  |  |
| --- | --- | --- |
| **Seq#** | **Actor’s Action** | **System’s Response** |
| **1-2** | Refer to Typical Course of Action | |
| **3** |  | Redo 1-2 steps |

##### 2.1.4 Modes of Operation

#### 2.2 System Analysis Rationale

The following is the Analysis Rationale section for the Fuppy project:

Based on our analysis of how the users interact with the system, we have identified 2 class of operational stakeholders.

1. User - These users include people of all ages who are willing to adopt a pet through Fuppy. These users requires authentication to log into Fuppy, and start search, filter, and appointment. This part of system can only be accessed when users connect to the network and use mobile application.
2. Shelter- these shelters include some private pets owner or pets agency or others organizations. These shelters requires authentication to use the Fuppy Web Application to upload the information of pets. And this part of system can only be accessed when users connect to network. And these shelters will receive notifications when users make appointments with them.

There are 2 external interface actors that interface with the Fuppy System. They are:

1. PetFinder interface - provides interface for Fuppy android application to fetch to data from PetFinder’s database.

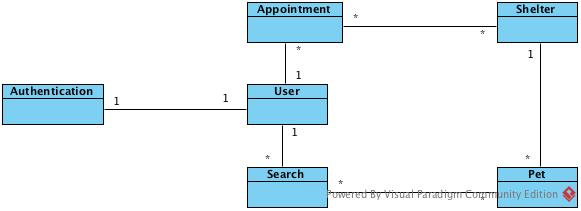
2. Google Map API - provides interface for Fuppy android application to implement map function.

The Web Fuppy systems are being built by another CSCI577 team. The Fuppy Android System is, therefore, being designed to integrate all interfaces together with basic UI and achieve a functional mobile application which can let users adopt pets through the application and let shelters receive appointment notifications.

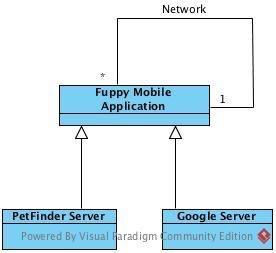
**3. Technology-Independent Model**

#### 3.1 Design Overview

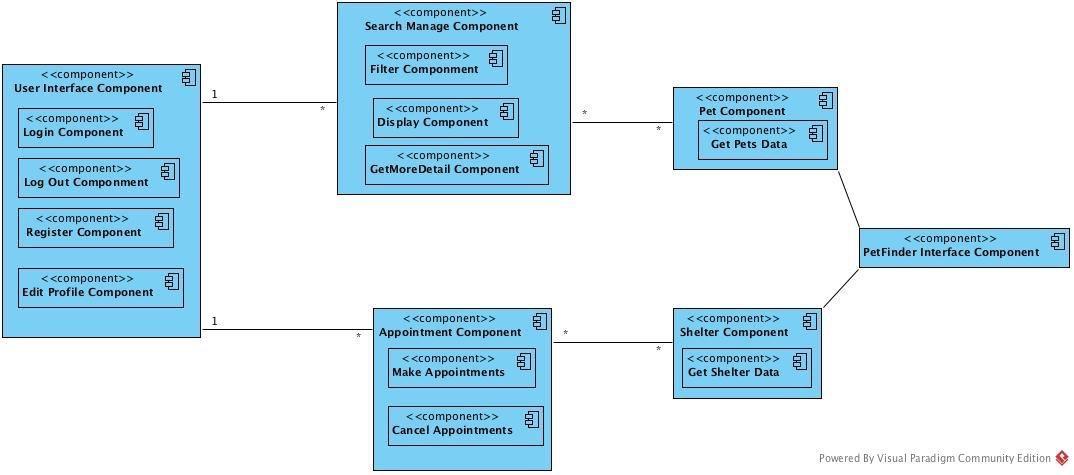
##### 3.1.1 System Structure



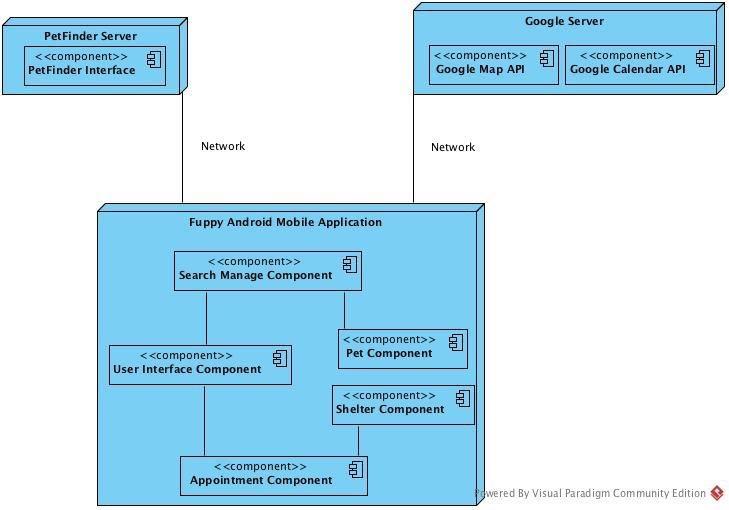
**Figure 4: Conceptual Domain Model**



**Figure 5: Hardware Component Class Diagram**



**Figure 6: Software Component Class Diagram**



**Figure 7: Deployment Diagram**

**Table 26: Hardware Component Description**

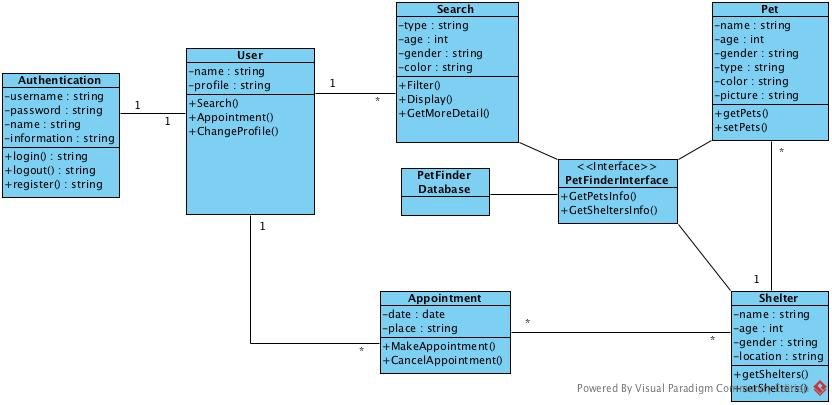
|  |  |
| --- | --- |
| **Hardware Component** | **Description** |
| Fuppy mobile Application | The Fuppy mobile Application is the application on which the majority of the Fuppy System components reside. The application provide user many interfaces so that users can communicate with others Server through network to get data. |
| PetFinder Server | The PetFinder Server is the server that provide a interface for Fuppy application to fetch data, and manage all the pets and shelters data. |
| Google Server | Google Server provide a Google Map API and Calendar API for Fuppy application, which lets Fuppy could locate users’ location. |

**Table 27: Software Component Description**

|  |  |
| --- | --- |
| **Software Component** | **Description** |
| User Interface Component | This component contains the users’ authentication operation and some basic operations, such like login, logout, register, edit profile. |
| Search Manage Component | This component manages the searching results, including use filter and display the result and show more detail of specific pet, through call PetFinder interface. |
| Appointment Component | This component manages the users make appointments and cancel appointments. Show, add, or delete appointments. |
| Pet Component | This component store the information of pets. |
| Shelter Component | This component store the information of shelters. |
| PetFinder Interface Component | This component provide a interface for user to communicate and fetch data from the PetFinder database. |

##### 3.1.2 Design Classes

###### <Classes n>

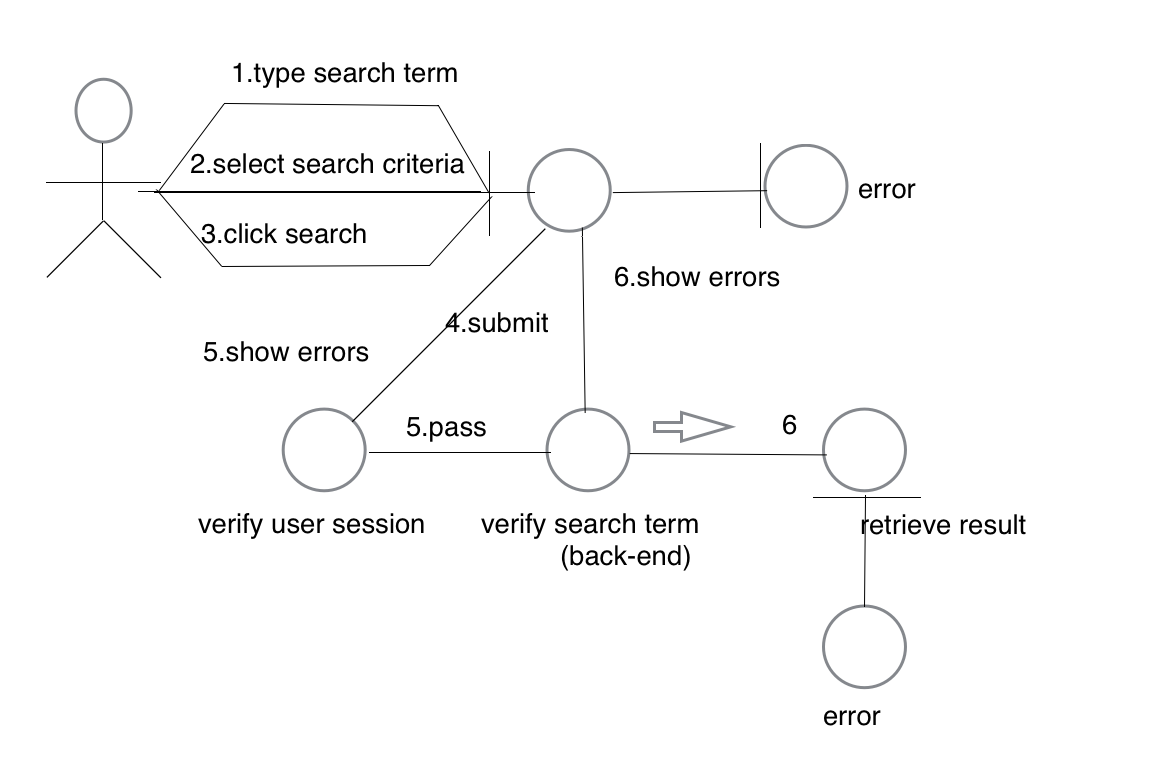


**Figure 8: Design Class Diagram**

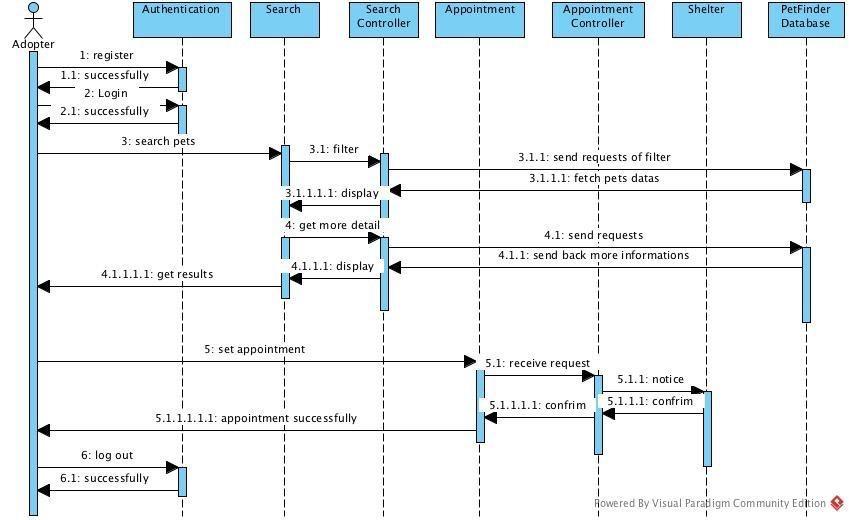
**Table 28: Design Class Description**

|  |  |  |
| --- | --- | --- |
| **Class** | **Type** | **Description** |
| User | component | Contains all operations for user to search, appointment and adopt pets. |
| Authentication | component | Contains all logical operation for user to access this application |
| Search | component | Contains all attributes and operations of search management. |
| Appointment | component | Contains all logical computation for make or cancel appointment |
| Pet | component | Contains all attributes and operations of pets. |
| Shelter | component | Contains all attributes and operations of shelters. |
| PetFinderInterface | interface | Provide interface between database and application. |
| PerFinderDatabase | entity | Contains all the datas of pets and shelters. |

##### 3.1.3 Process Realization



**Figure 9: Robustness Diagram**



**Figure 10: Sequence Diagram**

#### 3.2 Design Rationale

We adopt a 3-tier layers architecture, and base on MVC development pattern.

Authentication Layer

for users to login, logout and register

MainActivities Layer

including the activities like search pets, display pets’ information, get more pets’ details and shelters information, make and cancel appointment

DBInterface Layer

make a connection between application and backend

User Model, Pets Model, Shelter Models for store datas retrieve from application or database

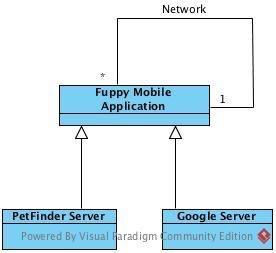
We adopt the architecture above for following reasons:

1. There has been existed a web back-end Fuppy, and client does not require us to develop a new backend to get the datas. So we decide to create a Petfinder interface, which can access the former backend server.
2. Our project mainly focus on the user User. So for another user shelters, they need to register on web application and to upload pets information, not in android mobile application.
3. Although user could start a quick sort without authentication, authentication is still necessary for the application and it needs a independent layer.
4. Although there will be interface to fetch the data from database, it still needs model to store the data temporarily, because in the view page, we do not show all the information in one page. So the model is necessary.

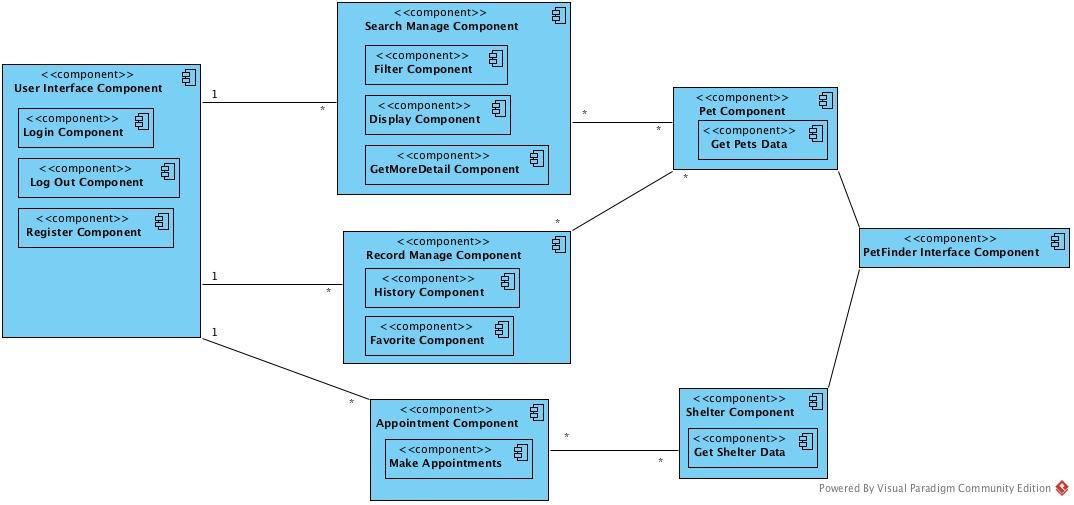
**4.Technology-Specific System Design**

#### 4.1 Design Overview

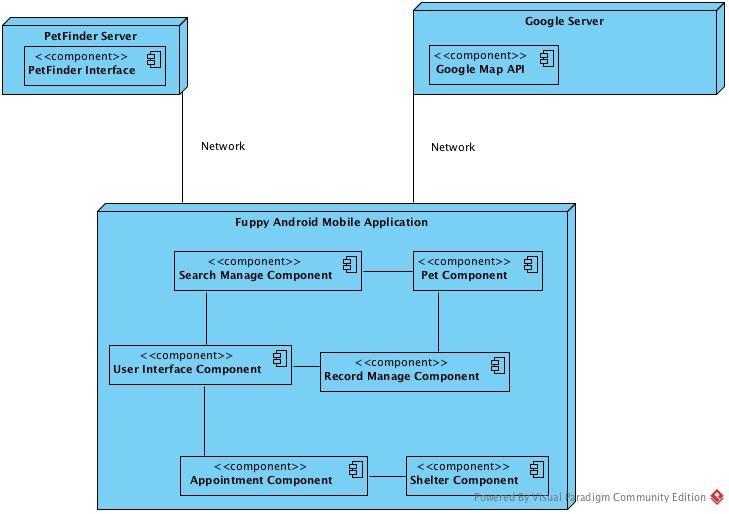
##### 4.1.1 System Structure



**Figure 11: Hardware Component Class Diagram**



**Figure 12: Software Component Class Diagram**



**Figure 13: Deployment Diagram**

**Table 29 Hardware Component Description**

|  |  |
| --- | --- |
| **Hardware Component** | **Description** |
| Fuppy mobile Application | The Fuppy mobile Application is the application on which the majority of the Fuppy System components reside. The application provide user many interfaces so that users can communicate with others Server through network to get data. |
| PetFinder Server | The PetFinder Server is the server that provide a interface for Fuppy application to fetch data, and manage all the pets and shelters data. |
| Google Server | Google Server provide a Google Map API for Fuppy application, which lets Fuppy could locate users’ location. |

**Table 30: Software Component Description**

|  |  |
| --- | --- |
| **Software Component** | **Description** |
| User Interface Component | This component contains the users’ authentication operation and some basic operations, such like login, logout, register, edit profile. |
| Search Manage Component | This component manages the searching results, including use filter and display the result and show more detail of specific pet, through call PetFinder interface. |
| Record Manage Component | This component manages the favorite and history records, including add or remove favorite records, automatically log user’s track on pets and clear records. |
| Appointment Component | This component manages the users make appointments and cancel appointments. Show, add, or delete appointments. |
| Pet Component | This component store the information of pets. |
| Shelter Component | This component store the information of shelters. |
| PetFinder Interface Component | This component provide a interface for user to communicate and fetch data from the PetFinder database. |

##### 4.1.2 Design Classes

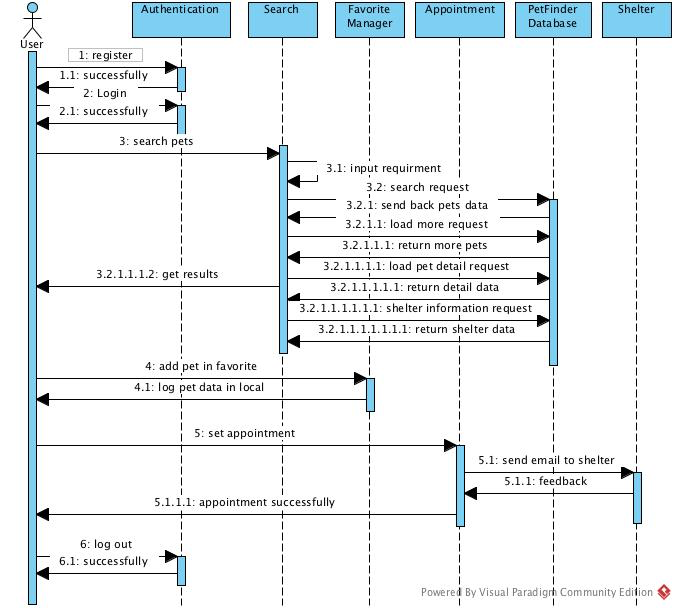
###### <Classes n>

**Figure 14: Design Class Diagram**

**Table 31: Design Class Description**

|  |  |  |
| --- | --- | --- |
| **Class** | **Type** | **Description** |
| User | component | Contains all operations for user to search, appointment and adopt pets. |
| Authentication | component | Contains all logical operation for user to access this application |
| Search | component | Contains all attributes and operations of search management. |
| Appointment | component | Contains all logical computation for make or cancel appointment |
| Pet | component | Contains all attributes and operations of pets. |
| Shelter | component | Contains all attributes and operations of shelters. |
| FavoriteManager | component | Contains basic operation for user to add or remove pet information in favorite part |
| History | component | Contains clear operation for user and log down all pets which user has scanned |
| PetFinderInterface | interface | Provide interface between database and application. |
| PerFinderDatabase | entity | Contains all the datas of pets and shelters. |

##### 4.1.3 Process Realization



**Figure 15: Process Realization Diagram**

#### 4.2 Design Rationale

We adopt a 3-tier layers architecture, and base on MVC development pattern.

Authentication Layer

for users to login, logout and register

MainActivities Layer

including the activities like search pets, display pets’ information, get more pets’ details and shelters information, make and cancel appointment

DBInterface Layer

Use petfinder Interface to access the database on petfinder, and fetch the data for relevant pets, pass the data to MainActivities Layer for showing.

User Model, Pets Model, Shelter Models for store datas retrieve from application or database

We adopt the architecture above for following reasons:

1. There has been existed a web back-end Fuppy, and client does not require us to develop a new backend to get the datas. So we decide to create a Petfinder interface, which can access the former backend server.

2. Our project mainly focus on the Adopter User. So for another user shelters, they need to register on web application and to upload pets information, not in android mobile application.

3. User need to take authentication first to access fuppy application, and then do activities like search or make appointments. So authentication is necessary for the application and it needs a independent layer.

4. Although there will be interface to fetch the data from database, it still needs model to store the data temporarily, because in the view page, we do not show all the information in one page. So the model is necessary.

5. For users, it is better for them to have a log record function such like favorite and history records, so that they could add some pets which they are interested in on local records. Therefore, they could check all the pets when they are available.

6. In order to provide simple and useful UI for users, it is a good idea to use navigation view, so users will clearly know how to operate.

**5. Architectural Styles, Patterns and Frameworks**

**Table 32: Architectural Styles, Patterns, and Frameworks**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Benefits, Costs, and Limitations** |
| Java and Android Studio | Android Studio provide a clear UI design and powerful functionality. And provide a emulator for easy test. | Provide clear UI, format and test environment. But it takes time for team to adapt to the new developing environment. |
| MVC | Design the pattern based on model, view, and controller, which could make the structure easy to understand and easy to develop. | Make the structure simple and easy to understand. Every one could develop its part, and finally integrate together.  However, it take lots of time to construct at early time, and also needs to define all variabilities earlier. |
| 3 layer architecture | Divide the architecture into 3 different layer, and each layer can communicate with other layer, not influence others. | Divide into 3 layer can make it more flexible to develop and to implement requirements. But, it also become more complex with project progressing. |