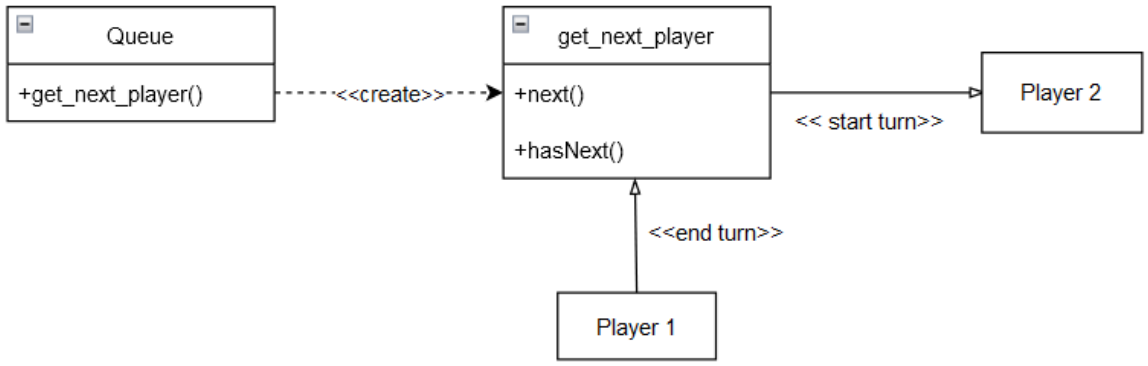
Detailed Design Document

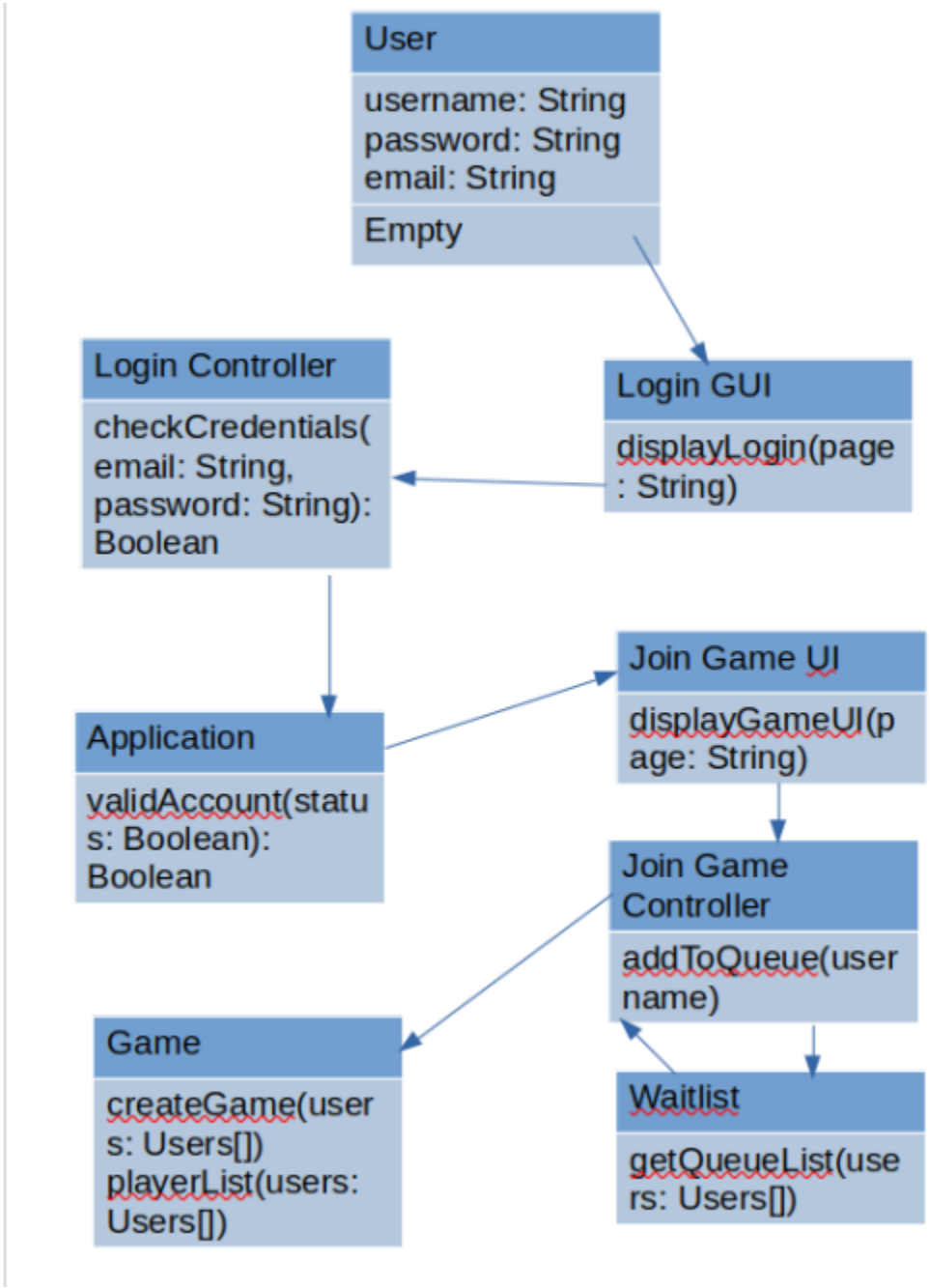
Iterator Class Diagram:



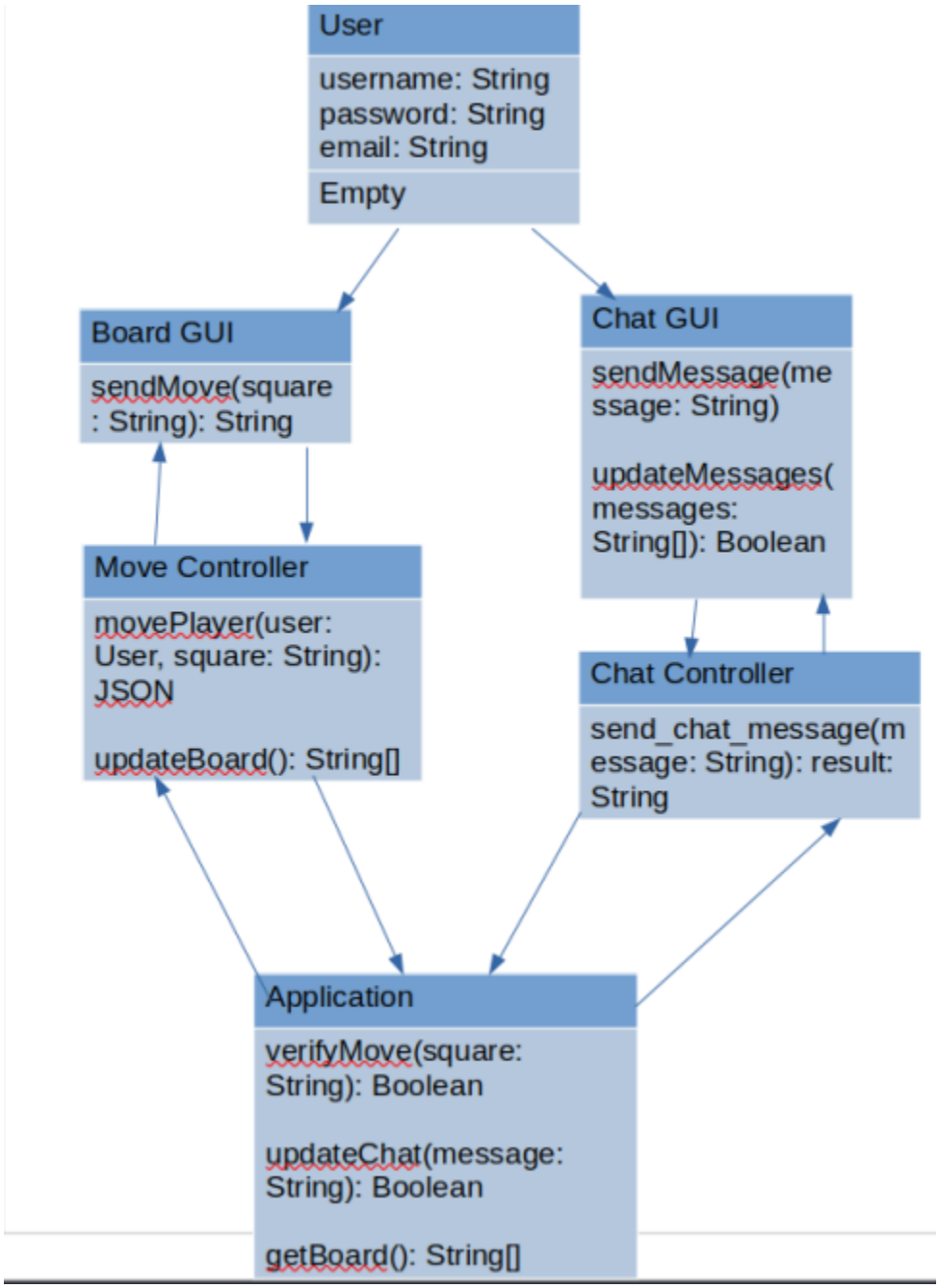
We chose an Iterator design diagram because we want to show how the users interact with recurring processes in a system to queue up players in the order they have their turns. What's occurring is when the player ends their turn, the queue sends the command to get the next player and then it sends a start turn command to player 2. In the iterator portion of the Queue, once a player is removed from it at the end of their turn, they are added back to the Queue to the back. This allows a round-robin way of easily keeping track of which player is to go next, while not having to store the players in an array elsewhere. Rather we simply call next to change the current moving player and handle this automatically.

Class Diagrams of the Application

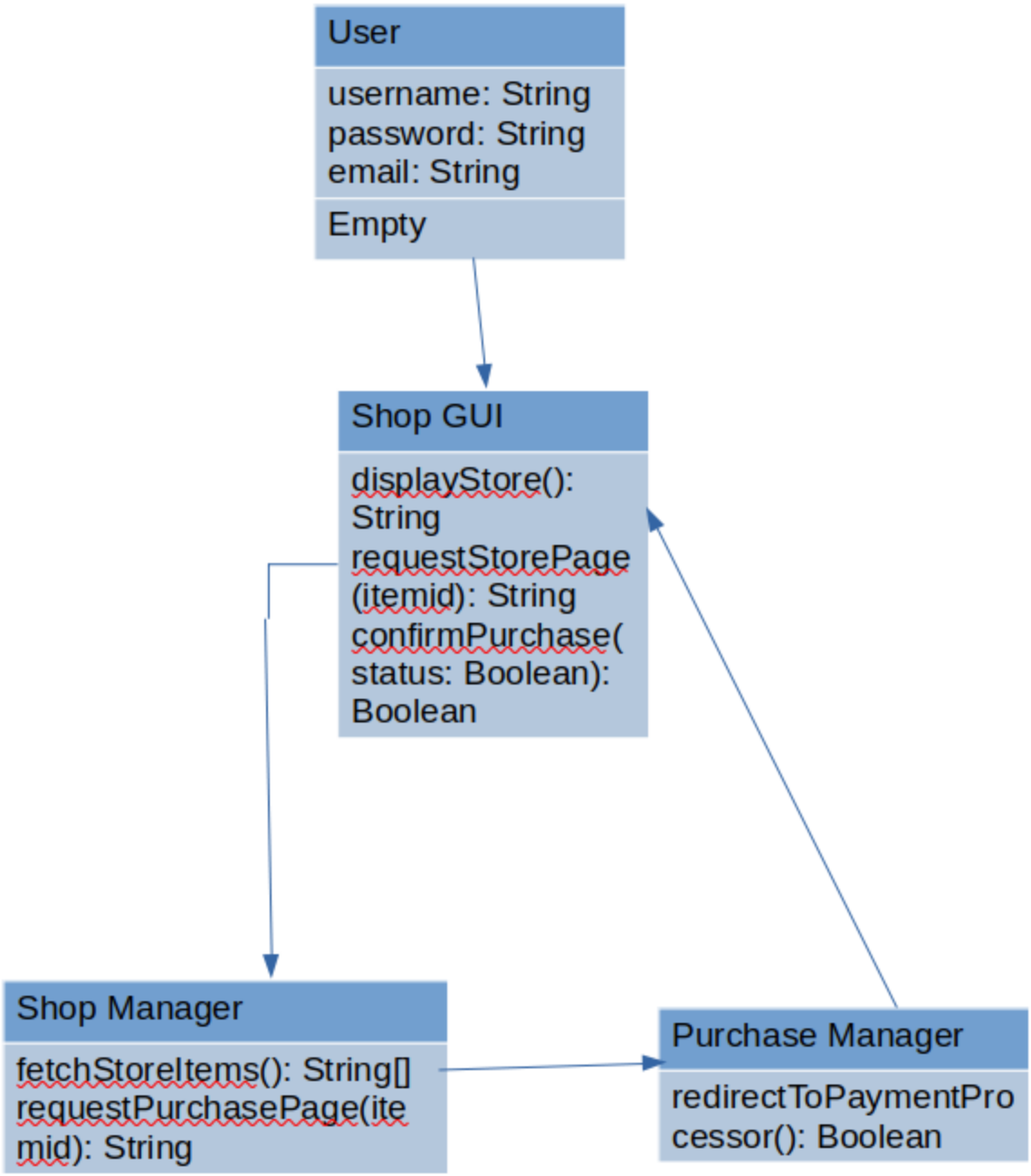
As the class diagram was being drafted, it was quickly realized that it would prove too difficult to read given the complex nature of a game such as ours, even if split into two diagrams. As a result, concurrent sequence diagrams were selected and descriptions provided in order to make it easy to understand the way our various systems come together. These are below:



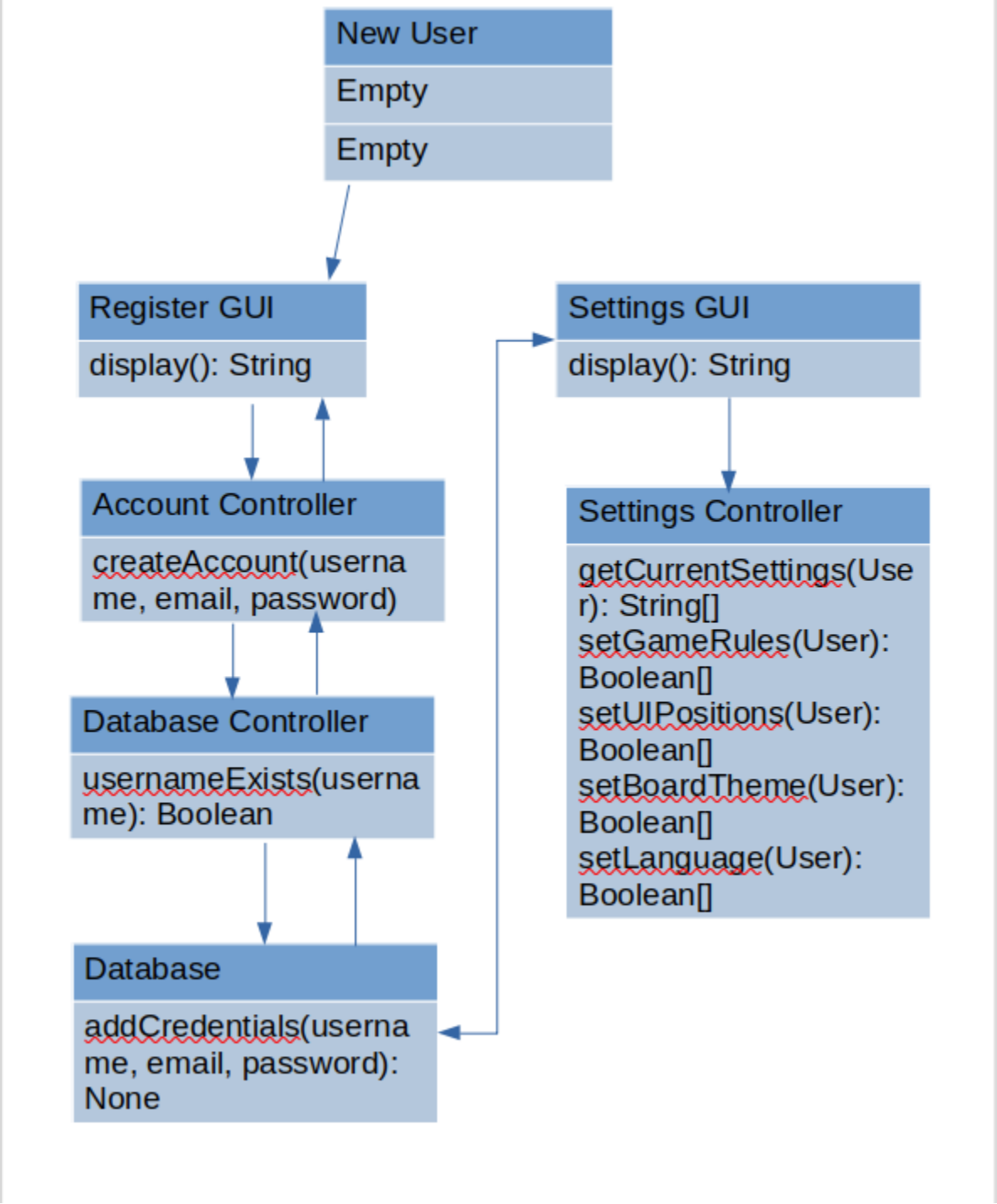
The first part of our DCD pertains to logging in and joining a game. Since only verified users can join a game, this coupling fits well. Firstly, users will be directed to the login GUI. There they will be shown the input form for their credentials. The login controller then checks these against the current users. If successful, this is sent to the application to tell the system that the user is valid. The user is then automatically redirected to the game itself, where they are sent to the Join Game controller and added to the Queue. This is where our **Iterator Design Pattern** comes in, as the Queue uses this in the waitlist to take the oldest player out and into the next available game. Eventually this will result in a full game of players taken to Game, where it'll promptly register them as players and begin the game itself.



The next part of our DCD is what happens after a player joins a game. For the sake of simplicity, the User from the past part is carried over. There the user gets two interactable objects: The board GUI (the game itself) and the Chat GUI (to talk to other players. For the Board, the user will click a square and call sendMove. The ID of the individual square's HTML is sent as a string to the Move Controller, where it's fed into the Application after ensuring it's not malformed (part of the movePlayer method). This is then forwarded to the main portion of the Application, where the game checks if the move sent was valid. If the move is valid, it's sent back to the Move Controller where it promptly calls updateBoard and all players see the move.
For the Chat, players can input into a text field to send a message. This goes to send_chat_message on the Chat Controller, where it gets filtered for XSS, HTML tags, and profanity. The message is then sent to the Application's updateChat segment, wherein the Application federates it to all the players in the game.



Another thing that Squared users can do is access a shop GUI. This is where in-app purchases can be made. Firstly, the Shop GUI calls the Shop Manager to fetchStoreItems so only current items can be viewed. From an individual item, the GUI calls requestStorePage, where the Shop manager will redirect the request to the Purchase Manager. This will redirect the user to a secure payment system where they can enter their payment information to buy the item. After this step, the user will be offered one final time if they want to make the transaction. If so, the user will have the item successfully added to their account.



This final component of our DCD pertains to account management and covers the creation of new SQUARED accounts and the management of settings within that account. The many controllers used for the initial creation of a SQUARED account communicate back and forth amongst each other in determining whether specific credentials that must be unique for each SQUARED account, such as an account's username or associated email account, are valid and have not yet been used, with respect to the input given by a user wishing to create a new

SQUARED account. The validity of these parameters is also communicated to the new account interface's graphical components, which duly communicate to end users that the information they wish to use for their accounts is either valid or invalid depending on whether that information has already been used for other SQUARED accounts or not. Once a new SQUARED account has been made, the SQUARED UI then manages another interface pertaining to account management after an account's creation, such as setting user preferences for how the game should be displayed and editing account information if necessary, where another settings controller communicates with the account management database to edit and save an account's settings, while reflecting any changes made to an account's personal settings on the SQUARED UI for future reference and editing.