Anirudh Kaluri
Date: 20 July, 2023

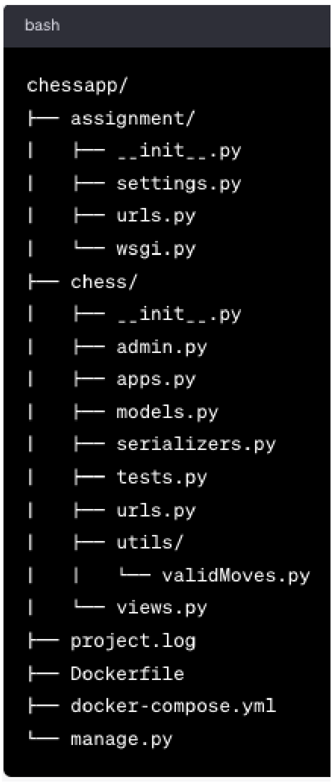
BRIEF OVERVIEW:

The project is a Django REST application with one exposed API. It takes a POST request of a Chess Board configuration and a slug (one of the chess pieces sent in the URL as a parameter) as an input. The response provides all valid moves for the slug assuming the rest of the pieces on the board are opponents.

Notable features:

- Robust logic to get valid moves
- Logging
- Caching
- Unit tests
- Serialization
- Django Rest framework

DIRECTORY STRUCTURE



```bash
chessapp/
├── assignment/
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
├── chess/
|   ├── __init__.py
|   ├── admin.py
|   ├── apps.py
|   ├── models.py
|   ├── serializers.py
|   ├── tests.py
|   ├── urls.py
|   ├── utils/
|   |   └── validMoves.py
|   └── views.py
├── project.log
├── Dockerfile
├── docker-compose.yml
└── manage.py
```

assignment: is the project folder. It has configuration file settings.py and top level url routing.
chess: is the App that encapsulates the functionality required for the problem.
chess/views.py: Implements PositionView which accepts the request and generates response.
chess/urls.py: Implements routing to PositionView when the API call is made.
chess/utils/validMoves.py: Hosts all the methods to compute a list of valid moves.
chess/serializers.py: Consists of PositionSerializer that validates and deserializes input request data.
chess/tests.py: Hosts unit tests to evaluate all the components.

project.log: Log file that logs INFO level and above messages.
Dockerfile: Consists of base image instructions to build the image.
docker-compose.yml: Configuration file specifying services, ports and commands to run after building the image.

MODELS/DATABASE SCHEMA:

No database persistence was used in the project since every call to the API provides data enough to produce the response.

LOGGING

Logging is configured in *settings.py* to provide with formatters, handlers and loggers. They log to *project.log* file present at the Project directory.

CACHING

Caching is to speed up the response if the board configuration and slug sent have already been processed previously. If the result is present in the cache, it will be retrieved and sent as response. If the board configuration and slug aren't in the cache, valid moves will be computed and added to the cache. An in-memory LocMemCache backend is configured in *settings.py* file. This can be easily extended to be used with Redis or Memcached.

SERIALIZATION/VALIDATION

The post method of the PositionView class uses PositionSerializer for validation and deserialization. The following are the validations:
1) No two pieces can have the same positions
2) There can only be one piece of each type
3) There can only be 4 types {Queen, Bishop, Rook, Knight}
4) The positions of the pieces must be valid:
    a) they must not fall outside the grid
    b) They must not be empty
5) The slug should a piece on the board as well as belong to the 4 types mentioned in (3)

In case of failure in validation, an exception will raised, handled and will be logged.

COMPUTATION FOR FINDING ALL VALID MOVES

The computations for finding all the valid moves is in *chess/utils/validMoves.py*

Algorithm in brief:

1) For a given piece find a list of all possible_moves regardless of other pieces attacking
2) Given all the possible_moves for the slug, check what all moves can be attacked by remaining pieces
3) Remove those positions that can be attacked from the possible_moves and return the answer

The important methods used to implement this algorithm in the project are:

get_valid_moves: returns list of valid movies i.e. the answer

get_all_possible_moves: returns a list of all possible moves assuming pieces obstruct but don't attack.

check_attack_on_positions: takes possible moves and checks if these positions can be attacked by other pieces. Remove the positions that can be attacked

Helper methods to implement the above methods are:

add_positions: helps *get_all_possible_moves* by adding possible positions to the list by travelling in a particular direction

direct_move_exists: checks if a "particular piece" can travel from one position to other in a "straight line in one move without obstruction"

obstruction_exists: checks if there is any chess piece between two points on the board using the concept of slope to look for alignment of 3 points

check_limit: checks if a given coordinate falls out of the board (out of range for an 8X8 grid)

get_coordinate_position: translates a chess position terminology to a coordinate plane matrix terminology Example: G1 as [1,1] = [row,column]

Since there is only one API, all these functions have been clubbed in one module *chess/utils/validMoves.py.* However, the functions above are reusable and can be used to implement other necessary chess based operations.

INSTRUCTIONS TO RUN THE PROJECT:

1) Clone the repository at https://bitbucket.org/infilectassignment/chessapp/
2) Open the terminal and navigate into the chessapp directory cloned to your local machine
3) Execute the following command:

```
$docker compose up -d --build
```

The command starts the app at 127.0.0.1:8000. Kill if any process is already runs on that address.

API End Point: 127.0.0.1/chess/<string:slug>
Method: POST
Body: JSON data with chessboard positions

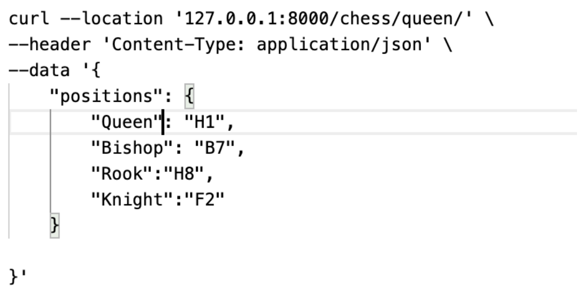
INSTRUCTIONS TO TEST THE PROJECT
After the App starts running, there are three ways to test:

1) UNIT TESTS
- Run the following command in the terminal

```
$docker compose exec web python manage.py test
```

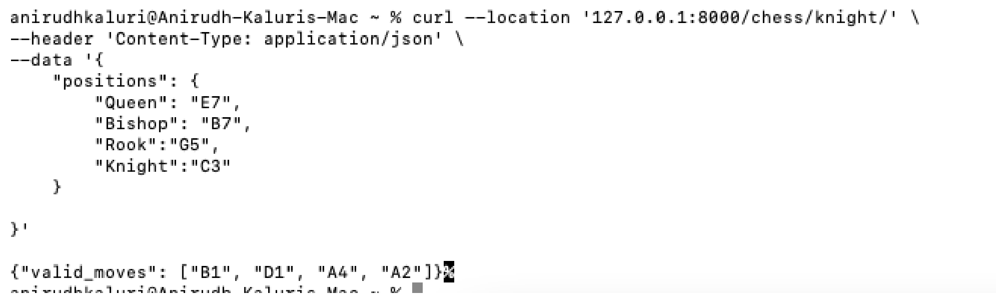
2) TESTING WITH CURL
- Open *CurlTests.txt* file present in the repository.
- Copy a curl command.
- Run it in the command line on the terminal.
- Verify the output with the expected output present in the same file.

For example copy the below command present in the C*urlTests.txt file* and paste it in the command line



```
curl --location '127.0.0.1:8000/chess/queen/' \
--header 'Content-Type: application/json' \
--data '{
    "positions": {
        "Queen": "H1",
        "Bishop": "B7",
        "Rook":"H8",
        "Knight":"F2"
    }

}'
```

The output is as follows:



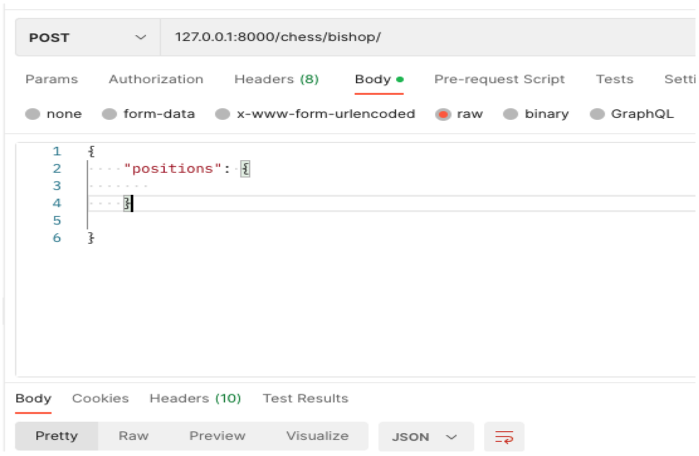
```
anirudhkaluri@Anirudh-Kaluris-Mac ~ % curl --location '127.0.0.1:8000/chess/knight/' \
--header 'Content-Type: application/json' \
--data '{
    "positions": {
        "Queen": "E7",
        "Bishop": "B7",
        "Rook":"G5",
        "Knight":"C3"
    }

}'

{"valid_moves": ["B1", "D1", "A4", "A2"]}%
anirudhkaluri@Anirudh-Kaluris-Mac ~ %
```

3) IMPORT TESTS INTO POSTMAN

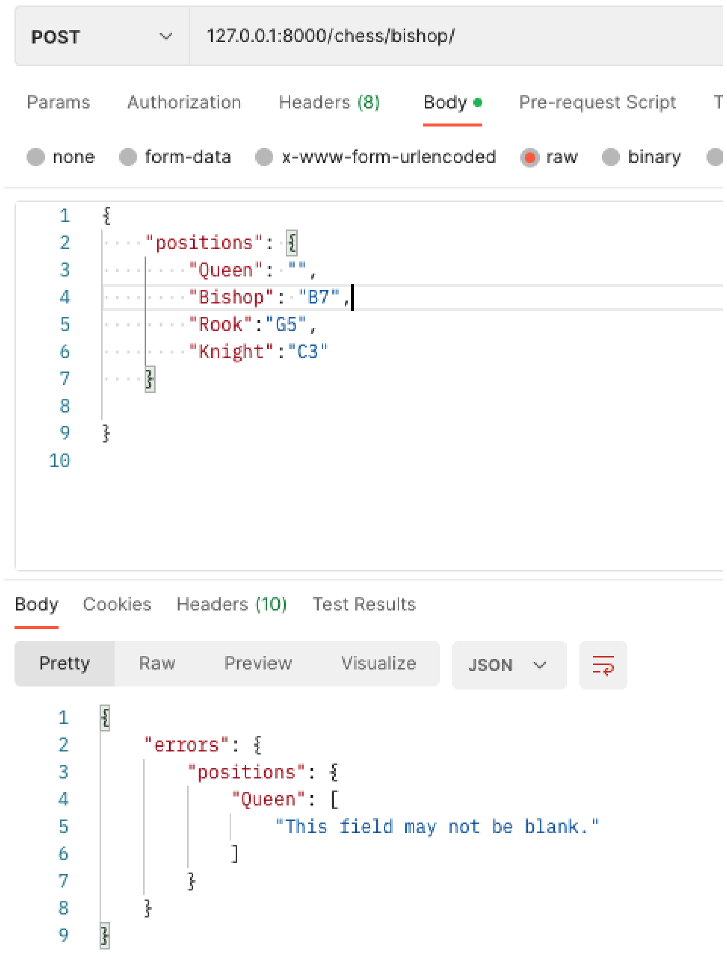
- Open Postman desktop application
- Click on File
- Select import
- Browse to the *Infilect.postman_collection.json* file present in the repository
- Add as a new collection
- Run each request in the collection

END NOTES
In case of any issues please contact me at anirudhkaluri@gmail.com

SOME SAMPLE POSTMAN SCREENSHOTS

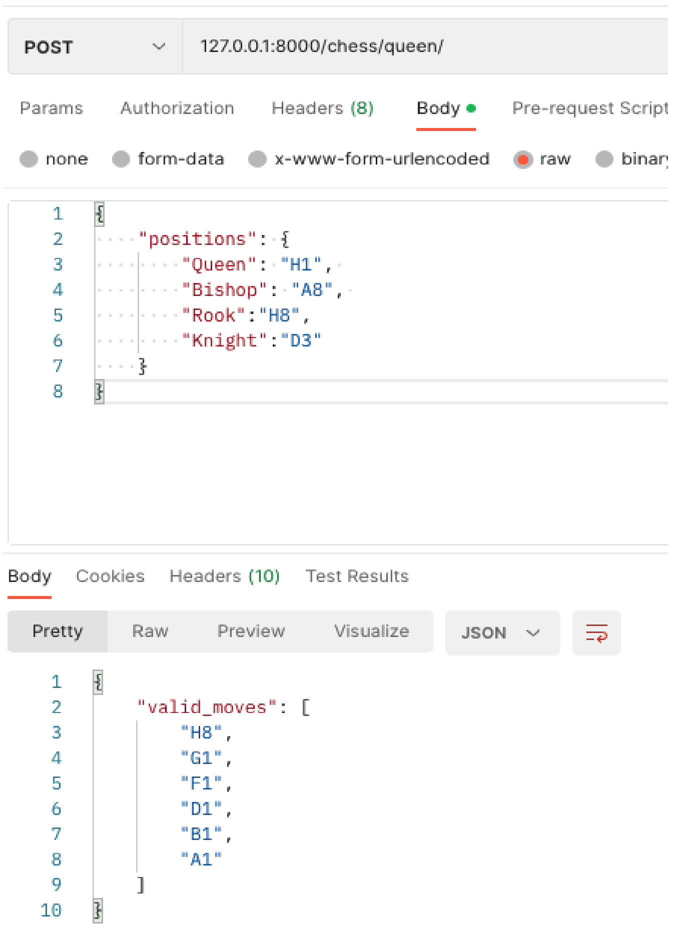


```
{
    "positions": {

    }
}
```



```
{
    "positions": {
        "Queen": "",
        "Bishop": "B7",
        "Rook":"G5",
        "Knight":"C3"
    }

}

```

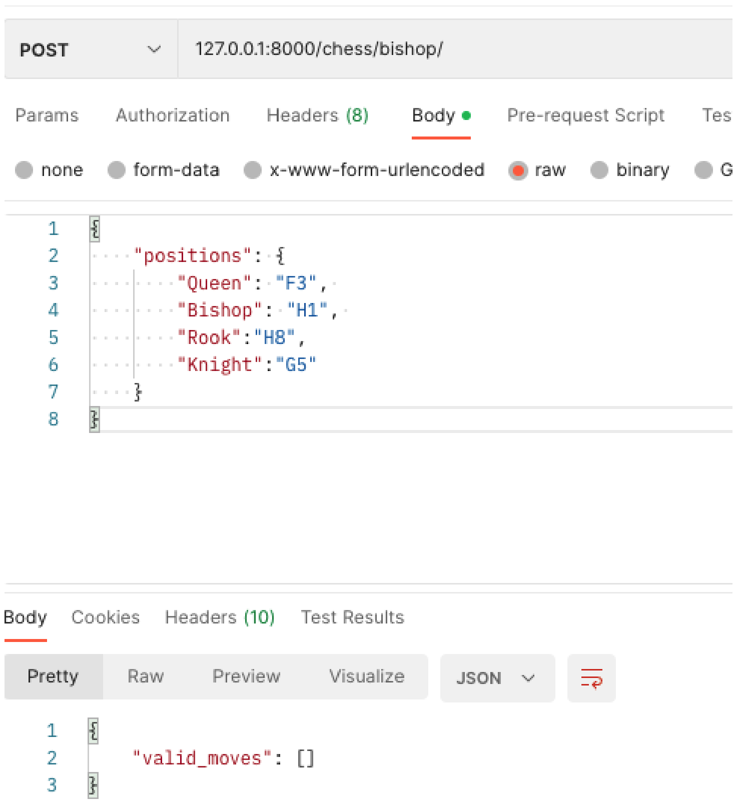
```
{
    "errors": {
        "positions": {
            "Queen": [
                "This field may not be blank."
            ]
        }
    }
}
```



127.0.0.1:8000/chess/queen/

```
{
    "positions": {
        "Queen": "H1",
        "Bishop": "A8",
        "Rook":"H8",
        "Knight":"D3"
    }
}
```

```
{
    "valid_moves": [
        "H8",
        "G1",
        "F1",
        "D1",
        "B1",
        "A1"
    ]
}
```



POST 127.0.0.1:8000/chess/bishop/

Params Authorization Headers (8) Body Pre-request Script Tes

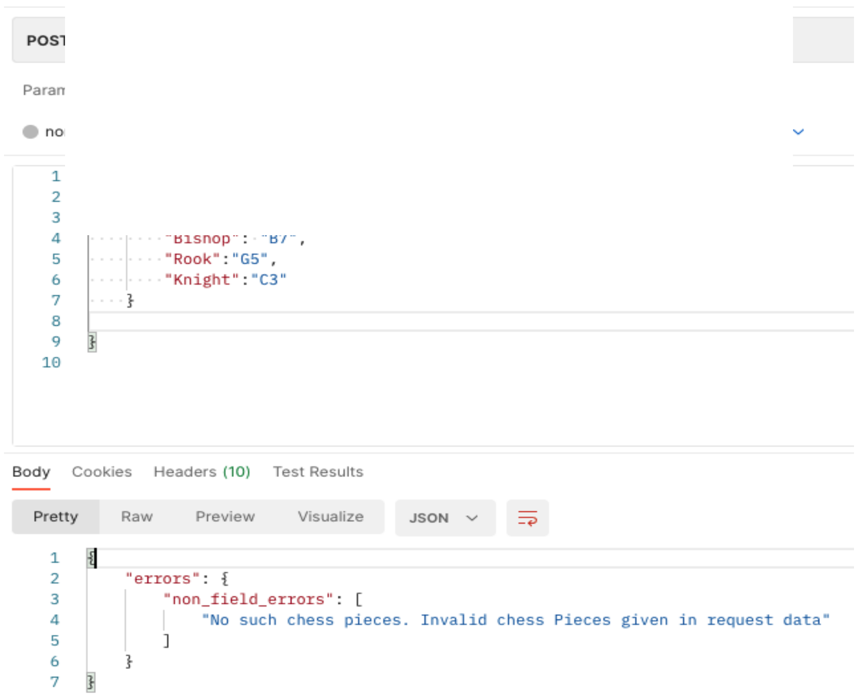
none form-data x-www-form-urlencoded [✓ raw] binary G

```
{
    "positions": {
        "Queen": "F3",
        "Bishop": "H1",
        "Rook":"H8",
        "Knight":"G5"
    }
}
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "valid_moves": []
}
```



POST

Param

no

```
        "Bishop": "B7",
        "Rook":"G5",
        "Knight":"C3"
    }

}
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "errors": {
        "non_field_errors": [
            "No such chess pieces. Invalid chess Pieces given in request data"
        ]
    }
}
```