FriendZone

Version 1.0

System/Subsystem/Software Requirements

2/21/2016

Prepared for:

Customer

Prepared by:

Group

Luccif@my.easternct.edu

Web Site

(add service website here)

Table of Contents

1 Introduction................................................................................................................. 1

1.1 Purpose of the System............................................................................................. 1

1.2 Scope of the System................................................................................................ 1

1.3 Objective and Success Criteria of the Project ......................................................... 1

1.4 Definition of Terms and Acronyms ........................................................................ 2

1.5 References ............................................................................................................... 2

2 System Context ........................................................................................................... 2

2.1 System Concept....................................................................................................... 3

2.2 Current System........................................................................................................ 3

2.3 System Design (If this is a component or subsystem of a larger system)............... 3

2.4 System (Subsystem) Overview ............................................................................... 3

3 Requirements .............................................................................................................. 1

3.1 Functional Requirements ........................................................................................ 1

3.2 Nonfunctional Requirements .................................................................................. 2

3.3 Pseudo Requirements ...............................................**Error! Bookmark not defined.**

3.4 Physical Requirements .............................................**Error! Bookmark not defined.**

3.5 Functional (Scenario) Analysis ............................................................................... 2

3.5.1 Actors................................................................................................................... 3

3.5.2 Use-Case X .......................................................................................................... 3

3.6 Analysis Model ....................................................................................................... 4

3.6.1 System Behavior .................................................................................................. 4

3.6.2 Object Structure and Behavior ............................................................................. 5

3.6.3 Data and Data Relations ....................................................................................... 6

Appendix A – Previous Versions........................................................................................ 7

Appendix B– Trade and Feasibility studies ....................................................................... 7

Page ii Team Name

List of Figures & Tables

Figure 1 - System Context Diagram ................................................................................... 4

Page iii Team Name

Revision History

*List all revisions here. Include the date of the revision, a description of what*

*changed, and by whom. Also, each week (or other time period depending on your*

*project), you will turn in a new version of this document. Please indicate any changes*

*from the previous week’s version using Microsoft Word’s change tracking system (which*

*can be turned on/off under the tools – track changes option). Then, accept last weeks*

*changes before starting the new weeks changes*

*Example revision history:*

September 13, 2001 – Added section 1 from Grant. Added section 1.1 from Adam.

September 15, 2001 – Added sections 3.1 – 3.3 from Amy.

September 16, 2001 – Added section 1.2 from Dev and section 2.3 from Scott. Added

Alan’s email address.

September 20, 2001 – Added modified requirements list from Scott.

September 21, 2001 – Added modified requirements list and modified sections 1-3 from

Amy.

September 24, 2001 – Added UI documentation and pieces of section 8.

September 28, 2001 – Added section 5, modified object diagram, and added “Notes” as a

component.

October 3, 2001 – Modified format of requirements section.

October 5, 2001 – Added section 6.

October 7, 2001 – Modified Object Diagram.

November 24, 2001 – Brought document up to date with the actual implementation.

December 1, 2001 – Final revisions made.

**1 INTRODUCTION**

*This document is design to provide the overview of a given system, subsystem, or*

*component. It is designed to support both software-intensive systems and hardwaresoftware*

*systems.*

*The SRDD captures both requirements and architectural design of a given system,*

*subsystem, or component. Typically, in the case of software, details of design and*

*implementation are captured in the code itself. In the case of hardware, details are*

*captured in individual hardware component specs or drawings.*

*Please reference all figures in the text. You must follow the outline numbering below.*

*If there is a section that you do not need, just put “N/A” (and eliminate all subsectons of*

*that section).*

*The introduction section gives the reader an understanding of what this*

*system/subsystem is. Keep this* ***brief*** *– a greater description of the system/subsystem and*

*its concept can be found in the next section. Example introduction:*

Friendzone is a web-based video chat designed to allow users to connect via webcam and chat room services. This system is designed to give the ability to have face to face discussions with friends, family and acquaintances over the internet.

**1.1 Purpose of the System**

*This section expands the introduction and gives the reader a sense of the problem and*

*the players. It should include:*

*Customer(s)*

*End-User(s)*

*Developer(s)*

*Concept statement (Problem/High-Level Solution) – This statement should be a*

*clear, short summary of what you are trying to do with this system.*

The Friendzone video chat system is web service designed to allow users to have text chats with one another while seeing each other through webcam feeds. The developers at Friendzone are working their hardest to provide a simple, easy to use product that is server-client based so that users don’t need to worry about downloading apps or any other software. Users will only need to have a computer with a browser, Internet Access, and a Web cam.

**1.2 Scope of the System**

*Clearly and* ***briefly*** *state the scope of the system. Indicate whether this is a standalone*

*system, a subsystem or component of a larger system.*

Friendzone is a standalone system, created using industry standard web technologies. (HTML,CSS, Angular.JS, Node.JS, Express, MongoDB). The only outside systems needed are a Webcam and a Browser with internet access.

**1.3 Objective and Success Criteria of the Project**

*Describe the central issue or problem. Also describe the general approach to solving*

*the problem. This is a good place to reiterate and expand upon statement of concept*

*(problem/high-level solution). Make sure that you really understand the real problem -*

*customers sometimes hide it under their own perceived solution. Example:*

Friendzone is designed so anyone should be able to sign up to be a member, invite their friends to join and chat with them in a few simple clicks. The main concept is to connect people and allow them to communicate over the internet without needing to download any software.

**1.4 Definition of Terms and Acronyms**

*List all terms with definitions. This section can be made an appendix if it gets too*

*long. Example:*

**App –** Angular app controls all MVC for html, two way data binding and page routing

**blocked user list** – list of all users who have request a block, and the user they have blocked

**blur** – when user clicks away from an HTML element that was in focus, a blur call is made to the webpage

**Client –** The computer running a browser that the User is using to connect to the Friendzone service.

**focus** – when user clicks on an input box or other HTML object, that object is brought into focus on the webpage

**friend –** two people who share a connection over Friendzone servers. Friend is a two-way relationship which allow data exchange

**Friend List** – list of all a user’s friends. stored on the server

**Friend Page** – page that allows User to see searchable friends, and make friend requests

**Friend Request** – one user sends a request to another to make a friend connection across Friendzone servers and share information

**friend requests list** – list kept by the server of all outstanding friend requests that have been sent but not answered

**Home/home page** – main page that an existing User is taken to upon login. This page shows friend list, friend status and video connections

**Icon –** each user can choose from a set of pictures that appears in their user profile and appears next to their screenname on a friend’s home page. Icon can be changed on Profile Page

**ID (request/message)** – a number assigned to server requests for retrieval. Not visible to users.

**Landing Page** – Friendzone Login page is the landing page. When no user has signed in, the app defaults to this page. On logout, user is sent back to this page.

**Login** – act of entering username and password for verification by Friendzone servers

**Logout** – releases all User information from active web page and sends update message to the servers

**Message** – user may send video recordings to their friends for later viewing. Video Messages are stored on servers for retrieval

**message list** – list of all messages that have been sent and received to all users stored by the server. The list has sender’s names, receiver’s names, an ID for retrieval and a link to the video message that was sent.

**MVC** – model-view-controller – design pattern where information is split into the *view* (what the user sees and interacts with) *model* (where information is stored) and *controller* (talks between the two)

**online(value)** – Boolean value set to true if a user is logged in. stored in server

**online** – any User who is logged in

**Password** – a unique character string that each user must create. This allows users to access their information but not another user’s passwords must follow a regular expression

**Password Regex** – user password must contain upper and lower case letters, one number and one special character, must be 4 – 8 characters in length

**Profile –** user settings such as username and screenname. Certain profile settings are visible to a user’s friends

**Profile Page** - where User can change their profile settings

**query –** portion of a server call containing information that pertains to the User. query\_strings are created in the html and javascript

**query\_string** – string objects created in javascript and sent to the server to either post User information, or request information from the server Node Express protocol *‘/request?query=value’*

**regular expression** (**regex**)– for character strings, it defines characters that must be present, how many must be present and if there is a particular order

**Requests(server requests)** – all server requests are handled via *$http.get* and *$http.post* using Node Express protocol *‘/request?query=value’*

**requests –** objects created by the server when it receives an *$http.get* or *$http.post* call from the webpage

**response** – object created by the server in execution of an *$http.get* or *$http.post* and sent back to the webpage

**Route** – all page change requests are handled internally by *Angular app* via *ng-Route* and ‘#/’ routing protocol

**route\_string –** any string containing information that will be passed back to app for a page route

**Screenname** – a name that is displayed to other users in Friendzone. two users may have the same screenname. On signup a user’s screenname defaults from their username

**Screenname Regex –** must be 4 characters long

**Server –** The hardware that the Friendzone service’s software is running on, contains a database and programs that provide functionality

**Signup** – act of a new user providing a unique username, a regular expression password, and valid email address is order to gain access to user portion of Friendzone servers to use Friendzone services.

**Status –** User displayed status lets friends know if you are available to chat, unavailable, or busy. Status can be changed in top-row control bar

**temporary user** – after an email is sent to a new user, before they validate their activation code, information is stored on server as a temporary user

**top-row control bar –** html code that appears at the top of all pages for a User who is logged in control bar allows user to change status, logout, and navigate to other pages

**two-way-data-binding** – user inputs are automatically updated in the model, and changes to the model are reflected in the view without user interaction or page refresh

**user(*general*) –** Any person who is using the Friendzone service.

**User (*specific*)** - person who is signed into Friendzone on their own system, user specific refers to current user

**Username** – a unique name that Friendzone identifies individual users by. No two Friendzone users may share the same username. Username is used for login, and on signup it becomes default screenname

**Username Regex –** name must be at least 4 characters long and must be unique

**user settings** – list of all settings for a User, these are passed from the server to the webpage on login

**user pages** – pages only accessible to users who login

**visible(value)** – Boolean value stored in the server. If a user sets visible to true, other users may see their profile in friend search and send them friend requests

**visible state** – set in a User’s profile section, decides whether other users can find this User by browsing the Friend Page

**webpage** – user portion of Friendzone software written in html, javascript and css, served to user on webbrowsers.

**1.5 References**

*B Dayley, Node.js, MongoDB and AnugluarJS Web Development, Upper Saddle River NJ AddisonWesley, 2014*

*B Green and S Seshadri, AngularJS: Up and Running, Sebastopol CA, 2015*

*Bootstrap Components available:* [*http://getbootstrap.com/components/*](http://getbootstrap.com/components/)

*AngularJS by Google available:* [*https://angularjs.org/*](https://angularjs.org/)

**2 SYSTEM CONTEXT**

*The system context section puts this system/subsystem/component into a larger*

*context. If this is a stand-alone system, then this section will focus on the system’s*

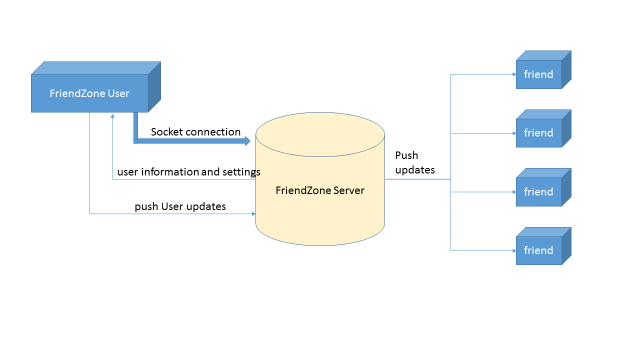
*interfaces to the external world. If this is a subsystem or component, this section will*

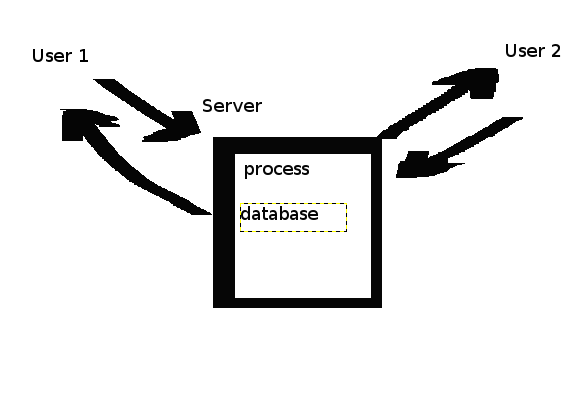
*describe the overall system and how this susbystem or component fits in.*

**2.1 System Concept**

*Describe the overall system concept. Give the reader a good background. This is a*

*good place to put diagrams that show the overall system concepts.*





**Figure 1 - Example System Diagram**

**2.2 Current System**

N/A*.*

**2.3 System Design (If this is a component or subsystem of a larger system)**

N/A

**2.4 System (Subsystem) Overview**

*Briefly describe the new system (or subsystem) - what it will do and how it will fit into*

*the current domain. This description may include a system context diagram. All*

*interfaces between the system and the domain should be shown on the context diagram in*

*simple form. This diagram should be directly extracted from the domain model or, if this*

*is a subsystem, from the system design diagrams. In other words, this is a simplification*

*to allow the user to clearly see the external interfaces.*

Friendzone provides a web chat service to other users over the internet. A person signs up for a membership, logs into the system. They can alter their user status, search their friends list, search for other members, send a request to chat. When the user logs into the server, it updates their friend status and assigned port # in the database and pushes the status change out to everyone on their friends list. When a user requests to chat with someone, the server checks the target user’s status in the database. If the target user is free, a request is generated and presented to the Target User. If the Target User accepts, the Server will update the database with the Target User and the Requesting User’s activity status. The server then takes the port # from the Requesting User and connects it with the Target User to allow transfer of video/chat data stream. If a User closes the browser, ends chat or is unavailable for a short amount of time the User is disconnected, the server changes both Users activity status and releases control of the disconnected users port to be free for another user in the future.

**INCLUDE DIAGRAM HERE!**

**Figure 2 - System Context Diagram**

**3 REQUIREMENTS**

**3.1 Functional Requirements**

*List all high-level functional requirements. These can be broken into groups but*

*avoid any implication of design. The capabilities/functions are best described as a list of*

*“shalls”, for example:*

*1. The software shall gather environmental data periodically*

*1.1. The software shall record temperatures every 5 minutes.*

*1.2. The software shall record pressure every 30 minutes.*

*Requirements can be organized into group or hierarchically. Each requirement*

*should be numbered. Also, sometimes requirements are captured that have not been*

*implemented. If you have requirements that are not implemented in you system, please*

*mark them with a \* and italicize them. Some requirements have specific range or values*

*– note those in the range/value column (this helps prevent the numbers from being lost in*

*the text)/*

*The source of the requirement should also be noted. The source is usually a user or a*

*customer. Be specific! Finally, discuss the verification method to be used to show that*

*the requirement has been meet. Verification methods usually involve some form of*

*formal testing but may also be meet with analysis, design, heritage (i.e. previously shown*

*to be meet).*

*Example:*

**# Description Range/**

**Values**

**Source**

1. Friend\_Zone Signup Functions
   * 1. User provides a new unique Username
     2. User provides new password that meets Regular Expression
     3. User verifies password
     4. User provides valid email address

address is validated through a code, emailed to user, and entered in signup page

1. Friend\_Zone Login Functions
   * 1. User provides their unique Username
     2. User provides their password
     3. Server verifies and allows user to access Friend\_Zone user pages
2. Friend\_Zone profile functions
   * 1. Friend zone stores user profile information for retrieval upon login
     2. Friend\_Zone provides access to some user information to that user’s friends
     3. users can update and change their profile settings and those update
3. Friend\_Zone Friends functions
   * 1. Friend zone stores a list of a user’s friends and shares information and settings between them
     2. Friend\_Zone allows users to decide if other users they are not friends with can search for them
     3. Friend\_Zone allows users to send friend requests to other users. These requests are stored until the receiving users either accepts or rejects them

If rejected the request is destroyed and no other action taken

If accepted a friend link is made between the two users and

* + 1. users will be able to break links between friends through either un-friend and or block functions

Un-Friend function will break friend connection but still leave user visible according to their settings

Block function will break any existing friend connection and leave that user not-visible regardless of their user settings

1. Friend\_Zone Message functions
   * 1. Friend zone users may send video messages to other users they have a friend connection with
     2. Friend\_Zone servers store messages

reference to the message ID are stored with both the sender and receiver

If a sender deletes a message before it is seen by the receiver both references to the ID are destroyed and the message removed from the server

once a receiver views a message both the sender and receiver must “delete” the message in order for the message to be removed from the server

One user deleting a message will only remove their access to the ID

* + 1. Message sent or received can be forwarded to other users and require the new user to be friends with only the user forwarding the message, not necessarily both the original sender and receiver
    2. Friend\_Zone server will alert users when they have unread messages in the top\_row\_controll\_bar and on the friend page

1. Friend\_Zone Video Functions
   * 1. Friend\_Zone will access a users video camera
     2. Friend\_Zone will open a port connection to the User so that friends may connect and view their video stream
     3. Friend\_Zone will allow users to record video, save it and send it specific friends, recordings, known as messages will be stored in the server according to Message Functions
2. Friendzone shall allow users to be grouped
   * 1. Users shall be able to add a person to a group
     2. Users shall be able to delete a person from a group
     3. Users shall be able to delete a group
     4. Users shall be able to Rename a group

**3.2 Nonfunctional Requirements**

*Describe any user-level requirements that are not directly related to functionality.*

*This includes performance, security, modifiability, error handling, hardware restrictions,*

*and physical environment. Also include user interface and human factor constraints,*

*documentation requirements, extreme conditions, quality requirements, and resource*

*constraints. Example:*

**# Description Range/**

**Values**

**Source**

**1** Those not registered with the system shall not be able to enter it. Only users

authorized to enter the system

2

**3.3 Functional (Scenario) Analysis**

*This section provides an in-depth look at the functional requirements. This can be*

*done through a number of methods including scenarios, view-point analysis, and usecase*

*analysis. If you use use-case analysis (recommended for this class), provide one or*

*more overall use-case diagrams. Then include a section describing each use case. Each*

*use-case or scenario may be supplemented by scenarios, state, activity, or sequence*

*diagrams. Remember, this is a* ***functional*** *view of the sytem. Activity diagrams can be*

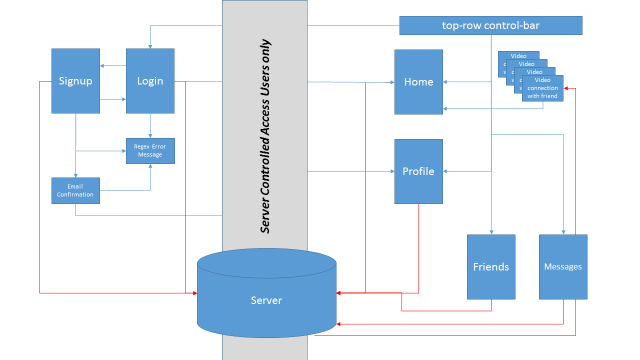
*particularly useful since normally, you haven’t yet defined objects. Sequence diagrams*

*are good for showing the flow of events between subsystems or other identified*

*components (avoid defining internal objects). However, sequence diagrams can be*

*difficult to use since they are based on objects!*

*Insert an overall Use-case diagram here. Example:*

**

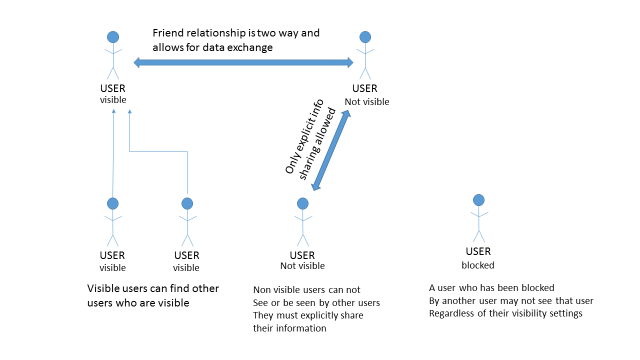
Page 3 Team Name

**3.3.1 Actors**

**Actor:** Friend Zone users

**Purpose:** Friend Zone users update their profiles, interact with “friends”, send video messages to “friends” or chat online via video connection

**Relations**: users can be related in several ways:



**Actor:** The User

**Purpose: “**The User” refers to any FriendZone user who is currently logged in. This relationship specifically refers to their interaction with user interface while they are logged in

**Relations**: The User interacts with the Friendzone server through the interface. User send updates about status and profile, sends and receives friend request and messages, and participates in video chats. All this is accomplished through the interface. Interfaces are connected through the server.

**Actor:** Friend

**Purpose:** Any Friendzone user that the User has a data sharing connection with is referred to as a Friend.

**Relations**: Friends send and receive messages and updates to the user.

**3.3.2 Use-Case 1**

****

****

**Use-Case Name:** Sign-Up

**Participating Actors:** User

**Entry Condition:** User enters Friendzone landing page and navigates to the signup section

**Flow of Events:**

User navigates to signup page. Every input area, for username, password and email have prompt sections which appear when webpage focus is brought to those areas.

User puts new username into input box. Server validates if name is available. User puts new password in input box, webpage verifies against regex. User puts confirmation password in input box. Webpage matches it to password input box. User enters email address in input box. When all fields are valid, webpage activates submit button.

For an invalid inputs, webpage will show user a message either prompting them to enter values in empty fields, or instructions on how to correct invalid entries.

User clicks submit, a message is sent to their email containing an activation code. The new username, password, email and code are stored on the server as a temporary user.

Server sends a response after temp user creation, webpage release a new input for validation code and email. If server cannot send an email to the address provided an error message and prompt for new email will appear on the webpage.

New user puts in the email and validation code and pushes submit button.

Server validates the code and the email address and a new User profile is created. If email and code do not match an error message will appear on the page.

**Exit Condition:** User is sent to Profile page upon completion of successful signup.

**Use-Case 2**

**Use-Case Name: Login**

**Participating Actors:** User

**Entry Condition:** User navigates to Login page. User Logs out. User tries to navigate to any page other than signup while they are not logged in.

**Flow of Events:** Login page has an input for username and one for password. User types in their unique username and password and clicks submit.

Server request is made with values from input fields. Server tries to match username and password. If the server responds with a match, it sends back a true response. Webpage login value is changed to true and User is allowed to navigate Firendzone pages.

If the response comes back with a false, a message appears on the page prompting the user to re-enter username and password, or navigate to sign-up.

**Exit Condition:** Upon successful login User is taken to Home Page

**Use-Case 3**

**Use-Case Name:** Activate Camera

**Participating Actors:** User

**Entry Condition:** User is on Home page, clicks “Start Stream” button

**Flow of Events:** On the Home page, inside the video section, user clicks on “Start Stream” button. An HTML request is sent to the user system to activate the camera device. If this is the first time, or if the user has not set their browser to remember settings the user may need to verify in their browser that they want “Friendzone” webpage to have access to the camera.

If this request is successful a new media object is created and logged with the server. The server sends a request to the User system to open a port for other users to access their video feed. Server will log that port for User friends to access on request.

If the system cannot access the camera or if it is denied on the webpage, a pop up will inform the User that Friendzone cannot access the camera at this time. If the feed is interrupted for any reason, the webpage will send a message to the server terminating the connection and closing the port. Messages will be sent to any connected friends that the connection has been terminated, and the User will be sent a popup telling them that Friendzone lost their video connection, and if they wish to restore it they should navigate to the home page and re-establish the connection. This will start the process from the beginning of this use case.

When the media object is successfully created, it is attached to an HTML object in the video section of the home page. A new tab labeled “My Feed” appears at the bottom and the video feed is shown on the user page.

**Exit Condition:** Connection cannot be made, is lost for any reason, or user clicks “Stop Stream” button, and the server destroys the media object and closes the port.

**Use-Case 4**

**Use-Case Name:** Connect to Friend Video Feed

**Participating Actors:** User, Friend

**Entry Condition:** User clicks on friend link to access their video feed

**Flow of Events:** User goes to their home page and selects a friend whose status is online from the friend list section.

If the User does not already have an active video feed one is created following the steps in ‘Use Case 3’. Upon successful completion of Use Case 3 a request is sent to the server for desired friend’s port connection. If the friend has successfully completed Use Case 3, the server will connect the two systems. The friends Media Object will be sent to the User and attached to an HTML object, placed in the video section and a new tab created with the friend screenname on it.

If the friend does not have an open feed, a message will be sent asking them if they want to connect to User. If the connection is accepted, the friend will go through Use-Case-3 and upon successful completion, follow Use Case 4 again. If not, User will receive a message that their friend is not available and the server will cease connection.

If at any time the connection is lost, both User and friend will be notified and given instructions to go to home page and reestablish connection following Use-Case-3, and Use-Case-4.

**Exit Condition:** User terminates connection by closing their feed or disconnecting friend feed. Message is sent to friend that the user has decided to end connection.

**Use-Case 5**

**Use-Case Name:** Update Profile

**Participating Actors:** User, friends

**Entry Condition:** User navigates to Profile Page

**Flow of Events:** User reaches profile page. Left hand side of screen shows an updated list of profile settings including, username, screenname, password and icon. Each setting has a change button next to it. When User clicks on a change button it brings up the options to change that specific user setting.

When user clicks on change for any profile element an input box and submit button appear in the right hand side of the screen. Number of input boxes may vary based on profile setting.

When submit is pressed, if the regex passed for that element, the webpage will change all instances of User profile setting, then send a message to the server alerting it of the change. The server will inform all User friends of the setting change if it is a piece visible to them. Any users who are online will have their webpages updated on the next refresh cycle. User Setting is always the last change to be made as a check against errors

For screenname, webpage gives the User one input box. The new screenname is checked in the webpage against screenname regex.

For password, User must input a new password in one input box, and match that input in a second. Webpage will check password against password regex, and check the password match against the password.

Email change will give the user one input box which checks for an input and then sends a confirmation email.

Icon change gives the user a list of all available icons. Clicking on any Icon will change all instances of User icon to new selected one.

Username cannot be changed.

On any server errors if the, server cannot update the User change for any reason, the server will first ensure that all instances in the server of all User profile settings match. If a non-match is found, server will change all instances of the profile setting based off User Setting object. Messages will be sent to User friends to update their pages.

A message will then be sent to User webpage updating the profile setting based on User Settings. Finally a message will appear on the User screen and in their message box telling them that an error occurred in the Profile update. This way the user can review their profile settings again and make any more changes they want.

**Exit Condition:** User navigates away from Profile page.

**Use-Case 6**

**Use-Case Name:** Send Messages

**Participating Actors:** User, friend

**Entry Condition:** Upon completion of Use-Case-3, User can record and send video

**Flow of Events:** User opens a good link to their system video camera and the server has registered a media object and port. From the home page User can select the record video button. When pressed, Friendzone will create a recorded video object with a start time and direct user system to begin capturing the current video feed. When User pushes stop record button Firendzone will tell the system camera to stop capturing video and create a video object from the recorded video that is sent to the server and stored.

User now has the option to attach friend’s screennames to the video object and send it as a video message. A url request is created, the video object is moved from stored videos in the server to sent messages, each sender and receiver’s username is stored with the video object ID, this allows them access to retrieve it through their messages. When a user “sends” the video and the server creates or updates the video object it alerts each person attached to the new update through their message center. Those users now have access to the video object in the server to either view, download or forward to other users they are friends with.

Any user with access to the message by virtue of being the original sender, or a receiver of the message may destroy their own link to the message by choosing “delete message” in their own message storage. When delete message has been selected, the user who selected it has their username removed from the video object in the server. If every attached username is removed, by those users, the video object will be deleted from the server.

If the original sender deletes the message before any other user has retrieved it, the video object will be deleted from the server along with all links to it.

**Exit Condition:** Server creates or updates a video object in the server and alerts users to its presence.

**Use-Case 7**

**Use-Case Name:** RetrieveMessage

**Participating Actors:** User

**Entry Condition:** User Navigates to Message Page

**Flow of Events:** If a User has video messages they can navigate to their message page to retrieve them. On the message page they will be shown a list of all the messages they have access to on the server. By clicking on one of those messages a request will be sent to the server to retrieve the message. The message will be displayed on the User’s webpage. By navigating away the User will remove that instance from their webpage but it remains with their link to it in the server. If the User clicks the delete button, a request is sent to the server to remove the User username from the access list for that message. If this User is the only user with access to it, the message will be deleted from the server.

**Exit Condition:** User navigates away from Message page

**Use-Case 8**

**Use-Case Name:** Friend Request

**Participating Actors:** User, friend

**Entry Condition:** User navigates to Friend Page

**Flow of Events:** User’s Friend Page shows them a list of all users they are not currently friend’s with, that have their visible state set to visible. Users may also search for another user through their screenname or username. A user who has their visible state set to not visible can only be found by searching for their username.

When a User clicks on another user icon they receive a message asking them to confirm they want to send this user a friend request. If they click yes a request will be sent to the server with the User username, the recipient username. The server will create an ID and store the object in the requests list.

When a user with a friend request logs in, the server will update all pending friend requests and flag them in the top bar controller. When a User responds to a friend request they can accept or deny it. If the User accepts, the request object in the server is deleted and a new friend connection is created between the sender and receiver. Both web pages and friend lists are updated.

If the request is denied, the request object is deleted and no further action taken.

A friend status can be terminated at any time through the home page friend list by clicking the x above the friend icon. In this case a request is sent to the server and the friend link is deleted. Both webpages a friend lists are updated by the server. The link may be terminated by either user at any time and does not require input from both users. A severed link can be re-established through the normal friend-request method.

A User can also block another user. If one User requests a block against another user, a request is sent to the server. Those users are added to the blocked user list as the requester, and the user they blocked. These two users are no longer visible to each other. They may not send messages to each other, may not friend each other or find each other’s profile on the friend page, and may not search each other’s username/screenname.

An email must be sent to admin to have a block lifted, and may only be requested by the user that initiated the block. A confirmation email will be sent to the email stored on the server and if the user responds, the two block will be removed from the list on the server.

A User can not block another user who has already blocked them.

**Exit Condition:** xxxxx

**Use-Case 1**

**Use-Case Name:** Sign-Up

**Participating Actors:** xxxxx

**Entry Condition:** xxxxx

**Flow of Events:** xxxxx (could be shown with activity, sequence, or state diagram).

Remember, the flow of events may be dependent on external events/responses. Use

scenarios for describing specific event/response flows.

**Exit Condition:** xxxxx

***Special Requirements:*** *xxxxx*

**3.3.2.1 Scenario X.1**

*A scenario is a specific set of events and responses of the more generalized use-case.*

*Sequence diagrams, activity diagrams, or an English description of the events and*

*responses are appropriate.*

Page 4 Team Name

*Add additional scenario sections (3.4.2.2, 3.4.2.3, etc.) as needed. Do not include*

*this section is you do not have any scenarios.*

*Examples of activity diagrams:*

*Examples of sequence diagrams:*

**3.4 Analysis Model**

*The analysis model is used to expand our understanding of the system (or subsystem).*

*Analysis should provide a better understanding of how the systems looks, what it contains*

*(objects, data), what their relationships are, and how it behaves. This section is used to*

*define the system* ***as understood by the outside world****!*

*A wide variety of methods may be used here. I have outlined three sections below*

*that I would recommend that you start with. Feel free to add additional models, sections*

*as you find useful.*

**3.4.1 System Behavior**

*Show the overall system behavior through a top-level state or activity diagram. Then*

*decompose the various states as useful. However, you may find that beyond the top-level,*

*it is easier to use the Object Structure and Behavior section below to decompose states.*

*State diagram example:*

Page 5 Team Name

POD

Communication

Send\_Via\_Collision

Non-Communication

/ Compute Expected Range

to Transfer Node

Believed\_to\_be\_in\_range

Listen

Pre-Transmit

PreTransmitDelay = random

[ PreTransmitDelayMin : PreTransmitDelayMax ]

{Recompute each time we enter or reenter state}

TransferNodePacket\_Received

Hear\_Another\_POD

Transmit

[ t > PreTransmitDelay and

Enough time to transmit]

Post-Transmit

Collision\_delay = t\_delay\_max

EndOfTransmit

Hear\_Another\_POD

[t <= Collision\_delay]

POD\_Believes\_Out\_Of\_Range

: Mark Data as not sent

[Data\_to\_be\_sent]

[Command\_For\_This\_POD]

Execute\_Command

[Data\_To\_Be\_Sent]

Decode

/ Decode TransferNodePacket

[Data\_To\_Be\_Sent]

Wait\_For\_Timeslice

Start\_Open\_Timeslices

[No\_Allocated\_Timeslice or

Transmit\_Failure]

Transmit

Start\_Allocated\_Timeslice

[Allocated\_This\_Timeslice]

Transmit\_Failure

Wait\_For\_Next\_Cycle

Decode

/ Decode TransferNodePacket

Transmit\_Success

TransferNodePacket\_Received

Packet\_Acknowledged / Mark Data as Sent

Packet\_Not\_Acknowledged

TransferNodePacket\_Not\_Recieved

[ Believe TransferNode Not in Range]

TransferNodePacket\_Not\_Recieved

[ Believe TransferNode in Range]

TransferNodePacket\_Not\_Recieved

[ Believe TransferNode in Range]

TransferNodePacket\_Not\_Recieved

[ Believe TransferNode Not in Range]

/ Mark any data as not sent

End\_Open\_Time\_Slices

End\_Open\_Time\_Slices

[ t>Collision\_Delay]

**Figure 3 - Example State Diagram**

**3.4.2 Object Structure and Behavior**

*This may include objects, components, or data items. For example, you might have*

*an object diagram that shows various data items and their relationships. You might use*

*an object diagram.* ***REMEMBER, THIS IS AS UNDERSTOOD BY THE OUTSIDE***

***WORLD!!!! IT IS NOT MEANT TO BE YOUR SOFTWARE DESIGN!!!1***

*Object Model Example:*

1 However, it is often the case that the software design does “fall out” of an analysis

model

Page 6 Team Name

**System Overview Diagram**

**Palantir-B**

Flight Software

Sensor Manager Power Manager

Event Manager

Data Manager COMM Manager Actual COMM

(Simulated in our

system)

Ground Station

Camera

Actual Sensors

Motion Detector

GPS/Altimiter

Humidity Sensor

Temperature

Wind

Microphone

Other

Schedule COMM

event, new event table

Operate,

Save State

Sensor data

Adjust

Parameters

Power

Status

Send/Receive from

Satellite, Save state

Have pending data?

data to be delivered

Save Data

COMM Manager

Send Data

Receive Data

Send

Data

Receive

Data

Data Bank

Event Manager

Send Event

Table

User Interface

command

Get

Current

Data

Save

Data

Actual System

Remote Site

Sensors

Flight

Software Ground

Station /

UI

Micro

Satellites

**Figure 4 - Example Object Diagram**

**3.4.2.1 Object Descriptions**

*For each element, include:*

***Description and Purpose*** *– Provide an external description of the element.*

***Characteristics*** *– List the various object characteristics and their possible values*

***Object Behavior*** *– Describe the object states that the external user would wish to*

*know. Feel free to use orthogonal states to represent different aspects of the*

*component.*

**3.4.3 Data and Data Relations**

*Another approach would be to model system data and their relationships – a good*

*starting point for data intensive systems like databases.*

*Data model example:*

Page 7 Team Name

**Figure 5 - Example Data Model**

**APPENDIX A – Previous Versions**

*This section is used to capture information about previous versions of the*

*system(subsystem). You should provide the following for the immediately previous*

*version:*

*Status of previous version (what was implemented, etc.)*

*Issues with the previous version*

*Analysis Model of the previous version*

*Functional Model*

*Structural Model*

*Behavior Model*

*Architectural Design Model of the previous version (Analysis and Design Models*

*may be merged).*

**APPENDIX B – TRADE AND FEASIBILITY STUDIES**

Page 8 Team Name

*Describe all trade and feasibility studies. Include all information relavent to making*

*requirement. IT IS CRITICAL THAT YOU INCLUDE INFORMATION NOT JUST*

*ABOUT WHY YOU CHOSE ONE WAY, BUT ALSO INFORMATION ABOUT WHY YOU*

*DID NOT CHOOSE ANOTHER WAY. Otherwise, subsequent developers may keep*

*revisiting the same ideas over and over again.*

*Trade and feasibility study information may be captured in the appropriate section*

*above or here in sub-appendix sections.*