# Evaluation

## Analysis of objectives

1. **Players will be able to register with a username and password, and log in/out. The users account information is then stored in the database**

The home page of the website has log in and sign up buttons for users not logged in. The buttons lead to web forms in which the users can enter their account information. If the data is not valid (e.g. not valid email format) or incorrect (wrong username password combination) the forms provide feedback and the user can enter it again. Once a user is logged in a log out button is given at the bottom of every page. The users account information is stored in the **CustomUser** table in the SQLite3 database. This objective has been fully met.

1. **A player’s total money is stored in a database.**

A player’s money is stored in their **CustomUser** record in the database.

* 1. **On entering a table, the buy in of the table is deducted from the user’s total money and is stored in a temporary table, in which the user can bet with**

On joining the table, the addPlayer function reduces the players money by the table buy in. The players current money in table is stored in the python Player object in the poker algorithm and in the temporary **Players** table.

* 1. **The money in the table is updated during play**

The money in table is stored in the poker algorithm Player object and is synced with **Players** every turn.

* 1. **Once a player leaves a table their money from the table is added to the player’s total money and the database is updated**

On disconnection from the poker socket in consumers the players total money is incremented by the players money in table.

* 1. **Players can reset their money if it reduces below 1000**

If a player’s money is below 1000, a button appears on the home page ‘Reset money’. Which redirects to reset-money/. This calls the resetMoney view, which checks the user’s money is below 1000 and if so sets the users money to 1000. It then redirects back to the home page.

1. **User passwords are stored with the SHA-256 hash**

The users passwords are stored with the SHA-256 hash in the database, using the PBKDF hashing algorithm

1. **Players will be able to create table, and the and join them through the home page where all current tables are displayed**

On the home page there is a button for logged in users to create a table which redirects to a web form to create a table. When the user submits a form which is valid, they are redirected to the table. From the home page users can join tables they have enough money to enter. If a user tries to join a table they have insufficient money to enter they are redirected back to the home page.

1. **If a table has not been used in 15 minutes, it is deleted**

On starting the server, a daemon thread is created that infinitely loops, checking every 10 seconds how long since a table has been last used. If the table hasn’t been used for over 15 minutes then it is deleted. The program updates a tables last use on when the last player left, but doesn’t delete tables with players still in.

1. **The pack in each table is shuffled between rounds**

The poker algorithm initializes a new deck array every round, and uses the random library to shuffle the array.

1. **The dealer button moves to the dealers left each round**

The dealer button is represented by an integer that dictates the position of the dealer in the playersInGame array and players array in Game. It is incremented by 1 each round and taken mod of the number of players.

1. **Tables have a maximum number of players (between 2 and 8) which can be specified by the table’s creator**

The restrictions are made in the **Table** model which the model form that creates the table inherits. The web form has an input to enter the maximum number of players and invalidates the form, prompting the user to enter it again if it is outside that range.

1. **Tables have a buy-in to enter (between 100 and 100,000,000), specified by the table’s creator which is used to determine how much a player must put in the table**

The table buy in is also specified in the **Table** model through the same procedures as above. The table buy is the amount of money that the player plays with in the table.

* 1. **Used to calculate the size of the blinds, the small blind 1/100th the size of the buy in.**

The buy in is used to calculate the size of the small blind, 1/100th of the buy in in the poker algorithm. The big blind is conversely twice the small blind so 2/100th of the buy in.

TODO – this objective may change

1. **Once in a table, players can play live poker according to the rules above**
2. **Players will only be able to see their pocket cards and the table’s community cards**

Each player is a member of the tables group socket, which sends pot and player actions and chat messages. Players also have a private socket that sends the players money and cards, and dealer. This way no player can see anyone else’s cards.

1. **If a player leaves a table, they will automatically fold**

When a player leaves the table, they disconnect from the web socket in which their **Players** record is deleted. On their turn, if no record relating to a player is found, it means the player has left and the player is folded.

1. **A leader board shows players ranked by their total money**

A SQL query is used to get all users usernames and money ordered by their money in descending order. This is then passed to the dynamic template which iterates through each of them adding them to a table to create a static web page. Each record of the table is a CSS button that redirects to the user’s profile.

1. **Players can interact with each other using an in-table chat**

The message is sent over poker web socket in each table. Once the user presses enter, the JavaScript takes the input from the chat box and sends it over the web socket. The algorithm appends <username>: to the front of the message and sends it to the players in the table, who receive it and pass it to the JavaScript which writes it to the top of the chat text box.

* 1. **The chat can be filtered so that swear words would be censored**

Once a player joins a table, the web socket reads a censored word file and adds each word to an array. Once a message is received from the user, it checks whether any words in the message are in the censored words array. If so it replaces the word with an appropriate number of asterisks.

1. **A how to play page is provided for new players**

A static web page is served at the URL how-to-play/ which contains the poker rules from the analysis. This is available to all users and is a static HTML resource.

1. **Multiple games on the website can be played at the same time**

Each game is running on its own poker thread and has its own **Room** and **Table** record, and each player in the table has a **Players** object, which has a foreign key to the corresponding **Room** they are in. This system can sustain as many simultaneous poker games as needed up to the servers computing power.

## End user feedback

When I performed my whole system test the overall feedback was positive and I feel like I’ve made a good base product that meet all the objectives I had for the project. The end users commented that it was always clear in a game what was currently happening, and the general UI of the website was clean and intuitive. However, some improvements I could implement would be that the game could use images of cards in the game and animations rather than the cards being text based and that the users could be shown to be sat round the table to easily show the betting order. I totally agree with this and these were a few of my extension objectives that I decided not to implement as it would not add much to the complexity of the program.

It was also suggested that the censored words in the text chat could be improved to match all cases and 1337 speak (e.g. replacing e with 3), and also match even with spacing and punctuation between letters. This could also apply to table names. We also found a minor bug where if players go all-in, and after which only 1 player has money left in the table, the river is shown in the poker log but not in the current community cards. This was easily fixed with a moving of a function and was a very minor bug and so after much testing I believe the underlying algorithms and programs are solid.

Users also suggested that there could be a way of adding friends on the app, which you could join and play with if they were online. They also suggested more stats to be shown on the player profile as it looks quite bare.

More stats in a player profile

Friends re buyin button

## Improvements

Use NO-SQL database for Players and Room tables, as they are temporary tables that are written to often, so for scalability if the website had more visitors, a NO-SQL database such as MongoDB would be ideal for these tables. The web application could also use HTTPS to secure the communication, which would be essential if it were available on the open internet. While my sign up and log in pages already have CSRF tokens, if I was to make it publicly available, I would need a re-captcha authentication to prevent bots, and using email confirmation to prove accounts are real.

The website could be hosted on the internet through a service such as AWS and purchasing a domain name. This would require some maintenance and the database would have to be migrated from SQLlite3 to a more stable system such as PostgreSQL.

Currently, the site is manually tested, which means I manually test that every part of the site works the way it should myself. As the application becomes larger, it becomes increasingly harder to check new updates don’t affect other existing parts of the site. Django by default includes the means to write automated tests, these are simply requests or inputs on parts of the website where the correct value is already known. (e.g. giving a hand with a pair to poker algorithm and testing it recognises the pair). This is known as ‘unit testing’.

As stated by my end users I could improve the GUI in the poker game, by using card and poker chip images and animations. The players could be shown in a circle around the table to easily show betting order and remove a layer of abstraction to the game. A rebuy button could be shown for when players run out of money so that they can easily keep playing instead of leaving the table and re-joining. A bet slider could be used to raise instead of a text box to better the user experience. Players could also decide on joining a table whether they want to sit down or spectate the game. A feature could be added that allows players to join multiple tables at once and switch between them.

Users could be able to add friends, in which they can view who is active and join them in tables if possible, from the ‘friends’ sidebar. Players could add friends through poker games, visiting their profile, or searching their username in the sidebar. Conversely players can invite their friends to tables. Another feature could be that players could create private tables that are only available to their friends or require a password to enter.

As stated by my supervisor, I could add user stats linked with each user to the stored on the database such as hands played, hands won, best hand. If a user visits a profile, they can view the players stats. I could also add a settings page so when a user visits their own profile, they can edit settings on their account, such as changing email and password or turning censored chat on or off.

I could have used a JavaScript framework on the frontend such as Vue.js to make the web app more reactive such as newly created tables showing on the home screen without refreshing. It could also reduce the amount of custom JS I have written.

The backend of the web application already uses Django Rest Framework to serialize table and player data for the frontend. I could use this to create an API that could power other services such as a mobile app.

A poker AI could be developed that users could optionally add to tables to fill player spots. When playing with an AI bot, the user would use temporary money that would not affect their total money they have gained so users cannot exploit the bots.

Instead of using fake money, the website could allow users to bet with real money against other players. However, this would require the website to be regularly and extensively security tested as the website would be storing bank information, which is a much higher security risk and also makes the website a target to hackers. It would also have to comply with government laws and have a system of verifying ID documents to prevent under 18s from using this feature. So realistically this would be extremely hard or impossible to maintain without a larger team and beyond the scope of this project.

Stats other players money