Design Document for:

StudyBuddy

Written by: Prepared by Dan D'Andrea, Robert Delgado, Ryan Hessen, Jacob Rabb, Monique Shotande, Max Ussin, Michael Van Wie, Renier Viega

Version # 1.00

9/18/2013

Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**

1.0 Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for software.

1.1 Goals and objectives

Our primary goal for the project is to have a functional web application, which can be used to help students find StudyBuddies. To accomplish this, we need to implement profile pages, as well as a method of searching through profiles so that a list of potential StudyBuddies can be returned to the user.

Our second set of goals relates to ease of use of our application. We would like to implement a grouping features, called “StudySpace,” which allows users to post messages to other users who have agreed to group with them.

1.2 Statement of scope

StudyBuddy is a web application for UF students who are looking for study partners. The application will also allow students to coordinate based on their busy schedules. Secondary purposes are related to convenience, accessibility, and the usability of StudyBuddy.

The major inputs are navigation to the various web pages of the application, clicking buttons, and submitting forms. Processing functionality includes back-end (controller, the “C” in MVC) processing to handle clicks and form submissions, saving data to the database and retrieving data from the database, and other business logic. The outputs are static and dynamic web pages, with the content based on items such as the state of a users web site session and the underlying database.

The following requirements will be designed and implemented:

- Create/Manage Profile - Essential
- Homepage - Essential
- Start Page - Essential
- Manage Schedule - Essential
- Find StudyBuddies - Essential
- StudySpace - Desirable
- Calendar/schedule matching - Desirable
- Chat - Future
- Profile picture - Future
- StudySpace file upload - Future

1.3 Software context

The purpose of this application is to create a system for students to be able to find a peer or group of peers. This is so that users can study and interact with said peers in an effort to perform better in their classes. The software will provide a tool to be able to compare conflicting schedules of students in an effort to be able to find a concurrent time where students can meet and study together for a set amount of time. Our application will be strictly web-based to provide access to the largest amount of individuals within the University of Florida.

1.4 Major constraints

One of the major constraints will be that the application will be written in a development server and thus, will not be deployable to an actual web page. Upon completion of the application, the decision will be made as to whether or not the application should be deployed to be used publicly under a functional web page.

The application will also be written using Django which is written using primarily Python. Additionally, the database design will be done within SQLite which will limit the span of the database and the amount of information that will be stored.

The developers within this design team also have very limited experience using both Python and Django and therefore some time will be spent becoming proficient with the software and programming language being used.

2.0 Data design

A description of all data structures including internal, global, and temporary data structures.

2.1 Internal software data structure

Classes:

- User: The user class describes users of our system. Properties: Name, Email, Password, Phone Number, A list of Schedule items, A list of StudySpaces, School Name, Year
- Schedule Item: An abstract class which other classes inherit from. Properties: Start time, End time.
- Class Schedule Item: Inherits from Schedule Item. Properties: Section Number, Professor Name, Semester, Class Code, Class Name, Class State*
- Other Schedule Item: Inherits from Schedule Item. Description (optional)
- StudySpace: Unique ID, List Users, List of Messages, StudySpace Name, Section Number, Class Code, School Name
- Message: Date, Time, Sender, Content

*Class State (Looking for a StudyBuddy, Not Looking for a StudyBuddy).

2.2 Global data structure

We are using the MVC implementation for our project, so our global data structure is a collection of models, views and controllers.

2.3 Temporary data structure

Classes (here we are using the word class as a university course, not a programming construct) will be a temporary data structure. This is because the software will clear users in a class after the semester is over. The message boards will use temporary data structure and the messages will get deleted when the messages exceed the allowed lifespan.

2.4 Database description

The following database tables will be created to support storing the data necessary for the application.

Table: User

Description: Holds user's name, email address, password, phone number, school name, and class year (1 = freshman, 2 = sophomore, etc)

Table: User schedule item
Description: Holds start time and end time of a schedule item plus whether or not it is a class schedule item, and if so, the section number, class code, class name, professor name, semester, and whether or not the student is looking for a StudyBuddy in that class

Table: User schedule
Description: Holds a list of each user’s schedule items

Table: StudySpace
Description: Holds unique ID for StudySpace plus list of users, list of messages, StudySpace name, section number, class code, and school name

Table: StudySpace message
Description: Holds a list of messages for each StudySpace; each message contains the identity of the sender, the date and time the message was sent, and the content of the message

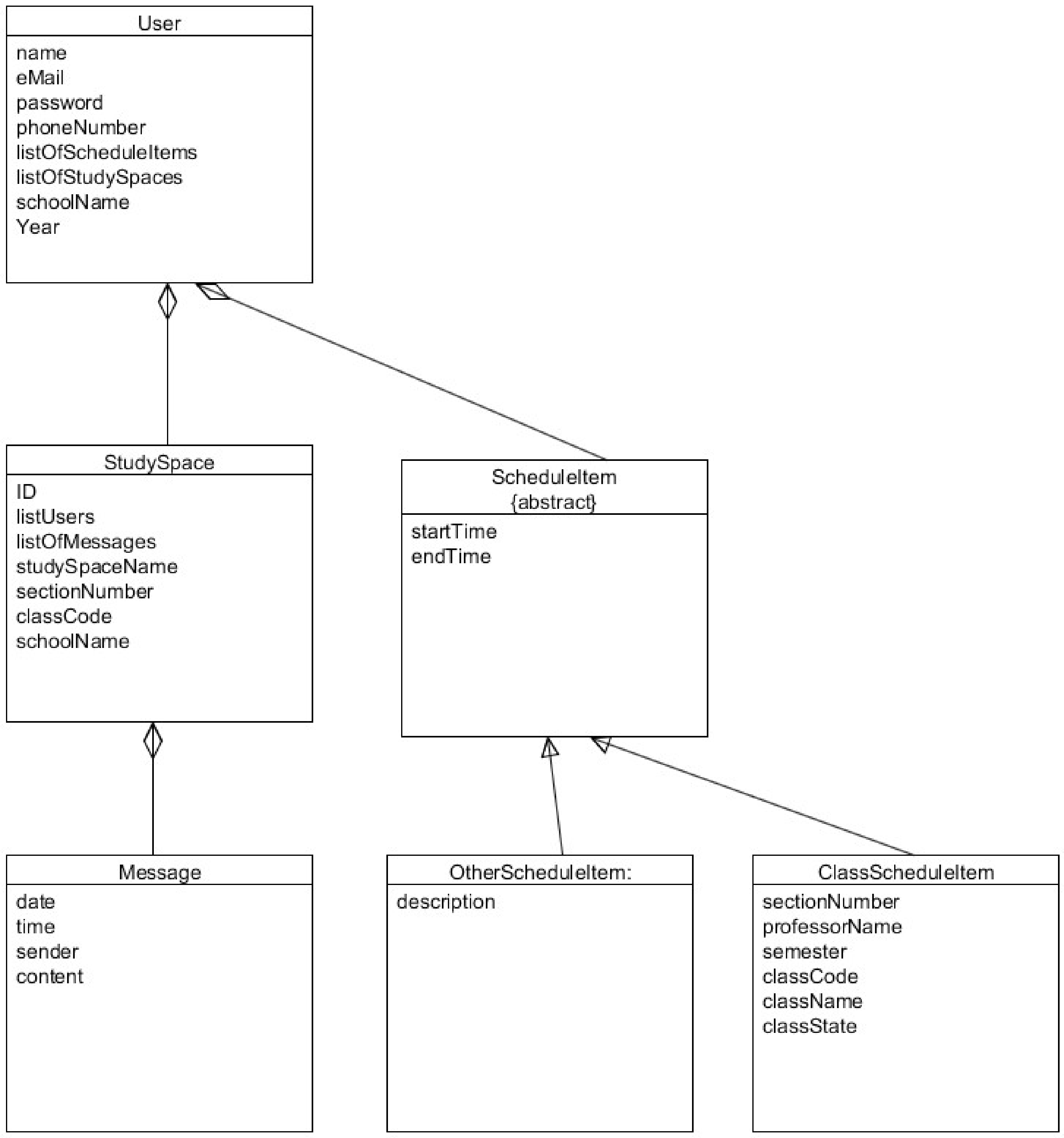
3.0 Architectural and component-level design

This application will be designed using the Django framework. This will allow simpler implementation with an MVC design pattern. The back-end programming will be done with Python and the data will be stored in a SQLite database.

3.1 System Structure

This website will be designed based on the Model-View-Controller (MVC) pattern. The Model contains the data to be rendered in the view. The View is in charge of the user interface (UI) and specifies how the model is to be rendered. The Controller prepares the Model for the View and also updates the persistent representation of the Model which is stored in the database.

3.1.1 Architecture diagram



3.2 Description for User Component:

The User Component represents and manages all the information relative to the users, such as profiles, schedules, and account information.

3.2.1 Processing narrative (PSPEC) for User Component:

The user component's responsibilities are: to provide the user a form on which data can be entered in order for the user's personal information to be stored in our database, to validate the data, and to inform the user how to fix incorrect data.

3.2.2 User Component interface description:

The user component's interface can be described as multiple pages with web forms. The web forms for the user profile consists of fields for data such as first name, last name,email, date of birth, School/University,Major, and "about me". This page also contains an upload photo button, and a submit button. Once the submit button is clicked, all data entered on the fields will be checked, and if all the data was entered correctly, then the data will be saved. Otherwise an error will inform the user which fields need to be corrected.

The log-in form is displayed to user via a pop-up window. The form for the user log-in consists of 2 fields: one for the username, and the other for their password. There will be two buttons on the form: one to submit the credentials that the user enters, and another to "sign up", which will redirect the user to the create profile page. Once the user inputs their log-in credentials and presses submit, the database will verify that their credentials are correct and grant the user permission to access the rest of the features that the site has to offer.

The schedule form consists of fields for data such as class name, period, section number, and time. Once all of the data is entered into the forms, the user presses the "add class" button which submits the information to the database. The schedule also displays the class information that has already been input into the database.

3.2.3 User Component processing detail

- The first and last name fields shall be checked as to ensure they do not contain an empty string. These fields shall also be checked to ensure that a duplicate entry in the database will not be created.
- The email field shall be checked as to ensure it does not contain an empty string. This field shall also be checked to ensure it contains the character '@' and the string ".com" or ".net" or ".gov" or ".edu". This field shall also be checked to ensure that the email being entered is not already in the database.
- The date of birth field shall be checked to ensure the user enters a valid date. The date entered in this field must allow the user to be between 1 and 100 years; a date in the future or a date far in the past should not be allowed.
- The school/University field can be implemented in different ways. If feasible, this field should display as a drop down and allow the user to pick a school from the choices presented. The second option for this field is to allow the user to type the name of his/her school in text box; in this case the field will be checked to ensure it does not

contain an empty string.

- The major field can also be implemented in two ways. If feasible this field should display as a drop down and allow the user to pick a major from the choices presented. The second option for this field is to allow the user to type his/her major in text box; in this case the field will be checked to ensure it does not contain an empty string.
- The "about me" field shall not be checked. This field allows the user to enter any information he/she wishes other users to see. The user shall not be forced to enter any data in this field.
- When creating a username and password, the username will be checked to see if it exists in our database and if it contains the correct amount of characters. If the username does not match our requirements, then we will inform the user that the input is not valid. The password shall be checked to see if it complies with the appropriate security requirements, such as a possible minimum length.
- When logging in, the username shall be checked to see if it exists in our database. If the username is found, then we look to see if the provided password matches our record on our database. In case the password doesn't match, then the user shall be notified.
- The schedule fields shall be checked. The class name, period, and section fields will be checked to see if they contain any invalid characters.

3.2.3.1 Design Class hierarchy for User Component:

The user component uses the user class and the abstract scheduleItem class, which are depicted in section 3.1.1. This component also makes use of the otherScheduleItem and classScheduleItem classes, which are both extensions of the scheduleItem class, in order to perform its functions.

3.2.3.2 Restrictions/limitations for User Component:

We will be limiting the amount of characters a user can enter in each field. Other restrictions are the server response time when querying the database and the amount of memory being used by the client.

We might restrict users who do not have an email with an ".edu" extension to register in order to protect our users and avoid people with wrong intentions to create accounts. By enforcing a person to have an email with an ".edu" extension we may also avoid spammers from stealing or accessing user information.

3.2.3.3 Performance issues for User Component:

Performance for the user component will depend partially on the performance of the user's computer since most of the work will be done through the use of a client side scripting language. The performance will also be dependent on the performance of our web server once we check the first and last names, email, and when the user submits the form.

3.2.3.4 Design constraints for User Component:

Its unlikely, but because the web application is being created with Django, there may be unexpected constraints involved with the implementation of the desired design.

3.2.3.5 Processing detail for each operation of User Component:

- createUser() will check to see if the user currently exists, and if not, will create a new user and add him/her to the database
- editUser() will check to see if the user exists, and if so, will modify the existing user's information
- deleteUser() will check to see if the user exists, and if so, will delete the corresponding user from the database
- login() will check the user's input credentials to see if the given account exists, and if so, will give the user access to the features of the website
- getSchedule() will return the user's full schedule and display all of the classes to them
- addScheduleItem() will check to see if the items already exists, and if not, add it to the user's schedule
- deleteScheduleItem() will check to see if the item exists, and if it does, delete it from the user's schedule
- updateScheduleItem() will check to see if the item exists, and if it does, update the information corresponding to it

3.2.3.5.1 Processing narrative (PSPEC) for each operation

- createUser() will create a new user
- editUser() will modify an existing user
- deleteUser() will delete a user
- login() will authenticate a user
- getSchedule() will displays the user's schedule
- addScheduleItem() will add a class to the user's schedule
- deleteScheduleItem() will delete a class from the user's schedule

3.3 Description for Networking Component:

3.3.1 Processing narrative (PSPEC) for Networking Component:

The networking component is responsible for communication between the our web application and the users. Users will take advantage of the network component to communicate with each other, through the server.

3.3.2 Networking Component interface description:

The interface will be comprised of Django, which is a high level Python web framework, including built in databases and Model-View-Controller management.

3.3.3 Networking Component processing detail

The processing detail shall allow the user to communicate to the server by sending requests. From the web page, the user shall be able to send requests to the server to access the databases (through a GUI).

3.3.3.1 Design Class hierarchy for Networking Component:

The hierarchy goes from the user GUI through Django, with Python code operating in the background accessing the user server for each request.

3.3.3.2 Restrictions/limitations for Networking Component:

The restrictions of the network component will be the limitation that Django will have on the programmer. For example, Django web framework is code is only compatible with Python.

3.3.3.3 Performance issues for Networking Component:

Unexpected performance issues might arise. For example, simultaneous requests being accessed or sent at the same time could be delayed or lost from traffic.

3.3.3.4 Design constraints for Networking Component:

Our group has not yet identified any design constraints for the networking component.

3.3.3.5 Processing detail for each operation of Networking Component:

- getStudyBuddy(): This operation will display the list of requests.
- findStudyBuddy(): This operation will look for user who are candidates for being part of a study group.
- createStudyBuddy(): Once the candidate is selected, we call the createStudyBuddy(). This operation will send a buddy request to another user.
- approveStudyBuddy(): This operation will add the user that sent the request to the list of users on a study space.
- denyStudyBuddy(): This operation will deny the user that sent the request to the list of users on a study space.

3.3.3.5.1 Processing narrative (PSPEC) for each operation

- getStudyBuddyRequests(): gets the StudyBuddies request list
- findStudyBuddy(): searches for StudyBuddies
- createStudyBuddyRequest(): create a StudyBuddy request
- approveStudyBuddyRequest(): approve StudyBuddy request
- denyStudyBuddyRequest(): deny StudyBuddy request

3.4 Description for StudySpace Component:

The StudySpace component represents and manages all the information relative to StudySpaces. It allows one to create a StudySpace and then interact with it. Interaction is done through the following: creation and retrieval of messages, the process of adding StudyBuddies, the process of retrieving the member list, and the process of getting the matching schedule for all of its members.

3.4.1 Processing narrative (PSPEC) for StudySpace Component:

The StudySpace component's responsibilities are to manage all operations that either create a StudySpace or alter its state. Forms will allow the user to input data and to display the updated state.

3.4.2 StudySpace Component interface description:

All input and output will be performed via web forms. These forms will allow users to do the following: see the StudySpaces that they are a member of, to see the messages previously posted to a StudySpace, to post new messages to a StudySpace, to get the list of members and their contact details, and to get the schedule which displays the blocks of time for which all members have overlapping free time.

3.4.3 StudySpace Component processing detail

- Message lists shall be rendered in a view, after retrieving them from a database. The details shall include the date and time that the message was created and the name of the user that created it. Each StudySpace shall have its own list of messages.
- A user shall be able to create a new message. This involves posting to a form, validating the form input, and then performing a database insert.
- Adding a StudyBuddy shall be performed after a StudyBuddy request is approved. That approval shall be performed by a user when they make approval decisions in their list of StudyBuddy requests. A button shall be provided in the user interface to indicate which StudyBuddy a user wants to study with and then subsequent database inserts shall be made by the controller to update the database to record this fact. If this is the first approval of a StudyBuddy for a particular school, semester, class, and section then a StudySpace shall be created in the database as well.
- A StudySpace shall display its members list. Getting the member list involves database queries to retrieve the member list for a given StudySpace. The member list shall be rendered in a view so that the user to see the names and contact details for each StudySpace member.
- A user shall be able to display matching schedule information. Getting the matching schedule involves an algorithm which takes in the schedule items of each member of a

StudySpace and then calculates all time blocks where no member has indicated that they are busy. This shall produce the list of time blocks for which all members have indicated that they are free to study.

3.4.3.1 Design Class hierarchy for StudySpace Component:

The StudySpace component uses the StudySpace class which is depicted in section 3.1.1. This component also makes use of the lists of Messages and Users on the StudySpace class in order to perform its functions. It in turn uses the lists of Schedule Items on each User class in order to compute the matching schedule.

3.4.3.2 Restrictions/limitations for StudySpace Component:

Restrictions for this component include only allowing access to valid, authenticated users and restricting all queries and inserts to the currently logged-in user.

3.4.3.3 Performance issues for StudySpace Component:

There are no major performance issues foreseen with this component. The schedule matching algorithm should perform well, based on the small input sizes that will typically be in use.

3.4.3.4 Design constraints for StudySpace Component:

There are no specific design constraints.

3.4.3.5 Processing detail for each operation of StudySpace Component:

- getMessageList() will perform a database query and then create a new model representation of the data and pass it to the view for rendering.
- createMessage() will perform a database insert to save the message.
- addStudyBuddy() will perform a database insert to add the new StudyBuddy to the given StudySpace; a new StudySpace will be created in the database if one does not already exist.
- getMemberList() will perform a database query and then create a new model representation of the data and pass it to the view for rendering.
- getFreeTimeBlocks() will query the database to retrieve the Schedule Items of all members of the given StudySpace and then use the schedule matching algorithm to find all blocks of time for which all users have no Schedule Items. It will then create a new model representation of the data and pass it to the view for rendering.

3.4.3.5.1 Processing narrative (PSPEC) for each operation

- getMessageList() gets StudySpace messages
- createMessage() creates a new StudySpace message
- addStudyBuddy() adds a StudyBuddy to a StudySpace and creates a new StudySpace

if one does not already exist

- getMemberList() gets the list of StudySpace members
- getFreeTimeBlocks() gets the list of common free time blocks for all StudySpace members

4.0 User interface design

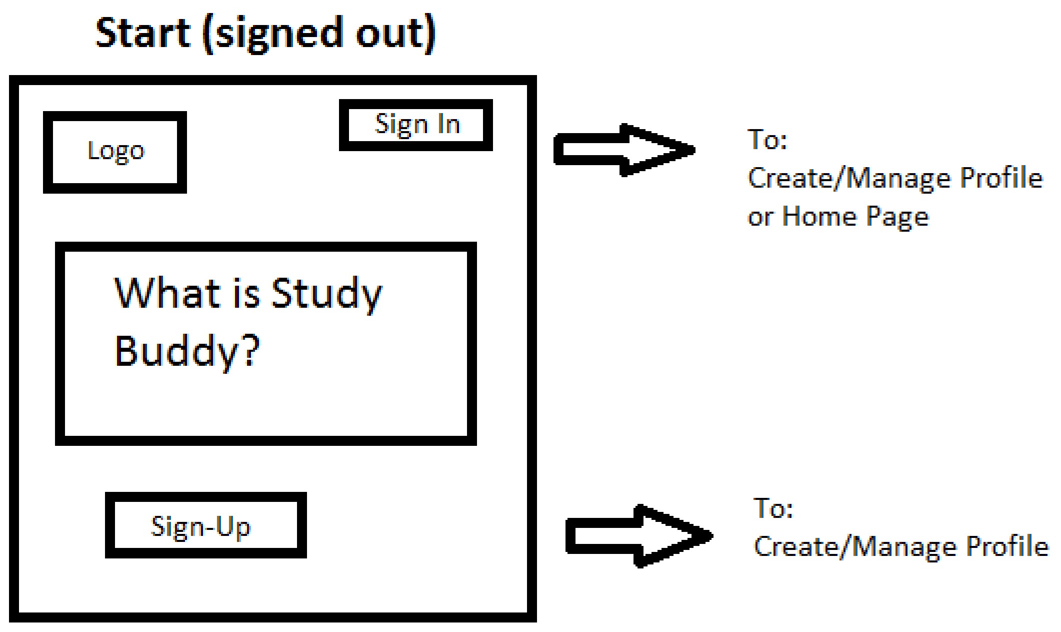
A description of the user interface design of the software is presented.

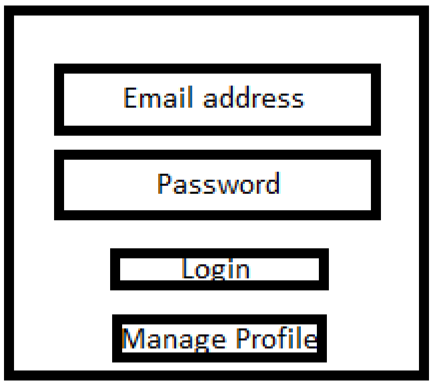
4.1 Description of the user interface

The user interface will consist of several pages which will allow the user to access all necessary functions of the application. There will be tabs which the user can use to easily navigate between pages.

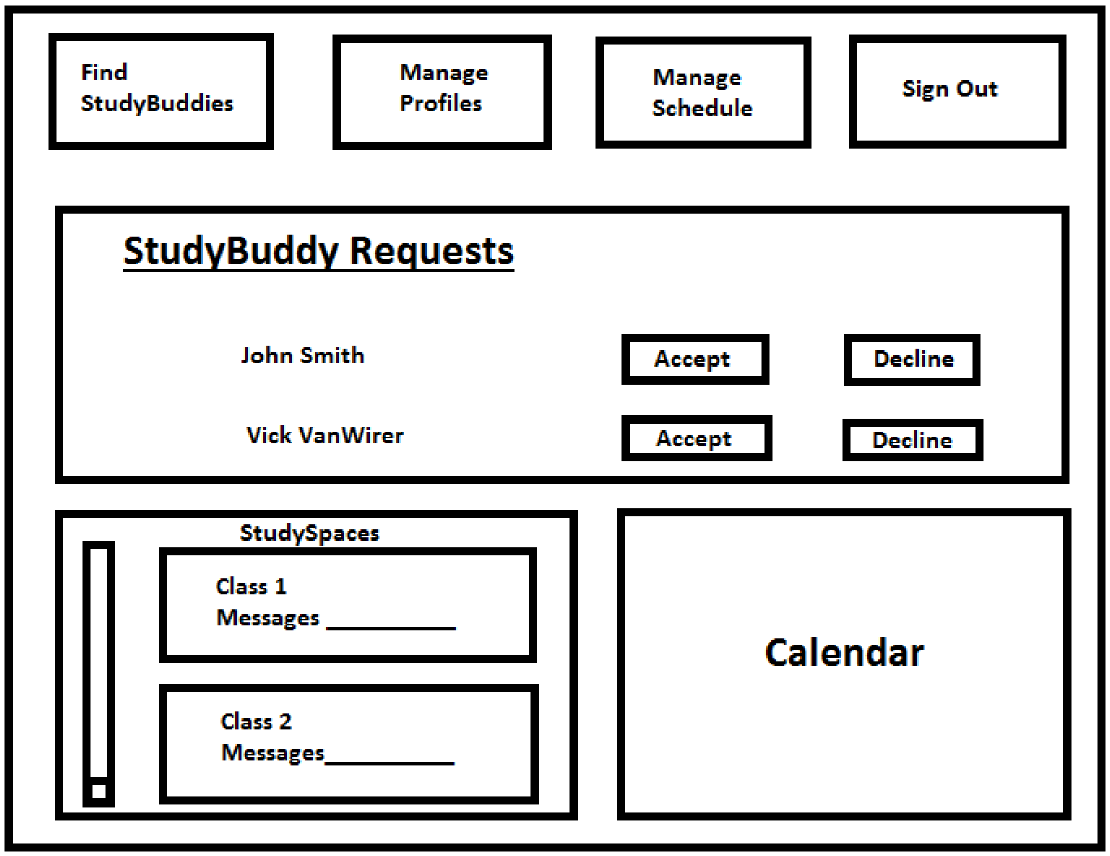
Considering the fact that StudyBuddy is a web application, interaction with the software will be cursor/keyboard based when accessing the website through a personal computer and the interaction will be touch based when viewed through a mobile device.

4.1.1 Screen images

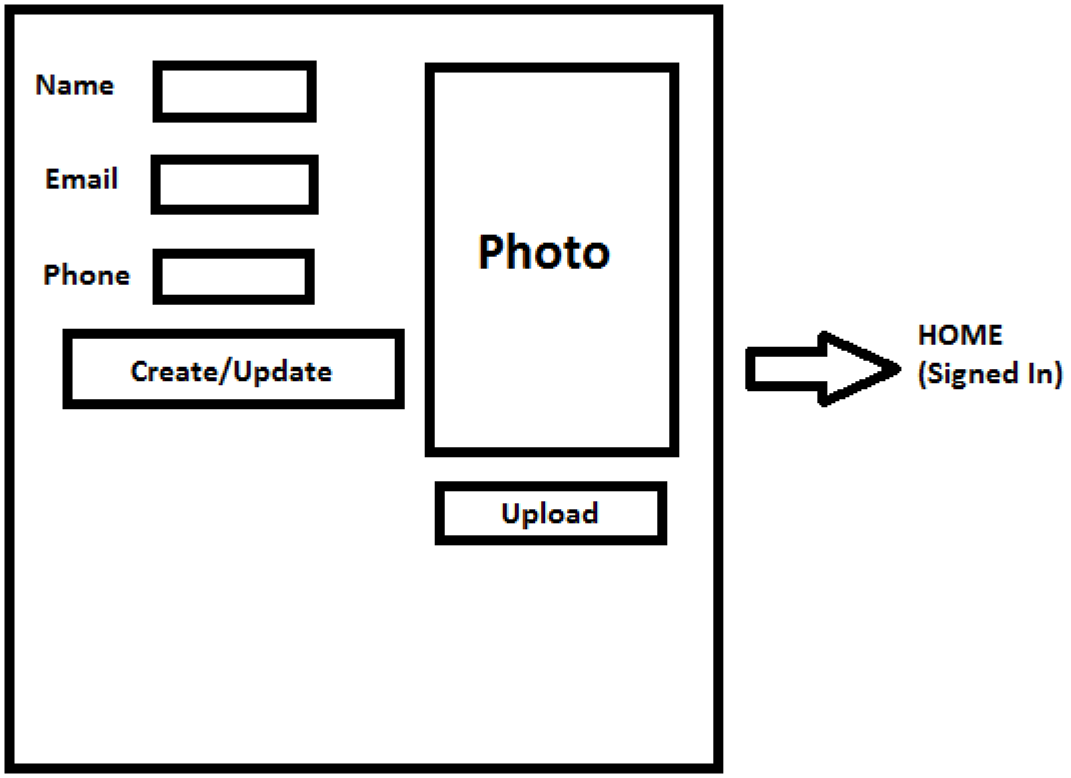

Start (signed out)
Logo
Sign In
What is Study Buddy?
Sign-Up
To: Create/Manage Profile or Home Page
To: Create/Manage Profile


Email address
Password
Login
Manage Profile

Home Page

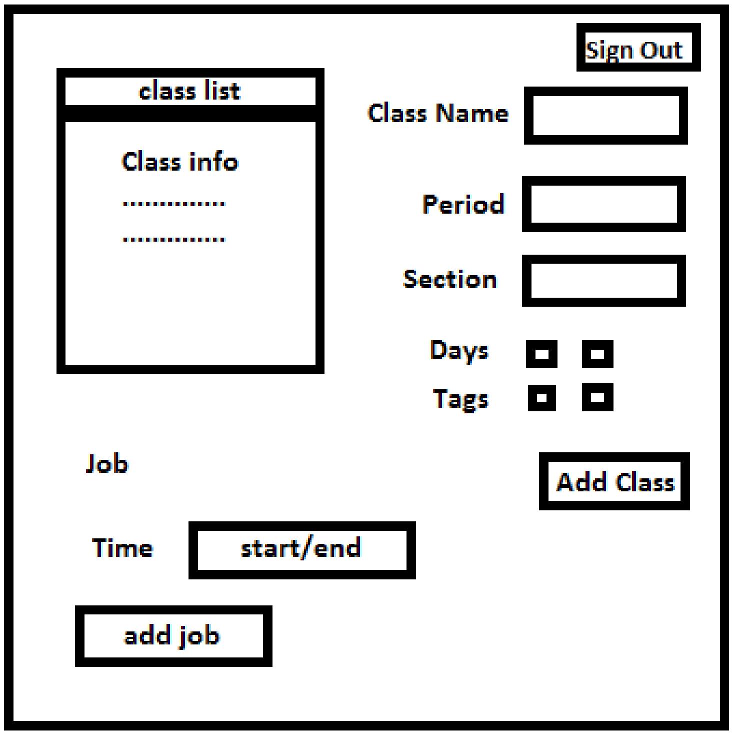

Find StudyBuddies
Manage Profiles
Manage Schedule
Sign Out
StudyBuddy Requests
John Smith
Accept
Decline
Vick VanWirer
Accept
Decline
StudySpaces
Class 1
Messages
Class 2
Messages
Calendar

Create/ Manage Profile



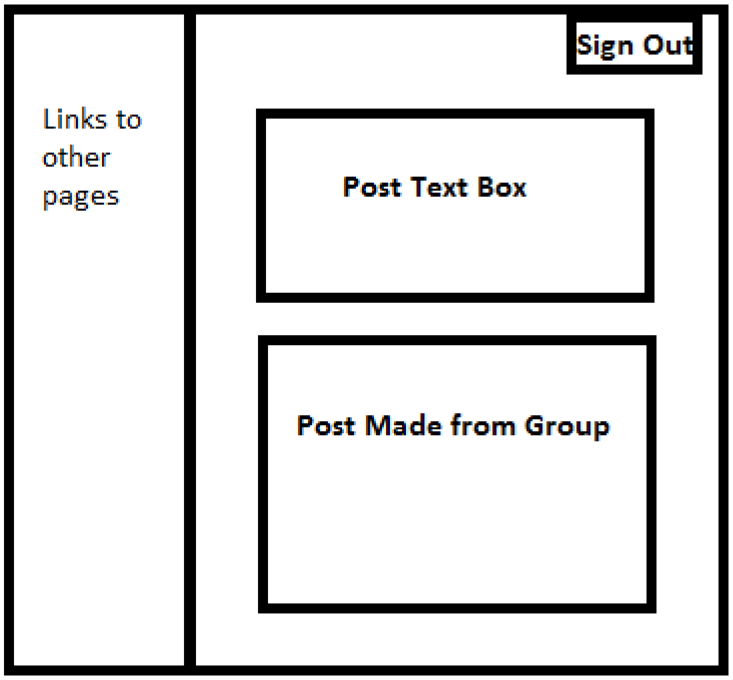
Manage Schedule

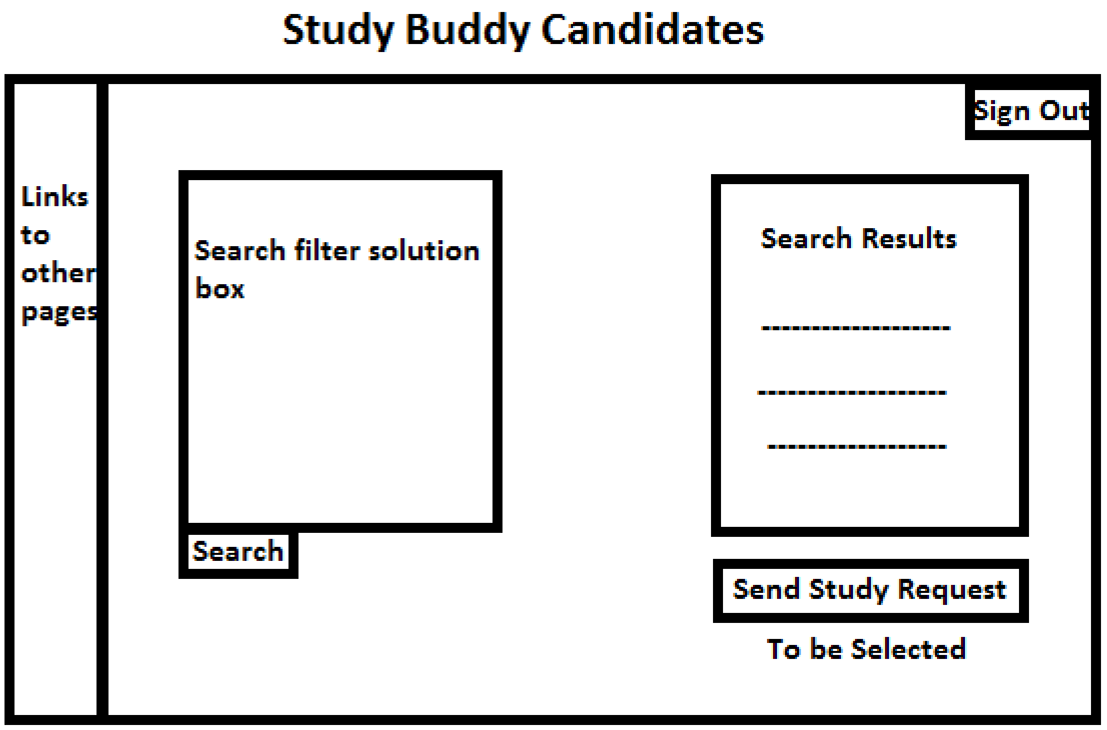
Links to other pages



Class info, LFG, and LFM flags will be color based

Study Space





4.1.2 Objects and actions

Users will log-in to their profiles by entering their email as a username and a password of their choice, into two separate pop up text boxes. Once the log-on button is clicked, the data entered into the boxes will be processed for validation. If their information passes the validation stage, the user will be sent to their home page. Otherwise, a message will be returned to them which explains that they have entered an invalid username or password. The user can also enter the “manage profile” page by entering their information in the pop up text boxes by clicking the “manage profile” button.

Users will be able to search for study buddies based on classes that they are requesting StudyBuddies for. Once potential StudyBuddies have been returned, the user will receive contact information.

4.2 Interface design rules

1. Strive for consistency: Our application should have a consistent response to input. Furthermore, we should use consistent terminology on each of the pages (StudyBuddy, StudySpace, etc), especially for our menus and forms. Finally, the pages should be rendered using a template so that they all have the same look and feel.

2. Enable frequent users to use shortcuts: On each page, there will be links to the other pages. In addition, we may allow our application to respond to input besides mice (keyboard shortcuts, or touch sensitive pages for a possible mobile version).

3. Offer informative feedback: Every operator input should warrant some feedback, even if this is as simple as a confirmation message on the screen.

4. Offer simple error handling: When the user enters erroneous data, they should be prompted to re-enter the data in a manner which makes more sense. Most other error handling will be internal, and only relevant to our group as the developers, although we will need to provide some sort of 404 page in case the user enters an invalid URL.

5. Permit easy reversal of actions: The user should not have to worry that the data they enter is permanent. We will allow a user to change their profile at any time. Furthermore, we will likely have some sort of “revert” button for schedule changes and the like.

6. Support internal locus of control: Experienced users should feel as if they are in control of the system, and the system should respond to their actions.

7. Reduce short-term memory load: We should not overload the user with information. We should keep our interface as simple and tidy as possible.

4.3 Components available

The components that will be available to the user are as follows:

- Home page will contain data pertinent to the current user’s study groups and notifications for new messages.
- Start page is a generic page the members and non-members alike will see that holds basic information about the site and text boxes to either login or register. This information will be stored in the database.
- Profile/account page contains the user’s information (including their schedule) and allows them to change this information.

4.4 UIDS description

HTML5 will be used to develop a template for Django to separate design, content and Python code.

5.0 Restrictions, limitations, and constraints

How the Study Spaces will be created and implemented is an issue that will further be discussed by the group and greatly impacts the user's experience when using this application. We are not yet sure if the application will create Study Spaces for the users based on current study buddies. Alternatively, the users can create and merge theses spaces on their own. The decision that we eventually make will have various design and user experience implications.

Here are some further constraints:

- Accessing class roster in order to load students to our database.
- Acquiring a list of school names.
- Acquiring a list of classes offered by each school.
- Verifying that the student is actually enrolled in a class.

6.0 Testing Issues

The testing for this application will be done entirely within a development environment and will not be done within an live, hosted web site. Testing will be done on a class by class basis to ensure the interaction between classes is optimized and error-free. Once the application is developed to a point of initial deployment, the web application will be tested as a whole, using UAT (user acceptance testing), to ensure that the application is concise and easy to use for the end-user.

Upon completion of UAT, any necessary debugging and code modification will be done to ensure the application is optimized for the end-user. Upon completion of testing within the development model, the application will be deployed to a public domain.

6.1 Classes of tests

The first test to be implemented, would be to create a profile and log-in as a user with all variables of the class “user” corrected. The second, is testing the SQL database is functioning properly and that the queries are finding the correct data. This will be done by creating profiles and searching for other user’s in the same class and sections.

6.2 Expected software response

The web application is expected to run in real time and therefore, it must be responsive in real time. The user must be able to traverse the application in optimal time. Database calls are expected to be time-efficient and done with limited delay.

6.3 Performance bounds

Performance should not be an issue for the end-user when it comes to using the web application properly. A network connection will be required in order to use the application. A suitable personal computer or mobile device will be sufficient to access the application.

The application must be able to make database calls in a timely fashion and limit the amount of time required to find a study partner for the end-user.

The application must not make unnecessary and time expensive calls to the database and reuse data to optimize the use of time. This can be done by the use of a client side scripting language such as javascript.

The web application shall be free of bugs and dead links. The use of a dedicated server is highly encouraged in the beginning phase. However if web traffic and/or the use of server resources increase, then the implementation of a dedicated server becomes imperative.

6.4 Identification of critical components

Testing of the database will be critical to ensure the proper retrieval of information in a time-efficient manner. The database schema will be thoroughly tested to create optimal data retrieval.

Upon implementation of the study spaces, it will be critical to maintain a process to remove unnecessary data that has become post-dated or irrelevant to the application. The algorithm used to search for various study partners will also have to be developed with time optimization in mind. The function will only look up relevant information for the user and limit the amount of bloat data that is provided.

7.0 Appendices

Presents information that supplements the design specification.

7.1 Packaging and installation issues

Installation of this application will be irrelevant to the user due to the software being a web-application. When modifications are made to the application, it will be deployed to the web server and the most recent update will be available for all users upon the time of its release.

The only issues that may present themselves would be if the website was down due to technical issues, or if updates were being made to the application prior to a new release.