Game Server Challenge

Deployment:

Github : https://github.com/darshan7parmar/gameserver
Heroku : https://gameserverdarshan.herokuapp.com

Technology Stack:

Base Language	Python
Framework	Python Django
API Framework	Django Rest Framework
Database	PostgreSQL
Deployment	Heroku

- I have used Django as base framework and on top of that have installed Rest Framework which provides very good support to develop REST APIs easily

.

- As a Database choice I have various options like MySQL, MongoDB, PostgreSQL, i have selected PostgreSQL because I required to use JSON field as a storage , which only MongoDB(No-SQL) database provides ,however Django framework by default don't support NO-SQL database, so have used PostgreSQL which supports relational as well as non-relational fields.

Installation:

Download source code and configure below things for local installation

Local Installation :

Django, Django Rest Framework, Postgres Database

Install VirtualEnv

```
pip install virtualenv
virtualenv myproject:
source myproject/bin/activate
```

Setup PostgresDB

Install PostgreSQL 9.5.4 and configure database [Note : lower version won't work]

Install Dependencies

pip install -r requirements.txt [It will install dependencies]

Configure database URL

Open gameserver/settings.py and configure your database url like below in ELSE part of ON_HEROKU condition like below,

```
        DATABASE_URL = 'postgres://admin:password@localhost:5432/gameserverdb'
```

Run Migrations and start server:

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

Heroku Installation[Easiest]:

```
Go to gameserver/settings.py
      Set variable ON_HEROKU = TRUE
```

Go to project root directory

Login to Heroku

```
heroku login
```

Create a heroku project

```
heroku create
```

Start Project

```
git push heroku master
```

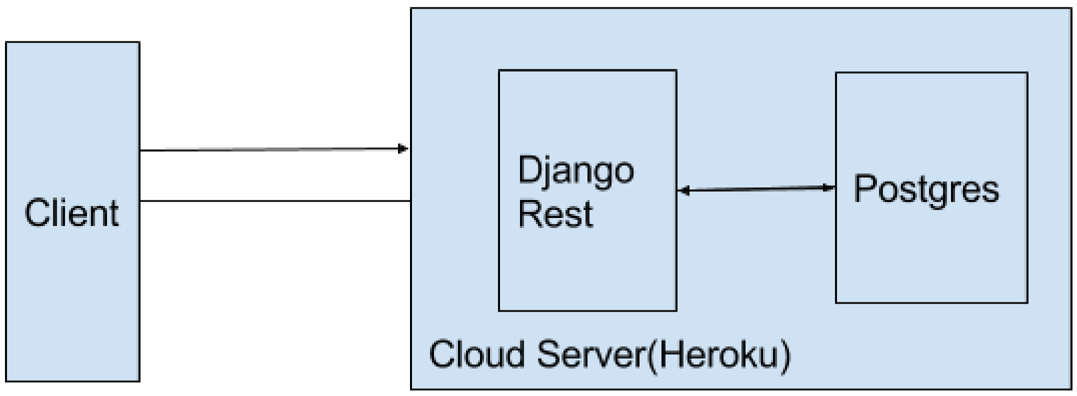
[it will show you link]

Run Migrations

```
heroku run python manage.py makemigrations

heroku run python manage.py migrate
```

Software Architecture



Scalable Solutions:

1) **Scalability can be achieved by configuring web server settings of caching etc**
2) **Database scalability can be achieved by clustering,sharding which postgres supports**

Database Modeling

```
class Player(models.Model):
        nick=models.CharField(max_length=20,blank=True)
        def __str__(self):              # __unicode__ on Python 2
                return str(self.nick)


class Board(models.Model):
        grid=ArrayField(ArrayField(models.CharField(max_length=1)),null=True)
        board_rows=models.IntegerField(default=15)
        board_cols=models.IntegerField(default=15)
        num_words=models.IntegerField(default=10)
        placed_words=JSONField()
        words_list=ArrayField(models.CharField(max_length=50))



class Game(models.Model):
        STATUS = (
    ('w', 'Waiting for start'),
    ('s', 'Started'),
    ('f', 'Finished'),)
        game_status=models.CharField(max_length=1,choices=STATUS)
        players = models.ManyToManyField(Player,related_name='player')
        admin_player=models.ForeignKey(Player,related_name='admin_player')
        turn_sequence=ArrayField(models.CharField(max_length=50))
        current_player=models.ForeignKey(Player,related_name='current_player',null=True)
        words_done=ArrayField(models.CharField(max_length=50))
        scores=JSONField()
        board=models.ForeignKey(Board,related_name="board",null=True)
        min_players=models.IntegerField()
        max_players=models.IntegerField()
        pass_count=models.IntegerField()

        def __str__(self):              # __unicode__ on Python 2
                return str(self.id)
```

- I have kept separate entities for Player,Game,Board so that all three can change on it's way.
- So that in future we can generate multiple games from same board.
- There is many to many relationship between Game and Player so a same player can also participate in multiple games.

APIs:

1. POST /gameserver/game/create

Info : Creates a new game on server

Parameters :

player_id [optional] if player is already registered.

Response :

201 : game_id game created

404 : player does not exists if player id provided.

Example:

```
Request : curl -H "Content-Type: application/json" -X POST
https://gameserverdarshan.herokuapp.com/gameserver/game/create
Response : {  "nick": "player83",  "game_id": 33,  "player_id": 83}
```

2. POST /gameserver/game/join

Info : a user can join game on server

Parameters :

game_id - game id

player_id[optional] - player_id of user

Response :

201 : Player created returns player_id

404 : Game does not exists | Player does not exists

406 : Maximum player already joined game

Example:

```
Request :curl -H "Content-Type: application/json" -X POST -d '{"game_id":"33"}'
https://gameserverdarshan.herokuapp.com/gameserver/game/join
Response :{"nick":"player100","player_id":100}
```

3. POST /gameserver/game/start

Info : starts game on server

Parameters :

game_id - game id

player_id - player id of admin

Response :

200 : Success returns game details

404 : Game does not exists | Player does not exists

401 : Player is unauthorized

417 : Please wait for more players to Join | Game already started or finished

Example:

```
Request :curl -H "Content-Type: application/json" -X POST -d
'{"game_id":"33","player_id":"83"}'
https://gameserverdarshan.herokuapp.com/gameserver/game/start
Response :{"turn_sequence": ["player83","player100"],"status": "Started","grid": [[...]]}
```

4. POST /gameserver/game/info

Info : Information about a game

Parameters:

game_id - game id

player_id - player id of any user who is entitled to play game

Response:

200 : Success returns game details

404 : Game does not exists | Player does not exists

401 : Player is unauthorized

Example:

```
Request :curl -H "Content-Type: application/json" -X POST -d
'{"game_id":"33","player_id":"83"}'
https://gameserverdarshan.herokuapp.com/gameserver/game/info
Response :{"game”: .., “board” :”grid”: .. } [ Game and Grid objects will be displayed]
```

5. POST /gameserver/game/play

Info : a player will play the game by providing different parameters

Parameters:

game_id - game id

player_id - player id of any user who is entitled to play game

word - word to be identified

direction - it can be either “DOWN” or “RIGHT”

start_loc -starting location of word

Response:

200 : OK returns message,score

404 : Game does not exists | Player does not exists | word can not be blank|Direction can be right or down|start location must be provided with two int values

401 : Player is unauthorized | game not started |game finished|It is not your turn to play

Example:

```
Request :curl -H "Content-Type: application/json" -X POST -d
'{"game_id":"33","player_id":"83","direction":"DOWN","word":"orbital","start_loc":[6,1]}'
https://gameserverdarshan.herokuapp.com/gameserver/game/play
Response: {"detail":"success","score awarded":1,"your total score":3}
```

6. POST /gameserver/game/pass

Info : a player will pass the game

Parameters:

game_id - game id

player_id - player id of any user who is entitled to play this game

Response:

200 : OK return success message

404 : Game does not exists | Player does not exists

401 : Player is unauthorized | game not started |game finished|It is not your turn to play

Example:

```
Request :curl -H "Content-Type: application/json" -X POST -d
'{"game_id":"33","player_id":"83"}'
https://gameserverdarshan.herokuapp.com/gameserver/game/pass
Response: {"detail":"Game successfully passed"}
```

```
7. POST /gameserver/game/locate [ THIS API can identify where all words  have been placed]
```

Info : you will able to locate where all words placed in grid

Parameters:

game_id - game id

player_id - player id of any user who is entitled to play this game

Response:

200 : OK return success message and returns list of words where it has been placed in grid with its direction

404 : Game does not exists | Player does not exists

401 : Player is unauthorized | game not started |game finished|It is not your turn to play

Example:

```
Request :curl -H "Content-Type: application/json" -X POST -d
'{"game_id":"33","player_id":"83"}'
https://gameserverdarshan.herokuapp.com/gameserver/game/locate
Response: {"placed_word":{"sanctioning":{"direction":"VER","location":[1,0]} … }
```

Main API Steps :

```
Create Game:
        fetch  request
        If player_id in request and player not already registered in db
                Return 404 player not exists
        create_player()
        create_board()
        create_game()
        add_player_to_game()
        Return game_id, player_id,nick

Join Game:
        fetch request
        Perform below validations
                check_if_game_exists()
                check_if_user_already_joined_game()
                check_if_max_player_joined_game()
        then
                 create_player()
                 add_player_to_game()
         Return success with {player_id,nick}

Start Game:
        fetch  request
        Perform below validations
                check_if_game_exists()
                check_if_player_exists()
                check_if_player_admin()
                check_if_game_already_started()
        then
                change_game_status()

Play Game:
        fetch request
        Perform below validations
                check_if_game_exists()
                check_if_player_exists()
                check_if_player_authorized()
                check_if_game_started()
                check_if_inputword_isvalid()
                check_if_startpos_isvalid()
                check_if_direction_isvalid()
                check_if_word_already_identified()
                check_if_word_not_in_dictionary()
        Then
                change_turn_sequence()
                Reset pass_count
                match=find_match_in_grid()
                If not match
                        Return  response message
                Else
                        update_score()
                        add_word_to_done_words()
                        If words_done=total_words
                                finish_game()
                        Return score
```

```
View Info:
        fetch request
        Perform below validations
                check_if_game_exists()
                check_if_player_exists()
                check_if_player_authorized()
        Then
                Return game and board details

Pass Turn:
        fetch request
        Perform below validations
                check_if_game_exists()
                check_if_player_exists()
                check_if_player_authorized()
                check_if_game_started()
                check_if_your_turn()
        Then
                update_pass_count()
                change_turn_sequence()
                If pass_count equals total_player_in_game
                        finish_game()
                Return success message of turn passed


Create_grid()
        Pick 10 random words from words array and filter out words by replacing special chars
        Sort words in reverse order by length ( largest to smallest)
        While not success:
                Pick next_largest_word()
                While word_not_place

                        Pick a random direction (horizontal or vertical)
                        If direction is horizontal
                                generate_random_row=random(0,len(gridrow))
                                genere_random_col=random(0,len(gridcol)-len(word))
                                If word_can_be_placed()
                                        place_word() and note_down_it's place
                                        break
                        Else
                                generate_random_row=random(0,len(gridrow)-len(word))
                                genere_random_col=random(0,len(gridcol))
                                If word_can_be_placed()
                                        place_word() and note_down_it's place
                                        Break
        If all words_placed
                Break outerloop
        fill_grid_with_random_chars()
        Return grid
```