# Design

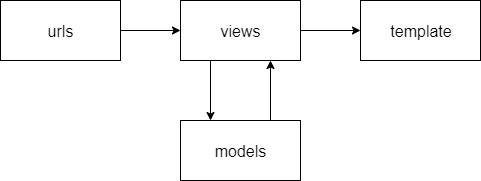
## Overview

I am designing a web-application in which users can log in and create poker tables, where they can play live poker against other players. In each table, there is also a chat, in which the players can interact. The users can then also view their total money and stats and compare them against other players in the leader board.

## The Django Framework

### Fundamentals of Django

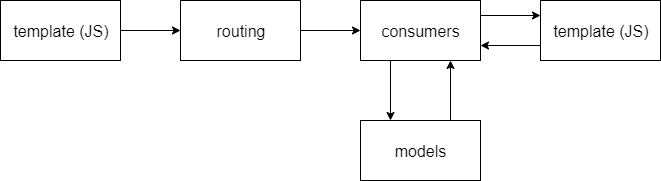
* models.py – Defines your data model and contains the fields and behaviours of the data you are storing. Each model maps to a database table and each attribute a field.
* urls.py – Uses regular expressions to capture URL patterns to retrieve a view.
* views.py – Called when a specific URL is accessed. Takes the web request and returns a web response. When rendering a web page, it can pass a dictionary of variables to the template
* template – a HTML file with special syntax describing how dynamic content can be inserted into the HTML



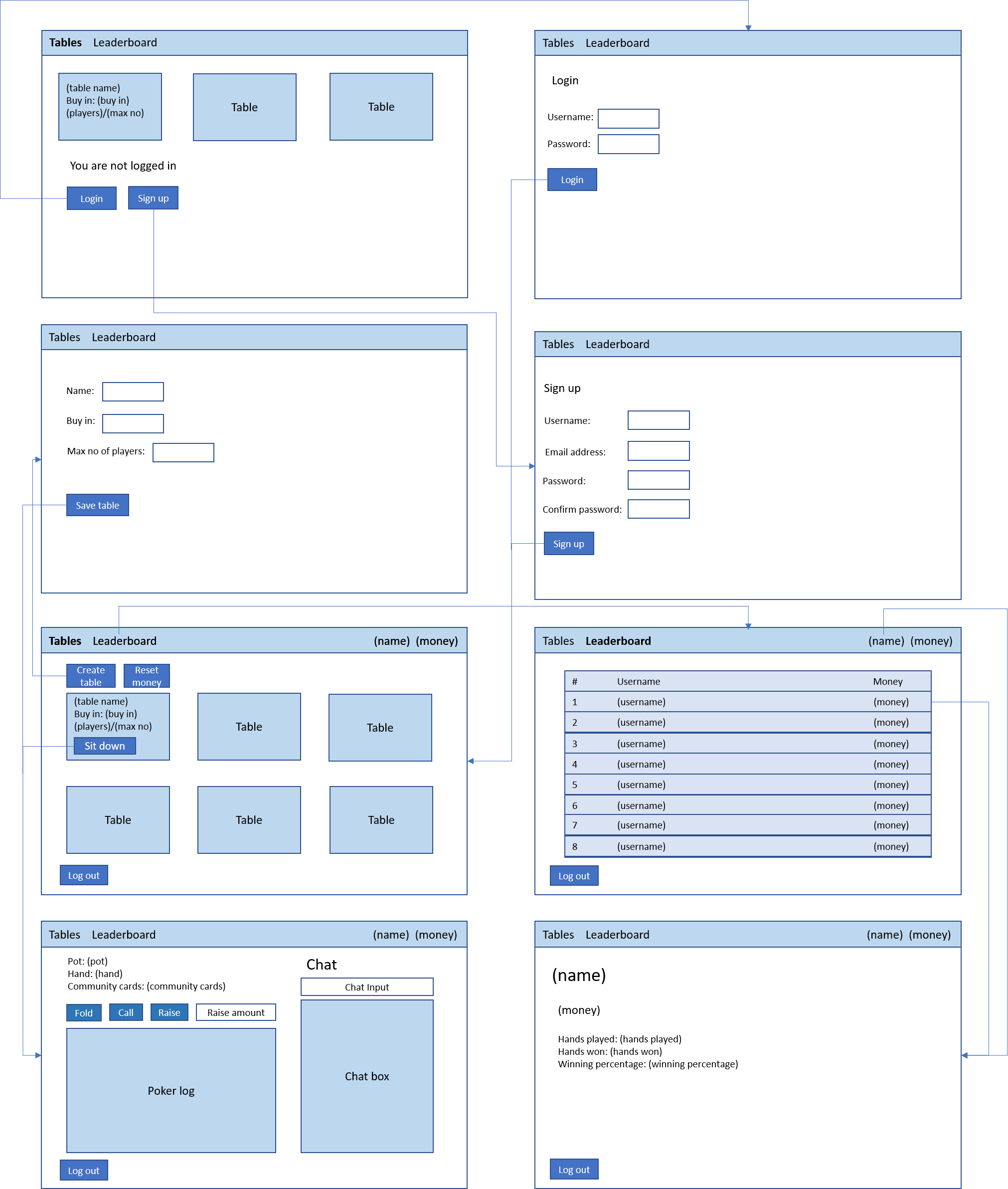
### Django Channels

A synchronous socket library for Django. Its base features work very similarly to Django:

* The templates’ JavaScript creates a web socket at a specific web socket address
* routing.py – uses regex to capture the web socket address retrieves a consumer
* consumer.py – can create web socket groups for to send data to multiple users e.g. the community cards, chat. Has connect disconnect and receive functions that are called when such events happen
* The templates’ JavaScript also contains a send function along with onmessage and onclose functions, that run when such an event is reached.



## User Interface Design



### Key

* Light blue box – object
* Dark blue box – button
* White box – input
* () – variables
* The leaderboard table are CSS buttons that link to each users profile

This is a wireframe of the web-app. The table page acts as a home page where users can view create and join tables. A public leaderboard displays every user by their money, and users can view other players profiles by clicking on them through the leaderboard. You can also view your profile by clicking on your username on the nav bar. The fold call and raise can be buttons are hidden if it is not your turn or you cannot perform that action at this stage.

## The Poker Algorithm

### Game view

When the user sits down at a table the program verifies that the user has sufficient money to play at the table, and that the table is not full. If so, it

login required to access

Function game

if users money >= tables buy in and players in table < max players in table

start daemon thread on poker main function

return render of game.html

return a redirect to the index view

### Adding player to table

When the poker algorithm runs, it determines whether it is the only player in the table, in which case initialises a new poker instance. Otherwise it just adds the player to the table and exits.

Function main

get Room object for Table

add player to Room

if Room does not exist

create Room object

add player to Room

startGame()

### Finding hand strengths

Function called before the game starts to determine the strength of each players’ hand

hand <- pocket cards and community cards

hand <- hand sorted by highest cards first

Function pairThree

strength <- 0

numPairs <- number of pairs in hand

three <- bool if hand contains three of a kind

four <- bool if hand contains three of a kind

if four

strength <- 7

if numPairs = 1

if three

strength <- 6

else

strength <- 1

elif numPairs = 2

strength <- 2

if three and strength < 3

strength <- 3

orderHand <- [cards included in final strength]

Function straightFlush

if 5 of same suit in hand and strength < 5

strength <- 5

orderHand <- [cards of the same suit]

append aces to hand as 1s

if 5 consecutive cards

if strength < 4

strength <- 4

orderHand <- [consecutive cards]

if consecutive cards are of same suit

strength <- 8

orderHand <- [consecutive cards]

if strength <- 8 and head of orderHand is ace

strength <- 9

### Dealing with split pots

If two players have the exact same strength they share the pot, and are added to playerWin in a single list.

playerWin <- []

win <- [players sorted by hand strength]

Function clash, splitWork, winStack

binary sort win

If win items are equal

items[0] <- [items]

remove all other items

for item in win

playerWin += [[item], hand strength in words]

### Determining the winner

After the game has finished, the winner function runs to determine how much each player has won.

Function winner

a <- 0

winners <- []

playerWin <- list of players sorted by hand strength

while pot != 0

for player in playerWin[a]

if player has not folded

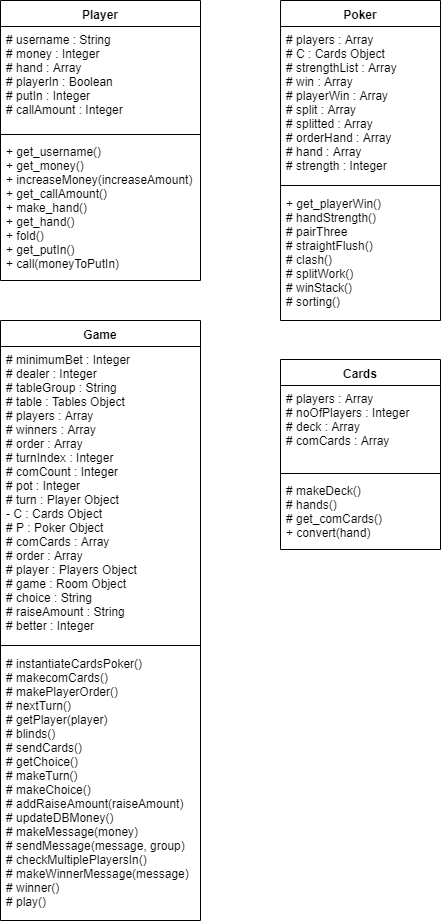
maxPrize <- amount of money the player put in

winners += [player, maxPrize]

add money from pot equally to each player in winners up to their maxPrize

pot -= amount of money given out

a+=1



DB

