Pricing simulation

Introduction

Pricing simulation is Django based Web application used for reel life truck auction. When we have a group of users, the certain amount of trucks(reel) are sourced for bidding. Some pre-requisite values are mentioned for the bidders/users of the application to bid proper values and earn maximum targets. The bidding is for head haul and backhaul with pre-defined number of trucks available. Certain values like loaded cost and empty cost broker's cost are also considered while calculating the net profit earned while each turn of auction. This auction carried out for several number of turns so that the net-contribution is calculated and from them the winner is decided. The winner from the bidding is decided based on the one who bids minimum, gets maximum trucks awarded and earns maximum profit out of it.

Setup and Dependency

The application is based on Django, Celery and RabbitMq.

Django:

Django-Django is a free and open-source web framework, written in Python, which follows the model-view-template architectural pattern. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Celery:

Celery is a task queue that is built on an asynchronous message passing system. It can be used as a bucket where programming tasks can be dumped. The program that passed the task can continue to execute and function responsively, and then later on, it can poll celery to see if the computation is complete and retrieve the data. While celery is written in Python, its protocol can be implemented in any language. By implementing a job queue into your program's environment, you can easily offload tasks and continue to handle interactions from your users. This is a simple way to increase the responsiveness of your applications and not get locked up while performing long-running computations.

RabbitMQ:

RabbitMq is a messaging broker - an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

Robust messaging for applications. Easy to use. Runs on all major operating systems. Supports a huge number of developer platforms. Open source and commercially support.

Basic Setup:

The database schema is maintained in models.py which depicts the exact replica of the database tables. We can customize our models by defining them in models.py.

Celery have its own models which could be merged when we define them in Settings file.

Settings.py file is the file which defines the structure, the models and the exact architecture of the application. It's the place where we get to know what are the dependencies of the application.

We define the entire code setup and other dependencies in setting.py file. E.g. email, modules like crispy-tags, celery, database setup, html templates setup, Frontend dependency location setup, OS compatibility setup etc.

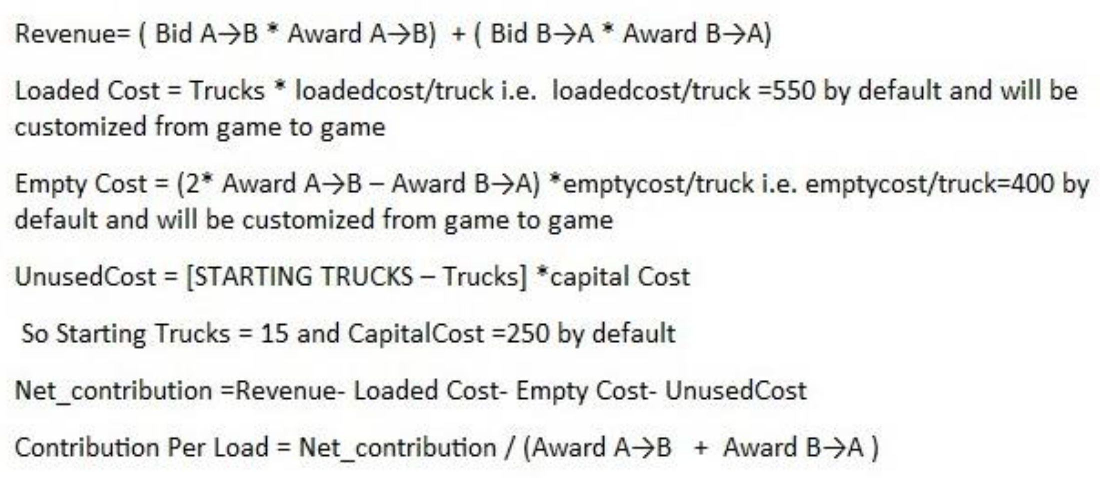
The game logic as a good practice is always written in views.py file. Urls are defined in urls.py for ease and understanding.

Django provides a admin file where we can customize all the admin rights and all the things to be seen only by admin users and not all users. We can sync the models and data of application and create DB we use following commands:

python manage.py makemigrations & python manage.py migrate

Python manage.py runserver 0.0.0.0:9000

Logic Calculation for Normal player:


Revenue= (Bid A→B * Award A→B) + (Bid B→A * Award B→A)

Loaded Cost = Trucks * loadedcost/truck i.e. loadedcost/truck =550 by default and will be customized from game to game

Empty Cost = (2* Award A→B – Award B→A) *emptycost/truck i.e. emptycost/truck=400 by default and will be customized from game to game

UnusedCost = [STARTING TRUCKS – Trucks] *capital Cost

So Starting Trucks = 15 and CapitalCost =250 by default

Net_contribution =Revenue- Loaded Cost- Empty Cost- UnusedCost

Contribution Per Load = Net_contribution / (Award A→B + Award B→A)

Models:

UserProfile:

This model is a part of User portal and extends user model.
Fields:
Company: char field for company
Activation_key: It is used for email verification
key_expires: Used for email verification
is_alive: used for checking online users
is_AI: For AI player availability

Game:

This model maintains the basic records and pre-requisite data of the game.
start_date_time: Tells when the game starts
game_status: Tells weather the game is active, ended or currently being played
number_of_players: Gives the total participants which includes human-players and AI players.
include_AIplayer: It Boolean value which permits the AI player if true
num_of_AIPlayers: AI player participants.
no_of_trucks_total: Trucks available per person
no_of_turns_per_player: Turn Limit
empty_cost_per_truck: Cost reserved for an empty truck
loaded_cost_per_truck: Cost reserved for a loaded truck usually greater than empty truck costs.

unused_capital_cost_perTruck: This is the minimum capital maintained for other cost recovery
created date: Gives the timestamp of creation
is_active = Checks weather a Game is active or not.

UserGameRecords:

This Models connects the game model in synchronization with Bidvalues, GameSimulationDetails.
status: It gives the details status of the player which changes during each turn.
no_of_turns_completed: It mentions the turn value during a particular process or operations.

Bid Values :

This model saves the bid details for each player at each turn
value_a2b: Bid value for head haul.
value_b2a: Bid value for back haul.
no_of_turn: turn number.

GameSimulationDetails:

The processed data after all the bids are generated is been stored in this models.
value_a2b: It copies the value of the Bid A-B
value_b2a: It copies the value of the Bid B-A
trucks_a2b_Awarded: It stores the trucks awarded
trucks_b2a_Awarded: It stores the trucks awarded
trucks_max_awarded: It stores the trucks awarded

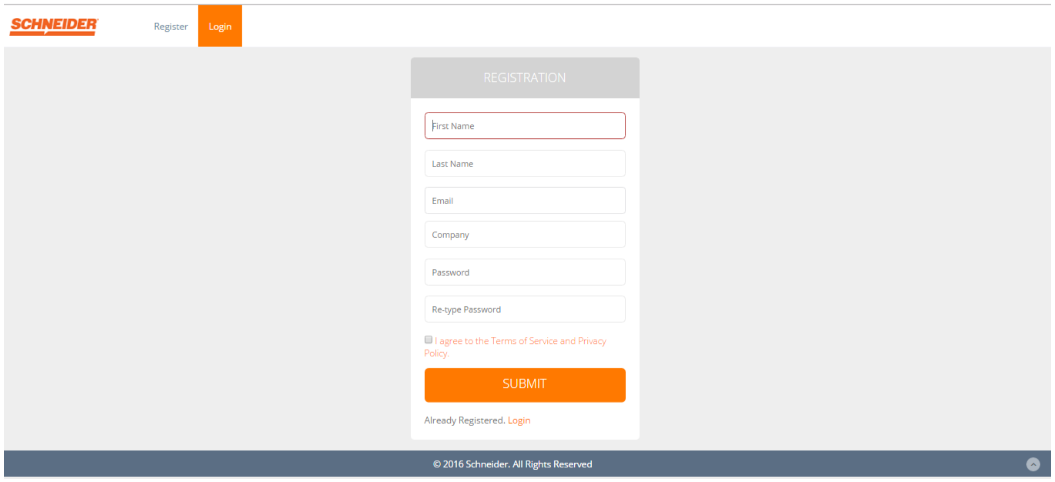
WinnerTable:

The name suggests it gives details of the winners for a particular game based on the maximum net contribution earned during all the turns.
player_total_net_contribution: Sum of all the net contributions earned by a player during a given turn limit gets saved here.
player_rank = Player is ranked based on maximum earnings.

Views:

home:

This view gives render the main login page

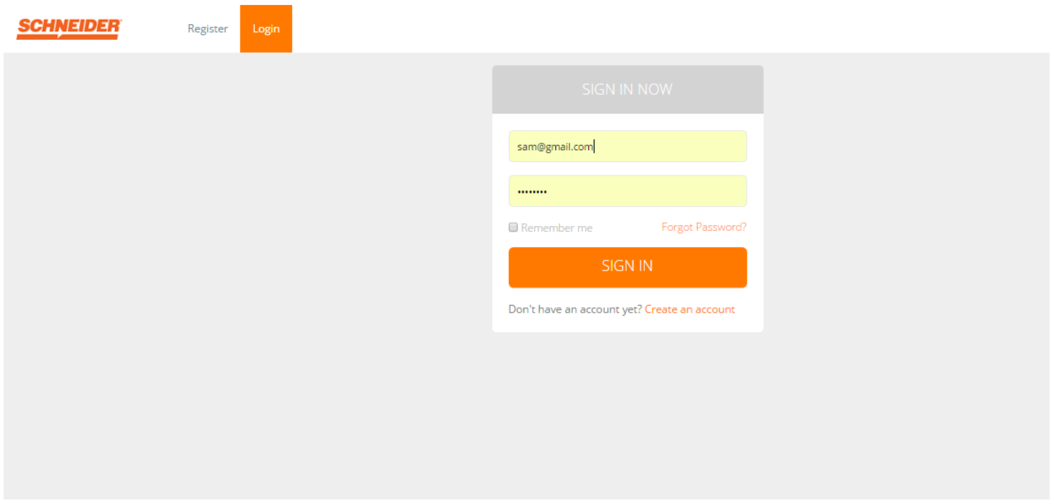

SCHNEIDER
Register
Login
REGISTRATION
First Name
Last Name
Email
Company
Password
Re-type Password
I agree to the Terms of Service and Privacy Policy.
SUBMIT
Already Registered. Login
© 2016 Schneider. All Rights Reserved

user_authentication:

This functions make decisions on the login authentications. If the user is not for it alerts a message stating invalid user or else, it will redirect page to Dashboard on successful authentication.

registration:

As the name suggests the function deals with registrations. Once the player is registered the link is sent to the user email for mail activation. Once the email is activated we can proceed with the game. If the email activation fails, the object stored in the DB will rollback.

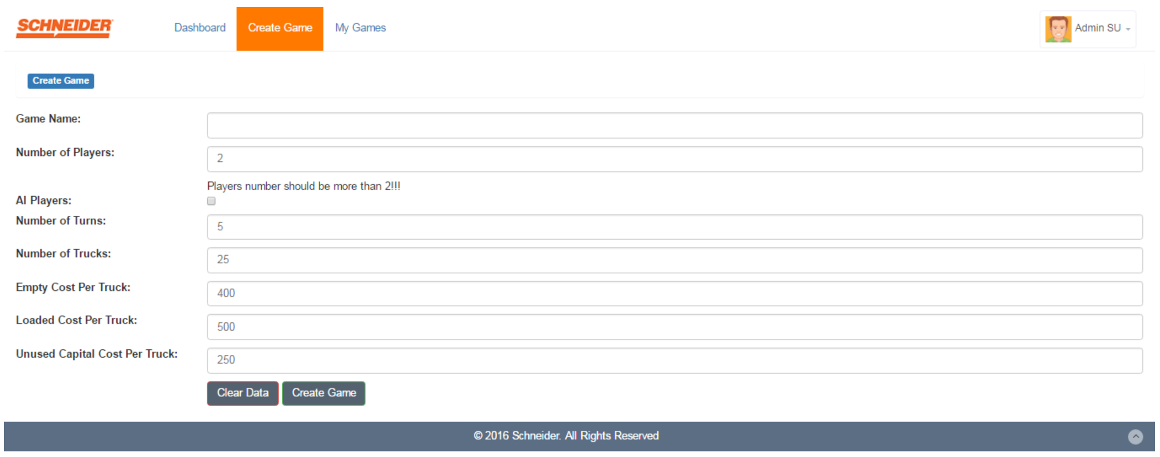


user_session_end:

It’s a logout function which redirects the login page when the session is logged out.

create_game:

This view function creates a new game. It needs certain details to be filled in the form on front-end and store the values in Game model.



Create Game

Game Name:	
Number of Players:	2
	Players number should be more than 2!!!
AI Players:	☐
Number of Turns:	5
Number of Trucks:	25
Empty Cost Per Truck:	400
Loaded Cost Per Truck:	500
Unused Capital Cost Per Truck:	250

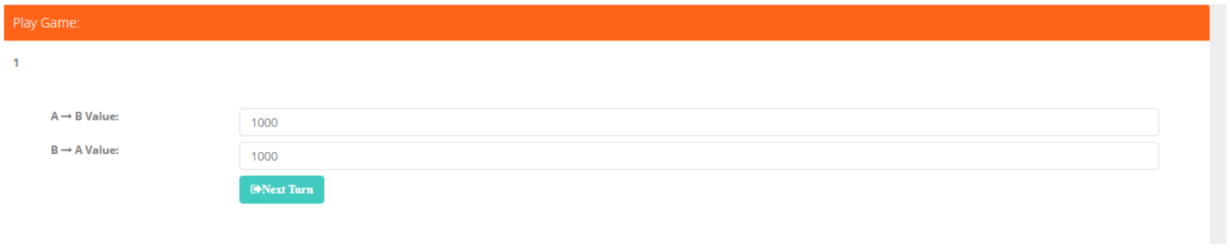
© 2016 Schneider. All Rights Reserved

update_turn_notify:

This function is calculating the GameSimulations for each turn for each player. This function takes the request from Button hit from the front-end and calls the task functions asynchronously every time the button is hit from the front-end template.

move_turn:

This function is triggered only once in every game. Once the creator of the game is aware that the available players have done the bidding, he hits the Next turn button and take all the players who bid for turn 1 to turn 2.



Play Game:

1

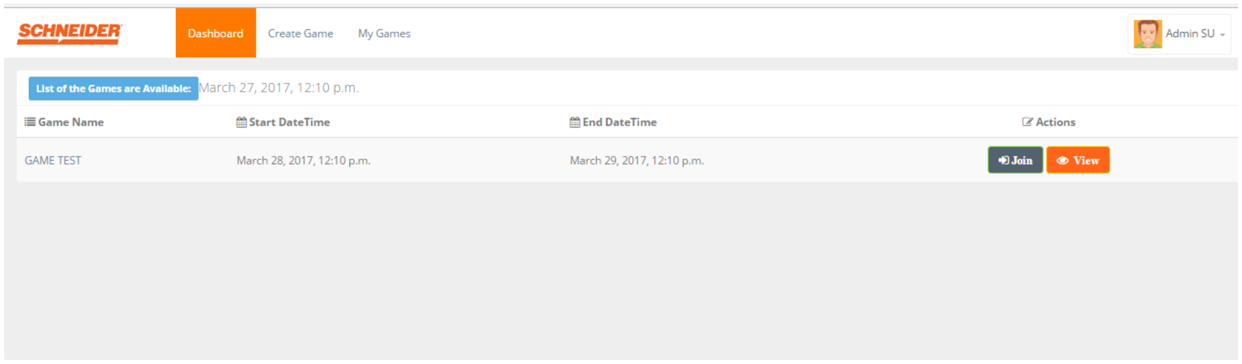
A→B Value:	1000
B→A Value:	1000

get_result:

This function generates the results from the GameSimulation table and stores them into WinnerTable. The results are generated only when all the players quit.

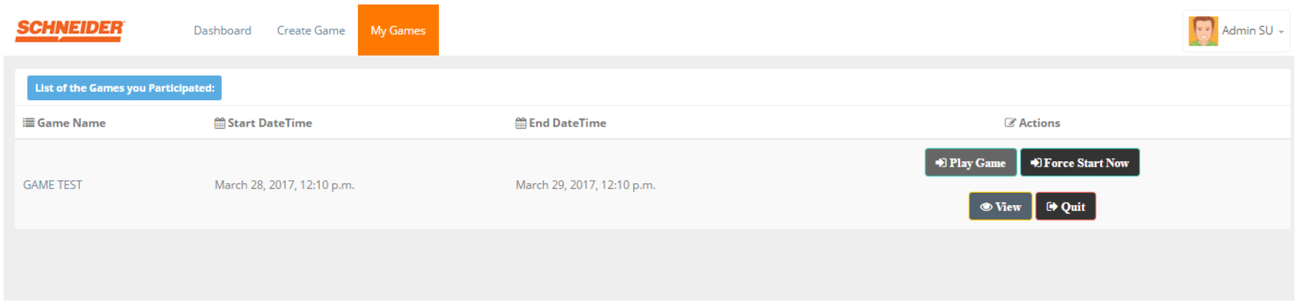
game_details:

This function takes care of all the actions taking place on myDashboard.html template. Its defines various conditions for Viewing, Joining, and seeing Results



List of the Games are Available: March 27, 2017, 12:10 p.m.

Game Name	Start DateTime	End DateTime	Actions
GAME TEST	March 28, 2017, 12:10 p.m.	March 29, 2017, 12:10 p.m.	Join View



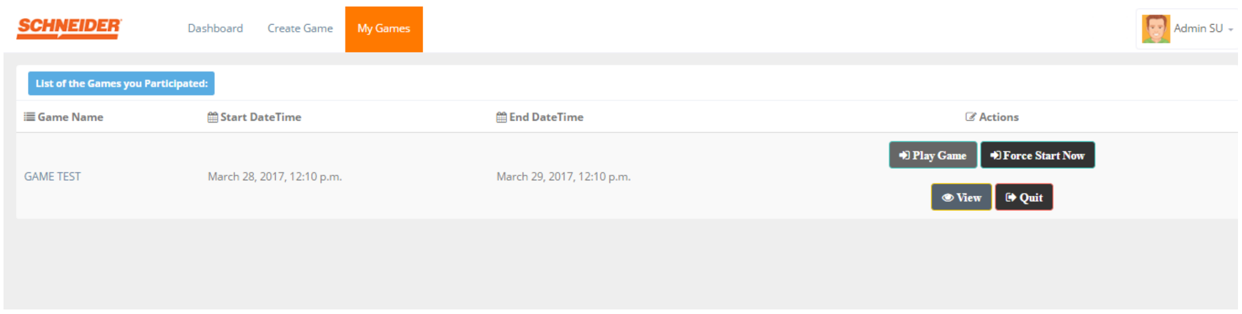
List of the Games you Participated:

Game Name	Start DateTime	End DateTime	Actions
GAME TEST	March 28, 2017, 12:10 p.m.	March 29, 2017, 12:10 p.m.	Play Game Force Start Now View Quit

Fig 1: Game Details

Play_game:

play_game function deals with play_game.html template. All the actions related to game bid are written in this function. It includes Force_play now, Quit, Place Bid etc.



List of the Games you Participated:

Game Name	Start DateTime	End DateTime	Actions
GAME TEST	March 28, 2017, 12:10 p.m.	March 29, 2017, 12:10 p.m.	Play Game Force Start Now View Quit

Celery Tasks:

force_start():

it is an asynchronous task use to start the game. Inside this task we update the bid values for AI players if present as participants. The

force_puch2():

It is an asynchronous task which is an action on button in Place Bid action.

It forces all the players to go to turn2.

calculate_formulas():

This function carries out the most important functionality and carries the efficient logic for the proper working of the game. The calculate function is called every 5seconds after the place bid button is clicked from the front end. It basically checks the player limit is matching the final number of players bided every turn. Once the condition is met it will carry-out the simulation calculations and move to the next turn.