

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

## **BETA Universe Systems Initiative Table Application**

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

Dr. David Feil-Seifer, Dr. Devrin Lee, Vinh Le

John Molt

February 18, 2022

# Table of Contents

<b>Abstract</b>	<b>2</b>
<b>Recent Project Changes</b>	<b>2</b>
<b>Updated Specification</b>	<b>3</b>
Summary	3
Updated Technical Requirements	3
Updated Use Case Modeling	6
<b>Updated Design</b>	<b>12</b>
Summary	12
Updated Level Designs	12
Updated User Interface Design	28
<b>Glossary</b>	<b>38</b>
<b>Engineering Standards and/or Technologies</b>	<b>40</b>
<b>Project Impact and Context Considerations</b>	<b>41</b>
<b>References</b>	<b>42</b>
<b>Work Contribution</b>	<b>44</b>

## Abstract

In tabletop role-playing games, players can be burdened by extraneous calculations and player order, which disrupt the flow and narrative of the game. The BETA Universe Systems Initiative Table Application is a mobile-optimized companion web application that aims to digitize and streamline gameplay components from the tabletop role-playing game "BETA Universe Systems." Players can register for accounts, manage player characters, and participate in game sessions that feature real-time communication via a chat log and combat loop. Gamemasters can also register for accounts, manage the non-player characters and manage game sessions along with running them. This document outlines the project's design and specification.

## Recent Project Changes

The recent changes that have been done to the project are that a dice rolling function has been added to allow for initiative value tracking. This makes it so that the purpose of initiative order can be met which was a mission for this project. Additionally, weapon creation, armor creation, and shield creation has been added to the project because we the goal of this project is to help minimize the distraction from the narrative in the game play and the group agreed that having all of this information inside the application with all combat features involved, then the calculations need for armor, weapons, and shields can be mitigated through the application.

# Updated Specification

## Summary of Changes

Several additions were made to our requirements in order to accommodate most of the features present within the BETA Universe Systems combat. Major changes are as follows: Weapons, armors, and shields have all been added as part of each character, and are stored in the database. Users are now able to define items for their characters to use within combat. A new library was added to the project in order to aid in generating more complex dice rolls.

## Updated Technical Requirements Specification

New functional requirements:

- ITA shall allow the user to create/edit/delete weapons, armors, and shields for each character.
- ITA shall generate dice rolls upon user request.
- ITA shall allow the user to edit characters while in a session.
- ITA shall guide the user through the attack loop.
- ITA shall keep track of player actions within a session.
- ITA shall calculate a player's initiative after they make an action.
- ITA shall allow the user to select the "repeat attack" action.

## Functional Requirements

FR01.	[1]	ITA shall allow the user to create an account.
FR02.	[1]	ITA shall allow the user to authenticate into their account.
FR03.	[1]	ITA shall allow the user to create a session.
FR04.	[1]	ITA shall allow the user to delete a session.
FR05.	[1]	ITA shall allow the user to join a real-time session.
FR06.	[1]	ITA shall allow the user to send messages to other users.
FR07.	[1]	ITA shall allow the user to create character profiles.
FR08.	[1]	ITA shall display a character creation interface.
FR09.	[1]	ITA shall allow the user to edit character profiles.
FR10.	[1]	ITA shall allow the user to delete character profiles.
FR11.	[1]	ITA shall display the initiative table data.
FR12.	[1]	ITA shall display a chat log.
FR13.	[1]	ITA shall allow the user to select an action.
FR14.	[1]	ITA shall allow the user to select one or more targets.
FR15.	[1]	ITA shall allow the user to confirm their action.
FR16.	[1]	ITA shall display the “attack” interface.
FR17.	[1]	ITA shall allow the user to input a value for an attack roll.
FR18.	[1]	ITA shall allow the user to select a weapon for an “attack” action.
FR19.	[1]	ITA shall display the “defense” interface for an attacked user to defend.
FR20.	[1]	ITA shall allow the user to input values for the “defense” action.
FR21.	[1]	ITA shall allow the user to select a defensive item for a “defense” action.
FR25.	[1]	ITA shall allow the user to select the “move” action.
FR26.	[1]	ITA shall allow the user to select the “hold” action.
FR27.	[1]	ITA shall allow the user to select the “wait” action.
FR28.	[1]	ITA shall display actions in the chat log for record keeping.
FR29.	[1]	ITA shall allow the game master to start and end sessions.
FR30.	[1]	ITA shall allow the game master to signal to players.
FR33.	[1]	ITA shall allow the user to create/edit/delete weapons, armors, and shields for each character.
FR34.	[1]	ITA shall generate dice rolls upon user request.
FR35.	[1]	ITA shall allow the user to edit characters while in a session.
FR36.	[1]	ITA shall guide the user through the attack loop.
FR37.	[1]	ITA shall calculate a player’s initiative after they make an action.
FR38.	[1]	ITA shall allow the user to select the “repeat attack” action.
FR22.	[2]	ITA shall display an intervention prompt for non-current action users.
FR23.	[2]	ITA shall display an “intervention” interface.
FR24.	[2]	ITA shall allow the user to input values for “intervention” action if qualified.
FR39.	[2]	ITA shall keep track of player actions within a session.
FR31.	[3]	ITA shall generate an auto XP tally report.
FR32.	[3]	ITA shall add endurance statistic tracking.

**Table 1:** All the requirements that ITA will meet by the end of Fall 2021 semester, [1], and Spring 2022 semester, [2], additional items that are not required, but nice to have, [3].

New non-functional requirements:

- ITA shall use an accessible color scheme to aid users.

### Non-Functional Requirements

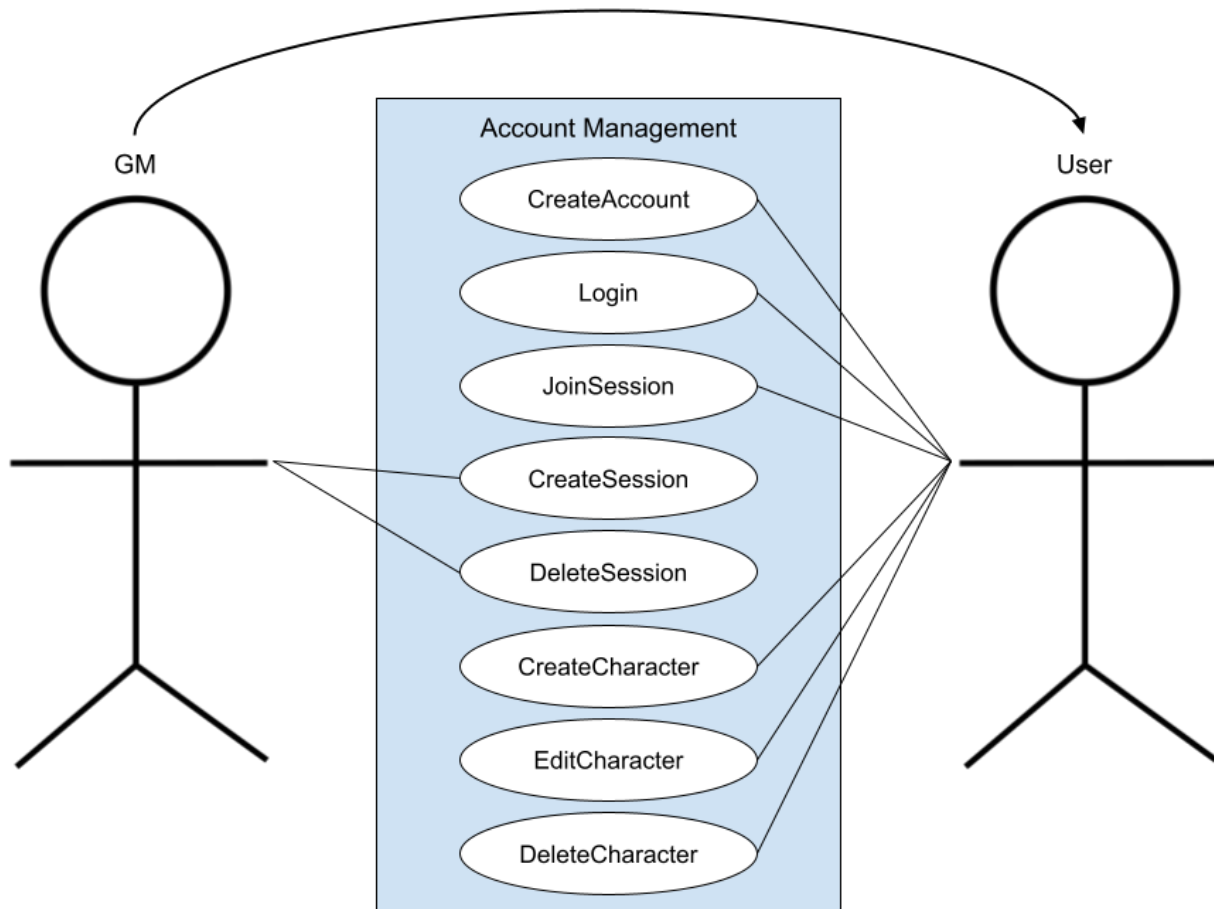
NFR01.	[1]	ITA will utilize Vue.js., Vuetify, Axios., ASP.NET Core. and SQL databases.
NFR02.	[1]	ITA will contain a home interface.
NFR03.	[1]	ITA will contain a session interface.
NFR04.	[1]	ITA will contain a last modified date field for all SQL table records.
NFR05.	[1]	ITA will contain a creation date field for all SQL table records.
NFR06.	[1]	ITA will contain a delete flag field for all SQL table records.
NFR07.	[1]	ITA will contain a user record creation field for all SQL table records.
NFR08.	[1]	ITA will be optimized on mobile devices.
NFR09.	[2]	ITA shall use an accessible color scheme to aid users.

**Table 2:** All the non-functional requirements that ITA will meet by the end of Fall 2021 semester, [1], and Spring 2022 semester, [2], additional items that are not required, but nice to have, [3].

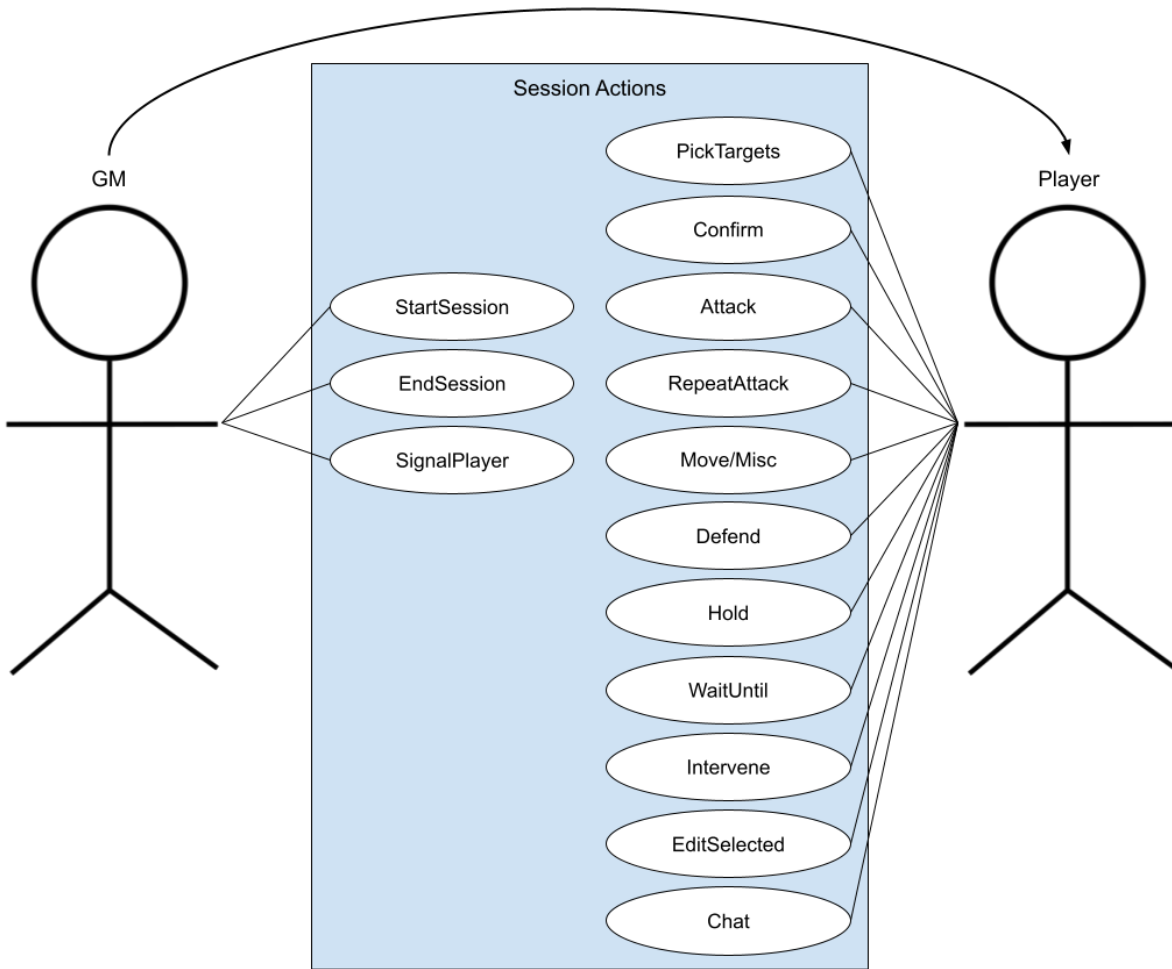
# Updated Use Case Modeling

Our use cases saw no changes since last semester.

## Use Case Diagram



**Fig. 1:** Use case diagram showing the available actions users have when managing their accounts.



**Fig. 2:** Use case diagram showing the actions players and GMs can take during a session.



## Detailed Use Cases

Note: Both players and GMs share certain actions. There is no difference in behavior between a player performing an action and a GM performing the same action.

### Account Management

ID	Use Case	Description
UC01	CreateAccount	Players can create an account by providing an email and password.
UC02	Login	Players can log into registered accounts by providing an email and password.
UC03	JoinSession	After login, users will be able to see all available sessions, which they can click on to join.
UC04	CreateSession	(GM only) GMs can create new sessions by providing a name.
UC05	DeleteSession	(GM only) GMs can delete existing sessions by clicking on a session's delete button.
UC06	CreateCharacter	Players can create characters to use in various sessions. Characters have various attributes like their name, equipment, and stats.
UC07	EditCharacter	Players can edit their characters' names, equipment, and stats.
UC08	DeleteCharacter	Players can delete their characters by clicking on that character's delete button.

**Table 3:** Detailed use case descriptions for the account management actions.

### Session Actions

ID	Use Case	Description
UC09	PickTargets	Certain actions, like attacking and chatting, require the player to select the target(s) for that action.
UC10	Attack	The active character takes an attack action against designated targets with a specified weapon.

UC11	RepeatAttack	A convenience option. Will launch a new attack with the same data/settings as the previous one.
UC12	Move/Misc	The player moves their character to a new position on the field, and their initiative will then be updated. Miscellaneous actions performed outside of the scope of the app will also be performed.
UC13	Defend	The character being attacked makes a defense roll against an attack with their current armor and/or shield.
UC14	Hold	The player decides to take no action on their turn. Holds after the first will decrease a player's number of actions by 1.
UC15	WaitUntil	The player declares a condition that may be met. Players in waiting have their turns skipped. If the condition is met, the player can immediately take action, even when it is not their turn.
UC16	Intervene	During certain actions, most notably attacking, players who aren't targets of the attack may jump in to defend their teammates.
UC17	EditSelected	Allows a selected character to be edited. Examples include changing their names, affiliations, or number of actions.
UC18	Chat	Players can communicate with each other through text. Messages can be sent both publicly and privately.
UC19	Confirm	The active character confirms their action.
UC20	StartSession	(GM only) The GM starts the session, preventing others from joining. Prompts all players to roll for their initial initiative values.
UC21	EndSession	(GM only) The GM ends the current session. All player actions are disabled and an optional ending message may be displayed to everyone.

UC22	SignalPlayer	(GM only) The GM selects a player to perform their action on the current segment. The selected player's controls will be activated.
------	--------------	---

**Table 4:** Detailed use case descriptions for the session actions.

## Requirement Traceability Matrix

The following tables contain our requirement traceability matrix. The matrix was divided into two tables to preserve formatting and for readability.

Requirement Traceability Matrix FR01 - FR20																						
	UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20	UC21	UC22
FR01	■																					
FR02		■																				
FR03				■																		
FR04					■																	
FR05			■																			
FR06																		■				
FR07						■																
FR08							■															
FR09								■										■				
FR10									■													
FR11										■												
FR12																		■				
FR13											■	■	■	■	■	■	■	■	■	■	■	■
FR14												■										
FR15																						■
FR16												■										■
FR17												■										
FR18												■										
FR19													■									■
FR20														■								

**Fig. 3:** Requirement Traceability Matrix for functional requirements 1 - 20.

Requirement Traceability Matrix FR21 - FR39																						
	UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20	UC21	UC22
FR21													■									
FR22																■						
FR23																■						
FR24																■						
FR25												■										
FR26														■								
FR27															■							
FR28										■	■	■	■	■	■	■		■				
FR29																				■	■	■
FR30																						■
FR31										■	■	■	■	■	■	■					■	
FR32						■	■															
FR33						■	■	■														
FR34							■			■			■			■						
FR35							■															
FR36										■	■		■			■						
FR37										■	■	■	■	■	■	■						
FR38											■											
FR39																		■				

**Fig. 4:** Requirement Traceability Matrix for functional requirements 21 - 39.

# Updated Design

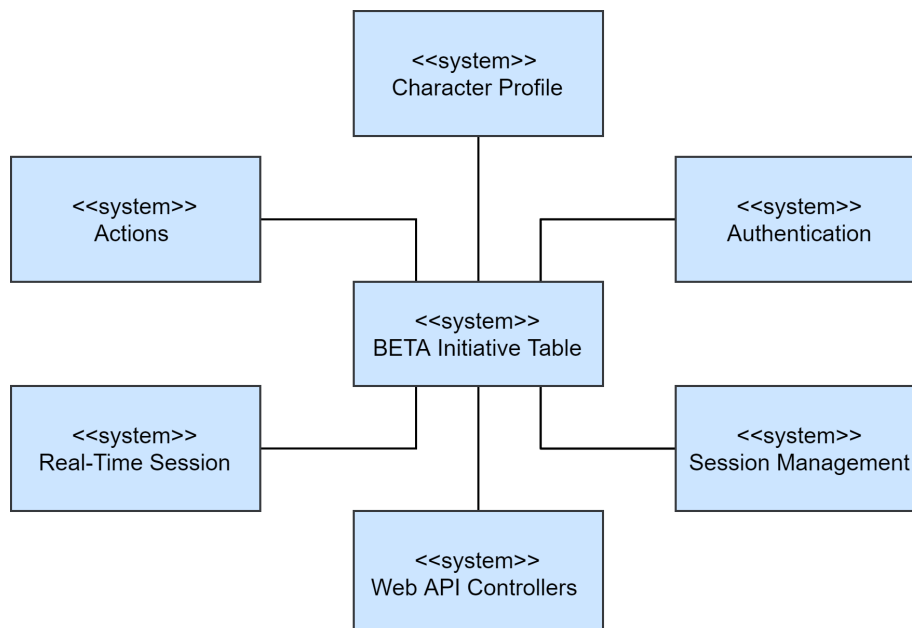
## Summary of Changes

Since our original project design, we have not had any major changes in the high-level structure of our systems. Instead, our program units have been further refined and adjusted to more accurately represent the features and systems that have been implemented or will be as development proceeds. Most of the program unit changes that occurred were in the actions system. Various program units were removed since they were no longer relevant or were simplified.

## Updated 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.



**Fig. 5:** 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.

<b>Authentication</b>	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)	<b>Inputs:</b> A player object based on our database players table. <b>Outputs:</b> A player object based on our database players table or an error. <b>Description:</b> This sends a request to our players web API controller to create a new record for the player in the table. <b>Program Units Called:</b> None <b>Exceptions:</b> An exception is thrown if the player object is empty or incorrect.
loginPlayer(player)	<b>Inputs:</b> A player object based on our database players table. <b>Outputs:</b> A player object based on our database players table or an error. <b>Description:</b> This sends a request to our players web API controller to check if the user exists and if the credentials match. <b>Program Units Called:</b> None <b>Exceptions:</b> An exception is thrown if the player object is empty, incorrect or if the credentials are incorrect.

sendSecurityEmail(passcode)	<p><b>Inputs:</b> A 6 digit random number.</p> <p><b>Outputs:</b> A success message or Error.</p> <p><b>Description:</b> 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.</p> <p><b>Program Units Called:</b> Calls registerPlayer upon successful entering of the number.</p> <p><b>Exceptions:</b> An exception is thrown if the email is invalid and cannot be sent.</p>
-----------------------------	--

**Table 5:** The table above lists the program units under the Authentication system, allowing users to register and log into accounts.

<b>Real-Time Session</b>	<p>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.</p> <p>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.</p> <p>All of the connected clients listen for the event signals and handle them once received.</p>
sendMessageToAll(message)	<p><b>Inputs:</b> A message object</p> <p><b>Outputs:</b> An event signal with the message attached to all the connected clients.</p> <p><b>Description:</b> Sends a message to the chat log that all the connected clients to based on the passed object.</p> <p><b>Program Units Called:</b> None</p> <p><b>Exceptions:</b> An exception is thrown if the message object is empty or incorrect.</p>

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



<p>updateInitiativeTable(tableData, sessionName)</p>	<p><b>Inputs:</b> An array of initiative table data [number, playerName] and string sessionName.  <b>Outputs:</b> An event signal with the tableData attached to all the connected clients.  <b>Description:</b> 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.  <b>Program Units Called:</b> None  <b>Exceptions:</b> An exception is thrown if the sessionName cannot be used to find a session in the sessions table.</p>
<p>SignalPlayer(playerID)</p>	<p><b>Inputs:</b> int playerID.  <b>Outputs:</b> An event signal.  <b>Description:</b> 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.  <b>Program Units Called:</b> None  <b>Exceptions:</b> An exception is thrown if the playerID cannot be used to find a player in the session.</p>
<p>CloseAnyOpenModals(playerID)</p>	<p><b>Inputs:</b> int playerID.  <b>Outputs:</b> An event signal.  <b>Description:</b> This is a generic helper method that will close any open modals for a given playerID. Typically used during an intervention.  <b>Program Units Called:</b> None  <b>Exceptions:</b> An exception is thrown if the playerID cannot be used to find a player in the session.</p>
<p>OpenATKDFNDModal(playerID)</p>	<p><b>Inputs:</b> int playerID.  <b>Outputs:</b> An event signal.  <b>Description:</b> This is a generic helper method that opens the attack/defend modal for a given playerID. Typically used during an intervention.  <b>Program Units Called:</b> None  <b>Exceptions:</b> An exception is thrown if the playerID cannot be used to find a player in the session.</p>

**Table 6:** 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.

<p><b>Actions</b></p>	<p>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.</p>
<p>Confirm(Action)</p>	<p><b>Inputs:</b> A selected action  <b>Outputs:</b> A confirmation message the action was selected.  <b>Description:</b> This finalizes the user's selected action.  <b>Program Units called:</b> The passed action  <b>Exception:</b> An exception is thrown if no action is selected.</p>
<p>Attack(Target)</p>	<p><b>Inputs:</b> Target (Player)  <b>Outputs:</b> A message once the action loop ends.  <b>Description:</b> This starts the attack process and signals the defender to defend.  <b>Program Units called:</b> Defend, roll, Initiative.  <b>Exception:</b> An exception is thrown if an invalid target is selected.</p>
<p>Defend()</p>	<p><b>Inputs:</b> None  <b>Outputs:</b> A message once the action loop ends.  <b>Description:</b> This player is being attacked and must prepare their defense, and make rolls.  <b>Program Units called:</b> Roll, Initiative.  <b>Exception:</b> An exception is thrown if they do not have an action.</p>

Roll(autoroll, diceFormula)	<p><b>Inputs:</b> A boolean if the user would like to autoroll and a string containing the dice formula for the current roll.</p> <p><b>Outputs:</b> An integer containing the final rolled value.</p> <p><b>Description:</b> This is a dice roller that outputs a number based on the input</p> <p><b>Program Units called:</b> none</p> <p><b>Exception:</b> An exception is thrown if the dice formula is in an invalid format.</p>
Initiative()	<p><b>Inputs:</b> None</p> <p><b>Outputs:</b> A message once the action loop ends.</p> <p><b>Description:</b> This shifts all the user turn orders based on the actions they have taken this round.</p> <p><b>Program Units called:</b> None</p> <p><b>Exception:</b> None</p>
Hold()	<p><b>Inputs:</b> None</p> <p><b>Outputs:</b> A message once the action loop ends.</p> <p><b>Description:</b> The user holds their action</p> <p><b>Program Units called:</b> Initiative</p> <p><b>Exception:</b> An exception is thrown if they do not have an action, unless in the second round.</p>
Wait()	<p><b>Inputs:</b> None</p> <p><b>Outputs:</b> A message once the action loop ends.</p> <p><b>Description:</b> The user waits until a certain event to occur before acting.</p> <p><b>Program Units called:</b> Initiative</p> <p><b>Exception:</b> An exception is thrown if they do not have action or do not specify an event.</p>

Location(roll)	<p><b>Inputs:</b> An integer</p> <p><b>Outputs:</b> An integer corresponding to a body map.</p> <p><b>Description:</b> It tells the group where an attack happened on the body chart depending on what was rolled.</p> <p><b>Program Units called:</b> Roll</p> <p><b>Exception:</b> An exception is thrown if no roll was given</p>
Repeat Attack()	<p><b>Inputs:</b> Nothing</p> <p><b>Outputs:</b> A message once the action loop ends.</p> <p><b>Description:</b> It repeats the users previous attack action</p> <p><b>Program Units called:</b> Attack</p> <p><b>Exception:</b> An exception is thrown if there is no previous attack.</p>
Intervention()	<p><b>Inputs:</b> None</p> <p><b>Outputs:</b> A message once the action loop ends.</p> <p><b>Description:</b> The user interrupts a defender by retargeting the attack to themselves.</p> <p><b>Program Units called:</b> Defend</p> <p><b>Exception:</b> An exception is thrown if they do not have an action</p>
Move(distance)	<p><b>Inputs:</b> An integer representing how far they want to move</p> <p><b>Outputs:</b> A message once the action loop ends.</p> <p><b>Description:</b> The user moves distance units.</p> <p><b>Program Units called:</b> Initiative</p> <p><b>Exception:</b> An exception is thrown if they do not have an action or invalid number</p>

**Table 7:** The table above lists the program units under the Action Systems. This lists many of the available actions the users can take each round.

<p><b>Web API Controllers</b></p>	<p>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.</p> <p>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.</p> <p>The player's Web API controller differentiates since it involves authenticating and registering users. The loadout Web API controller differentiates as well since it saves an entire character including all of their weapons, armors, and shields.</p>
<p>chrc_profilesController, sessionsController, session_usersController, chat_logController, initiativeTableController</p>	<p>Each Web API controller contains the following methods :</p> <ul style="list-style-type: none"> <li>● <b>GET(ID)</b>: Returns a single record with the provided ID.</li> <li>● <b>GET()</b> : Returns all records.</li> <li>● <b>POST(record)</b>: Adds a new record to the database table.</li> <li>● <b>PUT(record)</b>: Updates an existing record based on the passed record.</li> <li>● <b>DELETE(ID)</b>: Deletes an existing record with the provided ID.</li> </ul> <p>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.</p>
<p>playersController,</p>	<ul style="list-style-type: none"> <li>● <b>login(playersRecord)</b>: 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, return the playersRecord. An exception is returned if the password does not match</li> <li>● <b>register(playersRecord)</b>: Checks the passed record does not already exist. If it does not, the password is hashed and the playersRecord is added to the table.</li> </ul>

	An exception is returned if the record already exists.
loadoutController	<ul style="list-style-type: none"> <li>● <b>GET(charID):</b> Returns a character loadout record with the provided ID.</li> <li>● <b>POST(loadout):</b> Saves a character loadout to the database table.</li> <li>● <b>PUT(loadout):</b> Updates an existing character loadout based on the passed record.</li> <li>● <b>DELETE(charID):</b> Deletes an existing character loadout with the provided ID.</li> </ul>

**Table 8:** 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.

<b>Character Profile</b>	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.
--------------------------	---

<p>createCharacter()</p>	<p>When the user clicks the “Create New” button on the character menu, a prompt will appear asking for several values for the new character.</p> <p><b>Inputs:</b> Numerical values or text input from the user, to be used to initialize the character attributes.</p> <p><b>Outputs:</b> A completion message and an updated view of the characters list, with the new character visible.</p> <p><b>Description:</b> 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.</p> <p><b>Program Units called:</b> chrc_profilesController</p> <p><b>Exception:</b> None</p>
<p>editCharacter(id)</p>	<p>Within the character menu, users can click on the “Edit” button underneath a character’s name to begin the editing process.</p> <p>This process functions similarly to createCharacter(), except a PUT request is made instead of a POST.</p> <p><b>Inputs:</b> None</p> <p><b>Outputs:</b> A completion message and an updated view of the characters list.</p> <p><b>Description:</b> A PUT request is made to the server with the selected character’s ID, and the new values.</p> <p><b>Program Units called:</b> chrc_profilesController</p> <p><b>Exception:</b> None</p>

deleteCharacter(id)	<p>Within the character menu, users can click on the “Delete” button associated with each character to delete them.</p> <p><b>Inputs:</b> None  <b>Outputs:</b> A completion message and an updated view of the characters list.  <b>Description:</b> A DELETE request is made to the server with the selected character’s ID.  <b>Program Units called:</b>  chrc_profilesController  <b>Exception:</b> None</p>
---------------------	---

**Table 9:** The table above lists the functions of the Character Profile system.

<b>Session Management</b>	<p>Upon logging in, users are presented with all the available sessions that have been created by all players.</p> <p>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.</p>
joinSession(id)	<p>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.</p> <p>Note that sessions that have already started will not be able to be joined.</p> <p>This program unit can be thought of as the client-side version of the “JoinSession” unit from the Real-Time Sessions system.</p> <p><b>Inputs:</b> None  <b>Outputs:</b> Connects to the session and brings the user to the initiative table interface to begin playing.</p>



	<p>If the session is unable to be joined, an error message will appear, and the user will remain on the same page.</p> <p><b>Description:</b> Starts the process for joining a session.</p> <p><b>Program Units called:</b> JoinSession(playerName, sessionName)</p> <p><b>Exception:</b> None</p>
createSession()	<p>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.</p> <p>Upon clicking the “Create New” button, a prompt will appear, asking for a name for the new session.</p> <p><b>Inputs:</b> Text input from the GM.</p> <p><b>Outputs:</b> A completion message and an updated view of the sessions list.</p> <p><b>Description:</b> A POST request is sent to the server with the data from the user.</p> <p><b>Program Units called:</b> sessionsController</p> <p><b>Exception:</b> None</p>
deleteSession(id)	<p>Only GMs can delete sessions.</p> <p>If a session to be deleted is in progress, a prompt will be shown confirming if the GM wants to force-end the session.</p> <p><b>Inputs:</b> Confirmation from the GM</p> <p><b>Outputs:</b> 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.</p> <p><b>Description:</b> A DELETE request is sent to the server with the data from the user.</p> <p><b>Program Units called:</b> sessionsController</p> <p><b>Exception:</b> None</p>

**Table 10:** 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. All of our strong database entities have 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's primary key, name, e-mail, account password, and a flag to determine if the player is a game master. The character profile database table includes the character profile primary key, the identification of ownership, the character profile group name if they are a non-playable character, and the many stats. The sessions table consists of the session primary key, the session's name, and if the session is joinable. The initiative table includes the initiative primary key, the session foreign key, a player foreign key to determine which player has joined the session, and character profile foreign key for retrieving character information for the session and the initiative value used to determine the order of play. This weak entity is the primary data used in our real-time session system to determine the order of play and who is in a current session. Next, the chat log table includes the message primary key, the session foreign key for which session the message belongs to, the foreign key for where the message is being sent to, the foreign key for where the message came from, the message itself, and if it is a direct message or a global message. Any primary keys and foreign keys have been bolded for readability and formatting.

creationDate	lastEdit	creator	lastEditor	isDeleted
--------------	----------	---------	------------	-----------

**Table 11:** General metadata appended to all the database tables.

<b><u>playerID</u></b>	name	email	password	isGM
------------------------	------	-------	----------	------

**Table 12:** Player's database table.

<b><u>characterProfileID</u></b>	<b><u>characterProfile_playerID</u></b>	characterProfile GroupName	isNPC	PER	MD
SPK	AGL	STR	CON	HTP	LHTP
AIM	T.AIM	MOVE	FLY	PAIN	BID
T.PEN	P.PEN	Cur_HTP	MEM	WIS	MS
MR	MD	CHR	PB	MAN	RR

PR	SANITY	SEX	initBonus	numActions	
----	--------	-----	-----------	------------	--

**Table 13:** Character Profile database table. Contains the many statistics of character profiles.

<u>sessionID</u>	name	isJoinable
------------------	------	------------

**Table 14:** Sessions database table.

<u>messageID</u>	messageID_sessionID	messageID_TO_playerID	messageID_FRO_MplayerID	message	isDM
------------------	---------------------	-----------------------	-------------------------	---------	------

**Table 15:** Chat log database table

<u>initTableID</u>	initTableID_sessionID	initTableID_playerID	initTableID_character_ProfileID	initvalue
--------------------	-----------------------	----------------------	---------------------------------	-----------

**Table 16:** Initiative database table

<u>WeaponID</u>	WeaponID_character_ProfileID	weaponName	weapon Class	isRange
weaponStats	weaponShots	weaponDAM	weapon Skill	weaponChatMsg
weaponClass Bonus				

**Table 17:** Weapons database table

<u>ArmorID</u>	ArmorID_character_ProfileID	armorName	armorClass	armorLoc
armorASB	armorRFLT	TargetFactor (range only)		

**Table 18:** Armors database table

<u>ShieldID</u>	ShieldID__character ProfileID	shieldName	shieldClass	shieldLoc
shieldASB	shieldRFLT	shieldIntPercent		

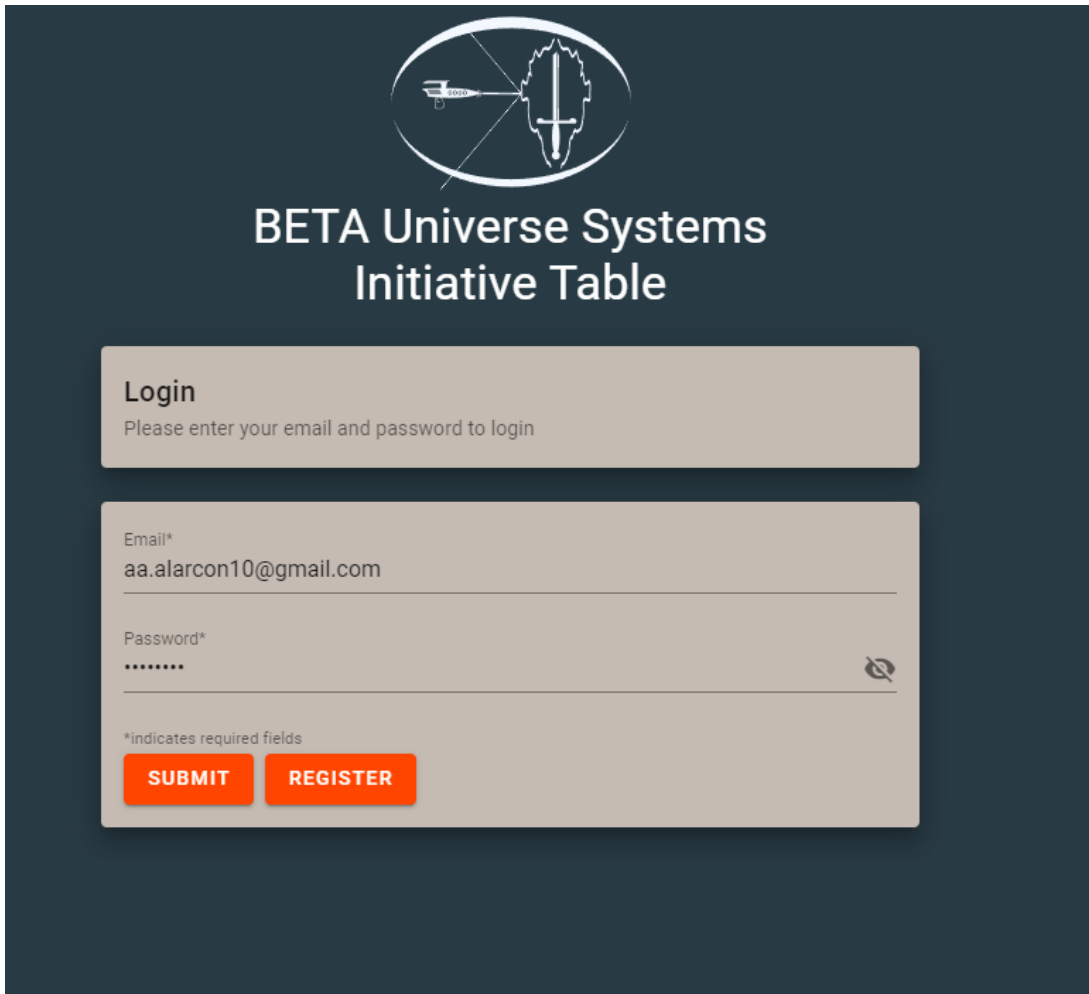
**Table 19:** Shields database table

## Updated 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.

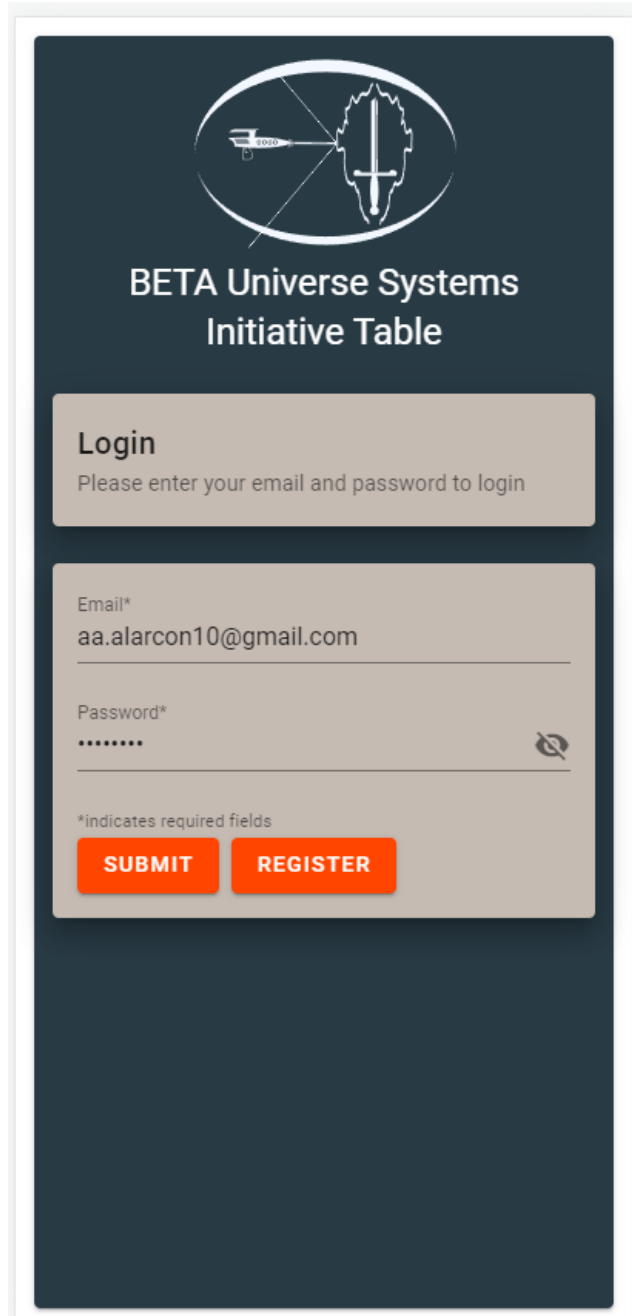
## Updated User Interface Design

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

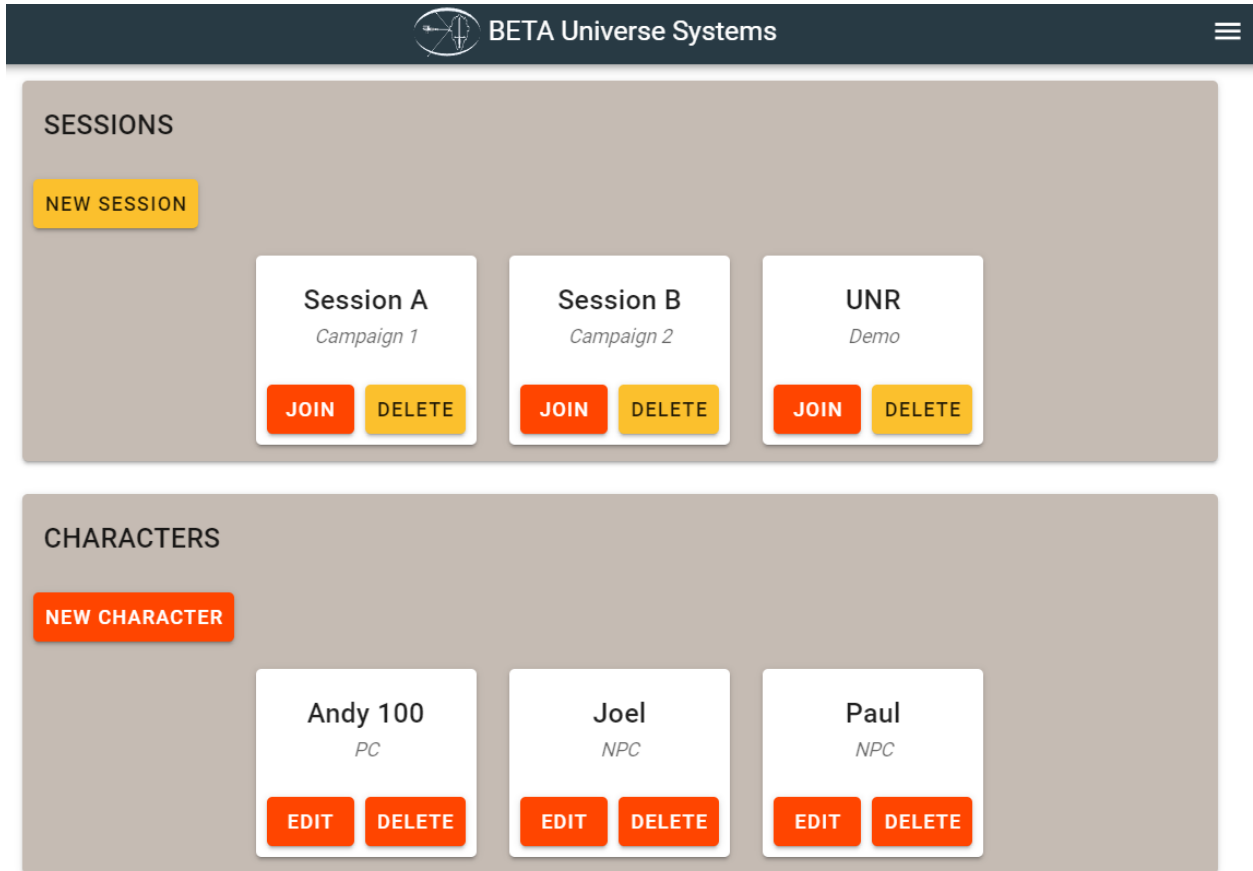


The image shows a desktop login interface for the BETA Universe Systems Initiative Table. At the top, there is a logo featuring a key and a sword within a circular frame. Below the logo, the title "BETA Universe Systems Initiative Table" is displayed in white text on a dark blue background. The main content area is a light gray box containing a "Login" section. This section includes a heading "Login", a sub-heading "Please enter your email and password to login", and two input fields: "Email\*" with the value "aa.alarcon10@gmail.com" and "Password\*" with masked characters "\*\*\*\*\*". A small eye icon is visible next to the password field. Below the input fields, there is a note "\*indicates required fields" and two orange buttons labeled "SUBMIT" and "REGISTER".

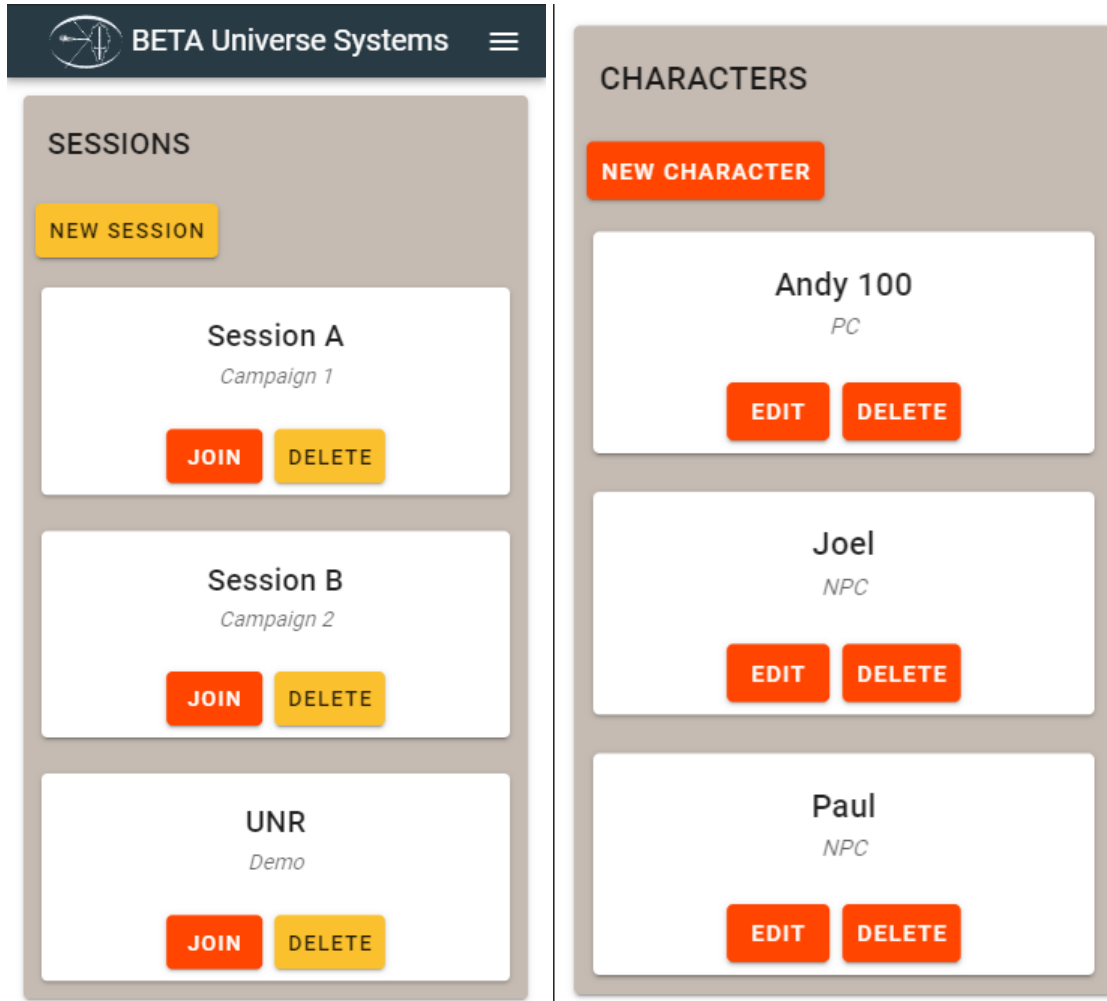
**Fig. 6:** Login UI (Desktop). The initial login screen for the website. Users with accounts can use their username and password to access their account and enter the main portion of the application. New users can access the registration through the register button.



**Fig. 7:** Login UI (Mobile). Mobile version of the login page. There is not much difference from the desktop version other than scaling the page size to the appropriate mobile aspect ratio.

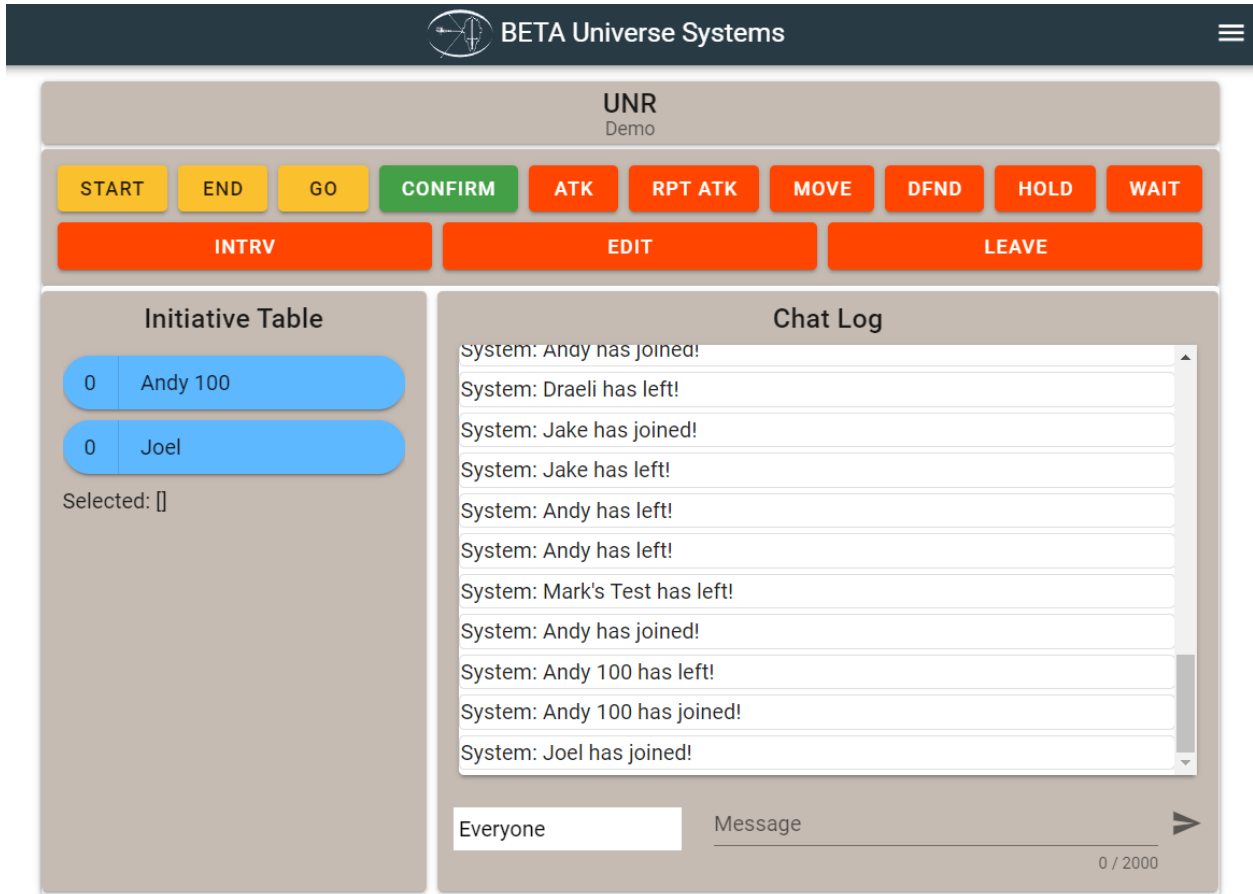


**Fig. 8:** Home page UI (Desktop). After logging in, the player is presented with a section of the available sessions. Each card in the section represents a session with a button to join. There is also a section for the characters the player has created. Each card represents a character, and there are buttons to edit and delete the individual characters. When logged in as a GM, there are two additional functions: creating a new session and deleting a session.

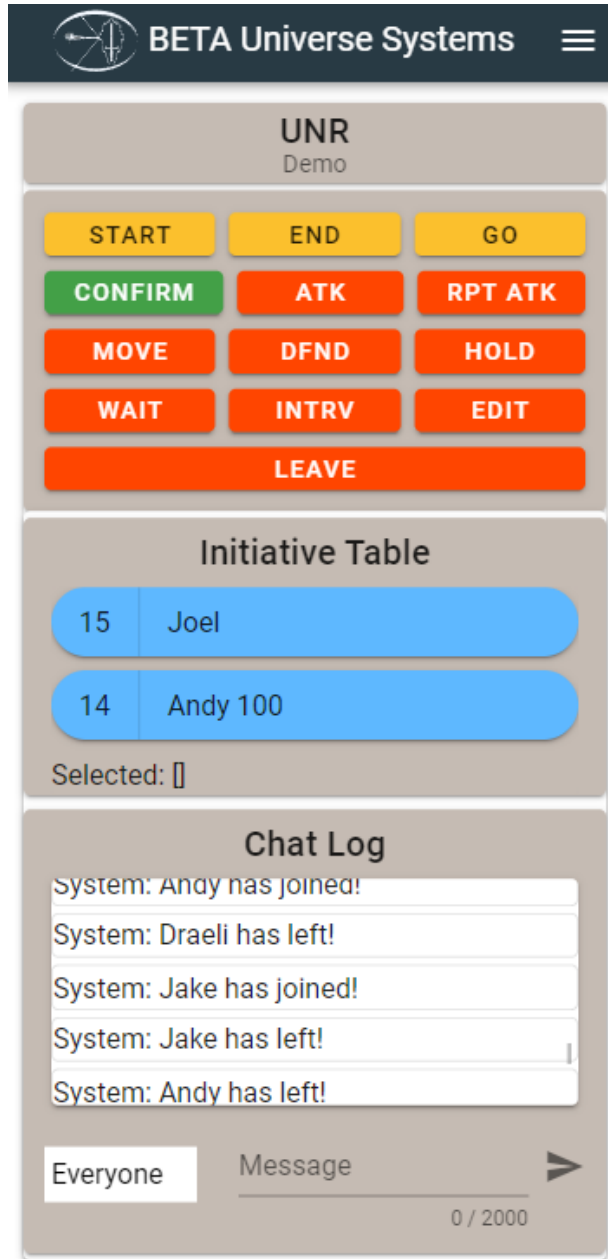


**Fig. 9:** Home page UI (Mobile). For the mobile version of the homepage UI, the same functionality is present; however, the two-session and character sections have been resized to fit a mobile size screen. The cards now take up less vertical space to make up for the change in screen size.





**Fig. 10:** 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 side bar 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 displays the chat log. The chat log displays actions that have been taken, 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.



**Fig. 11:** 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.

## Character

Please fill out the following information and click save

---

### Details

Character name\* 0 / 60

Group name\* 0 / 60

Would you like to have rolling automatically done for you?  
Yes ▼

Sex Team NPC

Male Us PC

---

ActionCount\* Initiative Bonus\*

0 0

0 / 3 0 / 3

---

### Stats

PER*	MD*	SPK*	AGL*
0	0	0	0
0 / 3	0 / 3	0 / 3	0 / 3

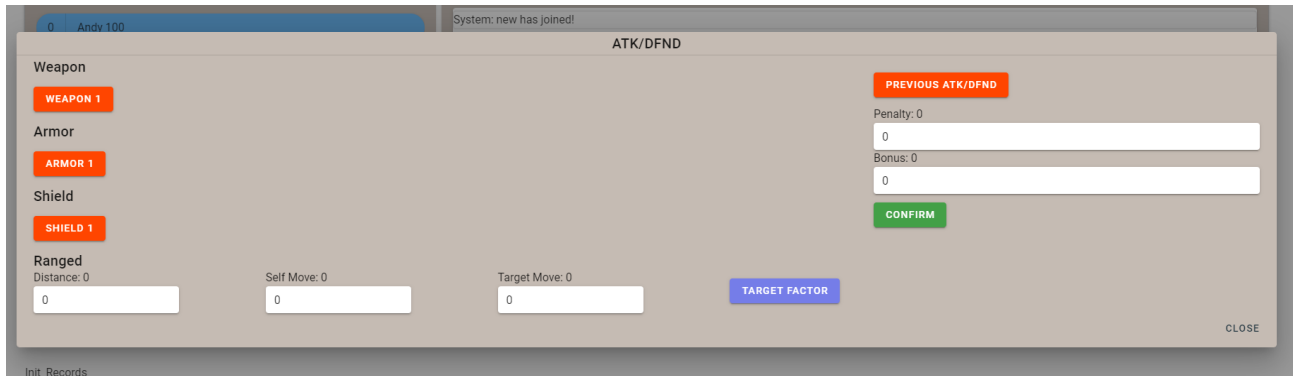
  

STR*	CON*	HTP*	LHTP*
0	0	0	0
0 / 3	0 / 3	0 / 3	0 / 3

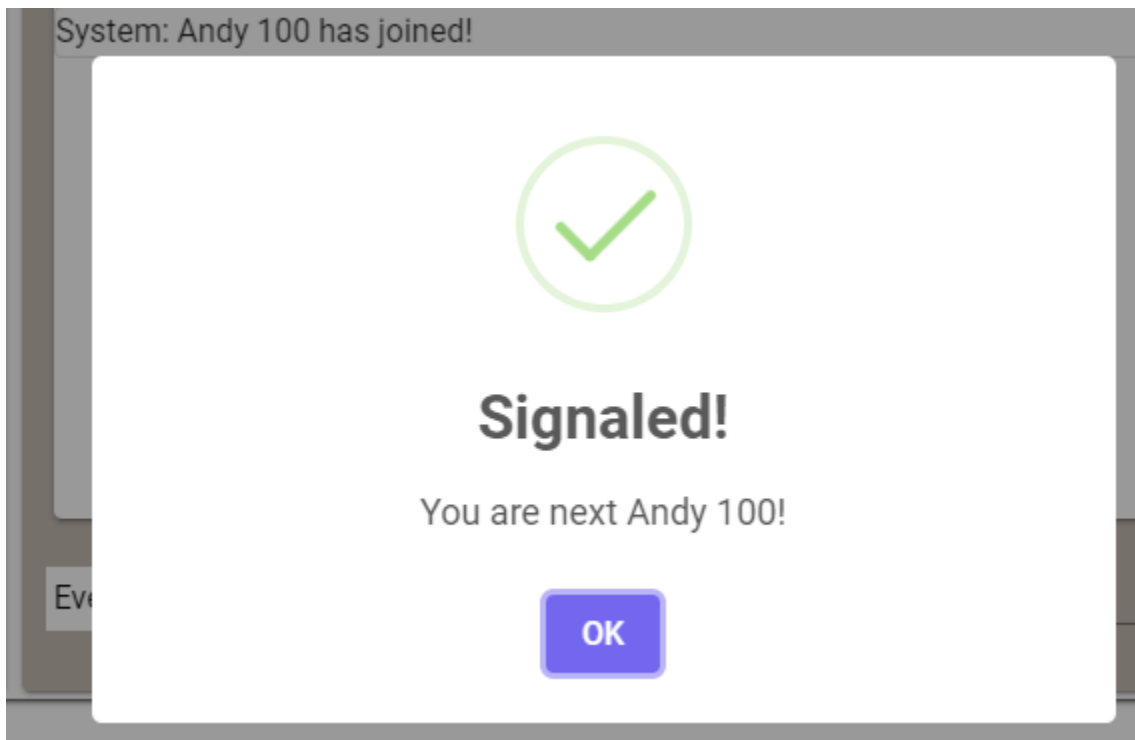
**Fig. 12:** Create a new character (Desktop). This is the screen where you create a new character. This is divided into five sections. We have the details section, the stats, the weapons, the armor, and the shield.

**Fig. 13:** Create a new character (Mobile). This is the screen where you create a new character. It is the same as the desktop, but has been scaled for mobile devices. Everything is readable and we have increased the row count so there is not as much at once.

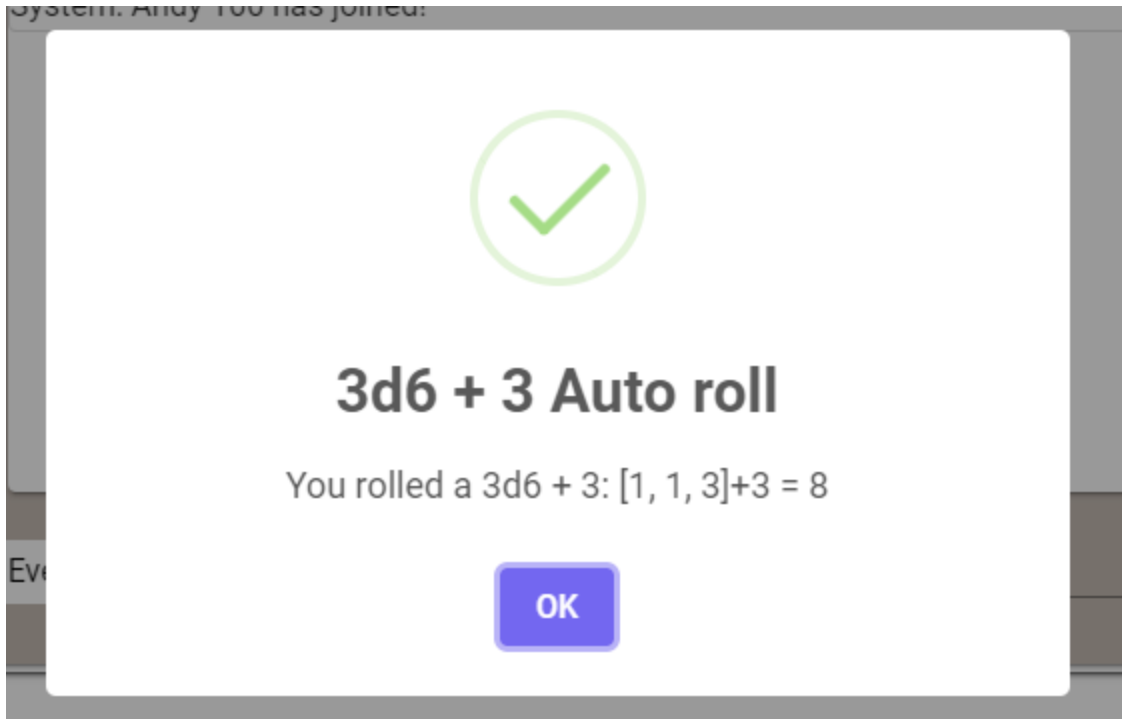
**Fig. 14:** Edit a Character: (Desktop). This is where you can edit a character while in a session. You choose one character from the list of characters you control.



**Fig. 15:** Attack Modal: (Desktop). This is the screen where you start the attack action. You choose your equipment and fill in all of the variables needed for the attack, target, range, movement, etc.



**Fig. 16:** Next Character Signal (Desktop). This shows that the next person, you, have been pinged to take their turn. The GM controls this with the “Go” button



**Fig. 17:** initiative Roll (Desktop). This is the initiative roll from an auto roller. The active character chose auto roll when making their character. It is 3D6 plus their initiative stat. This is different for every character.

# Glossary

Word	Definition/ Abbreviation
<b>Campaign</b>	An overarching story, a novel while the sessions are the chapter.
<b>Check</b>	A task the PC needs to perform in order to perform an action, such as jumping a chasm.
<b>END</b>	Endurance.
<b>GM</b>	Game master.
<b>Go</b>	Signals it is the next player's turn
<b>Hold</b>	Do nothing, in reference to an option on a character's turn.
<b>Initiative</b>	A stat with a dice roll that determines the order that the characters move in, higher number first.
<b>Intervention</b>	A character can attempt to interrupt another action.
<b>ITA</b>	Initiative Table Application.
<b>Location</b>	It is referring to where on the body you are interacting with.
<b>NPC</b>	A non-player character.
<b>PC</b>	Player character.
<b>Range</b>	The number of spaces from point a to point b.
<b>Round</b>	A full rotation of the initiative order.
<b>RPG</b>	Role playing game.
<b>Saving Roll</b>	A check in order to prevent someone from occurring, such as being poisoned.
<b>Segment</b>	The current initiative value.
<b>Segment Count</b>	The number of segments that have passed.
<b>Self Move</b>	How many units the active player moved that action.
<b>Session</b>	Each time the group meets to continue the game.
<b>Shield %</b>	The percent chance the attack hits your shield
<b>Stats</b>	The integer number corresponding to a specific skill, strength stat.

Word	Definition/ Abbreviation
<b>Target Move</b>	How many units the target moved in their last action
<b>Wait</b>	Player waits until an event occurs
<b>XP</b>	Experience points.

**Table 20 & 21:** The tables above contain 25 glossary terms surrounding the project's problem domain.



## Engineering Standards and/or Technologies

The first standard technology used by the team was the HyperText Markup Language or HTML, which is the standard markup language for documents designed to be displayed in a web browser. We will use this for the structure and outline of our entire front-end since our project is a web-based application. The second standard we will be using is the RFC 6455 WebSocket protocol. The WebSocket Protocol enables two-way communication between a client in a controlled environment to a remote host that has opted-in to communications purposely. We will be using this to facilitate real-time communication within our sessions.

Additional technologies used in our project include Visual Studio 2019 an integrated development environment (IDE) for .NET and C++ developers on Windows. This technology helped the team write code in a single coherent environment and perform debugging throughout the development of the project. Another technology that the team used is GitHub. GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. This technology allows the team to share the work in and distribute the workload as well as keep the work up to date. Next, the team used Vuetify which is a material design front-end framework that allowed the ease of components built from Hyper-Text Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript with minimal difficulty.

ASP. NET is another technology that the team used which is a web framework, created by Microsoft, for building modern web apps and services with .NET. The Team used ASP. NET to allow connection between the front-end and the database that way information could be stored properly and efficiently. In conjunction with ASP. NET the team used a MS SQL database which allowed for the team to store our data in a relational database. Additionally, SignalR was a technology used to make sure that the information used was updated in real-time on the application's interface. SignalR is a free and open-source software library for ASP.NET that allows server code to send asynchronous notifications to client-side web applications.

## Project Impact and Context Considerations

In terms of public needs, the application falls into the entertainment category. While not as urgent as health and safety, entertainment is crucial for mental health and productivity. The Initiative Table Application will have a significant impact on the social and cultural aspects of the tabletop RPG player community.

Socially, the application will make the client's game more accessible and easier to play. This could attract more players to the community and increase the social aspect of the community as a whole. The social impacts of this type of technology has the potential to become obvious and widespread. One of the limiting factors for this type of gameplay is the need to be in the same physical location with other players for long periods of time. Much of this is due to the lengthy calculations our project aims to eliminate, thus making gameplay much faster and more enjoyable, increasing the positive social impact of the entire gaming community.

Culturally, this application addresses a sub-culture found in the community of RPGs. Developing new technology, gameplay and content to this industry will strengthen and expand the subculture influence. It is unlikely that the application will have a significant global impact in the near future. However, there is a strong tabletop RPG in Europe and it is possible that the application may reach players from different countries in the future. Economically, the client intends to monetize this type of application in a novel way that may have some impact on the economy surrounding the tabletop gaming industry.

## References

*Jean Yang, Vijay Janapa Reddi, Yuhao Zhu, and Peter Bailis. 2016. Research for Practice: Web Security and Mobile Web Computing: Expert-curated Guides to the Best of CS Research. Queue 14, 4 (July-August 2016), 80–95. DOI:<https://doi.org/10.1145/2984629.3005356>*

The article above informs us how to better our security and authentication for our application. We are having users enter their email addresses and passwords; we want to make sure that information is protected so unauthorized users can not get access to their accounts.

*John S. Tonello. 2017. The full stack project. Linux J. 2017, 281, Article 1 (September 2017).*

This article is about the full stack project. Full stack development is used to help develop front end to back end development. Full stack development is also useful for mobile applications. This is how we are coding our project, one part will be a mobile application and another part will be a mobile application. Full stack development is also used for web based applications.

*John Molt. 2017. Beta Universe Systems Book of Player Character Combat.*

This is the book that we are digitizing. It is our reference material that our advisor has written for his tabletop role playing game. This book details every aspect of combat and everything that he wants us to automate. If we ever have a question we reference this book.

*Richard W. C. Lui. 2005. Security models for authorization, and delegation and accountability. Ph.D. Dissertation. University of Hong Kong (People's Republic of China). Order Number: AAI0809504.*

The article is about the different security models used for authentication and authorization. It also talks about the different roles users may have and each role has a different amount of access than other users. This is applicable in our case since we will have a player role and a GM role.

*Seikyung Jung. 2018. Web development with node.js. J. Comput. Sci. Coll. 33, 6 (June 2018), 154–156.*

We will be doing web development with Vue and node.js. This book is about web development with node.js and we will be using it as a reference for what to do. It is a tutorial on how to do web development with node.js. We are going to walk ourselves through it to get more familiar with it.

*Thomas Gustafsson and Jörgen Hansson. 2004. Dynamic on-demand updating of data in real-time database systems. In Proceedings of the 2004 ACM symposium on Applied*

*computing (SAC '04). Association for Computing Machinery, New York, NY, USA, 846–853.*  
DOI:<https://doi.org/10.1145/967900.968074>

This article is about updating data in real time. Our project deals with a lot of real time updates between each player and GM. This article goes over a strategy on how to solve this, on demand depth first traversal. We want to make sure everyone is updated at the same time without much lag.

<https://kastark.co.uk/rpgs/encounter-tracker/>

This website is similar to our project. It is an initiative tracker where you manually enter your name, initiative number, and HP. This differs from our project in that ours automates all of combat and not just helps with the initiative order.

<https://www.dndbeyond.com/>

This is a D&D reference website where you can store characters, books, classes, pretty much everything you need for D&D. We are aiming to make something like this, but for our role playing system. Our project is step one of that plan that our mentor has.

<https://roll20.net/>

This is a popular Dungeons and Dragons companion that shows a map, plays music, and has a chat log. After our part of the project is done our advisor wants to add a virtual map and music. It is used in a group setting similar to our web app. It is used as a companion, but is not a supplement for the game.

<https://dnd.wizards.com/charactersheets>

This is where you get D&D character sheets, which is similar to what our creator made. In our app we allow the user to digitize their character sheet and store it in our app. It is encouraged to use the app to its best ability.

## Work Contribution

	Andy Alarcon	Jacob Gayban	Mark Graham	Jacob Tucker	Griffin Wagenknecht
Project Assignment 2 Paper (Writing sections and formatting)	2.0	1.5	2.0	1.5	3.5
Total	2.0	1.5	2.0	1.5	3.5

**Table 22:** The table above shows the amount of time spent by each team member on each activity.