H.I.P.S.T.E.R

Humans in Programming, Science, Technology, Engineering, Research

Design Use Cases

Authors

Joe M' Deon, Stephen Gilardi, Sara Farsi, Saveen Chadalawada, Kann Chea, Vincent Yu, Sher Zahed, Francisco Aceves, Brandon Powers, & Mikey Cho

Table of Contents

[DUC-S01]	See List of Friends	03
[DUC-S02]	Add Friend	04
[DUC-S03]	Remove Friends	05
[DUC-S04]	Chat Box	06
[DUC-S05]	Search For a Friend	07
[DUC-P01]	Create an Account	08
[DUC-P02]	Login to the App	07
[DUC-P03]	Forgot Password	08
[DUC-P04]	Change Account Profile Info	09
[DUC-P05]	Unique User Names	10
[DUC-P06]	Profile Name	11
[DUC-P07]	Log Out	12
[DUC-P08]	Profile View	13
[DUC-G01]	Create a Group	14
[DUC-G02]	View Members of Groups I Am a Part Of	15
[DUC-G03]	View Details About Group Meetings	16
[DUC-G04]	View Groups I am In	17
[DUC-G05]	Create A Group Meeting	19
[DUC-G06]	View Single Group I Am In	20
[DUC-G07]	Invite People to a Group	21
[DUC-SC01]	Meeting Length	22
[DUC-SC02]	Recurring vs. One Time Event	23
[DUC-SC03]	Input Schedule Manually	24
[DUC-SC04]	Be Able to View Schedule	25
[DUC-SC05]	Start/End Dates for Recurring Meetings	26
[DUC-SC06]	Re-Schedule	27

Legend	
[UC-SXX]	Use Case - Social
[UC-PXX]	Use Case - Profile
[UC-GXX]	Use Case - Group
[UC-SCXX]	Use Case - Schedule

🟥	**INCOMPLETE**	🟩	**COMPLETE**

DUC-S01: See List of Friends

Description: This use case outlines the implementation details for the process of displaying a list of friends.

User Goals: The user wants to see a list of friends to be able to delete or organize his/her list.

Dependency Design Use Cases: Add/Remove Friends (UC-S02)

Details:

Priority: 2
Progress Status: Incomplete
Test Phase Status: Planned

Pre-Conditions:

1. User is registered and logged in.
2. User has friends in the database.

Post-Conditions:

1. No change in the database.
2. The user's view is updated with friend's list.

Trigger: User has clicked on the button to "View Friends"

Workflow:

1. HomeController.js verifies user is logged in.
2. ProfileController.js uses the Parse API to retrieve list of friends.
3. ProfileController.js updates the view with a list of the user's friends.

DUC-S02: Add Friend
Description: The user shall be able to add friends using their email address/username.
User Goals: The user wants to add a friend to his/her list, so he/she can find the friend using their friend's profile name instead of email address.
Dependency Use Cases: Create an Account (UC-P01), Search a Friend (UC-S05)
Details: **Priority:** 2   **Progress Status:** Incomplete   **Test Phase Status:** Planned
Pre-Conditions: 1. User is registered in the database.   2. User is logged in.
Post-Conditions: 1. FriendList in database is updated for current user.
Trigger: The user enters a username and clicks to confirm they wish to add friend.
Workflow: 1. ProfileController.js is called to add a friend to FriendList object   2. ProfileController.js sends request to Parse api to change the user's FriendList   3. ProfileController.js updates the view with new list of friends.

DUC-S03: Remove Friends
Description: The user shall be able to remove friends from the user's Friends List.
User Goals: The user wants to remove a friend if he/she does not wish to have them as friends.
Dependency Use Cases: See List of Friends (UC-S01), Search For a Friend (UC-S05)
Details: **Priority:** 2   **Progress Status:** Incomplete   **Test Phase Status:** Planned
Pre-Conditions: 1. User is registered in the database.   2. User is logged in.
Post-Conditions: 1. If the user chose to remove a friend, that friend shall no longer appear in list.
Trigger: User clicks the confirm button for removing a friend.
Workflow: 1. ProfileController.js is called to remove a friend from the FriendsList object.   2. ProfileController.js sends a request to Parse database to update the new FriendList object for the current user.   3. ProfileController.js updates the view with the new list of friends.

DUC-S04: Entering Message into Chat Box
Description: The user shall be able to chat with group members via an accessible chat box specific to a group they are a member of.
User Goals: The user wants to enter a message into chat box.
Dependency Use Cases: Create a Group (DUC-G01)
Details: **Priority:** 2; **Progress Status:** Incomplete; **Test Phase Status:** Planned
Pre-Conditions: 1. User is a registered member. 1. User is logged in. 2. User is a member of a group in the database.
Post-Conditions: 1. The group's row in the database is updated with new chat box.
Trigger: The user hits enter once their message is complete.
Workflow: 1. GroupController.js is called to update the Chat Box object with the new message. 2. GroupController.js sends a request to the Parse database to update the chat box database portion. 3. GroupController.js requests the chat box information from the database. 4. GroupController.js updates the view with the new chat box information.

DUC-S05: Search for a Friend
Description: Outlines the implementation for the system returning a user object when one has been searched for.
User Goals: The user wants to find people in his/her list of friends.
Dependency Use Cases: Create User Account
Details: **Priority:** 2   **Progress Status:** Incomplete   **Test Phase Status:** Planned
Pre-Conditions: 1. User is logged into the site.   2. The user knows the name or email of the user they're searching for.   3. The user being searched for exists in the User Database.
Post-Conditions: 1. If the searched-for-user exists, the user is shown.   2. If the searched-for-user does NOT exist, alert the primary User that the user friend doesn't exist.
Trigger: The user types in the person's name and clicks search.
Workflow: 1. ProfileController.js searches itself for the desired user.   2. ProfileController.js returns the desired user.   3. ProfileController.js updates the view with the information from the search.

DUC-P01: Create an Account

Description: Outlines the implementation for the system to create an account to have a personalized experience with the website.

User Goals: The user can access all other functions of the web app after logging in to their account.

Dependency Use Cases: Unique Usernames (UC-P04)

Details:

Priority: 1
Progress Status: Complete
Test Phase Status: Planned

Pre-Conditions:

1. The user isn't already signed in.
2. The user has entered their email, password, and username.

Post-Conditions:

1. The user is redirected to the homepage.
2. The user shall be able to insert a new schedule.
3. The user shall be able to create a new group.

Trigger: The user clicks on the "Sign Up" button.

Workflow:

1. LoginController.js determines that the username entered is valid.
2. LoginController.js determines that the email entered is valid.
3. LoginController.js determines that the password entered is valid.
4. LoginController.js packages the username, email, and password into a new User object.
5. LoginController.js submits the User Object to the database to be saved.

DUC-P02: Login to the App
Description: Outlines the implementation for the system to login a user using an email and password.
User Goals: After creating an account, the user wants to login in to the app every time he/she wants to use it to be able to revise/edit schedule or add/remove a group.
Dependency Use Cases: Create an Account (UC-P01)
Details: **Priority:** 1 **Progress Status:** Complete **Test Phase Status:** Planned
Pre-Conditions: 1. The user already has an account in the database. 2. The user has entered in their email and password.
Post-Conditions: 1. The user is redirected to the homepage.
Trigger: The user clicks on the "Log In" button.
Workflow: 1. LoginController.js searches the database to make sure the username and password match a previously stored user object. 2. LoginController.js sets the local user object to the one found in the database.

DUC-P03: Forgot Password
Description: Outlines the implementation for the system to change the password when password is forgotten.

User Goals: The user must want to recover a lost password.
Dependency Use Cases: Create an account(UC-P03)
Details: **Priority:** 1 / **Progress Status:** Completed / **Test Phase Status:** Planned
Pre-Conditions: 1. The user has an account in the database. 2. The user has typed in their email.
Post-Conditions: 1. An email will be sent to the user for password reset.
Trigger: The user must click on “Forgot Password” button.
Workflow: 1. LoginController.js submits the user's email to the Parse. 2. Parse’s servers change the user's password in the database.

DUC-P04: Change Account Profile Info
Description: This use case outlines the implementation details for the process of the user accessing and editing their profile settings to make it a better experience for themselves whenever they want.
User Goals: To edit certain profile information.
Dependency Use Cases: Have an account (DUC-P01)
Pre-Conditions: 1. The user must have an account.
Post-Conditions: 1. User information in database has been changed 2. Changes are reflected in the views presented to the user
Trigger: The user clicks the account information button.
Workflow: 1. ProfileController.js retrieves account info from database 2. ProfileController.js sends info to view, which presents user info to user 3. ProfileController.js changes info in database 4. ProfileController.js transfers reflected changes to view, which presents to user

DUC-P05: Unique User Names
Description: The system shall update the user's username if it is unique
User Goals: To create a distinct username.
Dependency Use Cases: Create an Account
Pre-Conditions: 1. System receives proposed username by user
Post-Conditions: 1. The system has distinct usernames for each user and the user can be identified uniquely by this username.
Trigger: User shall enter his/her email address as username.
Workflow: 1. LoginController.js checks database to verify username is unique Path 1:(The string matches another username string) 2. The system requests the user to input their username again. Path 2:(The string is unique) 2. The system stores the username in the database

DUC-P06: Profile Name
Description: The user shall be able to create a profile name that is not necessarily unique and that will be shown in their groups.
User Goals: The user wants to be able to set a username that will serve as their default display name in groups.
Dependency Use Cases: Create an Account (DUC P01)
Pre-Conditions: 1. There should be an entry for the user in the database
Post-Conditions: 1. The system shall have a profile name stored in the parse database for the user. 2. The system shall display the user's profile name in their profile view and in their groups unless otherwise set. (set group display name)
Trigger: The user the be setting click on the sign-up tab in the welcome screen.
Workflow: 1. ProfileController.js receives profile/display name entered by user from the view 2. ProfileController.js updates user's profile/display name in the database

DUC-P07: Log Out
Description: The user shall be able to log out of the app at any time.
User Goals: The user wants to be able to set a username that will serve as their default display name in groups.
Dependency Use Cases: Create an Account (DUC P01)
Pre-Conditions: 1. The user is logged in
Post-Conditions: 1. Tbe user is logged out
Trigger: The user the be setting click on the sign-up tab in the welcome screen.
Workflow: 1. HomeController.js calls Parse API to log the user out

DUC-P08: Profile View
Description: The user shall be able to view their profile upon request.
User Goals: The user wants to be able to set a username that will serve as their default display name in groups.
Dependency Use Cases: Create an Account (DUC P01)
Details: **Priority:** 2 / **Progress Status:** Incomplete / **Test Phase Status: Planned**
Pre-Conditions: 1. The user shall input a desired profile name. 2. The user shall be trying to create an account.
Post-Conditions: 1. The system shall have a profile name stored in the parse database for the user. 2. The system shall display the user's profile name in their profile view and in their groups unless otherwise set. (set group display name)
Trigger: The user the be setting click on the sign-up tab in the welcome screen.
Workflow: 1. ProfileController.js retrieves the user's from their name in their database.

DUC-G01: Create a Group
Description: The user shall be able to create a new group.
User Goals: The user wants to create a new group to see meeting times.
Dependency Use Cases: Create an account (DUC P01), Login into the App (DUC P02), Search for a Friend (DUC S05)
Details: **Priority:** 1   **Progress Status:** Incomplete   **Test Phase Status:** Planned
Pre-Conditions: 1. The user shall have an account.   2. The user shall click "add" group button.   3. The user shall input a group name.
Post-Conditions: 1. The system shall store the group in the parse database for the user.
Trigger: The user shall click on the "add group" button.
Workflow: 1. GroupController.js displays a popup with a group name field, an add friend field, and an add member field.   2. GroupController.js checks that the group name is not already used by the user.   3. GroupController.js checks that the friend is not already in the group.   4. GroupController.js adds non-friends to the group by adding emails.   5. GroupController.js checks that the person is not already in the group.   6. GroupController.js creates a group in the parse database.   7. GroupController.js displays a confirmation window.

DUC-G02: View Members of Groups I am a Part Of

Description: The user shall be able to view a comprehensive list of the group roster.

User Goals: The user shall gain information about who their groupmates are for a specific group.

Dependency Use Cases: Create an account (DUC P01), Login into the App (DUC P02)

`Details:

Priority: 2
Progress Status: Incomplete
Test Phase Status: Planned

Pre-Conditions:

1. Be a member in the group

Post-Conditions:

1. The user will be able to see their groupmates for a specific group

Trigger: The user shall click on the "members" button in the specific group's profile view

Workflow:

1. GroupController.js redirects the user from the the user profile screen to the group profile view in question.
2. GroupController.js displays the group's profile view.
3. GroupController.js displays the group leader followed by the group members.

<u>Path 1:</u> The user stays on the group info display view

5. GroupController.js displays information about a member that the user requested about.

<u>Path 2:</u> The user leaves the group info display view

5. GroupController.js displays the group's profile view

DUC-G03: View Details About Group Meetings
Description: The user shall be able to view details such as time, location, and the topic of the meeting.
User Goals: The user shall gain all the information about group meetings
Dependency Use Cases:
Details: **Priority:** 2   **Progress Status:** Incomplete   **Test Phase Status: Planned**
Pre-Conditions: 1. The user must be a member of the group in question
Post-Conditions: 1. The system shall display all details pertaining to upcoming group meetings to the user.
Trigger: The user shall click on the “info” button in the specific group’s profile view.
Workflow: 1. GroupController.js displays the group’s profile view   2. GroupController.js displays information about upcoming events in the group.   Path 1: The user stays on the group info display view   3. The user shall click on a member to view info about them.   Path 2: The user leaves the group info display view   5. The system shall display the group’s profile view

DUC-G04: View Groups I am In
Description: This use case outlines the implementation details for the process of a user seeing the groups in which they are a member of.
User Goals: The user shall be able to see a list of the groups they a member of.
Dependency Use Cases: Nothing
Pre-Conditions: 1. The user must be logged in. 2. The user exists in the database.
Post-Conditions: 1. All groups that the user is in is retrieved from the database. 2. The system shall display the groups that the user is a member of.
Trigger: The user shall click on the “groups” tab from their profile view.
Workflow: 1. GroupController.js accesses data fields of the group object. 2. GroupController.js displays all the groups the user is a member of.

DUC-G05: Create a Group Meeting

Description: This use case outlines the implementation details for the process of a user creating a meeting time in a simple and efficient way.

User Goals: The system shall find the best meeting time, create an event, and automatically enter it into all member's personal schedule.

Dependency Use Cases: Login to the App (DUC-P02), Re-diff Schedules (DUC-SC07), Providing Alternative Meeting Times (DUC-SC08)

Pre-Conditions:

1. The user must be able to create an event.
2. The system shall be able to access all member's schedules and find the best meeting time.

Post-Conditions:

1. The system shall display as many time options as it can find with the given criteria.
2. The system shall display a weekly view of all possible meeting times for group members to pick.

Trigger: The user must click on "submit" button after entering all the required fields for creating a meeting in a group.

Workflow:

1. GroupController.js accesses the schedule of all members.
2. GroupController.js compares all the schedules.
3. GroupController.js finds all possible times according to criteria entered and schedule of all members.
4. GroupController.js displays the available times where least conflicts occur to hold this event.

DUC-G06: View single Group I am In
Description: This use case outlines the implementation details for the process of a user viewing a single group they are a member of.
User Goals: The system shall present a profile view of a single group, showing all the members, a chat box, and all other details of a group.
Dependency Use Cases: Login to the App (DUC-P02), Re-diff Schedules (DUC-SC07), Providing Alternative Meeting Times (DUC-SC08),
Pre-Conditions: 1. The user must be able to choose a group they are a member of.
Post-Conditions: 3. A single group's data is retrieved from the database. 4. The system shall display the group with all its details.
Trigger: The user click on a group.
Workflow: 1. GroupController.js accesses the data fields of a single group object. 2. GroupController.js displays the group's profile.

DUC-G07: Invite People to Group

Description: This use case outline the implementation details for the process of the user inviting their friends into a group they are a member of.

User Goals: The user adds a friend to their group.

Dependency Use Cases: See List of Friends (DUC-S01), Search for Friend (DUC-S05)

Pre-Conditions:

1. The user must be logged in.
2. The database should have the capability to edit the members list of a group.

Post-Conditions:

1. The database should either add the new member to the members list or not change the members list based on the group leader's discretion.

Trigger: A member of a group would like to add a friend to the same group.

Workflow:

1. groupController.js sends a message to the group leader asking them if they are OK with the new member.

Path 1: If the group leader accepts the new member

2. groupController.js sends a notification to the new member informing them that they are now a member of the group.
3. profileController.js displays the new group in the user's group overview section.

Path 2: If the group leader does not accept the new member

2. groupController.js notifies the original user that their friend was not added for a reason given by the group leader.

DUC-SC01: Meeting Length

Description: This use case outline the implementation details for the process of the user selecting a meeting length for an event they are creating or editing.

User Goals: The user can customize the length of an event to their liking.

Dependency Use Cases: The user has created an account. (DUC-P01)

Details:

Priority:	1
Progress Status:	Incomplete
Test Phase Status:	**Planned**

Pre-Conditions:

1. The user must have an account and the user must be in a group.
2. The user must be trying to create an event.

Post-Conditions:

1. The database should update the events object with a new event.

Trigger: The user clicks create an event button.

Workflow:

Path 1: The user is creating an event for their personal profile.

1. profileController.js will set the meeting length of the new event to what the user wants.

Path 2: The user is a group leader and is creating a meeting for their group.

1. groupController.js will set the meeting length of the new meeting to what the user wants.

DUC-SC02: Recurring vs. One-Time Event
Description: This use case outline the implementation details for the process of the user selecting whether an event is set to recur or be one-time only for an event they are creating or editing.
User Goals: The user wants to make a meeting
Dependency Use Cases: Have an account. (DUC-P01), Login to the account (DUC- P02)
Details: **Priority:** 1   **Progress Status:** Incomplete   **Test Phase Status: Planned**
Pre-Conditions: 1. The user needs to have an account.   2. The database's event object should have a recurring boolean field to set
Post-Conditions: 1. The database should update the events object with the new event.
Trigger: The user clicks the create a meeting button.
Workflow: 1. profileController.js accepts one or more new events and populates event objects with the user's data.   2. profileController.js adds the new event object(s) to the database.   3. profileController.js refreshes the user's view to display the new event in the user's profile page.

DUC-SC03: Input Personal Schedule Manually
Description: This use case outline the implementation details for the process of the user inputting their schedule manually into their profile.
User Goals: The user wants to enter his/her schedule manually instead of importing them from another source, so he/she can have complete control over the schedule.
Dependency Use Cases: Create an Account, Login to the Account
Details: **Priority:** 1   **Progress Status:** Incomplete   **Test Phase Status:** Planned
Pre-Conditions: 1. The user must be logged in.   2. The database must have the capability to store and call upon the inputted data.
Post-Conditions: 1. The database should store and be able to call upon the inputted data.
Trigger: The user has to click on "Add Event", enter the details of the event, and click on "Create Event".
Workflow: 1. profileController.js should add the new event(s) into the database.   2. profileController.js should update the user's profile with the new events.

DUC-SC04: Be Able to View Schedule
Description: This use case outline the implementation details for the process of the user viewing their weekly schedule.
User Goals: The user wants to view his/her schedule to be able to add/remove/update an event or just to see what he has to do for the day.
Dependency Use Cases: UC-S03
Pre-Conditions: 1. User must have an account. 2. Database must have user schedule existing.
Post-Conditions: 1. The database must contain the user's schedule.
Trigger: The user clicks on profile view button.
Workflow: 1. HomeController.js hands off view to ProfileController.js 2. ProfileController.js displays user's profile and schedule

DUC-SC05: Start/End Dates for Recurring Schedule
Description: This use case outline the implementation details for the process of the user choosing the start and end dates for their recurring events so that the algorithm doesn't consider them after they are over.
User Goals: The user wants to specify a start and end date for repetitive events so they do not show in their schedule after they are done.
Dependency Use Cases: Create a Group, Create a Meeting, Present the Best Time
Pre-Conditions: 1. User must have an account in the database. 2. Database must store user's start/end date input into the event object in their schedule.
Post-Conditions: 1. The event is added to the schedule for a certain period of time and it is not shown before or after the start and end time in the database.
Trigger: The user shall enter the dates manually when adding an event.
Workflow: 1. ProfileController.js accesses data fields of the schedule object. 2. ProfileController.js adds new dates to the schedule object. 3. ProfileController.js displays confirmation of schedule addition.

DUC-SC06: Re-schedule

Description: The system shall allow the user to re-compare schedules in the group while reflecting changes in each user's schedules.

User Goals: To be able to compare a new personal schedule with a group schedule and run the algorithm on them.

Dependency Use Cases: Create Profile, Create Schedule, Join a Group

Pre-Conditions:

1. There are at least two schedules to compare in the database for the group.
2. Database compares with algorithm and stores available times into the group's schedule object.
3. Database confirms storage action to group in the view.

Post-Conditions:

1. A diff of the group schedule and the users' schedules is returned.

Trigger: The user requested to diff the personal schedules with the group schedule.

Workflow:

1. The GroupController.js will request the schedule of each member in the group from the database.
2. The GroupController.js will use an algorithm to find the available times for each member and store them into the schedule object of the Group.
3. The GroupController.js will display the new available times in the view (accessed from the database).