

**Computer Science and Engineering**

# \_

**ShelterHelp**

**Software Design Description (SDD)**

# Version 2.0

Document Number: SDD-002

Team Number: B10

Team Members (Name and NET-ID) Raj Ramrup rr2661, Jeton Koka jk4697, Zulkifl Arefin za453, Mohammed Mahmud mnm376, Nabil Ahmed na1489

# REVIEW AND APPROVALS

|  |  |  |  |
| --- | --- | --- | --- |
| **Printed Name and Title** | **Function (Author, Reviewer,**  **Approval)** | **Date** | **Signature** |
| Fred Strauss | Author | May 3rd 2018 |  |
| Nabil Ahmed | Agile Coach | May 3rd 2018 | NA |
| Zulkifl Arefin | Scrum Master | May 3rd 2018 | ZA |
| Jeton Koka | Lead Developer | May 3rd 2018 | JK |
| Mohammed Mahmud | Project Manager | May 3rd 2018 | MNM |
| Raj Ramrup | Quality Assurance | May 3rd 2018 | RR |

**REVISION LEVEL**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision Number** | **Purpose** |
| March 21st 2018 | Version 1.0 | Initial Release |
| May 3rd  2018 | Version 2.0 | Final Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Table Of Contents**

1. [INTRODUCTION 1](#_bookmark0)
   1. [PURPOSE 1](#_bookmark1)
   2. [SCOPE 1](#_bookmark2)
   3. [IDENTIFICATION 1](#_bookmark3)
   4. [DOCUMENT SUMMARY 1](#_bookmark4)
   5. [SYSTEM OVERVIEW 1](#_bookmark5)
   6. [DOCUMENT OVERVIEW 1](#_bookmark6)
2. [REFERENCE DOCUMENTS 2](#_bookmark7)
3. [SYSTEM WIDE DESIGN DECISIONS 2](#_bookmark8)
   1. [SOFTWARE COMPONENT ARCHITECTURAL DESIGN 2](#_bookmark9)
   2. [SOFTWARE ARCHITECTURE GENERAL DESCRIPTION 2](#_bookmark10)
   3. [SOFTWARE ITEM COMPONENTS 2](#_bookmark11)
   4. [COMPONENT INTERFACE IDENTIFICATION 3](#_bookmark12)
   5. [SOFTWARE COMPONENT CONCEPT OF EXECUTION 3](#_bookmark13)
4. [SOFTWARE ITEM DETAILED DESIGN 3](#_bookmark14)
   1. [STRUCTURE 3](#_bookmark15)
      1. [Software Unit Detailed Design 3](#_bookmark16)
   2. [STATIC RELATIONSHIP OF SOFTWARE UNIT 3](#_bookmark17)
      1. [Run-time Object Instances 3](#_bookmark18)
   3. [BEHAVIOR 3](#_bookmark19)
      1. [Sequence Interaction Diagrams 3](#_bookmark20)
      2. [Collaboration Diagrams 3](#_bookmark21)
      3. [Activity Diagrams 3](#_bookmark22)
   4. [CONCEPT OF EXECUTION 3](#_bookmark23)
   5. [INTERFACE DESIGN 4](#_bookmark24)
      1. [Interface Identification and Diagrams 4](#_bookmark25)
      2. [Unique identifier of Interface 4](#_bookmark26)
5. [IMPLEMENTATION ARCHITECTURE (NOT REQUIRED) 4](#_bookmark27)
   1. [ALL ACTIVE AND PASSIVE CLASSES ASSIGNED TO COMPONENTS 4](#_bookmark28)
   2. [DIAGRAMS OF PHYSICAL PACKAGING OF LOGICAL COMPONENTS 4](#_bookmark29)
6. [DEPLOYMENT ARCHITECTURE 4](#_bookmark30)
   1. [PHYSICAL DEPLOYMENT ARCHITECTURE DIAGRAM 4](#_bookmark31)
7. [DICTIONARIES 4](#_bookmark32)
8. [SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION 4](#_bookmark33)
9. [REQUIREMENTS TRACEABILITY 4](#_bookmark34)
   1. [SOFTWARE COMPONENT-LEVEL REQUIREMENTS TRACEABILITY 5](#_bookmark35)
10. [SYSTEM DESIGN TESTING 5](#_bookmark36)
11. [RATIONALE 5](#_bookmark37)
12. [NOTES 5](#_bookmark38)
13. [APPENDICES 5](#_bookmark39)
    1. [DICTIONARIES 5](#_bookmark40)
    2. [UML DIAGRAMS, IF NOT INCLUDED IN THE BODY OF THE DOCUMENT 5](#_bookmark41)
    3. [SCHEDULE TRACKING 6](#_bookmark42)
    4. [DEFECT TRACKING 7](#_bookmark43)

# INTRODUCTION

# Purpose

The document is used to communicate overall quantitative and qualitative system characteristics to operations management, technical support, training, and operators. It is to provide a description of a system fully enough to allow for software development to proceed with a proper understanding of what is to be made and how. Its purpose is to document the architecture and detailed design of the system.

# Scope

The process of adopting and saving animals scattered throughout different shelters has not been addressed, relying on old techniques such as phone calls and paper forms to document people who have an interest in adopting pets. Furthermore, the lack of communication between shelters themselves has not been addressed, with institutions having their own internal systems to rely on cross-shelter communication. More often than not, this leads to overcrowded shelters that turn to killing of animals to preserve space rather than reallocation of said animals to shelters with available capacity. ShelterHelp will provide an interface for shelters to interact with each other, serving as a centralized hub between various shelters, ultimately providing a means of communication between the shelters to avoid animal euthanasia. Furthermore, the interface will allow individuals looking to adopt a pet with a repository of up to date shelter listings and respective animal health records.

# Identification

Team B10 Software Design Description, SDD-002, Version 2.0

# Document Summary

This document description serves multiple purposes. It is primarily to describe the functional structure, data and algorithms to be implemented. With these structures, the document identifies required system resources for the project along with producing test cases to very compliance with requirements.

# System Overview

## Context Diagram

## 

Note: Shelters and adopters could be people accessing the front-end of a web app or they could be other systems. Shelters could be a database that a shelter uses and we pull and sync records from them. Adopters could be a different system that adopters use that we sync with.

## Additional Descriptive Items

At a high level our product needs to:

* + to help shelters organize their information
  + share some of this with other shelters
  + share animal information with potential adopters
  + navigate people with strays to shelters that can take of them
  + organize transfers of animals between shelters

As currently planned specific features include:

* 1. Animal capacity system will monitor available space in shelter to avoid overcrowding.
     1. Suggestions on possible shelter transfers will be made available to administrators within each respective shelter.
  2. Animal health record system will allow easy transfer of animal health or behavioral information between shelters.
  3. Shelters will be able to reserve or make appointments for adoption inquiries concerning specific animals.
  4. A list of animals in each shelter will be available to potential animal adopters and will serve as the interface between shelters and adopters.

The users of the system will be divided up into two general categories: shelter staff and administration, and potential adopters. The staff/administration will not require any technical expertise besides being able to enter animal information into the system through the means of form input fields. The staff will need to have experience with the animals in order to accurately input proper information. On a similar note, the potential adopters will not need a technical background to use the application, and they will only need to create an account that involves a few personal questions to find matching animal profiles.

The need to interface with existing shelter infrastructure will limit automated database table entry. Moreover, the database table design will be constrained to the available information found across all shelters to ensure that a standardized table design.

Constraints on the developer options are fairly minimal as this will mostly be an organization application. Shelters are not expected to have modern computers as they were mostly dealing with paper forms earlier. As a result most of the work will be done on the server and puts a heavy strain on it. This means that the language/framework chosen will need to be an efficient one.

# Document Overview

The rest of this document contains Section:

2) Reference Documents

3) System Wide Design Decisions

4) Software Item Detailed Design

5) Implementation Architecture (Not Required)

6) Deployment Architecture

7) Dictionaries

8) Software Item Computer Resource Utilization

9) Requirements Traceability

# REFERENCE DOCUMENTS

Software Project Management Plan Version 1.1, February 17¨†2018

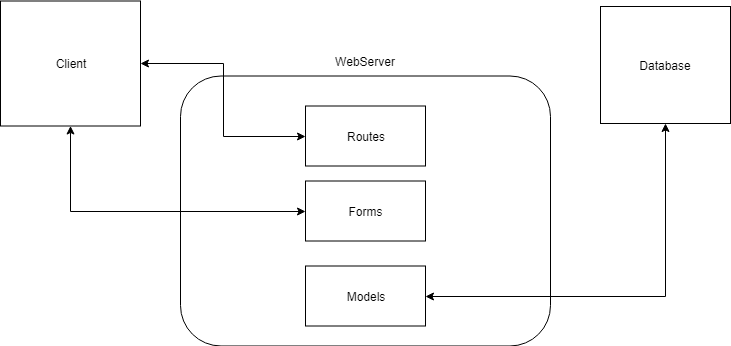
Requirements and Analysis Specification Version 1.1, February 9, 2018

Project Proposal Version 1.1, February 6, 2018

SDD 1.0, March 21, 2018

# SYSTEM WIDE DESIGN DECISIONS

# Software Component Architectural Design



# Software Architecture General Description

ShelterHelp is designed to be a database management platform for local shelters and adopters. ShelterHelp will be able to facilitate transfers of animals effectively through this system. It will also help adopters find out descriptions of animals that are from a variety of shelters in a formatted page.

To facilitate all the functional requirements, the client will interact with the server and the database server through interfaces.

# Software Item Components

Models: These are SQLAlchemy models that abstract our database usage. These models hold data after they are retrieved from the database.

Database: Postgres database used for storing data in a persistent manner.

Forms: Structured method for user to interact with client. Clients interact with the forms via the webpage the specific form is contained, and the forms are sent back in that format.

Routes: Manages a function to a route and route to a webpage, where a function is a normal python function and a route is a URI.

Client: Displays html code sent by the server.

# Component Interface Identification

# Models

# We need an interface with the database to hold the data that our queries return. We also need to have an interface to format our data to send to the database. Luckily the python library SQL Alchemy does a lot of this for us. We define classes called 'Models' that hold this data in a compatible way. It also abstracts database calls for us. We can query an entire table, filter it, and do joins without ever writing a line of SQL. This allows SQL Alchemy to abstract *what* database we are dealing with. We could be using Postgres in production but quickly switch and use SQLITE for quick testing without access to the internet. Our interface for our database is SQL Alchemy and specifically their 'Models'

# Forms

# Handling input from the user is a perilous task. They can put SQL or JavaScript code and hope it gets executed. They can also (not maliciously) try to put in bad data repeatedly. Handling user input needs to be robust. The python library wtf-forms handles this process for us. We define classes called forms that use special data types. We can pass these forms into html files where they are turned into html code. When the user tries to input into the page then their input will be validated as defined in the class and sent back in a structured form. The python library wtf-forms allows us to handle user input in a robust way. Our interface for our client input is wtf-forms specifically their 'Forms'

# Software Component Concept of Execution

# The webserver will be run on a computer on a network that allows incoming traffic. If on a home network, then you need to port forward port 5000. The database can be hosted anywhere where the webserver can reach it. The database should be broadcasted so that it can be reached by others. From here the client can access the webserver by the IP address of the computer running the webserver.

# Unzip the file that is submitted. Install Python 2.7, [], Postgres. Open dbInit.py, insertTestData.py, main.py. Replace the string "root:root" with your Postgres username:password. Run command python main.py.

# Frontend components: Client, Application Interface

# The user initiates a request by using the application interface

# After receiving the information/responses from the backend components, the frontend then relays it to the user by either committing to some action or displaying to the screen

# Backend components:

# External: External API

# The user requests some specific information or request from the external API

# The API returns the information requested or returns an error

# Internal: Server, Database

# The user sends a request to retrieve some information from the database or post new information into the database

# Request Information:

# If the user is requesting information, the database either returns the information using the parameters provided or returns an error if the data isn't found

# Post Information:

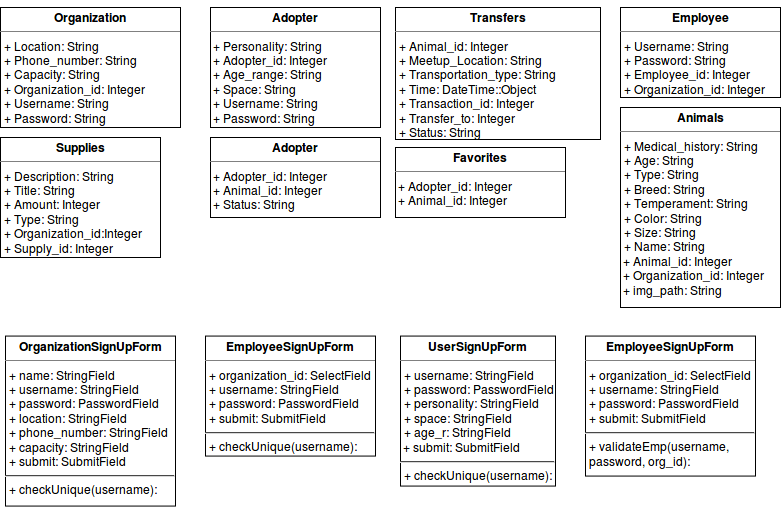
# If the user is trying to put new information, the database records the information and sends back a status message

# SOFTWARE ITEM DETAILED DESIGN

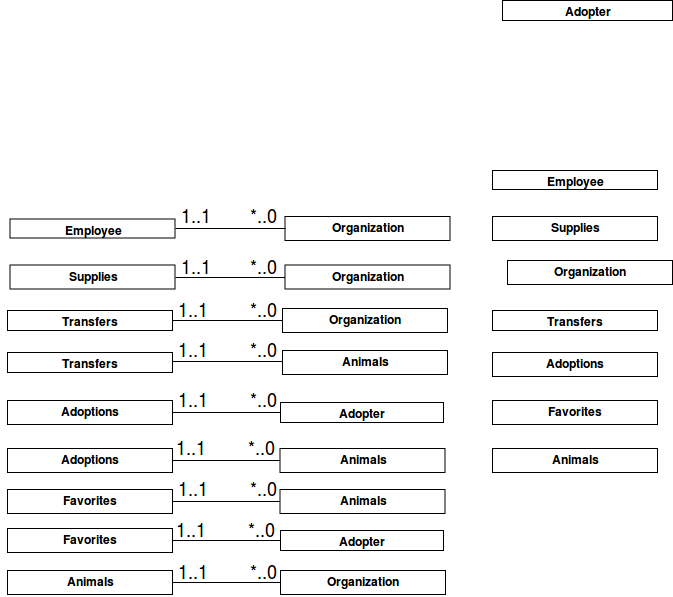
This section is also referred to as the Process Architecture

# Structure

## Software Unit Detailed Design



We have changed some libraries since the last document. We are using SQLAlchemy's convenient models which are python classes that we define that model the Database tables and are our interface to the database. Since our classes so closely relate to the database we don't really have any OOP relations. Our classes are linked by ids and foreign key constraints defined ***in python.*** To show the cardinalities between our classes an ER diagram for the table versions is more appropriate. It would be redundant to redefine the fields of our tables since they're in our data dictionaries. Below is the cardinality constraint diagram for our tables/ mapped python classes.



# Static Relationship of Software Unit

## Run-time Object Instances

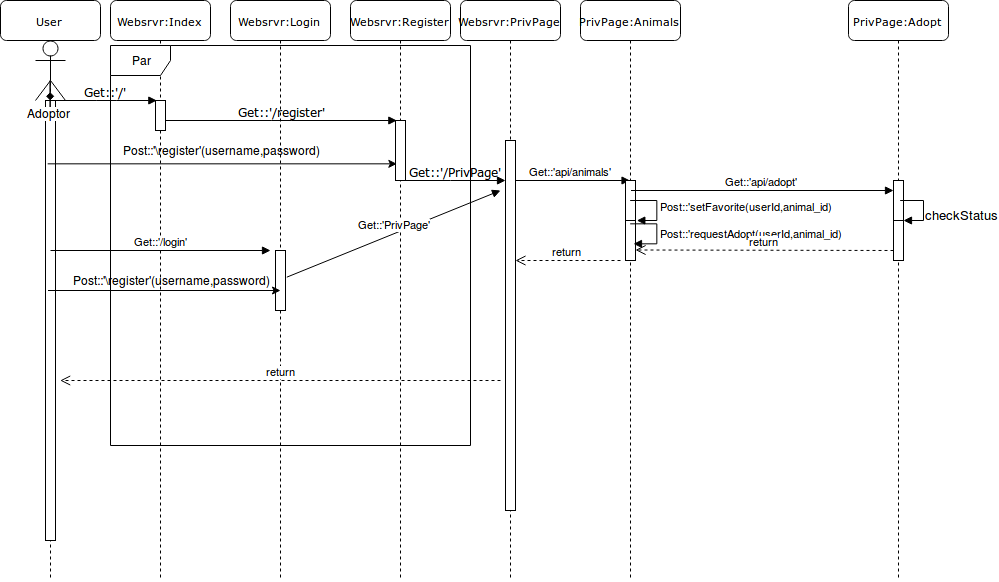
For the time being the server will be running only one thread, so all the object will be within the same thread. For queueing, there is none implemented now. We will be relying on the database for data lookups, so thing will not be populated in memory until accessed. In the future it would be faster to have things stored in memory. For example, when an account is created when should store the account object in memory so that when they login we don't have to make a call to the database and their login is faster. The obvious problem here is that memory space is fairly limited compared to databased space and that would open us up to a DOS attack if we didn’t limit the number of things in memory. When we start storing things in memory we will need queueing mechanisms to keep certain object in memory while dropping others. A likely choice for how we will do this is by trying to keep object that are going to be accessed a lot in memory. Failing that we will keep things in memory that are likely to be accessed soon. When we do implement this, we will have queues of similar object types.

# Behavior

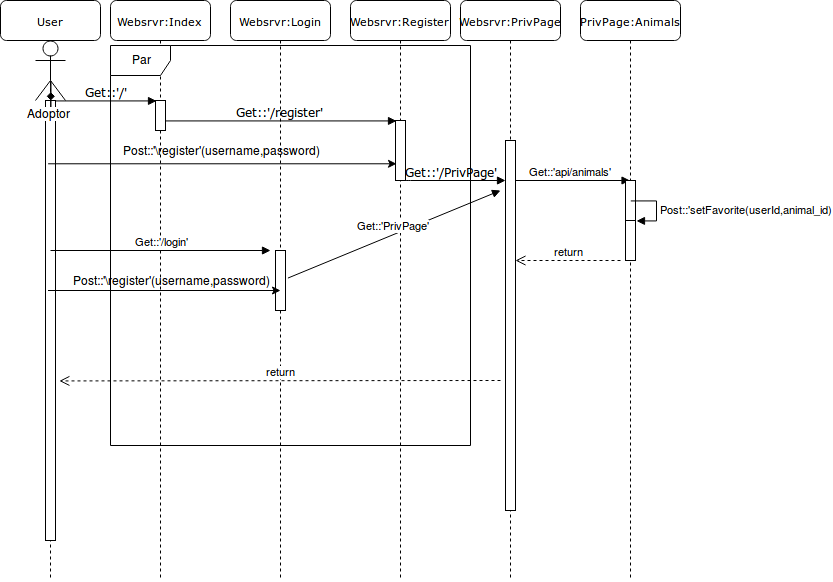
## Sequence Interaction Diagrams

## Visit section 9.1 for a brief explanation of our major functions and requirements. Look at our past documents such as our RAS for use cases

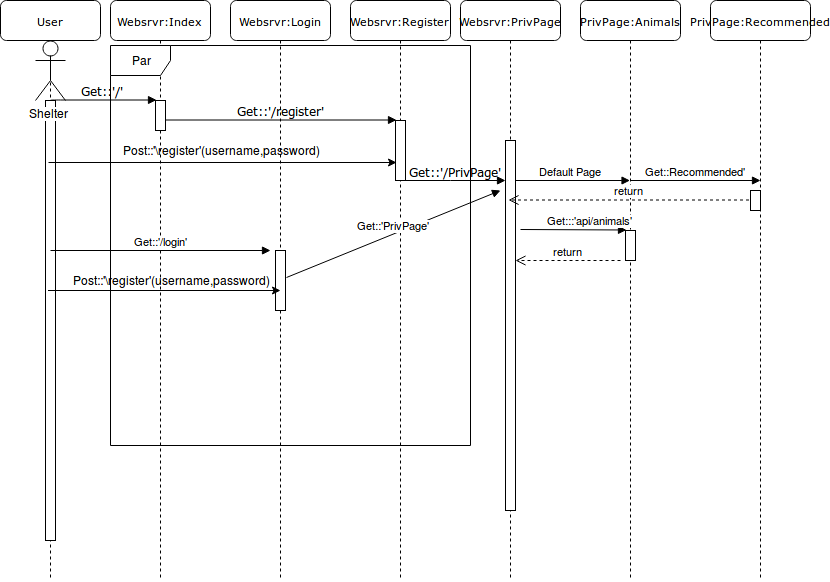
## Sequence Diagram for Adoption



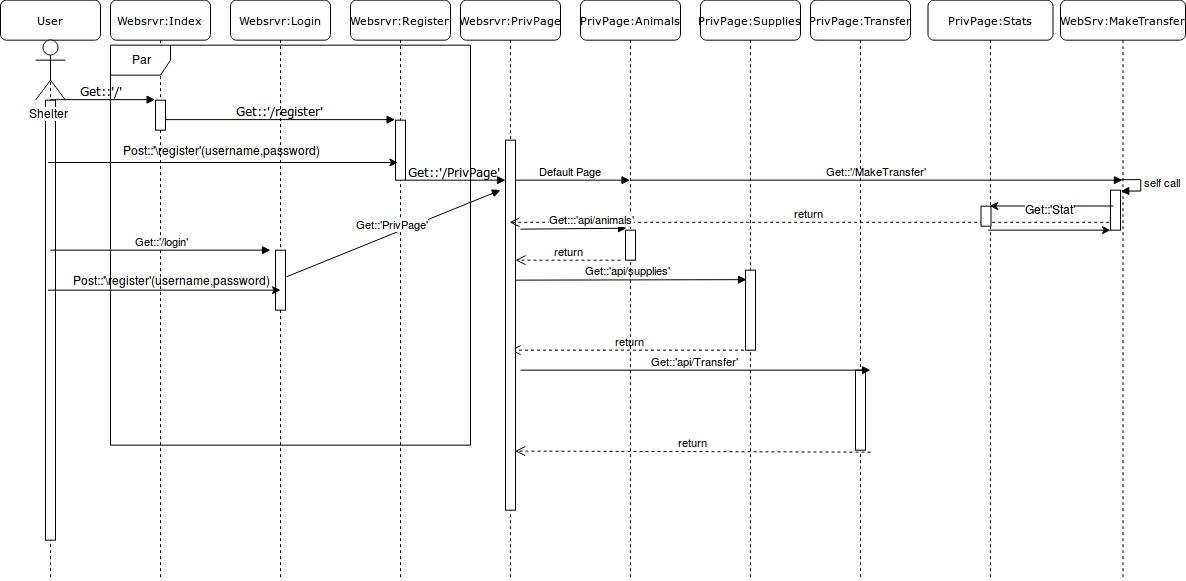
## Sequence Diagram for Favorites



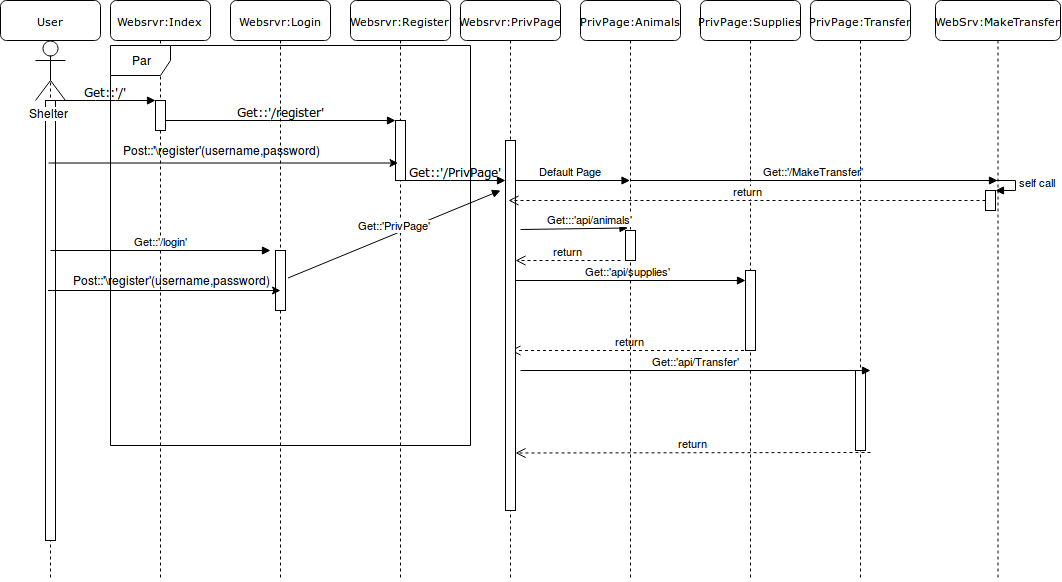
## Sequence Diagram for Matching



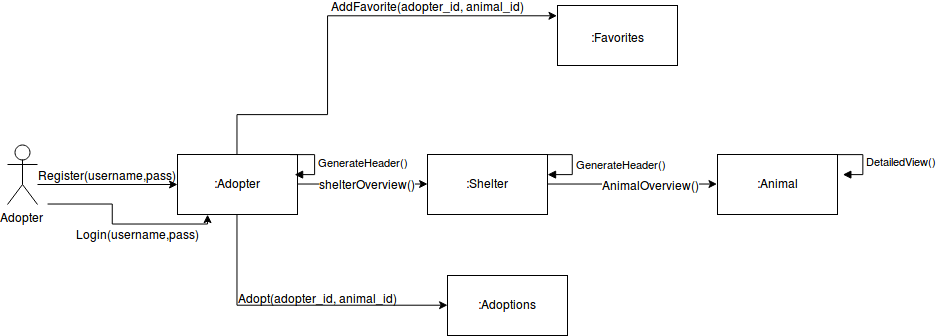
## Sequence Diagram for Stats

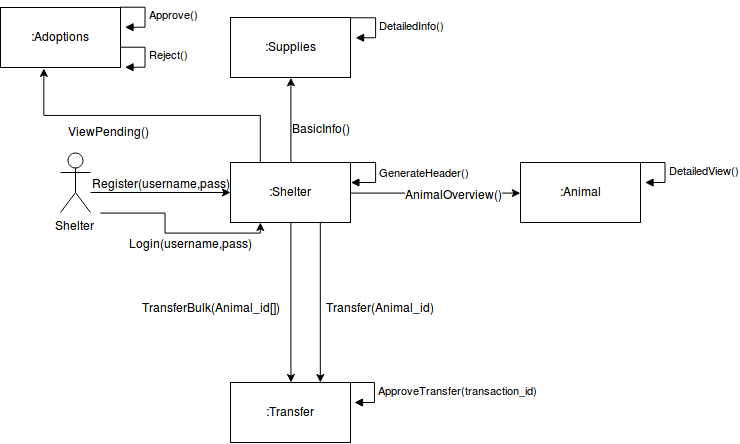


## Sequence Diagram for Transfer

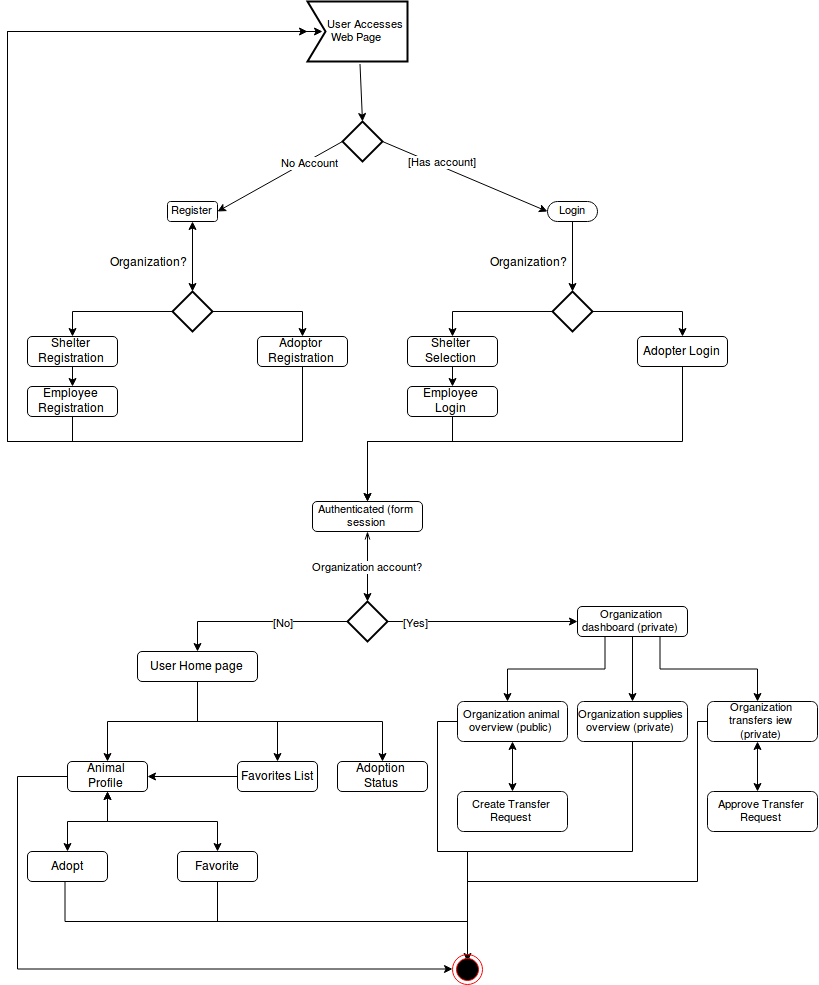


## Collaboration Diagrams





## Activity Diagrams



# Concept of Execution

# The webserver will be run on a computer on a network that allows incoming traffic. If on a home network, then you need to port forward port 5000. The database can be hosted anywhere where the webserver can reach it. The database should be broadcasted so that it can be reached by others. From here the client can access the webserver by the IP address of the computer running the webserver.

# Unzip the file that is submitted. Install Python 2.7, [], Postgres. Open dbInit.py, insertTestData.py, main.py. Replace the string "root":"root" with your Postgres "username":"password". Run command python main.py.

# Frontend components: Client, Application Interface

# The user initiates a request by using the application interface

# After receiving the information/responses from the backend components, the frontend then relays it to the user by either committing to some action or displaying to the screen

# Backend components:

# External: External API

# The user requests some specific information or request from the external API

# The API returns the information requested or returns an error

# Internal: Server, Database

# The user sends a request to retrieve some information from the database or post new information into the database

# Request Information:

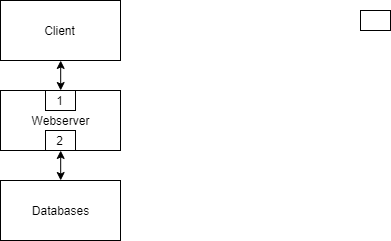
# If the user is requesting information, the database either returns the information using the parameters provided or returns an error if the data isn't found

# Post Information:

# If the user is trying to put new information, the database records the information and sends back a status message

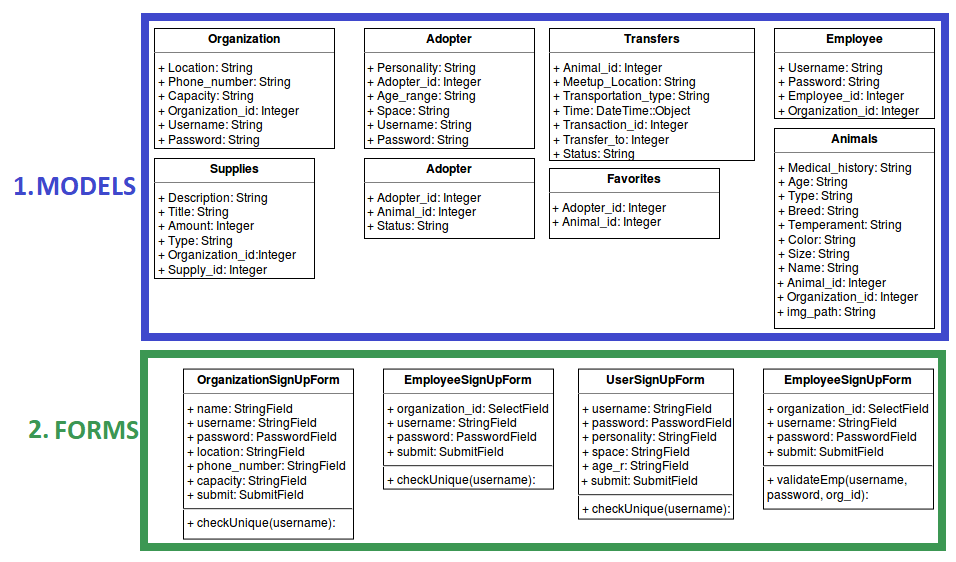
# Interface Design

## Interface Identification and Diagrams



1. Forms
2. Models

## Unique identifier of Interface



# IMPLEMENTATION ARCHITECTURE (NOT REQUIRED)

# *Omitted.*

# All Active and Passive Classes Assigned to Components

# *Omitted.*

# Diagrams of Physical Packaging of Logical Components

# 

**Above is the file layout of our application. The bottom white rectangle (starts with adoptAnimal.html) is our templates group. This group is a collection of all the webpages, which are the user interface for our web application.**

**The next group (Going from the bottom) contains models.py which contain our model classes which make our "model" interface.**

**The group above that consisting of main.py has our routes. This file is the webserver.**

**The group above that is our forms.py which has our forms. This are our "forms" interface. These control how we handle input from the user.**

# DEPLOYMENT ARCHITECTURE

# Physical Deployment Architecture Diagram

# 

As shown above, there are 3 major parts of our application. They can all be separate machines or the same machine (like shown in our demonstration). Regardless, each part has its own requirements.

The client must have at least:

500 megabytes of RAM (preferably 2 gigabytes)

1GHz CPU

200 megabytes of free disk space

0.5 Megabits per second network connection

The ability to run a modern web browser (google chrome, mozilla firefox, microsoft edge) that has JavaScript enabled.

The webserver must have at least:

1 gigabyte of RAM (preferably 2 gigabytes)

1GHz multicore CPU

16 gigabytes of free disk space

1 Megabits per second network connection

The ability to install python, pip, and various python libraries

The database must have at least:

1 gigabyte of RAM (preferably 2 gigabytes)

1GHz CPU

16 gigabytes of free disk space

1 Megabits per second network connection

The ability to install PostgreSQL

# DICTIONARIES

Please refer to Section 13.1 of this document.

# SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION

ShelterHelp will require application interfaces, servers and databases. For every adopter and animal shelter, their computers or laptop devices will act as the application interface, where they will send their requests: add animal, remove animal, transfer animal and for the adopter: adopt animal. For every animal shelter, there would be more and more servers and databases, especially for each town/city. Also, if users become a lot, more and more databases would be needed to store user information.

# REQUIREMENTS TRACEABILITY

# Software Component-Level Requirements Traceability

The 5 major features in the development of our code have been developed and refined throughout the design of our application. All features of our application below all exist in our documentation in one form or another but to be clear they will be briefly re-described in this section. Our 5 major features are:

**Feature**: Transfer

**Short Description**: Allow shelters to transfer an animal to another shelter

**Requirements**: Shelters must be allowed to submit a transfer request to any other shelter. Shelters must be allowed to deny or approve any incoming transfer request. One a transfer is complete the animal must be moved from the source shelter to the destination shelters inventory.

**Feature**: Favorite

**Short Description**: Allow adopters to favorite an animal so that may revisit

them quickly

**Requirements**: Adopters must be able to favorite any animal they desire. Adopters must be able to view their favorited animals in a convenient manner, faster way than can reach all other animals. Adopters must be able to request an adoption for any animal they favorite.

**Feature**: Filter

**Short Description**: Animals should be recommended to adopters based on information about the adopter

**Requirements**: Adopters must be given a set of animals that are predicted to compatible with them. Adopters must be able to favorite or request an adopt any of these animals.

**Feature**: Stats

**Short Description**: Shelters should be able to quickly visualize the current and max capacities of other shelters

**Requirements**: Shelters must be able to view how full other capacities are in an easy to understand and visual manner

**Feature**: Adopt

**Short Description**: Adopters should be able to adopt animals from shelters

**Requirements**: Adopters must be able to request an adoption with a shelter. Adopters must be able to view the status of their pending adoption request.

For tracing software components, each chunk and each file of code will be named well, commented on well and finally, will have a document with each file name and what the file's major function is.

After each completion of code, run test runs to ensure slowly but surely the requirements are being met one by one.

# SYSTEM DESIGN TESTING

To maintain the best quality for our product, our team will commit to weekly inspections and tests on each artifact. Each session will be overseen by our Quality Assurance Manager. To ensure that the best possible product is released each defect will be taken care of as soon as detected.

# RATIONALE

None

# NOTES

None

# APPENDICES

# Dictionaries

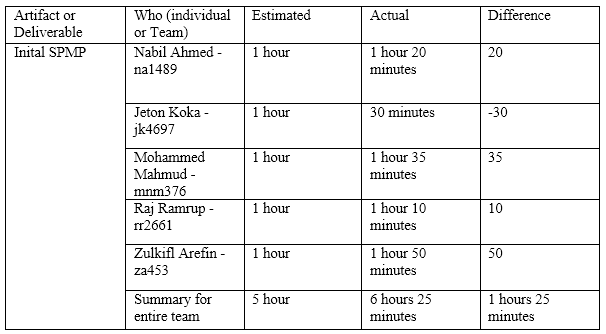
|  |  |
| --- | --- |
| Term | Definition |
| Back End - | Frameworks/services running on our server that allow app interaction to programmer to communicate to each other |
| Database | Collection of data structured to facilitate the storage, retrieval, modification, and deletion of data in conjunction with various dataprocessing operations |
| Front End - | Frameworks/services that allow app interaction with user to communicate to each other |
| JavaScript | Programming language used for OOP on web servers. |
| CSS | Client framework to produce dynamic web pages |
| RAS | Requirements Analysis Specification |
| SDD | Software Design Document |
| SPMP | Software Project Management Plan |

|  |  |  |
| --- | --- | --- |
| Entity Type | Field Name, Data type | Field Description |
| Adopter |  |  |
|  | Personality=Column(String) | Personality description of Adopter entity |
|  | Space=Column(String) | Amount of space Adopter cannot accommodate |
|  | Age\_range=Column(String) | Desired age range of animal |
|  | Adopter\_id=Column(Integer,primary\_key=True,autoincrement=True) | Id for Adopter account |
|  | Username=Column(String,unique=True) | Username for profile creation |
|  | Password=Column(String) | Password hash for profile |
|  |  |  |
|  |  |  |
| Favorites |  |  |
|  | Animal\_id=Column(Integer,ForeignKey('Animals.Animal\_id'),primary\_key=True,nullable=False) | Animal id that Adopter wishes to favorite |
|  | Adopter\_id=Column(Integer,ForeignKey('Adopter.Adopter\_id'),primary\_key=True,nullable=False) | Adopter id that has favorited the respective animal |
|  |  |  |
| Animals |  |  |
|  | Medical\_history=Column(String) | Medical history of animal |
|  | Age=Column(String) | Age of animal |
|  | Temperament=Column(String) | Temperament of animal |
|  | Name=Column(String) | Name of animal |
|  | Animal\_id=Column(Integer,primary\_key=True,autoincrement=True) | Animal id |
|  | Organization\_id=Column(Integer,ForeignKey('Organization.Organization\_id'),nullable=False) | Organization id to which animal belongs to |
|  | img\_path=Column(String) | Path of animal image on server |
|  |  |  |
| Employee |  |  |
|  | Username=Column(String,unique=True) | Username of Employee account |
|  | Password=Column(String) | Password hash of Employee account |
|  | Employee\_id=Column(Integer,primary\_key=True,autoincrement=True) | Employee id within organization |
|  | Organization\_id=Column(Integer,ForeignKey('Organization.Organization\_id'),nullable=False) | Organization id to which Employee belongs to |
|  |  |  |
| Organization |  |  |
|  | Location=Column(String) | Physical location of Organization |
|  | Phone\_number=Column(String) | Phone number of Organization |
|  | Capacity=Column(String) | Animal capacity of Organization |
|  | Organization\_id=Column(Integer,primary\_key=True,autoincrement=True) | Organization id |
|  | Username=Column(String,unique=True) | Master username for organization |
|  | Name=Column(String) | Organization Name |
|  | Password=Column(String) | Master password hash for organization |
|  |  |  |
| Supplies |  |  |
|  | Description=Column(String) | Description of supply item |
|  | Title=Column(String) | Title (name) of supply item |
|  | Amount=Column(String) | Quantity of supply item |
|  | Type=Column(String) | Type of supply item |
|  | Organization\_id=Column(Integer,ForeignKey('Organization.Organization\_id'),nullable=False) | Organization id to which supply item(s) belong to |
|  | Supply\_id=Column(Integer,primary\_key=True,autoincrement=True) | Id of supply item type |
|  |  |  |
| Transfers |  |  |
|  | Animal\_id=Column(Integer,ForeignKey('Animals.Animal\_id'),nullable=False) | Animal id for transfer |
|  | MeetUp\_Location=Column(String) | Planned meet-up location for the transfer to take place |
|  | Transportation\_type=Column(String) | Transportation type that will pick up animal |
|  | Time=Column(String) | Time transferred completed |
|  | Transaction\_id=Column(Integer,primary\_key=True,autoincrement=True) | Id of transaction |
|  | Transfer\_to=Column(Integer,ForeignKey('Organization.Organization\_id'),nullable=False) | Organization id to transfer to |
|  | Status=Column(String) | Status of transfer |
|  |  |  |
| Adoptions |  |  |
|  | Animal\_id=Column(Integer,ForeignKey('Animals.Animal\_id'),primary\_key=True,nullable=False) | Animal id for adoption |
|  | Adopter\_id=Column(Integer,ForeignKey('Adopter.Adopter\_id'),primary\_key=True,nullable=False) | Adopter id that is looking to adopt |
|  | Status=Column(String) | Status of adoption |

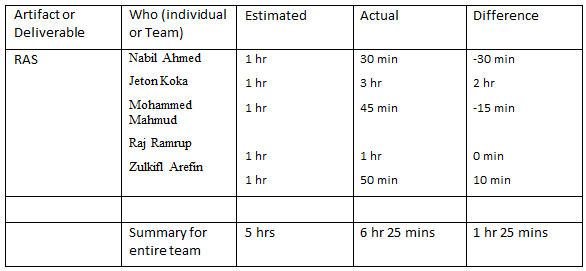
# UML diagrams, if not included in the body of the document

# Schedule Tracking

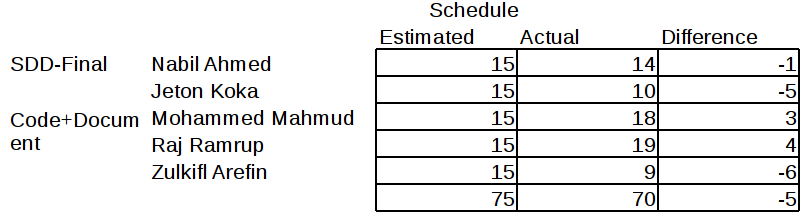
Time (hours)



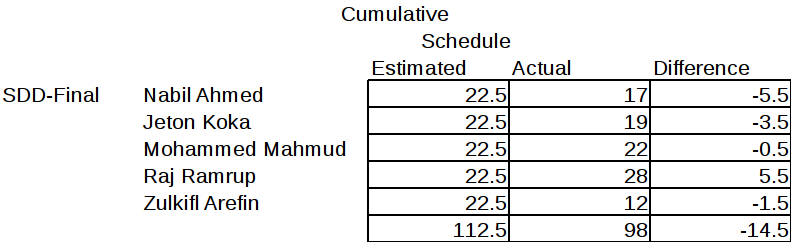
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or Deliverable | Who (individual or Team) | Estimated | Actual | Difference |
| Updated SPMP | Nabil Ahmed - na1489 | 30 minutes | 50 minutes | 20 minutes |
|  | Jeton Koka - jk4697 | 30 minutes | 30 minutes | 0 minutes |
|  | Mohammed Mahmud - mmm376 | 30 minutes | 1 hour | 30 minutes |
|  | Raj Ramrup - rr2661 | 30 minutes | 20 minutes | -10 minutes |
|  | Zulkifl Arefin - za453 | 30 minutes | 1 hour 15 minutes | 45 minutes |
|  | Summary for entire team | 2 hours 30 minutes | 3 hours 55 minutes | 1 hour 25 minutes |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or Deliverable | Who (individual and team) | Estimated | Actual | Difference |
| SDD | Nabil Ahmed | 5 hours | 3 hours | -2 hours |
|  | Jeton Koka | 5 hours | 9 hours | 4 hours |
|  | Mohammed Mahmud | 5 hours | 4 hours | -1 hour |
|  | Raj Ramrup | 5 hours | 9 hours | 4 hours |
|  | Zulkifl Arefin | 5 hours | 3 hours | -2 hours |
|  | Team summary | 25 hours | 28 hours | 3 hours |

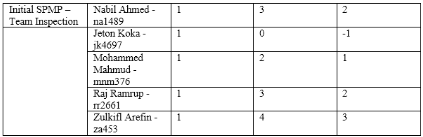


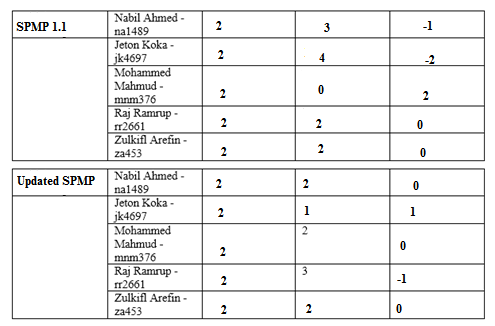
**Cumulative**

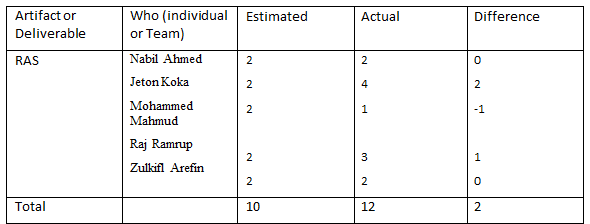


# Defect Tracking

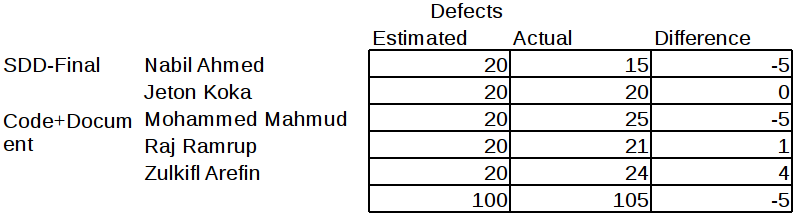
Counts







|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact or Deliverable | Who (individual and team) | Estimated | Actual | Difference |
| SDD | Nabil Ahmed | 2 | 0 | -2 |
|  | Jeton Koka | 2 | 3 | 1 |
|  | Mohammed Mahmud | 2 | 2 | 0 |
|  | Raj Ramrup | 2 | 4 | 2 |
|  | Zulkifl Arefin | 2 | 2 | 0 |
|  | Team summary | 10 | 11 | 1 |



**Cumulative**

