**Requirements Analysis Document**

Study Bear

CSCI 4712 Senior Capstone Project

Spring 2015

Georgia Regents University

Augusta, GA

Date: 2/1/15

Version 0

Team Members

Jeremy Scott

Chad Reynolds

Darcie Odom

**Abstract**

This document contains the requirements, analysis and design artifacts for Study Bear. Study Bear is a software system designed to aid its users in finding study partners for their class work.

The users will register for the system using an Android application. Once they have access, they will be able to enter or update information for the system to use to match them up with study partners. Matched users will be able to communicate using a message inbox system. Users will be also be able to manually search for other users.

This document describes the requirements, analysis and design of the Study Bear. The rest of this document is structured as follows. Chapter 1 contains the introduction. This chapter presents a brief description of the system. Chapter 2 outlines the functional requirements of the system.

**Table of Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | INTRODUCTION…………………………………………………...... | | | 5 |
|  | 1.1 | OVERVIEW OF SYSTEM…………………………………………... | 5 |  |
|  | 1.2 | SCOPE OF SYSTEM………………………………………………… | 5 |  |
|  | 1.3 | OVERVIEW OF DOCUMENT……………………………………… | 5 |  |
| 2 | REQUIREMENTS OF SYSTEM…………………………………...…. | | | 6 |
|  | 2.1 | FUNCTIONAL REQUIREMENTS………………………………..….. | 6 |  |
|  | 2.2 | USE CASES…………………………………………………………….. | 7 |  |
|  | 2.3 | USE CASE DESCRIPTIONS………………………………………….. | 8 |  |
|  | 2.4 | REQUIREMENTS ANALYSIS………………………………………… | 14 |  |
|  |  | | |  |
|  |  |  |  |  |

# Introduction

## Overview of System

Study Bear is a system for finding study partners. Its actors are *Users*, the actual users of the system. The matchmaking system gives the functionality to find and plan with study partners with minimal effort on the part of the user.

Study Bear allows the users to register new accounts in the system, input and manage information about what classes they are taking and have taken, be matched automatically by the system with other users based on their information, and also search for other users manually by name and email. Study Bear is accessed through an Android capable device running the Study Bear application, and is supported by a backend server accessed via network connection.

## Scope of system

The matchmaking system provides the functionality for users to manage accounts and information to aid the system in finding study matches. The data store is externally located on a server accessed through a network. The system requires this server for data storage, processing and generating responses to user matchmaking and search requests, and out-of-band communication with the user for features like account validation and password resetting.

## overview of document

The rest of this document is structured as follows. Chapter 2 outlines the functional requirements of the system. Within this chapter is a list of functional requirements of Study Bear. It also includes a use case model of those functional requirements. A detailed description of each functional requirement then follows.

# requirements of system

## Functional Requirements (so many, where can we simplify and condense?)

## Login -

## Logout -

## RegisterAccount -

## PasswordReset -

## ValidateUserAccount -

## SearchUsers -

## FindStudyPartner -

## ViewMessages -

## ComposeMessage -

## SendMessage -

## DeleteMessage -

## ManageBlockList -

## UpdateUserProfile -

## EditClasses -

## AddToBlockList -

## RemoveFromBlockList -

## (to-Do) NavigationBar -

## (To-Do) ShowUserProfile - need to add ShowUserProfile use case and NavBar use case

|  |  |
| --- | --- |
| *Use case name* | FindStudyPartner |
| *Participating*  *actors* | User |
| *Flow of events* | 1. The User initializes use case by clicking on the “match” button on the user profile screen.  2. StudyBear replaces the current screen with the FindStudyPartner screen that initially has a loading matches message and a back button. Then StudyBear sends a request to the server to find matches for the User.  The server searches the data store for matches to the User, returning the results of the search to StudyBear.  StudyBear displays the first match to the user.  3. The User has three different responses to the match: positive, negative, or block.  4. StudyBear temporarily stores the User’s response and reacts in one of three ways:  a. Positive - The SendMessage use case is initialized from the User to the match they selected  b. Negative/Block - StudyBear displays the next match  c. Out of Matches - StudyBear sends the temporarily stored responses of the user to the server for saving in the data store, and then sends another request to find matches for the User.  5. The User clicks the back button to indicate they are done on the FindStudyPartner screen.  6. StudyBear sends the temporarily stored responses of the User to the server, and then closes the FindStudyPartner screen and displays the user profile screen  The server stores the User responses in the data store. |
| *Entry condition* | * The User has logged in to StudyBear (or currently viewing some screen). |
| *Exit condition* | * The User has started the SendMessage use case or is on some screen. |
| *Quality*  *requirements* | * The server should not spend more than x amount of time searching for matches. |

Instead of match/back buttons on the user profile screen there should be a reference to the navigation bar

|  |  |
| --- | --- |
| *Use case name* | PasswordReset |
| *Participating*  *actors* | User |
| *Flow of events* | 1. The User initializes use case by entering an email and clicking the password reset button on the Login screen.  2. StudyBear sends the password reset request to the server and displays a message to the User that a password reset email is being sent. The server generates a reset code linked to the User’s account. The server sends an email with a hyperlink using the reset code for resetting the User’s password.  3. The User receives the email and visits the password resetting page.  4. The server displays a page based on the reset code in the hyperlink: an input box for entering a new password an input box for confirming the new password, and a button for submitting.  5. The User enters a new password and clicks the submit button.  6. The server queries the database with the User submitted information and the reset code, verifying if the password should be reset.  a. If the server verifies the information, then the new User password is saved in the data store. The server displays a success message to the User on a new page.  b. If the server does not verify the information, then the server displays an error message to the User on a new page. |
| *Entry condition* | * The User is on the Login screen. |
| *Exit condition* | * The User’s password has been reset. |
| *Quality*  *requirements* | * Reset code should only be valid for a limited amount of time. |

Update wording around 5/6 where two passwords are entered to make this more clear

|  |  |
| --- | --- |
| *Use case name* | SearchUsers |
| *Participating*  *actors* | User |
| *Flow of events* | 1. The User intializes the use case by clicking on the search button on the user profile screen.  2. StudyBear closes the user profile screen and brings up the search screen with an input box and submit button.  3. The User enters a name or email in the input box and clicks the submit button.  4. StudyBear displays a message that results are loading and then sends a request to the server with the user input.  The server searches for user names and emails that are like the user input and returns them as a result to StudyBear.  StudyBear updates the screen to display the results as a list for the user to view.  5. The User can click on a result or hit the back button.  6. StudyBear either:  a. Initializes the SendMessage use case with the user chosen by the searching User.  b. Closes the search screen and displays the user profile screen. |
| *Entry condition* | * The User has logged in to StudyBear. |
| *Exit condition* |  |
| *Quality*  *requirements* |  |

update based on navigation bar

|  |  |
| --- | --- |
| *Use case name* | ValidateUserAccount |
| *Participating*  *actors* | User |
| *Flow of events* | 1. The server initializes this use case after creating a new account in the pending state. The server generates a validation code for the new user’s account. Then it sends an email to the user with a hyperlink using the generated validation code.  2. The User receives the email and visits the account validation link.  3. The server displays a page based on the validation code in the hyperlink: an input box for the user to enter their email address and a submit button.  4. The User enters their email address and hits the submit button.  5. The server queries the data store with the entered information and the validation code to verify if the account should be validated.  a. If the validation code and user email match, then the account is updated from pending validation to full status and the user is able to log in to their user profile page on StudyBear.  The User is taken to a new page with a message displaying that their account is validated and that they can now log in.  b. If the validation code or user email do not match, then the user is taken to a new page with a message displaying that the entered information was not valid and the user should try again. |
| *Entry condition* | * The user has completed the RegisterAccount use case in StudyBear. |
| *Exit condition* |  |
| *Quality*  *requirements* |  |

There should be a way to get the email sent again, or for the pending account to be wiped after x amount of days to not lock the user out of the system if something goes wrong.

|  |  |
| --- | --- |
| *Use case name* | Login |
| *Participating*  *actors* | User |
| *Flow of events* | 1. StudyBear displays the login form. The login form contains two textboxes for username and password, register link, and a login button.  2. Users enter their login information and then clicks the login button  3. StudyBear queries webserver using the user’s login information and checks for validity.  a. If invalid user information is submitted, StudyBear displays an error message and the user is able to repeat the login process again.  b. If valid login information is submitted, StudyBear closes login screen and displays the User’s profile |
| *Entry condition* |  |
| *Exit condition* | * The User’s profile screen is shown |
| *Quality*  *requirements* |  |

There should be more detail about how the server is queried. The server is part of the system that we are implementing and we have to account for it.

Message displayed if the account is created but unvalidated.

|  |  |
| --- | --- |
| *Use case name* | Logout |
| *Participating*  *actors* | User |
| *Flow of events* | 1. The user initializes use case by clicking logout button from the menu screen.  2. StudyBear closes menu screen and initializes the Login use case. |
| *Entry condition* | * The user is currently viewing the menu screen. |
| *Exit condition* | * The user is logged out. |
| *Quality*  *requirements* |  |

|  |  |
| --- | --- |
| *Use case name* | ViewMessages |
| *Participating*  *actors* | User |
| *Flow of events* | 1. The User initializes use case by clicking the Messages button from the menu bar.  2. StudyBear displays the Messages form to the User. Messaging form contains an icon to compose a message, a button to view inbox messages, and a button to view outbox messages.  a. If the User clicks the inbox button, StudyBear displays a list of received messages from other Users.  i. If the User selects a message thread, StudyBear displays the messages in a text view.  ii. If the User holds down on a message thread, the User can tap the delete the message option. If tapped, the DeleteMessage use case is initiated.  b. If the User clicks the outbox button, StudyBear displays a list of messages sent to other Users.  i. If the User selects a message, StudyBear displays the messages in a text view.  ii. If the User holds down on a message thread, the User can tap the delete the message option. If tapped, the DeleteMessage use case is initiated.  c. If the User clicks the compose message button, the ComposeMessage use case is initiated. |
| *Entry condition* | * The User is at a screen that contains the menu bar |
| *Exit condition* | * The User is at a screen that is not within messaging |
| *Quality*  *requirements* |  |

Could the view message screen be skipped and the user is taken directly to an inbox/outbox screen in order to simplify the process, and condensing the repetition of options between the two?

All user interaction steps must be listed in the left column, followed by a system response. a, b, c, and their sub-sections all include responses to User input without any User input being listed. There should be multiple back and forths.

We should remove any implementation specific details such as tapping, holding, and other gestures. Those lock in choices unless we update the documentation. We can list that the user has multiple options without saying how they choose them. FindStudyPartner has an example of this.

Steps need to be included for retrieving messages through the server from the data store.

|  |  |
| --- | --- |
| *Use case name* | DeleteMessage |
| *Participating*  *actors* | User |
| *Flow of events* | 1. User initiates use case by holding down on a message in the inbox or outbox.  2. StudyBear displays a delete message button.  3. User clicks the delete message button.  4. StudyBear sends a delete message request to the server.  Server deletes messages from data store and returns list of remaining messages. |
| *Entry condition* | * StudyBear is displaying a list of messages |
| *Exit condition* | * StudyBear is displaying a list of messages |
| *Quality*  *requirements* |  |

|  |  |
| --- | --- |
| *Use case name* | ComposeMessage |
| *Participating*  *actors* | User |
| *Flow of events* | 1. User initiates use case by clicking the compose message button from the messaging form or by clicking the message button on a Users profile.  2. StudyBear displays the compose message form to the user. Compose message form contains a textbox for the recipient, a text view to write a message, and a send button.  a. If the use case is initiated from the messaging form, the recipient field is blank.  i. User fills in the recipient field and types a message in the text view.  ii. User clicks the send button. SendMessage use case is initiated.  b. If the use case is initiated from a user profile, the recipient field is pre-populated with the User’s profile.  i. User fills in the recipient field and types a message in the text view.  3. User clicks the send button. SendMessage use case is initiated. |
| *Entry condition* | * StudyBear is displaying messaging form or another User’s profile. |
| *Exit condition* | * Message was sent or the User pressed back to retrieve previous screen. |
| *Quality*  *requirements* |  |

See notes on ViewMessages about showing all User input and server steps

|  |  |
| --- | --- |
| *Use case name* | SendMessage |
| *Participating*  *actors* | User |
| *Flow of events* | 1. User initiates use case by pressing the send button from the Compose Message Form.  2. StudyBear makes a send message request to the server.  3. Server saves message in data store for the addressed User.  4. Server sends notification to recipient of the message. |
| *Entry condition* | * Study Bear is displaying Compose Message Form |
| *Exit condition* | * Notification sent to recipient |
| *Quality*  *requirements* |  |

|  |  |
| --- | --- |
| *Use case name* | RegisterAccount |
| *Participating actors* | User |
| *Flow of events* | 1. StudyBear displays the login screen. The login screen contains two textboxes for username and password, a sign up link, a forgot password link, and a login button.  2. User clicks the sign up link.  3. StudyBear displays the create/update profile screen: text boxes to enter username, email, password, confirm password, name, and bio, an area to upload a photo, a save button, and a back button.  4. User enters information and clicks the save button.  5. StudyBear sends user entered information to the server.  The server saves the user information in the data store, setting the account status to “pending email validation.” The server responds to StudyBear that the account has been created and initiates the ValidateUserAccount use case.  StudyBear then displays a message that the User must validate their email.  6. The User acknowledges the message  7. StudyBear closes the create/update profile screen and displays the login screen. |
| *Entry condition* | * Login Screen is displayed |
| *Exit condition* | * Login Screen is displayed |
| *Quality*  *requirements* |  |

I do not believe we need implementation details such as which fields are options. –Removed implementation details DO

I updated the end based on the ValidateUserAccount use case.

We do not want to include specific details about how messages are displayed or what text they contain. Those details lock us in during the coding. Staying general allows us to shift design/implementation details without having to update the documentation. –Got it DO

|  |  |
| --- | --- |
| *Use case name* | UpdateUserProfile |
| *Participating*  *actors* | User |
| *Flow of events* | 1. StudyBear displays screen with menu bar  2. User clicks settings button. Menu is displayed. User clicks the Update User Profile link.  3. StudyBear displays Create/Update Profile Screen: text boxes to enter username, email, password, confirm password, name, and bio, an area to upload a photo, and a save button. The server queries the data store. If User has already saved information in the data store, that information is loaded to the form.  4. User enters information and clicks save button.  6.  a. If invalid information has been submitted, StudyBear shows highlighted fields with errors and User can try again.  b. Once valid information is submitted and verified, the server stores information in the data store. |
| *Entry condition* | * User is logged in to StudyBear |
| *Exit condition* |  |
| *Quality*  *requirements* |  |

See notes on RegisterAccount, general is better than specific. What if we want to change from a gear to another image? Or if a name becomes required? –Removed implementation details DO

Both steps are still on the same side of the action. A new number denotes that the use case has moved from actor actions to system response or vice-versa –Got it DO

\*Should have steps that reach out to the server and retrieve user profile information for loading –Got it DO

|  |  |
| --- | --- |
| *Use case name* | ManageBlockList |
| *Participating*  *actors* | User |
| *Flow of events* | 1. StudyBear displays screen with menu bar  2. User clicks settings button. User clicks the manage block list link.  3. StudyBear displays Manage Block List Screen: Instructions for adding/removing users, listbox of blocked users, textbox to enter usernames, an add button, and a remove button.  5. a. User implements AddToBlockList use case  b. User implements RemoveFromBlockList use case |
| *Entry condition* | * User is logged in to StudyBear |
| *Exit condition* |  |
| *Quality*  *requirements* |  |

See notes on RegisterAccount and UpdateUserProfile –Got it DO

|  |  |
| --- | --- |
| *Use case name* | AddToBlockList |
| *Participating*  *actors* | User |
| *Flow of events* | 1. StudyBear displays Manage Block List Screen: Instructions for adding/removing users, listbox of blocked users, textbox to enter usernames, an add button, and a remove button.  2. User enters a username in the “Add user” textbox and clicks the add button.  3. StudyBear adds username to listbox and clears “Add user” textbox. The server stores information in the data store. |
|  | * Manage Block List Screen is Displayed |
| *Exit condition* | * Changes have been made to block list |
| *Quality*  *requirements* |  |

See notes on RegisterAccount and UpdateUserProfile –Got it DO

Should StudyBear automatically send and save usernames added to the block list? Does it make sense to click the add button and THEN have to confirm the update? There is not list of what has been done, so the user does not know what information –Got it DO

|  |  |
| --- | --- |
| *Use case name* | RemoveFromBlockList |
| *Participating*  *actors* | User |
| *Flow of events* | 1. StudyBear displays Manage Block List Screen: Instructions for adding/removing users, listbox of blocked users, textbox to enter usernames, an add button, and a remove button.  2. User clicks username in listbox to select username then clicks the remove button  3. StudyBear removes selected usernames from listbox. The server stores information in the data store. |
| *Entry condition* | * Manage Block List Screen is Displayed |
| *Exit condition* | * Changes have been made to block list |
| *Quality*  *requirements* |  |

See notes on RegisterAccount and UpdateUserProfile –Got it DO

|  |  |
| --- | --- |
| *Use case name* | EditClasses |
| *Participating*  *actors* | User |
| *Flow of events* | 1. User clicks edit button (pencil) on User Profile Screen.  3. StudyBear queries data store of valid classes for user’s school and valid times for availability. StudyBear displays User Profile Screen in Edit Mode with dropdown menus for user to chose valid classes and available times.  4. User updates classes and times  a. User clicks add course/time and makes selections from drop-down menu  b. User clicks the remove button next to previous courses/times  5. User clicks saves button  3. StudyBear saves information to the data store.  4. StudyBear displays User Profile Screen in Display Mode and loads information from data store. |
| *Entry condition* | * User Profile Screen is displayed (in Display Mode) |
| *Exit condition* | * User Profile Screen is displayed (in Display Mode) |
| *Quality*  *requirements* |  |

The display mode terminology may be unclear. They are mostly two separate screens, that share similar arrangement and meaning. But to change the user profile screen from display to edit you essentially have to replace all the elements. If we consider them two separate screens it may make the language clearer, and we can implement the “swap” however we would like