Department of Computer Science & Engineering, University of Nevada, Reno

BETA Universe Systems Initiative Table Application

Team 7: Taking Initiative

Jacob Gayban, Mark Graham, Andy Alarcon, Jacob Tucker, Griffin Wagenknecht

Dr. Sergiu Dascalu, Vinh Le, Dr. Devrin Lee

John Molt

November 16, 2021

Table of Contents

Abstract

The BETA Universe Systems Initiative Table is a mobile-optimized web application that aims to digitize and streamline gameplay components from the tabletop role-playing game "BETA Universe Systems," and limit narrative disruption. The players are able to create and sign in to an account, manage characters profiles, and join a game session started by the gamemaster. The gamemaster can create and sign in to an account, manage the non-player character profiles, and manage game sessions. The app embeds all the main components of player combat for the game and maintains a real-time chat log during game sessions. The chat log records the current events that have occurred in the session, allows players to message each other, and presents the turn order for all the session users.

Introduction

The Initiative Table application for "BETA Universe Systems" provides a host of quality-of-life features that handle many of the administrative and organizational aspects of the "BETA Universe Systems" table-top RPG game. Through the app, game masters can create sessions of a game and invite other players to play in real-time. The significance of this project is that players can enjoy the role-playing narrative aspect of the game without being disrupted by whose turn it is and without the extraneous calculations that, for example, can determine if their attack successfully connected or if it failed. This application is optimized for both desktop and mobile viewing to allow for portability and comfort when interacting with the game to meet the description as mentioned earlier. As described by the client, the lengthy calculations will be reduced by the application's built-in rule correlating and equation calculating matrix to distribute appropriate numbers and information to the player efficiently. The Initiative Table application will meet this objective by giving the player several interfaces for which they can fill in the missing information for proper calculations.

Our current implementation continues to use Vue.js, Vuetify, and Axios for client-side (front-end), and Microsoft's ASP.NET Core framework for server-side (back-end). Most of the visual and logical design is complete at this stage, and the team has begun working on implementation. Since the previous report, the team has already started working on the UI and has implemented the authentication system. When it comes to changes regarding our project, we have only had minor changes to our database tables. Nothing else has changed regarding the project.

High-level and medium-level design

System-level diagram

The following diagram is of our system's context model, which illustrates the high-level components of the system. The character profile system allows the creation and management of character profiles. The authentications system enables players to register and log into their accounts. The session management system will allow players to join sessions and the GM to manage them. The real-time sessions system is the backbone of our sessions which enables real-time communication. The actions system allows players to take actions during sessions, and our web API controllers system enables all our systems to interface with our databases.

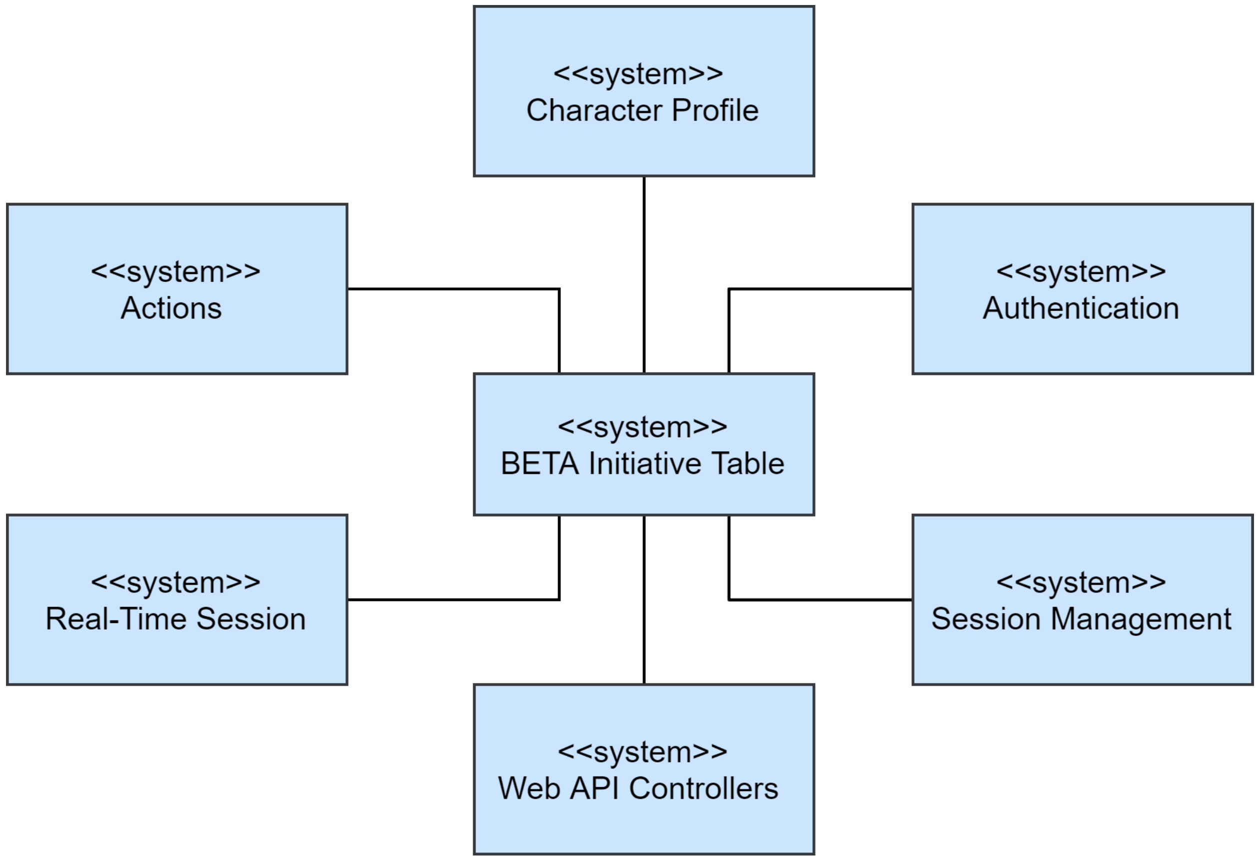

<<system>> Character Profile
<<system>> Actions
<<system>> Authentication
<<system>> BETA Initiative Table
<<system>> Real-Time Session
<<system>> Session Management
<<system>> Web API Controllers

Fig. 1: The table above shows the context model of the Initiative Table application for "BETA Universe Systems".

Program units

The Initiative Table application for “BETA Universe Systems” is a non-object-oriented solutions project. Most of these systems share data and are connected and feature a hierarchy. Our authentication system appears first at the highest level, followed by the character profile and session management system. Then our actions, real-time session, and web API controllers systems are at the lowest level. The following tables show the different program units that are required to make each of the systems work.

Authentication	The authentication system primarily consists of a Vue component that provides the user with a user interface and makes AJAX requests based on the user input to our web API controllers, interacting with our database. The system will allow users to register for an account and log in.
registerPlayer(player)	**Inputs:** A player object based on our database players table. **Outputs:** A player object based on our database players table or an error. **Description:** This sends a request to our players web API controller to create a new record for the player in the table. **Program Units Called:** None **Exceptions:** An exception is thrown if the player object is empty or incorrect.
loginPlayer(player)	**Inputs:** A player object based on our database players table. **Outputs:** A player object based on our database players table or an error. **Description:** This sends a request to our players web API controller to check if the user exists and if the credentials match. **Program Units Called:** None **Exceptions:** An exception is thrown if the player object is empty, incorrect or if the credentials are incorrect.

sendSecurityEmail(passcode)	**Inputs:** A 6 digit random number. **Outputs:** Nothing. **Description:** This sends an email with the number to the player's email when registering to verify the email is valid—waits for the correct number to be entered. **Program Units Called:** Calls registerPlayer upon successful entering of the number. **Exceptions:** An exception is thrown if the email is invalid and cannot be sent.

Table 1: The table above lists the program units under the Authentication system, allowing users to register and log into accounts.

Real-Time Session	The real-time session system primarily consists of a SignalR Hub. The hub contains all of the methods that will send events to our connected clients to control the flow of gameplay and send messages to the chat log of the current session. These methods are called initially on the client-side and then handled by SignalR. SignalR determines which connected clients need to receive the event signal and sends it to them with any optional data attached. All of the connected clients listen for the event signals and handle them once received.
sendMessageToAll(message)	**Inputs:** A message object **Outputs:** An event signal with the message attached to all the connected clients. **Description:** Sends a message to the chat log that all the connected clients to based on the passed object. **Program Units Called:** None **Exceptions:** An exception is thrown if the message object is empty or incorrect.

sendMessageTo(message, playerID)	**Inputs:** A message object and int playerID. The playerID corresponds to the identifier used internally by SignalR when they connect initially. **Outputs:** An event signal with the message attached to only the playerID that is connected. **Description:** Sends a direct message to only the chat log of the playerID that was passed. **Program Units Called:** None **Exceptions:** An exception is thrown if the message object is empty or incorrect.
JoinSession(playerName, sessionName)	**Inputs:** string playerName and string sessionName. **Outputs:** An event signal to all connected clients with a list of the currently connected players names attached (includes new player). **Description:** Adds the playerName to the current session based on the passed name and assigns them an internal playerID. SignalR establishes a connection. **Program Units Called:** Calls sendMessageToAll to let everyone know that a new player has joined the current session. **Exceptions:** An exception is thrown if the sessionName cannot be used to find a session in the sessions table.
Override OnDisconnectedAsync(playerID, sessionName)	**Inputs:** int playerID. **Outputs:** An event signal to all connected clients with a list of the currently connected players names attached (includes the player that left/ended their connection). **Description:** Overrides the SignalR disconnect function. Removes the player from the current session based on the passed playerID and sessionName. **Program Units Called:** Calls sendMessageToAll to let everyone know that a passed player has left the current session. **Exceptions:** An exception is thrown if the sessionName cannot be used to find a session in the sessions table.

updateInitativeTable(tableData, sessionName)	**Inputs:** An array of initiative table data [number, playerName] and string sessionName. **Outputs:** An event signal with the tableData attached to all the connected clients. **Description:** When the initiative table data has been modified by an action this method is called to send the updated data to all the connected clients. **Program Units Called:** None **Exceptions:** An exception is thrown if the sessionName cannot be used to find a session in the sessions table.
SignalPlayer(playerID)	**Inputs:** int playerID. **Outputs:** An event signal. **Description:** Once a player's initiative is next, or when they need to roll the GM signals, it is their turn. Enables the user interface for the player. **Program Units Called:** None **Exceptions:** An exception is thrown if the playerID cannot be used to find a player in the session.
CloseAnyOpenModals(playerID)	**Inputs:** int playerID. **Outputs:** An event signal. **Description:** This is a generic helper method that will close any open modals for a given playerID. Typically used during an intervention. **Program Units Called:** None **Exceptions:** An exception is thrown if the playerID cannot be used to find a player in the session.
OpenATKDFNDModal(playerID)	**Inputs:** int playerID. **Outputs:** An event signal. **Description:** This is a generic helper method that opens the attack/defend modal for a given playerID. Typically used during an intervention. **Program Units Called:** None **Exceptions:** An exception is thrown if the playerID cannot be used to find a player in the session.

Table 2: The table above lists the program units under the Real-Time Session System, which allows our real-time session system to control the flow of the session gameplay and chat log.

Actions	The actions system primarily consists of a Vue component that provides the user with a user interface and all of the potential actions the user can take. The methods are called and defined on the client-side. After execution of each process, it sends data or calls additional methods in other systems or the current one depending on the action.
Confirm(Action)	**Inputs:** A valid action **Outputs:** A message **Description:** This finalizes the users selected actions. **Program Units called:** The selected action **Exception:** If no action is selected
Attack(Target)	**Inputs:** Target **Outputs:** A message **Description:** This starts the process of the attack process and it signals the defender. **Program Units called:** Defend, roll, Initiative, weapon, **finish** **Exception:** If an invalid target is selected
Defend()	**Inputs:** None **Outputs:** A message **Description:** This unit is being attacked and must select their defense, skills, and make rolls. **Program Units called:** Roll, Armor, Initiative **Exception:** If they do not have an action

Roll(amount, die)	**Inputs:** The number and type of dice **Outputs:** an integer **Description:** This is a dice roller that outputs a number based on the input **Program Units called:** none **Exception:** If the amount or die are blank
Weapon()	**Inputs:** None **Outputs:** An item call **Description:** This selects the weapon the user is attacking with and uploads all of the weapon's data **Program Units called:** Melee, ranged **Exception:** If no action is selected
Armor()	**Inputs:** None **Outputs:** An item call **Description:** This selects the armor the user is defending with and uploads all of the armor's data **Program Units called:** The selected action **Exception:** If no action is selected
Skill(Skill)	**Inputs:** A selected skill **Outputs:** A message **Description:** The user selects a skill and the skill performs an event. **Program Units called:** The skill selected **Exception:** If no skill is selected
Initiative()	**Inputs:** None **Outputs:** A message **Description:** This shifts the users turn order based on the actions they have taken this round. **Program Units called:** None **Exception:** None

Hold()	**Inputs:** None **Outputs:** A message **Description:** The user holds their action **Program Units called:** Initiative **Exception:** They do not have an action, unless in the second round.
Wait(event)	**Inputs:** A specified event **Outputs:** A message **Description:** The user waits until a certain event to occur before acting. **Program Units called:** Initiative **Exception:** They do not have action or do not specify an event.
Melee(weapon)	**Inputs:** The weapon they are attacking with **Outputs:** A message **Description:** The user attacks with a melee weapon. **Program Units called:** Roll **Exception:** An invalid weapon is selected
Ranged(weapon)	**Inputs:** The weapon they are attacking with **Outputs:** An event call **Description:** The user attacks with a ranged weapon. **Program Units called:** Roll **Exception:** An invalid weapon is selected.
Location(roll)	**Inputs:** An integer **Outputs:** An integer corresponding to a body map. **Description:** It tells the group where an attack happened on the body chart depending on what was rolled. **Program Units called:** Roll **Exception:** No roll was given

Repeat Attack()	**Inputs:** Nothing **Outputs:** A message **Description:** It repeats the users previous attack action **Program Units called:** Attack **Exception:** There is no previous attack.
Intervention()	**Inputs:** None **Outputs:** A message **Description:** The user interrupts a defender by retargeting the attack to themselves. **Program Units called:** Defend **Exception:** They do not have an action
Move(distance)	**Inputs:** An integer representing how far they want to move **Outputs:** A message **Description:** The user moves distance units. **Program Units called:** Initiative **Exception:** They do not have an action or invalid number
Skip()	**Inputs:** None **Outputs:** A message **Description:** The user skips their action **Program Units called:** Initiative **Exception:** They do not have an action
Aim()	**Inputs:** None **Outputs:** A message **Description:** The user aims at a target and increases their chance of hitting the target during their next action. **Program Units called:** Initiative **Exception:** They do not have an action.

Table 3: The table above lists the program units under the Action Systems. This lists many of the available actions the users can take each round.

Web API Controllers	The Web API controllers system will provide each of our database tables with an interface that our Vue components can persist and retrieve our data. Each Web API controller is a module with its own program units. Each controller has similar standard Web API HTTP methods and has been condensed for readability and formatting. The players Web API controller differentiates since it involves authenticating and registering users.
chrc_profilesController, sessionsController, session_usersController, chat_logController, initiativeTableController	Each Web API controller contains the following methods : ● **GET(ID)**: Returns a single record with the provided ID. ● **GET() :** Returns all records. ● **POST(record):** Adds a new record to the database table. ● **PUT(record):** Updates an existing record based on the passed record. ● **DELETE(ID):** Deletes an existing record with the provided ID. Exceptions can occur during each of these methods if a record is not found or cannot be added to the database due to being empty or incorrect.
playersController,	● **login(playersRecord):** Retrieves the record from the players table based on the passed record. Hashes the password and checks if it matches with the hashed password in the database. If so, returns the playersRecord. An exception is returned if the password does not match ● **register(playersRecord)**: Checks the passed record does not already exist. If it does not, the password is hashed and the playersRecord is added to the table. An exception is returned if the record already exists.

Table 4: The table above lists the program units under the Web API Controllers system. This lists the different controllers this system will have and the program units within each one.

Character Profile	The Character Profile system will allow users to manage the characters that they have made to use in various sessions. The character management system allows users to perform certain actions regarding their characters, like creating new ones and updating existing ones. Users must be logged in in order to access these options.
createCharacter()	When the user clicks the “Create New” button on the character menu, a prompt will appear asking for several values for the new character. **Inputs:** Numerical values or text input from the user, to be used to initialize the character attributes. **Outputs:** A completion message and an updated view of the characters list, with the new character visible. **Description:** A POST request for the new character is made with the inputted information, and the new record is saved to the database under the user’s account. **Program Units called:** chrc_profilesController **Exception:** None
viewCharacter(id)	Within the character menu, users can click on the names of their characters to see a detailed view of them. **Inputs:** None **Outputs:** A modal showing all the relevant information associated with the selected character. **Description:** A GET request is made to the server with the selected character’s ID.

	Program Units called: chrc_profilesController **Exception:** None
editCharacter(id)	Within the character menu, users can click on the “Edit” button underneath a character’s name to begin the editing process. This process functions similarly to createCharacter(), except a PUT request is made instead of a POST. **Inputs:** None **Outputs:** A completion message and an updated view of the characters list. **Description:** A PUT request is made to the server with the selected character’s ID, and the new values. **Program Units called:** chrc_profilesController **Exception:** None
deleteCharacter(id)	Within the character menu, users can click on the “Delete” button associated with each character to delete them. **Inputs:** None **Outputs:** A completion message and an updated view of the characters list. **Description:** A DELETE request is made to the server with the selected character’s ID. **Program Units called:** chrc_profilesController **Exception:** None

Table 5: The table above lists the functions of the Character Profile system.

Session Management	Upon logging in, users are presented with all the available sessions that have been created by all players. All users are allowed to join a session, however if one is logged in as a GM, then they will gain access to a “Create New” button and “Delete” buttons associated with each session.
joinSession(id)	Each session has a “Join” button that users can click to join a real-time session with other players wishing to join the same session. Note that sessions that have already started will not be able to be joined. This program unit can be thought of as the client-side version of the “JoinSession” unit from the Real-Time Sessions system. **Inputs:** None **Outputs:** Connects to the session and brings the user to the initiative table interface to begin playing. If the session is unable to be joined, an error message will appear, and the user will remain on the same page. **Description:** Starts the process for joining a session. **Program Units called:** JoinSession(playerName, sessionName) **Exception:** None
createSession()	Only GMs are allowed to create new sessions. Doing so will create a new item that will be displayed on the homepage for logged-in users. Upon clicking the “Create New” button, a prompt will appear, asking for a name for the new session.

	Inputs: Text input from the GM. **Outputs:** A completion message and an updated view of the sessions list. **Description:** A POST request is sent to the server with the data from the user. **Program Units called:** sessionsController **Exception:** None
deleteSession(id)	Only GMs can delete sessions. If a session to be deleted is in progress, a prompt will be shown confirming if the GM wants to force-end the session. **Inputs:** Confirmation from the GM **Outputs:** All players within the session are kicked, and the SignalR hub associated with the session is closed. The session is then removed from the database. A completion message is shown and the sessions list is updated. **Description:** A DELETE request is sent to the server with the data from the user. **Program Units called:** sessionsController **Exception:** None

Table 6: The table above lists the functions of the Session Management system.

Database tables

The following are the database tables for players, character profiles, sessions, session users, chat log, and initiative table. Each of our database tables has general metadata attached; this includes the record's creation date, the date of the last edit made to the record, the record's creator, the last editor to the record, and if the record is deleted. The player's database table includes the player identification, the player's name, e-mail, the account password, and a flag to determine if the player is a game master. The character profile database table includes the character profile identification, the identification of ownership, the character profile group name, and if they are a non-playable character. The sessions table consists of the session identification, the session's name, and if the session is joinable. The session users table includes the user's identification, the user's session identification, a player identification for being able to determine if they joined the session, and character profile identification for retrieving character information for the session. Next, the chat log table includes message identification, the session identification for which session the message belongs to, the identification for where the message is being sent to, the identification for where the message came from, the message itself, and if it is a direct message or a global message. Finally, the initiative table includes the table identification, the session identification for which session the table belongs to, the character profile identification for which character the initiative value belongs to, and the status. Any primary keys and foreign keys have been bolded for readability and formatting.

creationDate	lastEdit	creator	lastEditor	isDeleted

Table 7: General metadata appended to all the database tables.

playerID	name	email	password	isGM

Table 8: Player's database table.

character ProfileID	**characterProfile _playerID**	characterProfile GroupNameTe	isNPC	PER	MD
SPK	AGL	STR	CON	HTP	LHTP
AIM	T.AIM	MOVE	FLY	PAIN	BlD
T.PEN	P.PEN	Cur_HTP	MEM	WIS	MS
MR	MD	CHR	PB	MAN	RR
PR	SANITY	SEX	InitBonus	Num_actions	

Table 9: Character Profile database table. Contains the many stats of character profiles.

sessionID	name	isJoinable

Table 10: Sessions database table.

sessionUserID	**sessionUsersID_ sessionID**	**sessionUsersID_ playerID**	**sessionUsersID_ chrcProfilesID**

Table 11: Session user's database table

messageID	**messageID_ sessionID**	**messageID_ TOplayerID**	**messageID_F ROMplayerID**	message	isDM

Table 12: Chat log database table

initTableID	**initTableID_sessionID**	**initTableID_chrcProfilesID**	initTableStatus

Table 13: Initiative database table

WeaponID	**WeaponID_character ProfileID**	weaponName	weapon Class	isRange
weaponStats	weaponShots	weaponDAM	weapon Skill	weaponChatMsg

Table 14: Weapons database table

ArmorID	**ArmorID__character ProfileID**	armorName	armorClass	armorLoc
armorASB	armorRFLT	TargetFactor (range only)		

Table 15: Armors database table

ShieldID	**ShieldID___character ProfileID**	shieldName	shieldClass	shieldLoc
shieldASB	shieldRFLT	shieldIntPercent		

Table 16: Shields database table

Detailed design

The following figures contain our detailed design diagrams.

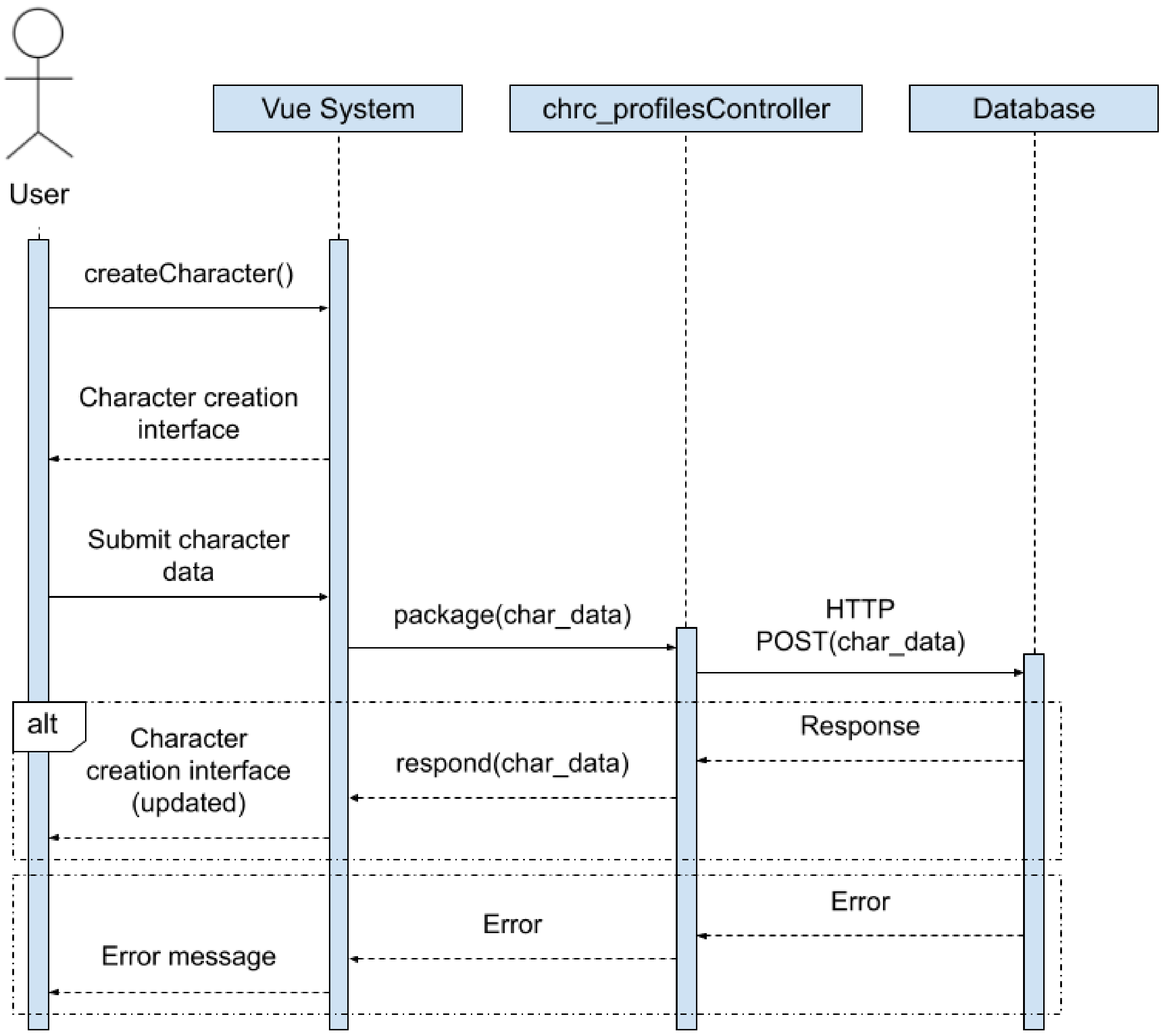


Fig. 2: The figure above contains a sequence diagram that demonstrates the process of creating a new character.

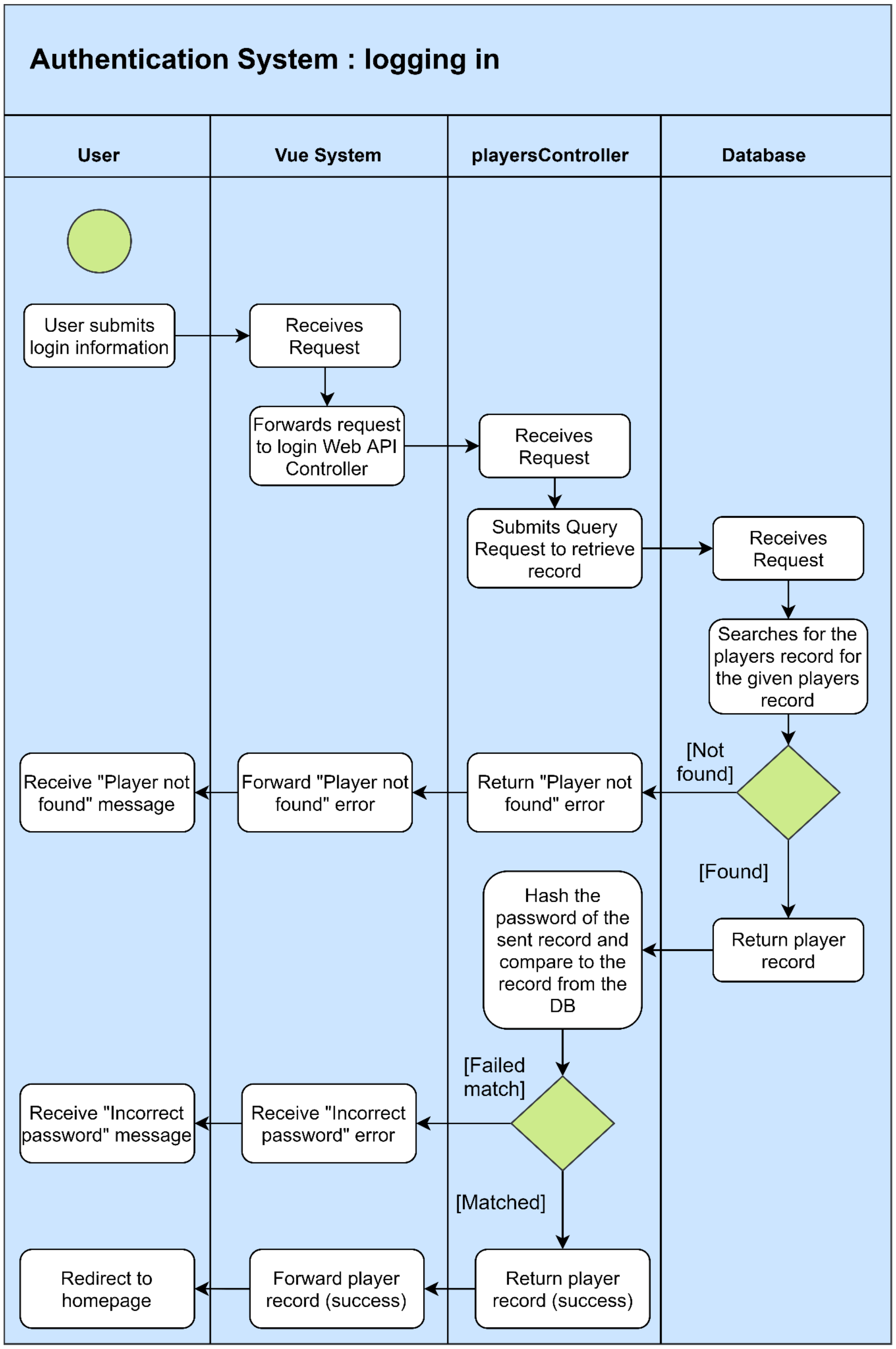

Authentication System : logging in
User
Vue System
playersController
Database
User submits login information
Receives Request
Forwards request to login Web API Controller
Receives Request
Submits Query Request to retrieve record
Receives Request
Searches for the players record for the given players record
[Not found]
[Found]
Return "Player not found" error
Forward "Player not found" error
Receive "Player not found" message
Return player record
Hash the password of the sent record and compare to the record from the DB
[Failed match]
[Matched]
Receive "Incorrect password" error
Receive "Incorrect password" message
Return player record (success)
Forward player record (success)
Redirect to homepage

Fig. 3: The figure above contains an activity diagram that demonstrates how a login request from a player is handled.

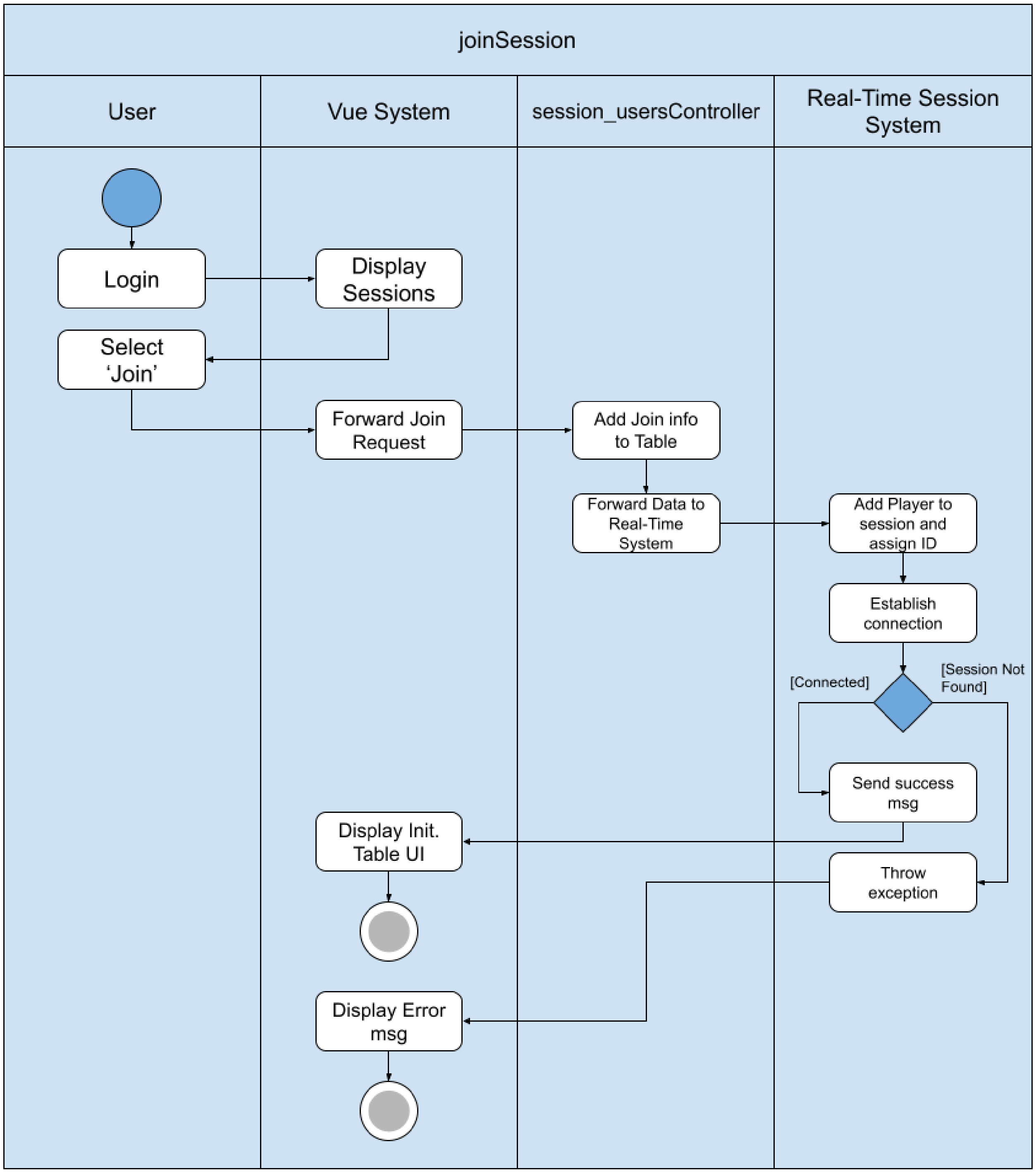


Fig. 4: The figure above contains an activity diagram that demonstrates what goes on when a user joins a session.

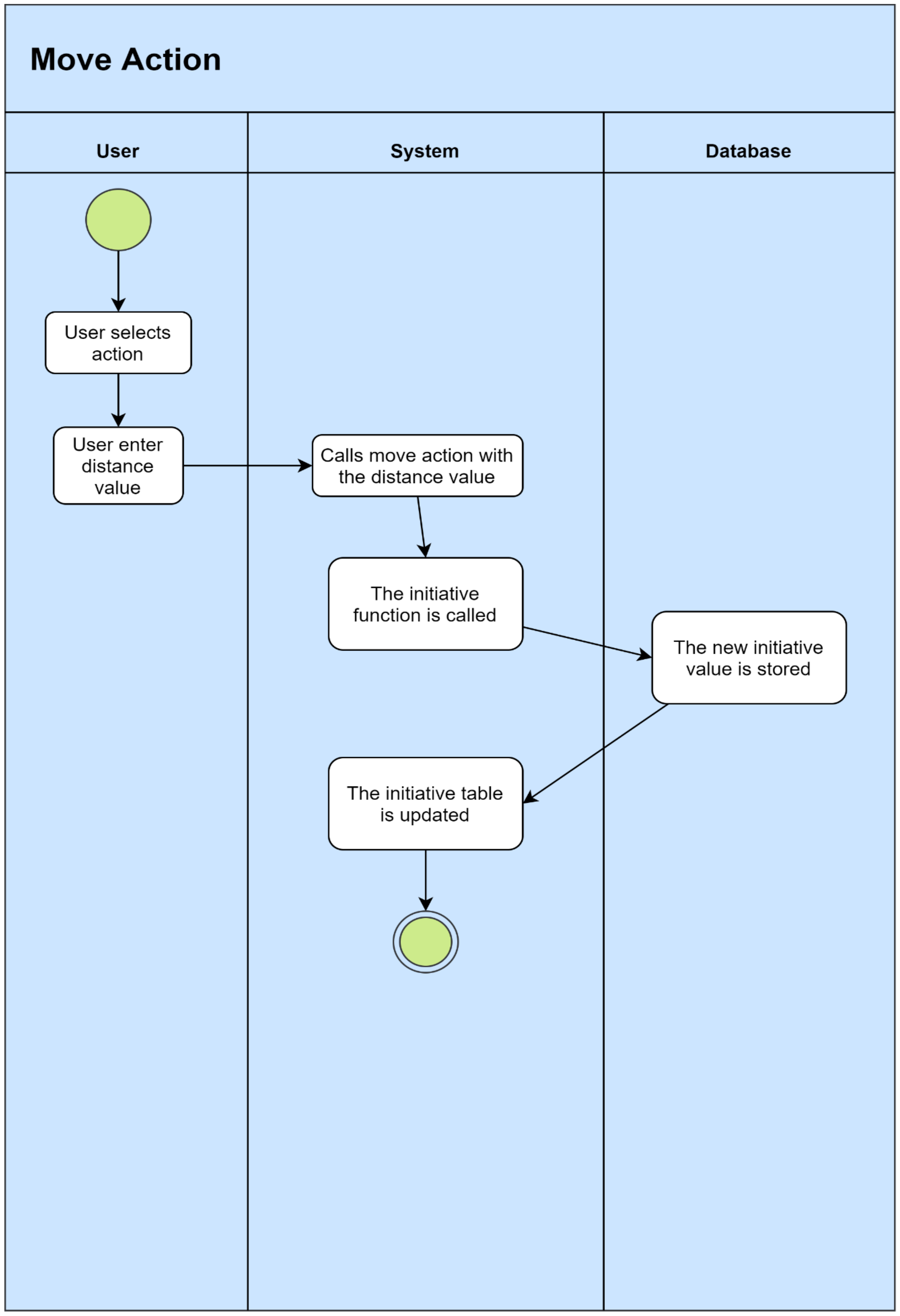

Move Action
User
System
Database
User selects action
User enter distance value
Calls move action with the distance value
The initiative function is called
The new initiative value is stored
The initiative table is updated

Fig. 5: The figure above contains an activity diagram that demonstrates how a move action from a player is handled.

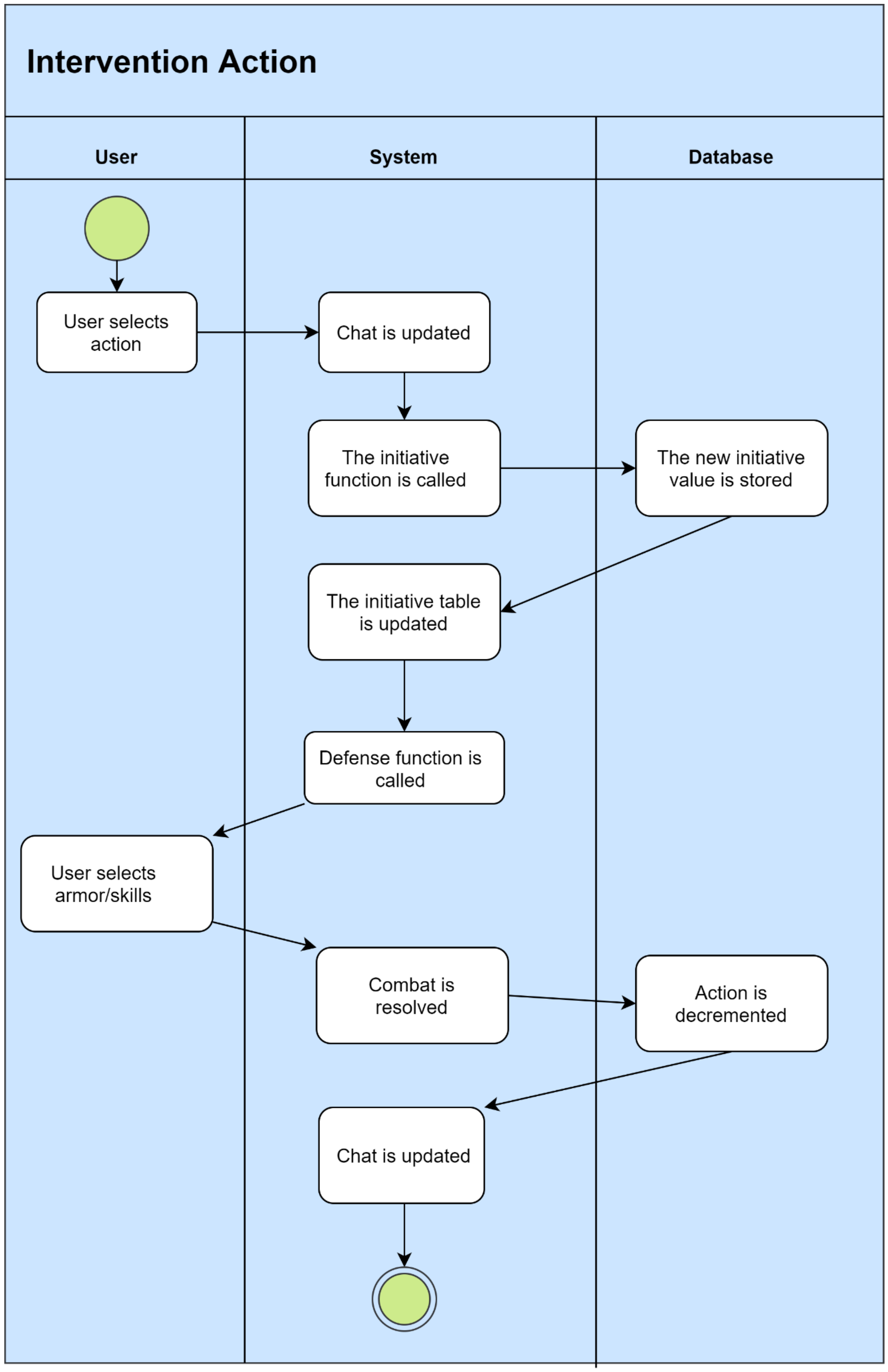

Intervention Action
User
System
Database
User selects action
Chat is updated
The initiative function is called
The new initiative value is stored
The initiative table is updated
Defense function is called
User selects armor/skills
Combat is resolved
Action is decremented
Chat is updated

Fig. 6: The figure above contains an activity diagram that demonstrates how an intervention action from a player is handled.

Hardware design

The initiative table application does not have much in terms of hardware. It is primarily a software solution, but it does make use of server hardware services for hosting. The client has provided access to a hyper-v web server and MS-SQL Database server. The database server is an instance of abstracted hardware. Other than these configurations, the team's focus is on implementing a software-based solution to the client's needs.

User interface design

The following figures are snapshots of the system's potential user interface. Note: The user interface slightly varies between the two prominent roles GM and player. Buttons colored in yellow only appear for the GM; everything else appears for both. Buttons colored in purple represent dropdown menus.

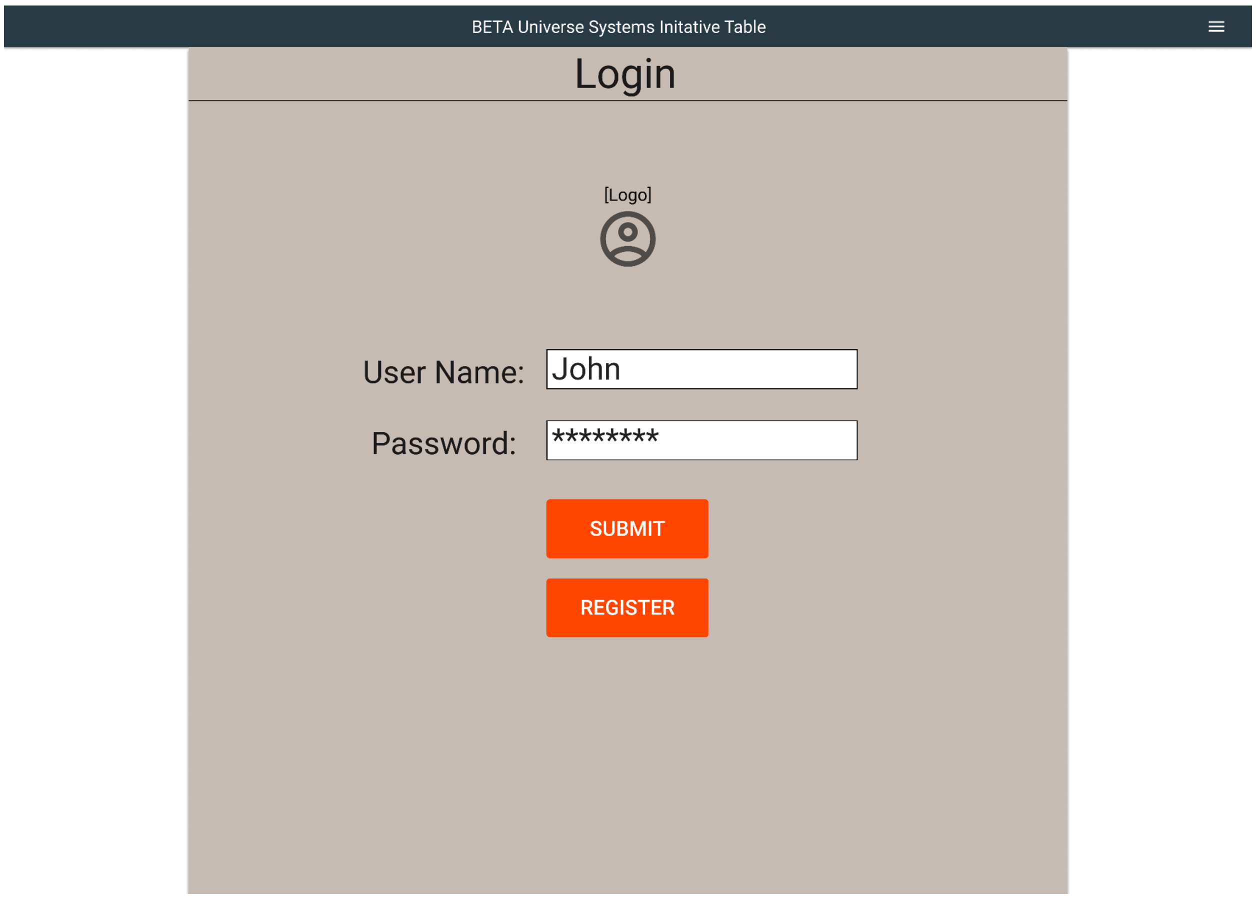


Fig. 7: Login UI (Desktop). The initial login screen for the website. Users with accounts can use their username and password to access their account and enter the main portion of the application. New users can access the registration through the register button.

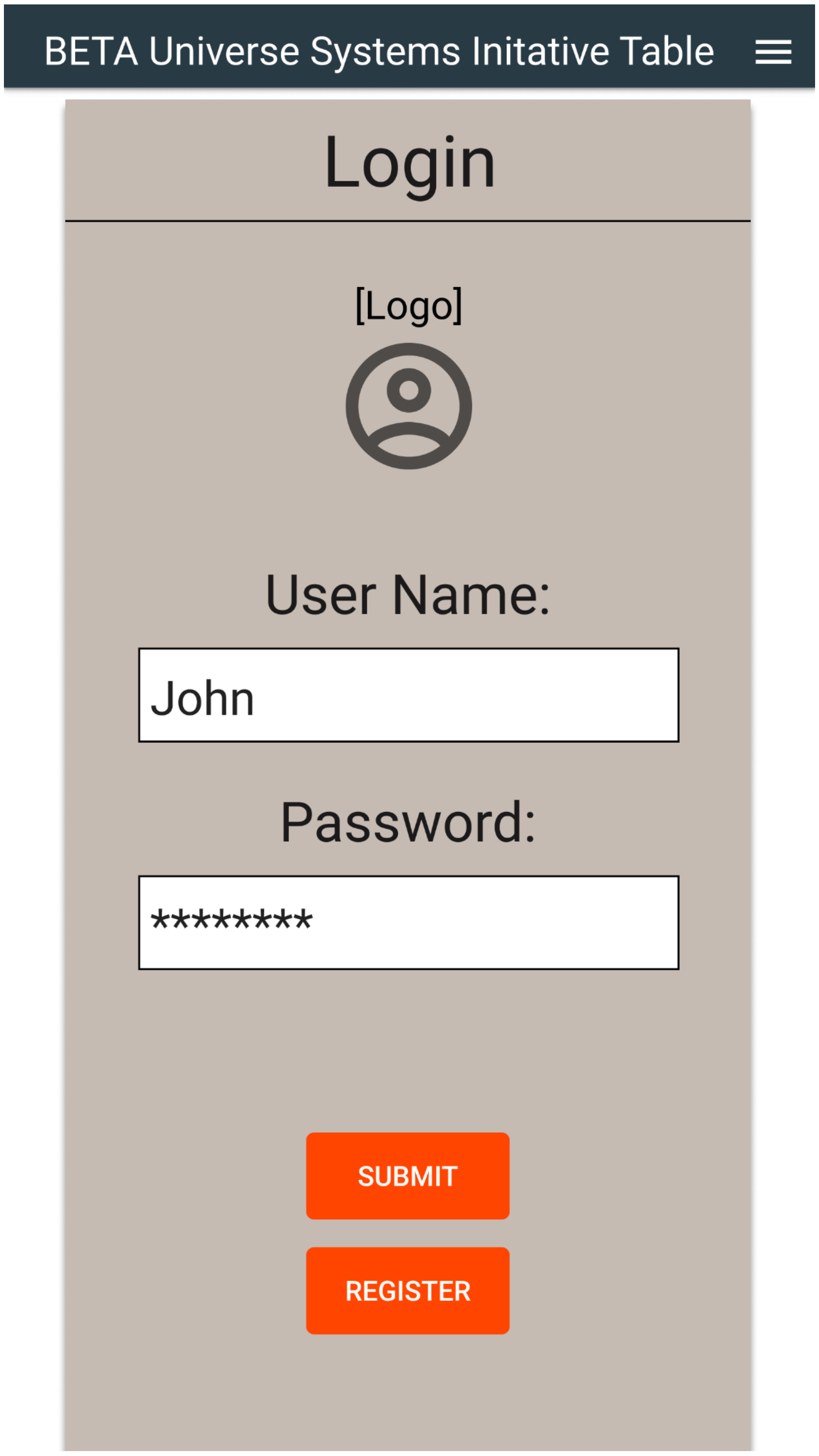
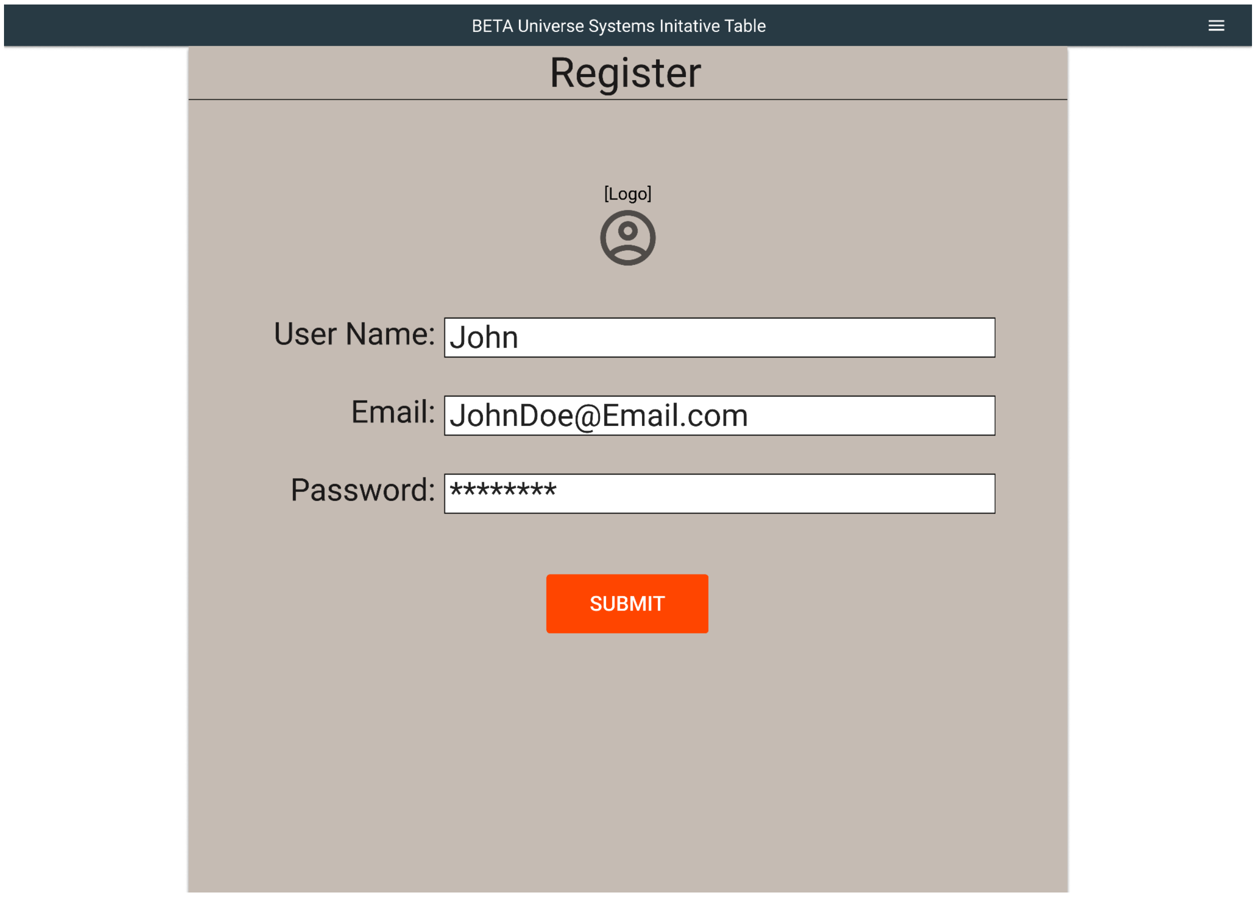


Fig. 8: Login UI (Mobile). Mobile version of the login page. It is not much different from the desktop version other than scaling the size to the appropriate mobile aspect ratio.

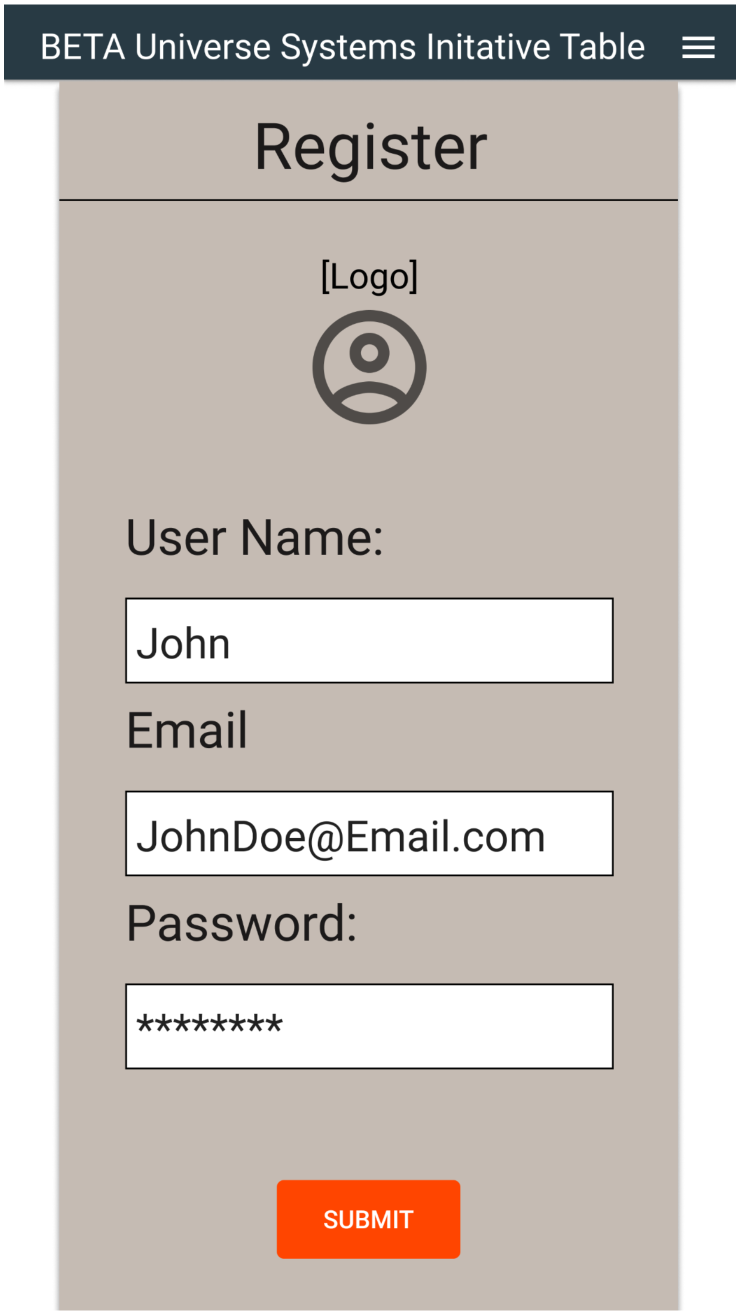


User Name: John

Email: JohnDoe@Email.com

Password: ********

Fig. 9: Register UI (Desktop). This screen allows the user to register their email as an account for logging in to the Beta Universe Initiative Table App. The user can provide a username, email address and password.


BETA Universe Systems Initative Table

Register

[Logo]

User Name:

John

Email

JohnDoe@Email.com

Password:

****

Fig. 10: Register UI (Mobile). This is the same functionality as the Desktop register screen. The only difference is the mobile optimized layout of the fields.

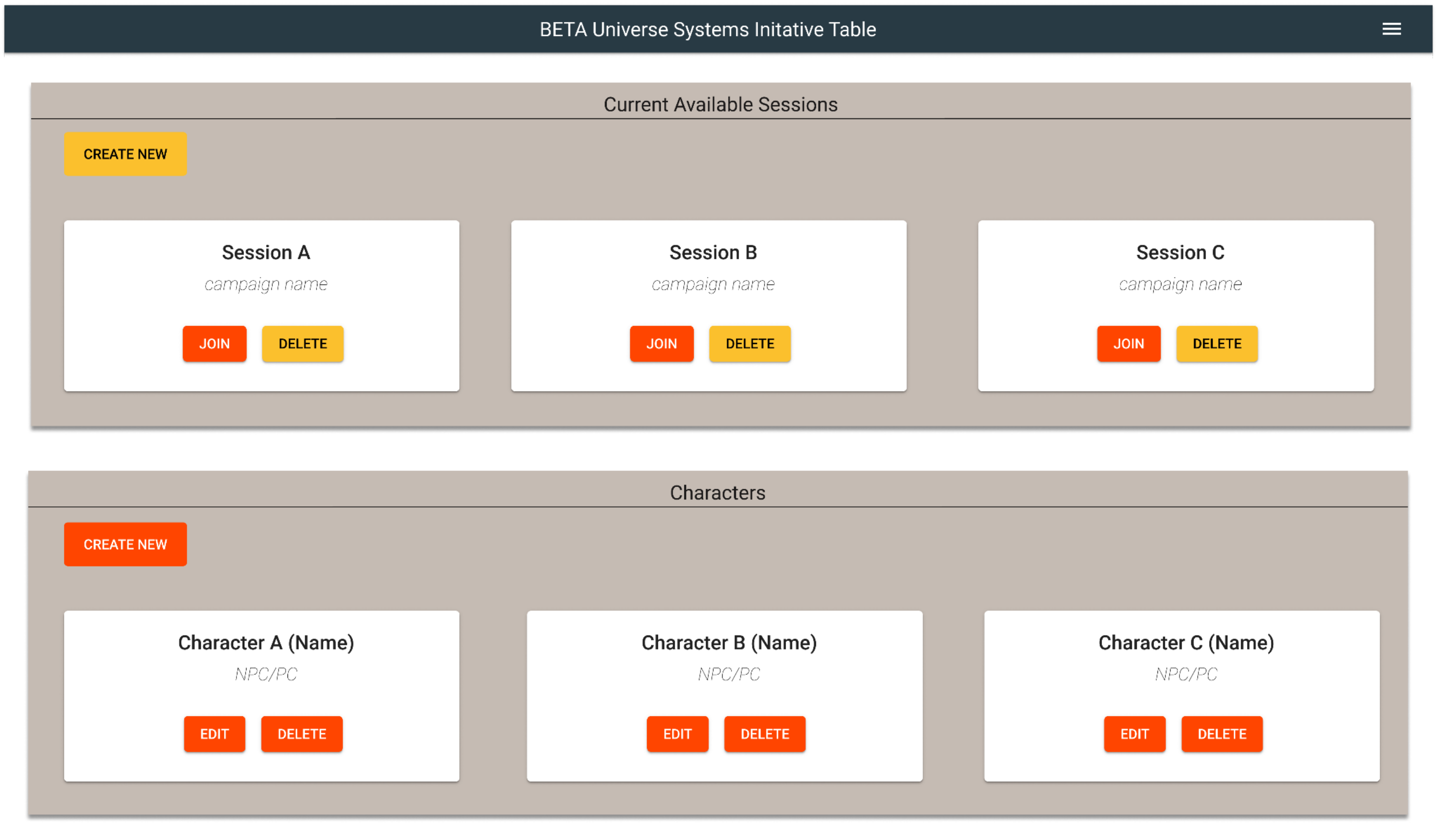


Fig. 11: Home page UI (Desktop). After logging in as a regular player, the player is presented with a section of the available sessions. Each card in the section represents a session with a button to join. There is also a section for the characters the player has created. Each card represents a character, and there are buttons to edit, delete and add the individual characters. When logged in as a GM, there are two additional functions: creating a new session and deleting a session.

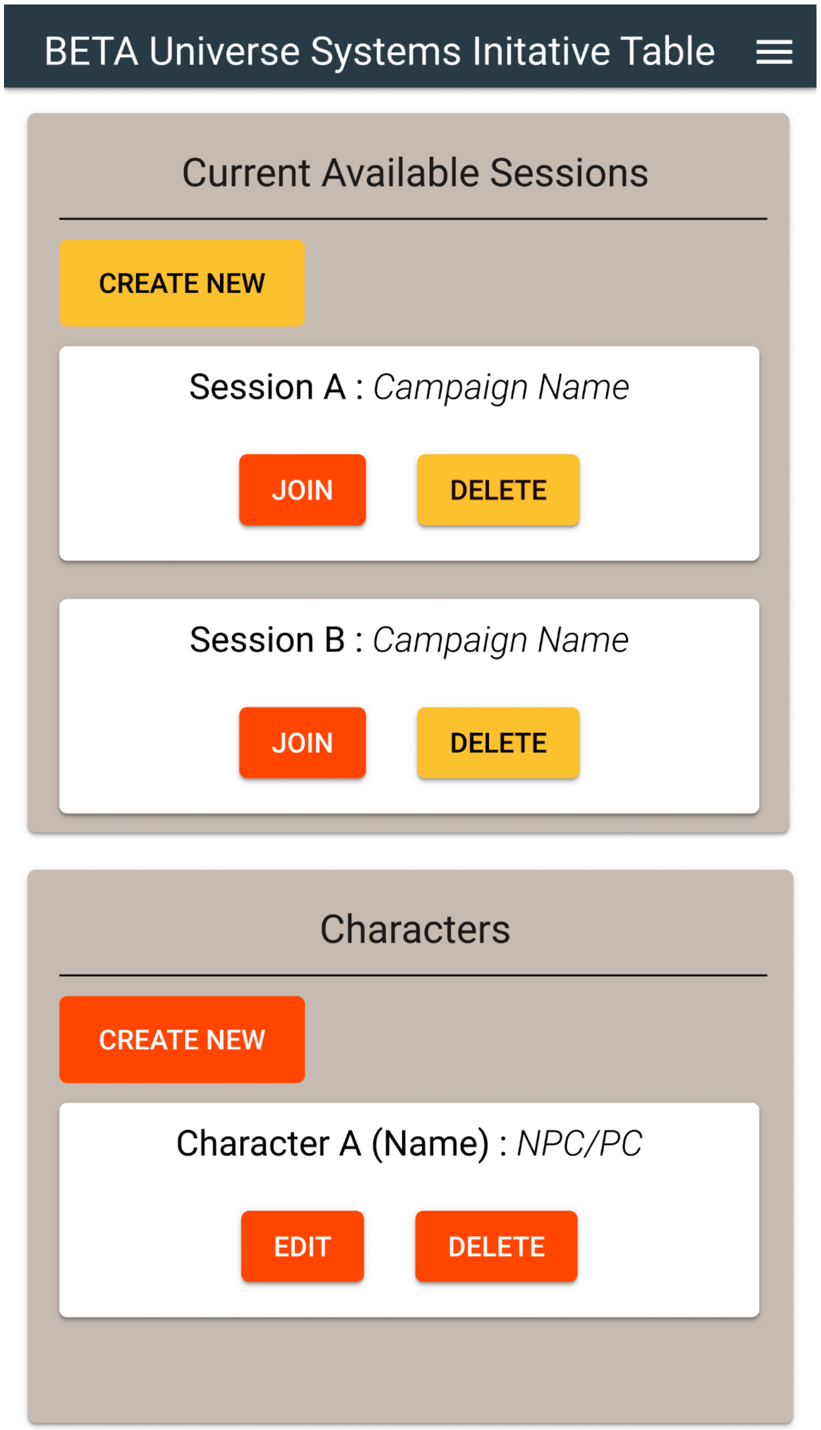
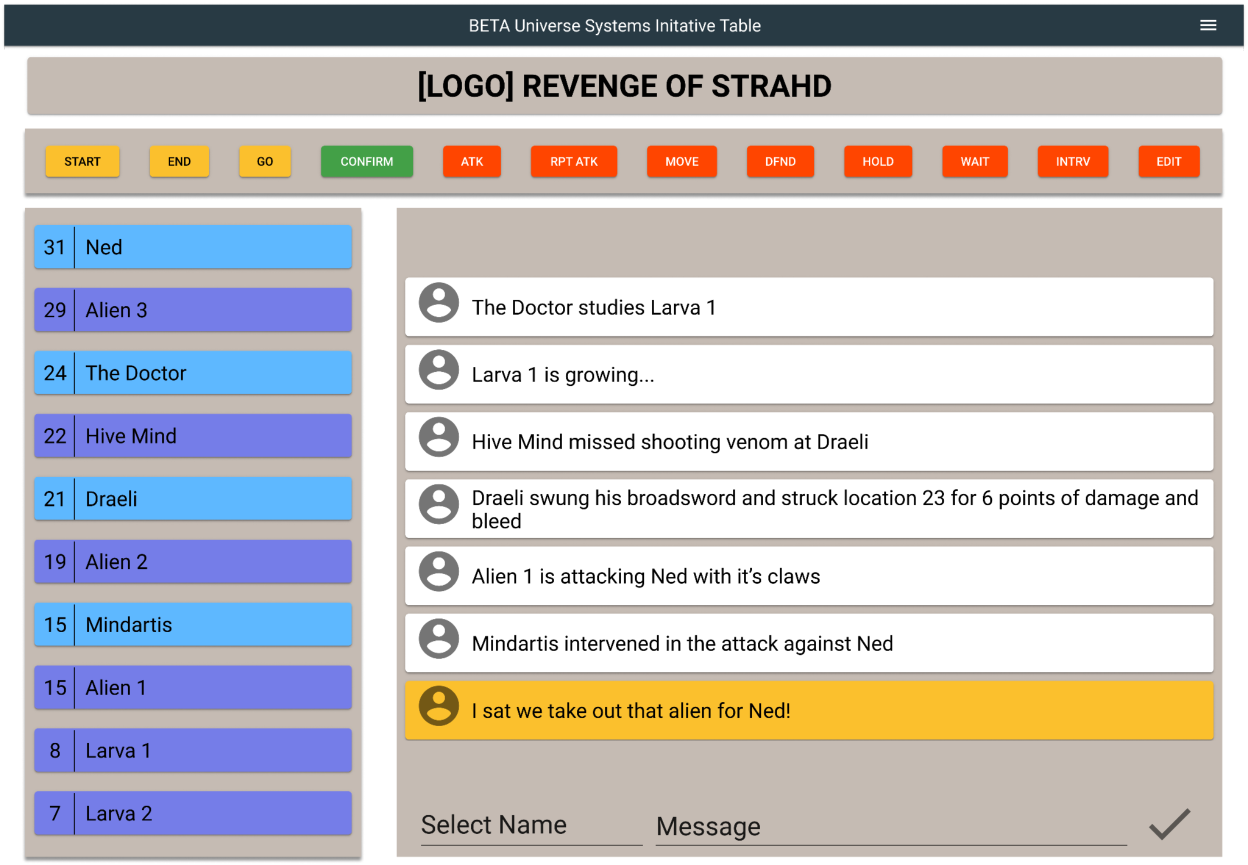

BETA Universe Systems Initative Table
Current Available Sessions
CREATE NEW
Session A : *Campaign Name*
JOIN
DELETE
Session B : *Campaign Name*
JOIN
DELETE
Characters
CREATE NEW
Character A (Name) : *NPC/PC*
EDIT
DELETE

Fig. 12: Home page UI (Mobile). For the mobile version of the homepage UI, the same functionality is present; however, the two-session and character sections have been resized to fit a mobile size screen. The cards now take up less vertical space to make up for the change in screen size.



31	Ned
29	Alien 3
24	The Doctor
22	Hive Mind
21	Draeli
19	Alien 2
15	Mindartis
15	Alien 1
8	Larva 1
7	Larva 2

The Doctor studies Larva 1

Larva 1 is growing...

Hive Mind missed shooting venom at Draeli

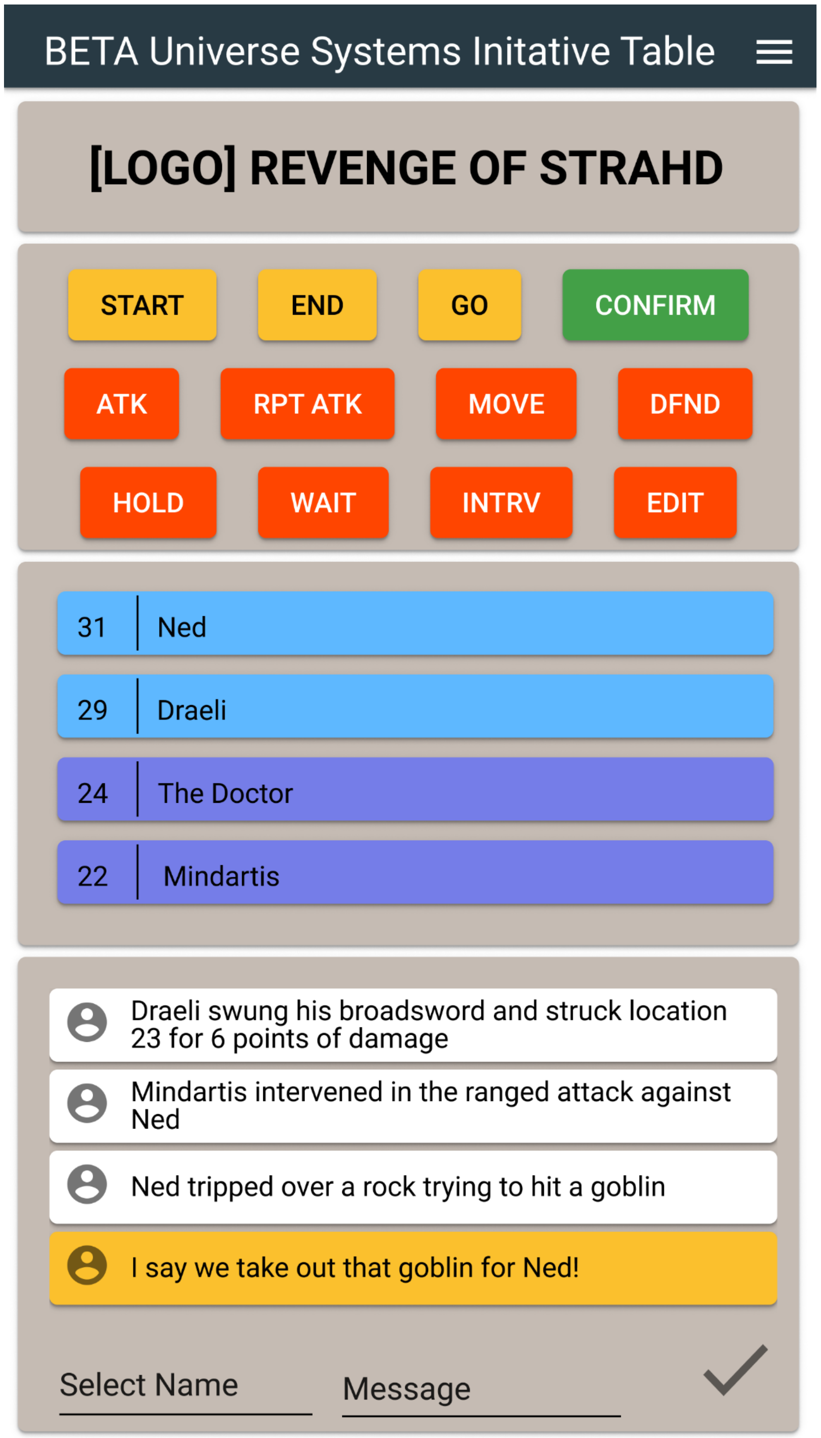
Draeli swung his broadsword and struck location 23 for 6 points of damage and bleed

Alien 1 is attacking Ned with it's claws

Mindartis intervened in the attack against Ned

I sat we take out that alien for Ned!

Fig. 13: Session UI (Desktop). The session UI is one of the critical components of the application. This UI comes after a player joins a session and displays game-specific information and interaction. The top bar shows the campaign name chosen by the GM. The bar just underneath the campaign name is the button palette, which gives the players interactivity to make actions within the game. The left sidebar holds the initiative table information, which keeps track of the player turn order. The color of each name denotes the different teams. The right and largest area display the chat log. The chat log displays actions that have been taken, the results of those actions, and player chat messages. The bottom of the chat area is the area where players can choose who to send a message, type in their message, and click the send icon.



31	Ned
29	Draeli
24	The Doctor
22	Mindartis

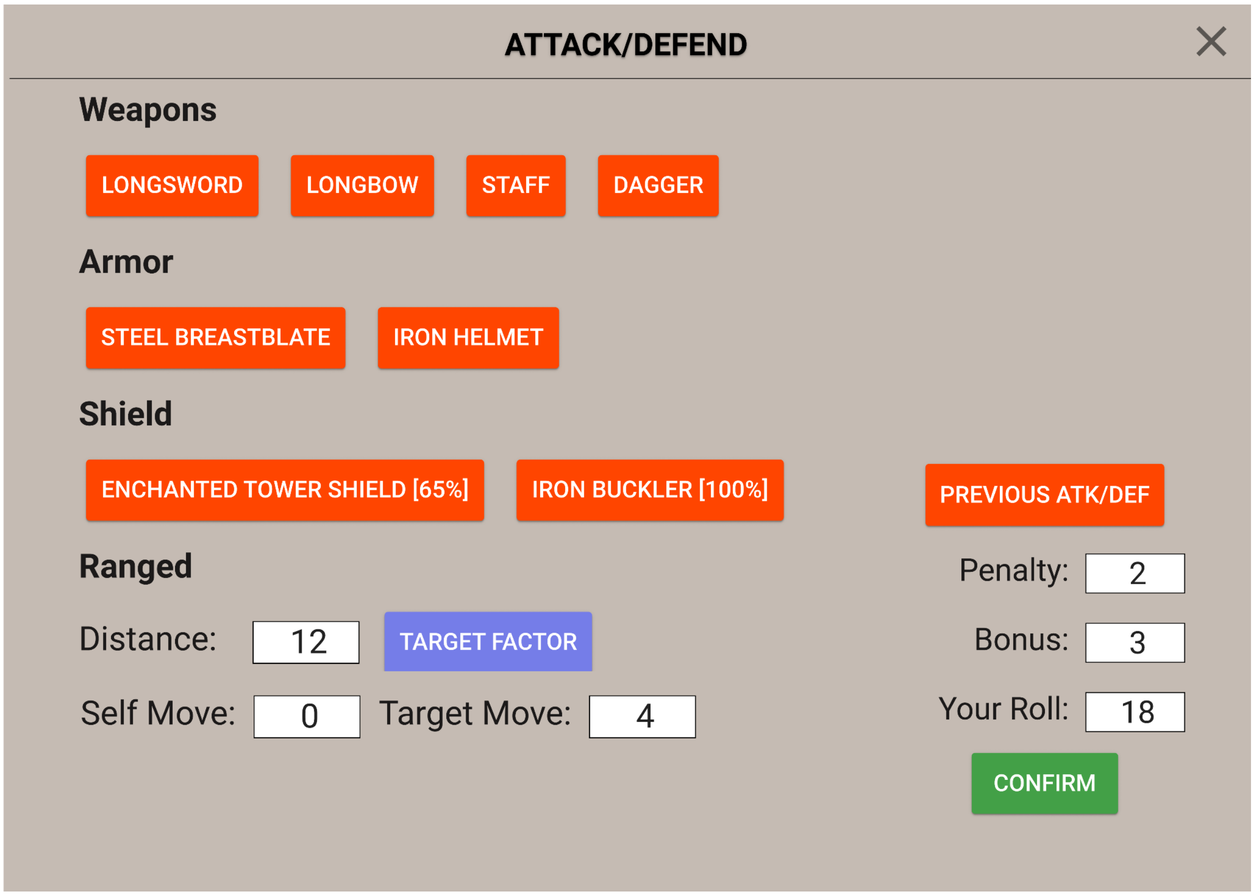
Draeli swung his broadsword and struck location 23 for 6 points of damage

Mindartis intervened in the ranged attack against Ned

Ned tripped over a rock trying to hit a goblin

I say we take out that goblin for Ned!

Fig. 14: Session UI (Mobile). The same functionality is present for the mobile version of the session UI; however, the four main sections have been condensed vertically to present the information. The button palette now wraps to make sure they are all accessible.



Penalty: 2
Bonus: 3
Your Roll: 18
Distance: 12
Self Move: 0
Target Move: 4

Fig. 15: Attack/Defend Modal (Desktop). This modal prepares players for the main combat loop. After declaring an attack from the session screen, the player can set their weapon, armor, shield, and data for a ranged attack. The player can also select "Previous ATK/DEF" to auto-fill the selection from the last attack or defense the player used. The player can enter data for the roll penalty, bonus and roll if the player chose to roll their own dice. Ranged data should only appear if the player has chosen a ranged weapon. The blue "Target Factor" includes a drop-down list of various calculations based on the player's choice of attack. The same screen will appear for a player who is being attacked, so that they may set their own data for their defense.

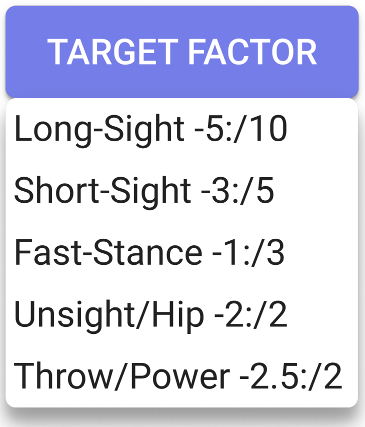
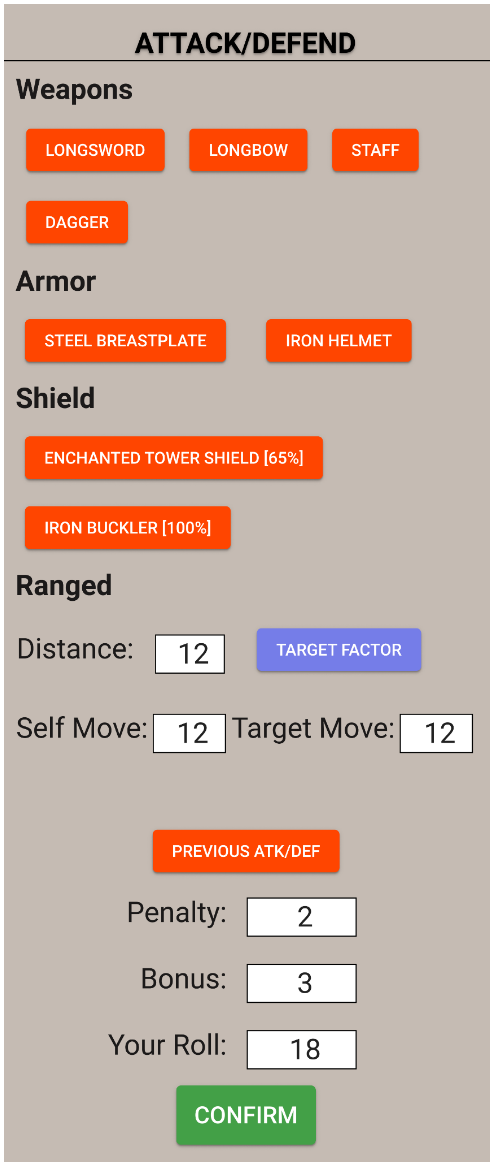


Fig. 16: Dropdown menu for Target Factor in ATK/DFD Modal. This is an example of the target factor calculations for the dropdown menu.



Distance: 12

Self Move: 12 Target Move: 12

Penalty: 2

Bonus: 3

Your Roll: 18

Fig. 17: Attack/Defend Modal (Mobile). This includes the same information and interactions as the desktop version. This screen can contain a large amount of information so the player will be able to scroll through all the data before confirming.

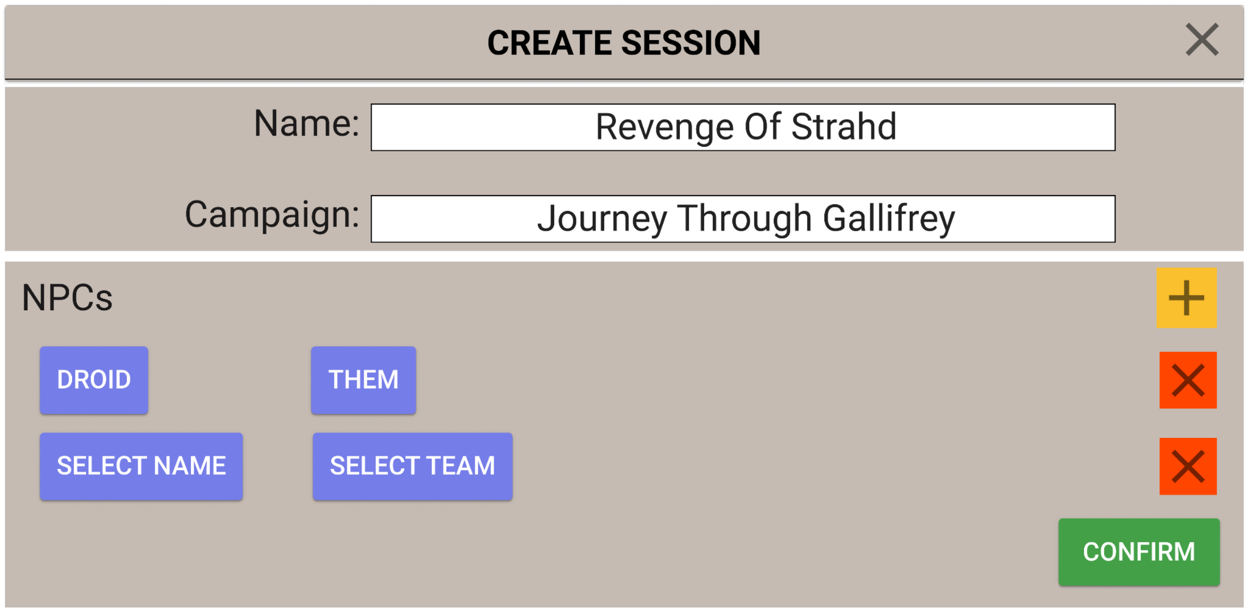


Fig. 18: Create Session Modal (Desktop). This modal comes from the GM selecting to create a new session from the home screen. The GM can enter the name and campaign for the session. The GM can also add NPCs to the session from this screen by selecting the plus sign. This will cause a new row to appear under the NPCs area. Each row has options to select an NPC name from a list and set their team from a list. The name list will contain names of character profiles previously created by the player.

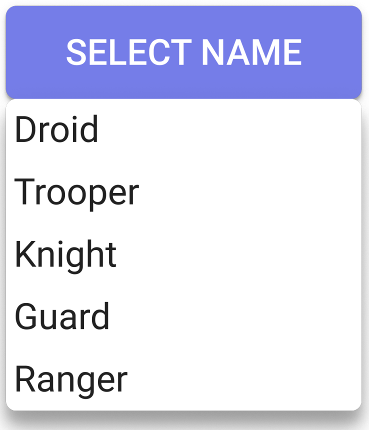


Fig. 19: Example dropdown for Select Name in Create Session modal.

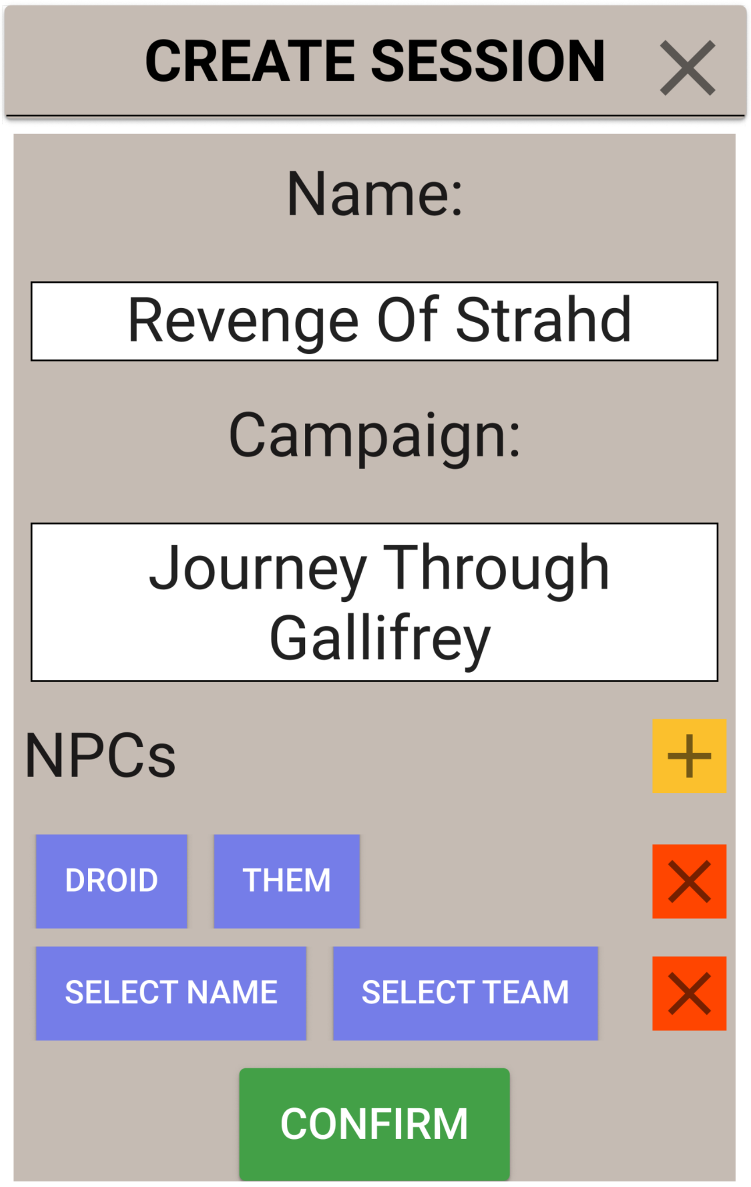


Fig. 20: Create Session modal (Mobile). There is not much difference between the mobile and desktop versions other than screen size.

Version control

The client informed the team which informed the graders/evaluators for the classes of Computer Science (CS) 425 - Software Engineering and Computer Science (CS) 426 - Senior Project in Computer Science at the University of Nevada, Reno that the Intellectual Property (IP) belongs to the client, and the client wishes for the code and implementation to be private. A compromise has been made that the repository on GitHub will be private and that the graders/evaluators will have access to the repository but will only view the repository for grading purposes. Due to this compromise, a link is not added to this documentation, but invitations have been sent out to the parties involved.

Work Contribution

	Andy Alarcon	Jacob Gayban	Mark Graham	Jacob Tucker	Griffin Wagenknecht
Project Assignment 3 Paper (Writing sections and formatting)	2.0	2.0	4.0	2.0	2.0
System-Level Diagram	2.0	0.0	0.0	0.0	0.0
Detailed Design Charts	2.0	3.0	0.0	0.0	2.0
Program Units	2.0	0.0	0.0	0.0	4.0
UI Design	0.0	0.0	0.0	3.5	0.0
Database Tables	0.5	0.0	2.5	2.0	0.0
Researching Web Development	1.0	4.0	3.0	2.0	0.0
Team meetings	2.0	2.5	2.0	2.0	2.0
Total	11.5	11.5	11.5	11.5	10.0

Table 17: The table above shows the amount of time spent by each team member on each activity.