# 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 Poker Algorithm

### Determining the winner

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

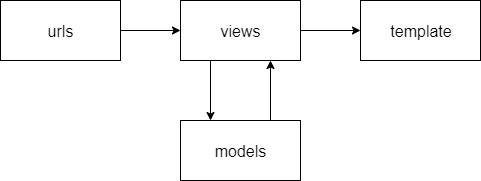
a+=1

### Dealing with split pots

## The Django Framework

The fundamentals of Django:

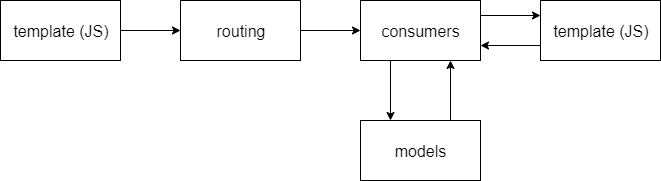
* models.py – Defines your data model and contains the fields and behaviours of the data you are storing. Each model maps to a database table and each attribute a field.
* urls.py – Uses regular expressions to capture URL patterns to retrieve a view.
* views.py – Called when a specific URL is accessed.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



Django Channels

A 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
* templates JavaScript also contains a send function along with onmessage and onclose functions, that run when such an event is reached.



Split pseudocode

Function clash, splitWork

splitted <- []

win <- [players sorted by hand strength]

BinarySort win

if items are equal

firstItem <- [items]

otherItems <- ''

splitted += [items]

Function winStack

for player in win

if player in splitted

add splitted players at player position

delete all splitted players from list

delete splitted[players]

Strength pseudocode

hand <- pocket cards and community cards

hand <- reverse sort hand

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

Game view

login required to access function

Function game

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

start daemon thread on poker main function

return game.html render

return redirect index

main function

Function main

get Room object for Table

add player to Room

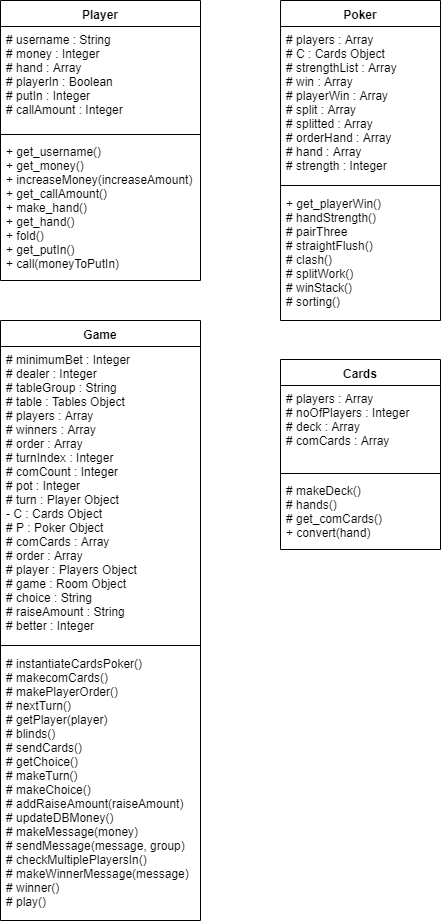
if Room does not exist

create Room object

add player to Room

startGame()

OOP



DB

