International Institute of Information Technology, Bangalore

AuctionBID – Host A Online Tournament Auction

(In the Guidance of Professor B. Thangaraju and Sameer Khurd)

Subhodeep Sahoo
(MT2020129)

Pushpendra Kumar
(MT2020135)

Table of Contents

Objective 3

Technology Used 3

I. Development Walkthrough 4

1. Setting up Django and React Environment 4
2. Creating Django Project & React Framework 5
3. Setting up REST API 9
4. Integration of Frontend and API 12
5. Auction Functionality 17

II. Ops Walkthrough 19

1. Source Control Management 20
2. Building Docker Image 21
3. Continuous Integration 24
4. Building Project from Docker Image 26
5. Continuous Deployment 27
6. Continuous Monitoring 27

Chalenges Faced 31

References 32

Tournament Auction using DevOps

Objective :

Let say Karnataka Cricket Board wants to organize a T20 tournament. They are planning to host the tournament between 8 teams. As, organising the player auction can cost millions of rupees they can host the auction online. Our web application will give them the opportunity to do so. The steps are:

1. First an organiser needs to register in the web portal where they can host the auction.

2. Then it can add all the teams and players those are interested to participate in the tournament.

3. Now the teams can register and signin in the tournament and can visit auction page where they can bid for a player.

4. The organizer also need to visit the auction page when the auction started where they can finish the bid for a player and bring a new player for bidding. The organiser can also see the details of completed and current players' auction status.

5. All the teams in this tournament can bid for the ongoing player and can see the status of current and ongoing player.

Technology Used :

1. **Django –** Django is used to as a Web framework to run our website. We used that because it is redicolously fast and we can make as many app to support our website.

2. **React –** React is used to make to create interactive User Interfaces. It has thousands of templates which helps us to build a beautiful web application.

3. **Ops –** We are using Git and Github for source control management, Docker Image for creating the container, Jenkins for continuous integration, Ansible for continuous deployment and ELK stack for continuous monitoring.

I. Developement Walkthrough :

The steps are as follows –

1. Setting up django and react environment :

We are using virtual environment to run the django project. To create the virtual environment we are using conda. First we need to setup conda. The steps are following :

- Download the bash file 'Anaconda-latest-Linux-x86_64.sh' from 'https://www.anaconda.com/products/individual'.
- Now open a terminal and run the following commands

  ```
  $ sudo chmod +x Anaconda-latest-Linux-x86_64.sh
  ```

  ```
  $ bash Anaconda-latest-Linux-x86_64.sh
  ```

- Create a virtual environment using following command

  $ conda create -n **djangoenv** python=3.6 anaconda

  we are using python 3.6 to develop our project

- Activate the virtual environment

  ```
  $ source activate djangoenv
  ```

- Install django and django rest framework in this virtual environment

  ```
  $ pip3 install django djangorestframework
  ```

- Now we will install nodejs and npm to setup react environment using following commands

  $ sudo apt-get install nodejs

  $ node -v (to check nodejs version)

  $ npm -v (to check npm version)

2. Creating django project & react framework :

- We will create our project using command

  $ django-admin startproject auction

- Now we will create an app inside auction directory that will handel our all api by using

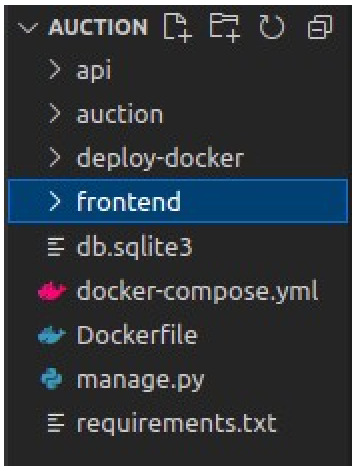
  $ cd auction

  $ django-admin startapp api

- Now we will create another app named frontend that will handel all frontend web views.

  $ django-admin startapp frontend

- Now our folder structure look like as below



```
AUCTION
  > api
  > auction
  > deploy-docker
  > frontend
    db.sqlite3
    docker-compose.yml
    Dockerfile
    manage.py
    requirements.txt
```

(Ignore all docker files and folders. We will discuss it later)

- Now inside frontend directory we will create some folder

  (i) templates (ii) src (iii) static

- Inside template directory we will create a folder

  – frontend (all html file will be here)

  Inside src directory we will create a folder

  – components (all react related javascript file will be here)

  Inside static directory we will create some folders

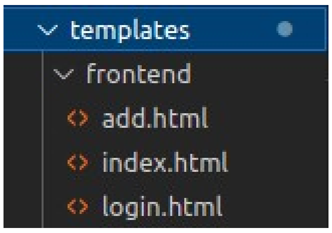
  – css (all stylesheet file will be here)

  – images (all image file will be here)

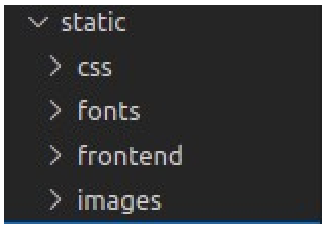
  – frontend (all javascript file related to html will be here)

  – fonts (all font family will be here)

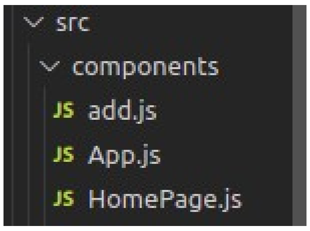
  The structure will look like as follows -



```
templates
  frontend
    add.html
    index.html
    login.html
```



```
static
  css
  fonts
  frontend
  images
```



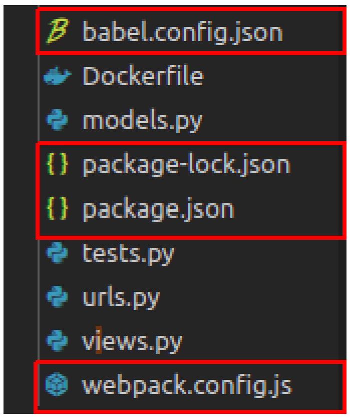
```
src
  components
    add.js
    App.js
    HomePage.js
```

- Now we will install all react packages inside frontend

```
$ npm init -y

$ npm i webpack webpack-cli --save-dev

$ npm i @babel/core babel-loader @babel/preset-env
@babel/preset-react --save-dev

$npm i react react-dom --save-dev
```

```
$ npm install @material-ui/core

$ npm install @babel/plugin-proposal-class-properties

$ npm install react-router-dom

$ npm install @material-ui/icons
```

Now we will configure some files ‘babel.config.json’, ‘package-lock.json’, ‘package.json’, ‘webpack.config.js’. The configurations are written in the files inside the fronend directory.



babel.config.json
DockerFile
models.py
package-lock.json
package.json
tests.py
urls.py
views.py
webpack.config.js

(for more details watch youtube video : Django & React (Tech with Tim) Tutorial 3)

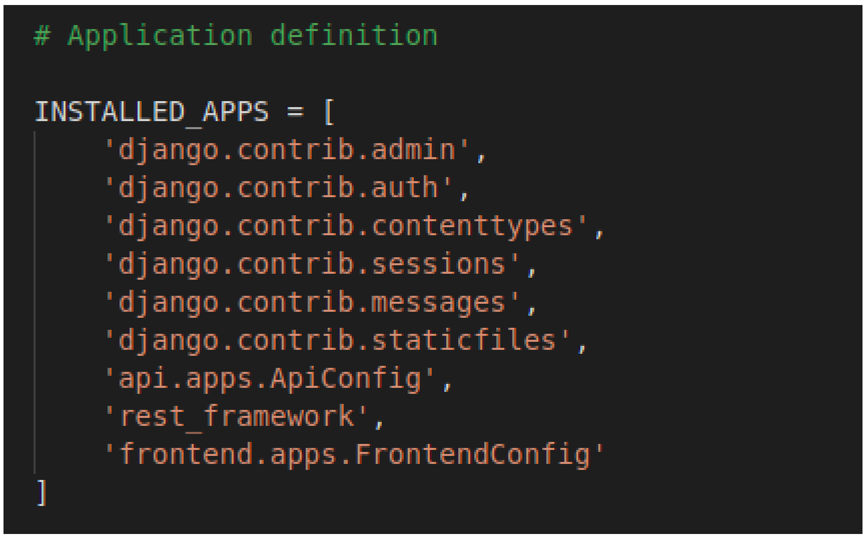
- Now we will setup the settings.py and urls.py files that is inside auction directory.

We will add our apps inside INSTALLED_APPS in settings.py as follows:

```
'api.apps.ApiConfig',

'rest_framework',

'frontend.apps.FrontendConfig'
```

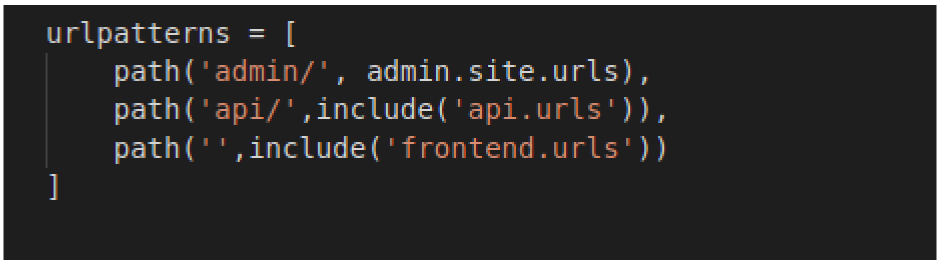
Now we will add our BASE_DIR as

BASE_DIR = Path(__file__).resolve().parent.parent



```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api.apps.ApiConfig',
    'rest_framework',
    'frontend.apps.FrontendConfig'
]
```

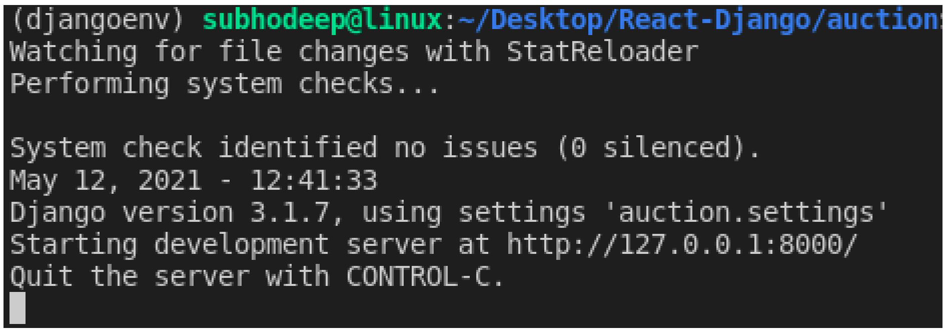
Now in urls.py we will add path of all urls as below :



```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/',include('api.urls')),
    path('',include('frontend.urls'))
]
```

- Now we can run our django server by following command,

$ python3 manage.py runserver



```
(djangoenv) subhodeep@linux:~/Desktop/React-Django/auction
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 12, 2021 - 12:41:33
Django version 3.1.7, using settings 'auction.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

We can run the webserver in browser using the url, http://127.0.0.1:8000/

3. Setting up REST API :

A REST API is a standardized way to provide data to other applications. Those applications can then use the data however they want. Sometimes, APIs also offer a way for other applications to make changes to the data. We are using following REST API requests :

- GET — The most common option, returns some data from the API based on the endpoint you visit and any parameters you provide.

- POST — Creates a new record that gets appended to the database.

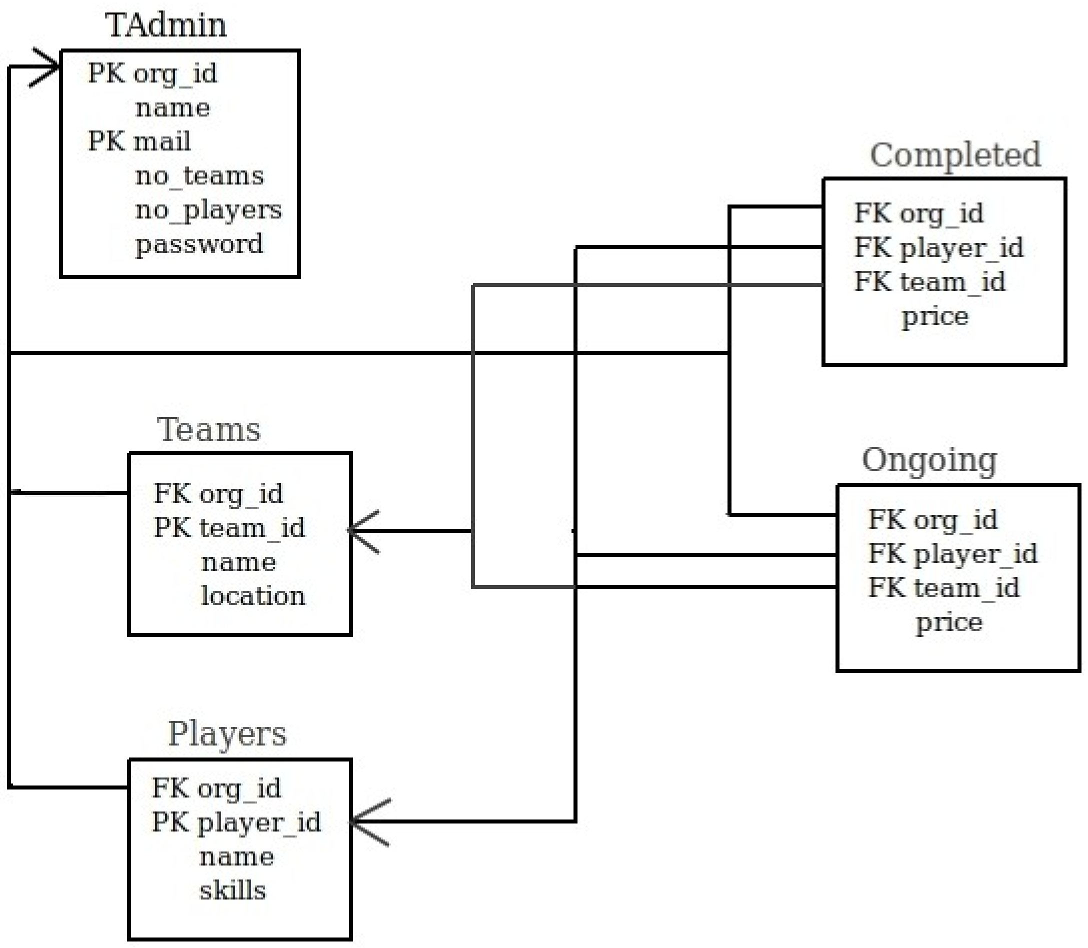
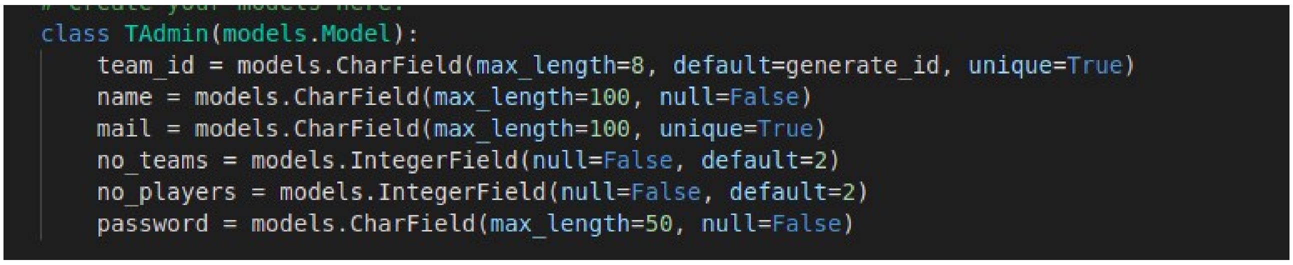


FIG : Schema of auction database

Now we will access the database using django rest framework using following 4 files

- **models.py :** A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.

  The example model defines the team organizer as TAdmin table



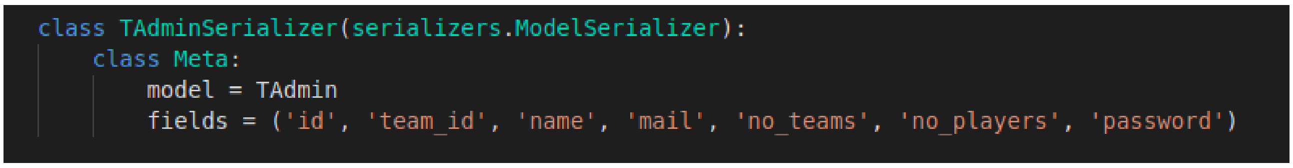
```python
class TAdmin(models.Model):
    team_id = models.CharField(max_length=8, default=generate_id, unique=True)
    name = models.CharField(max_length=100, null=False)
    mail = models.CharField(max_length=100, unique=True)
    no_teams = models.IntegerField(null=False, default=2)
    no_players = models.IntegerField(null=False, default=2)
    password = models.CharField(max_length=50, null=False)
```

  Now we need to use following command for migration of table

  $ python3 manage.py makemigrations

  $ python3 manage.py migrate

- **serializers.py :** Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data. Declaring a serializer looks very similar to declaring a form,

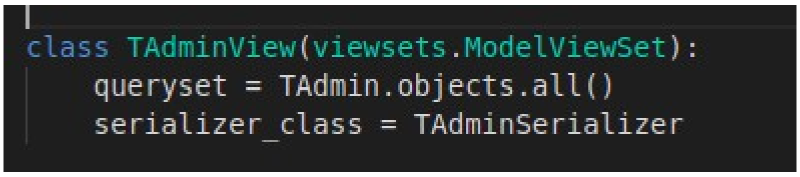


```python
class TAdminSerializer(serializers.ModelSerializer):
    class Meta:
        model = TAdmin
        fields = ('id', 'team_id', 'name', 'mail', 'no_teams', 'no_players', 'password')
```

- **views.py :** A view is a callable which takes a request and returns a response. This can be more than just a function, and

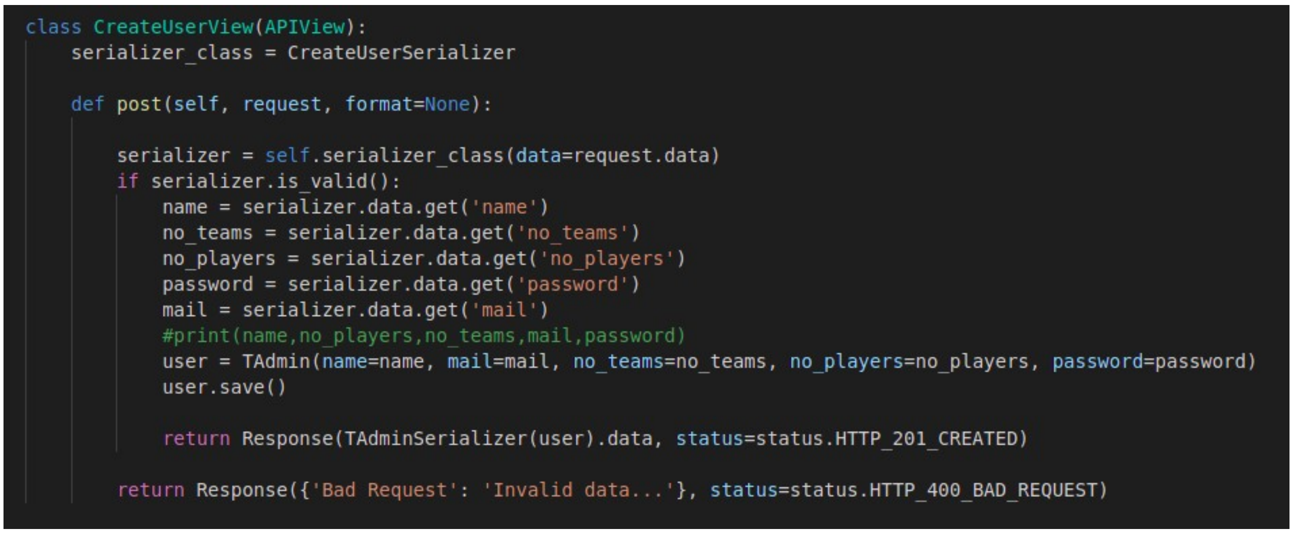
Django provides an example of some classes which can be used as views. These allow you to structure your views and reuse code by harnessing inheritance and mixins.

We are accessing TAdmin table data for user login verification using GET request. That table stores all registered tournament organizer. **views** for the request as follows,



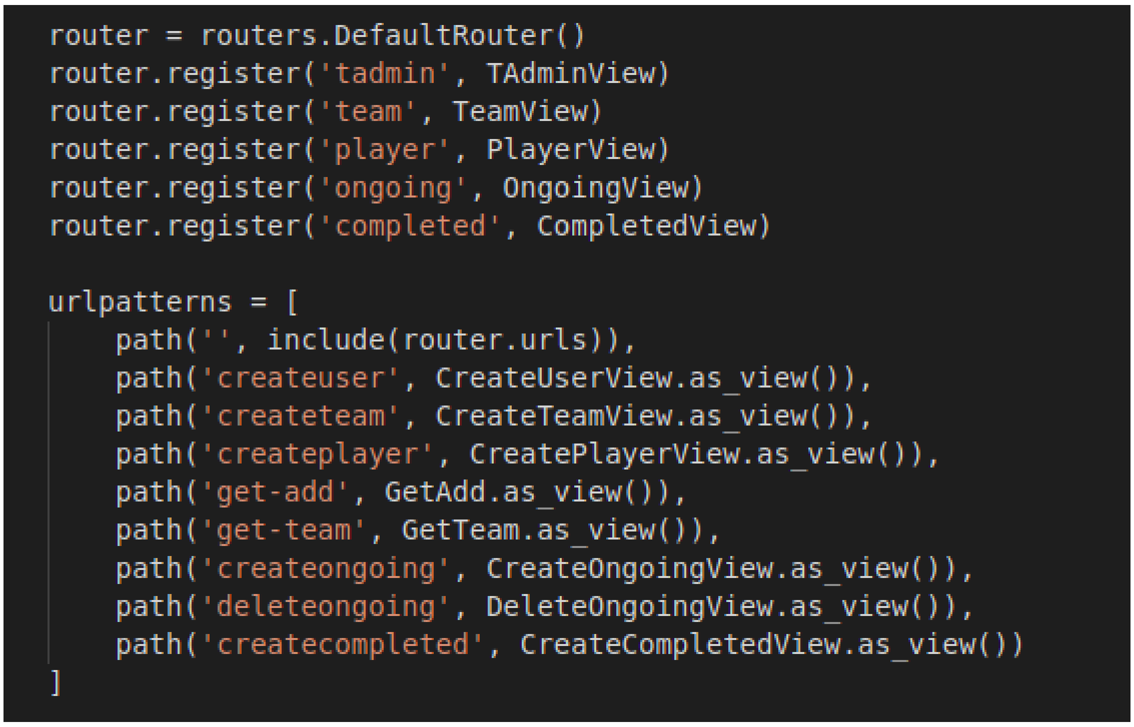
```
class TAdminView(viewsets.ModelViewSet):
    queryset = TAdmin.objects.all()
    serializer_class = TAdminSerializer
```

We are using createuserview for tournament organizer registration using POST request. **views** for the request as follows,



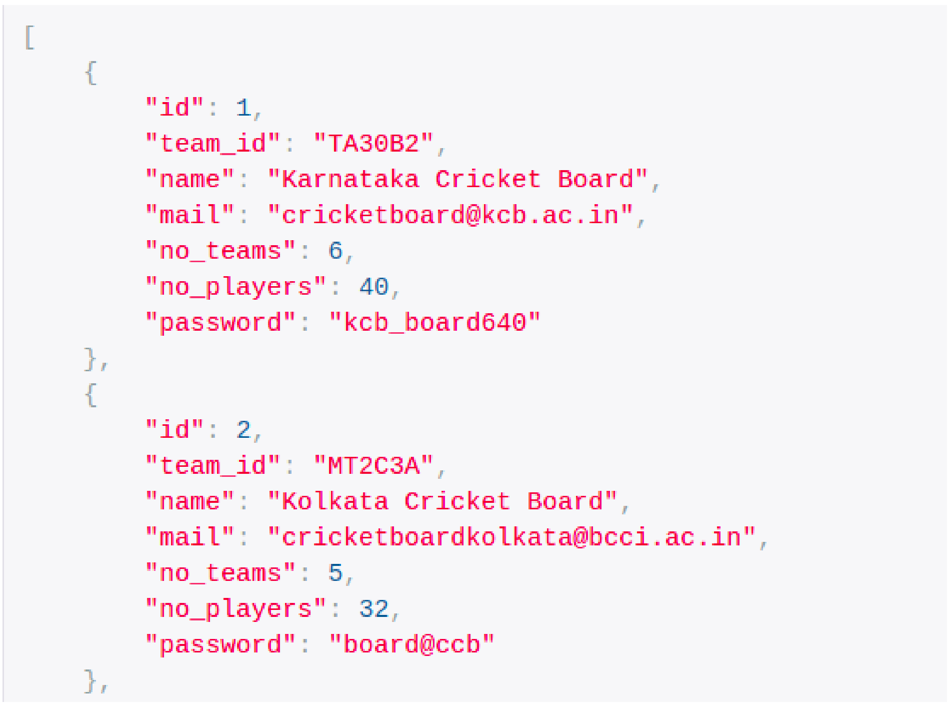
```
class CreateUserView(APIView):
    serializer_class = CreateUserSerializer

    def post(self, request, format=None):

        serializer = self.serializer_class(data=request.data)
        if serializer.is_valid():
            name = serializer.data.get('name')
            no_teams = serializer.data.get('no_teams')
            no_players = serializer.data.get('no_players')
            password = serializer.data.get('password')
            mail = serializer.data.get('mail')
            #print(name,no_players,no_teams,mail,password)
            user = TAdmin(name=name, mail=mail, no_teams=no_teams, no_players=no_players, password=password)
            user.save()

            return Response(TAdminSerializer(user).data, status=status.HTTP_201_CREATED)

        return Response({'Bad Request': 'Invalid data...'}, status=status.HTTP_400_BAD_REQUEST)
```

- **urls.py :** Finally we are creating some api urls to view those database data as json file format in our web browser.



```
router = routers.DefaultRouter()
router.register('tadmin', TAdminView)
router.register('team', TeamView)
router.register('player', PlayerView)
router.register('ongoing', OngoingView)
router.register('completed', CompletedView)

urlpatterns = [
    path('', include(router.urls)),
    path('createuser', CreateUserView.as_view()),
    path('createteam', CreateTeamView.as_view()),
    path('createplayer', CreatePlayerView.as_view()),
    path('get-add', GetAdd.as_view()),
    path('get-team', GetTeam.as_view()),
    path('createongoing', CreateOngoingView.as_view()),
    path('deleteongoing', DeleteOngoingView.as_view()),
    path('createcompleted', CreateCompletedView.as_view())
]
```

Now we can get all tables details as json file format in web browser as below



```
[
    {
        "id": 1,
        "team_id": "TA30B2",
        "name": "Karnataka Cricket Board",
        "mail": "cricketboard@kcb.ac.in",
        "no_teams": 6,
        "no_players": 40,
        "password": "kcb_board640"
    },
    {
        "id": 2,
        "team_id": "MT2C3A",
        "name": "Kolkata Cricket Board",
        "mail": "cricketboardkolkata@bcci.ac.in",
        "no_teams": 5,
        "no_players": 32,
        "password": "board@ccb"
    },
```

FIG : API View of TAdmin table in chrome web browser

4. Integration of frontend and api :

- In frontpage there are some portals where a tournament organiser can visit register and login page and also tournament participant clubs or teams can login through the portal.

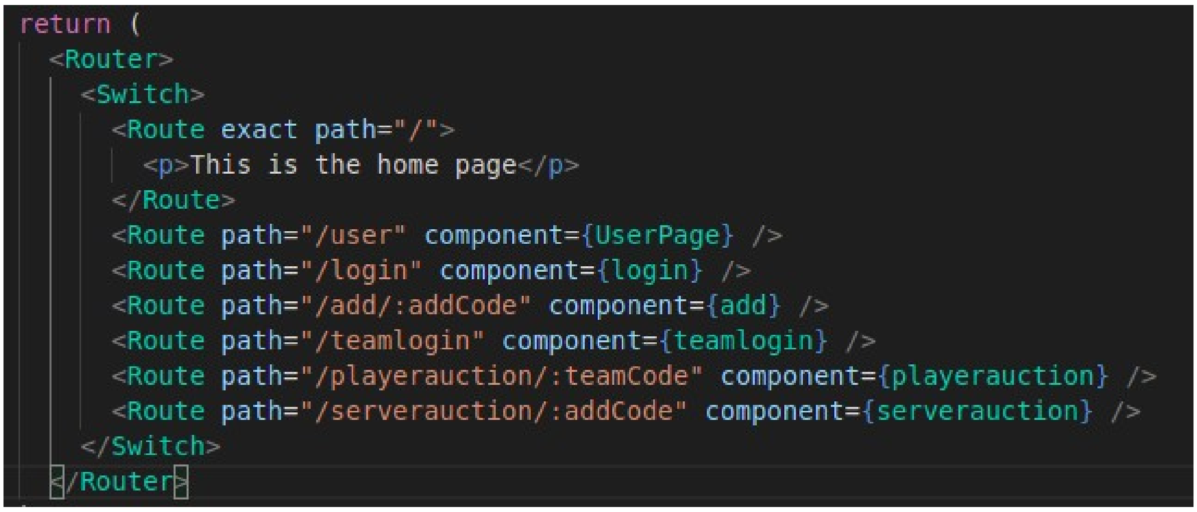
- For other pages we are using material ui (an react based library) to design our forms and tables in our webpages. There are some steps to connect our reactapp with our frontend

  i. In our html file we are creating an id tag called "app" that can be called from our App.js file as

     const appDiv = document.getElementById("app");

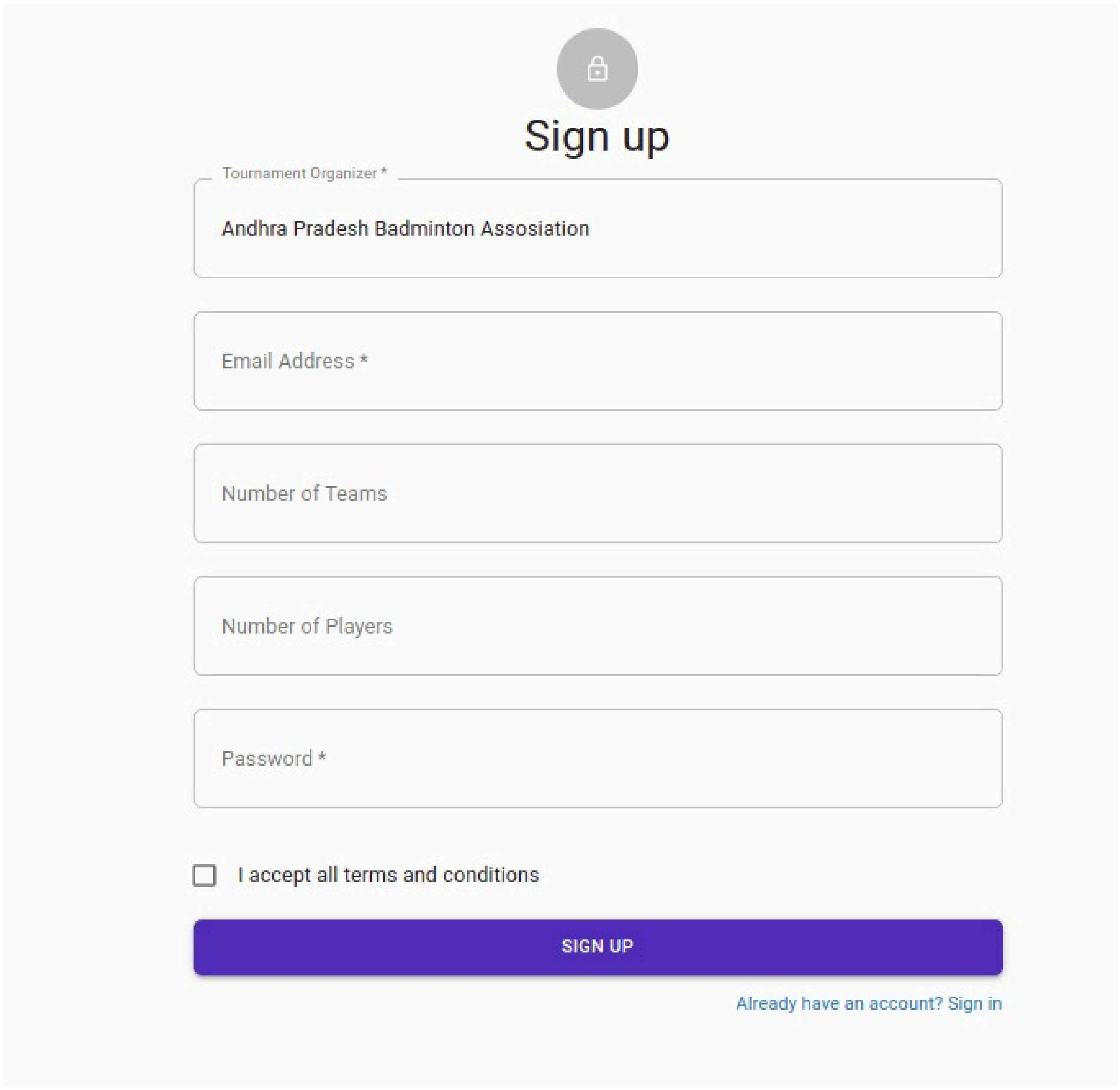
     and now we can render the app.

  ii. We have also a Homapage.js file that is routing all the reactapp to our front page.



```
return (
  <Router>
    <Switch>
      <Route exact path="/">
        <p>This is the home page</p>
      </Route>
      <Route path="/user" component={UserPage} />
      <Route path="/login" component={login} />
      <Route path="/add/:addCode" component={add} />
      <Route path="/teamlogin" component={teamlogin} />
      <Route path="/playerauction/:teamCode" component={playerauction} />
      <Route path="/serverauction/:addCode" component={serverauction} />
    </Switch>
  </Router>
```

  iii. Now in user page (user.html & UserPage.js file) for organiser registration we are creating a form using material ui.


Sign up

Tournament Organizer *
Andhra Pradesh Badminton Assosiation

Email Address *

Number of Teams

Number of Players

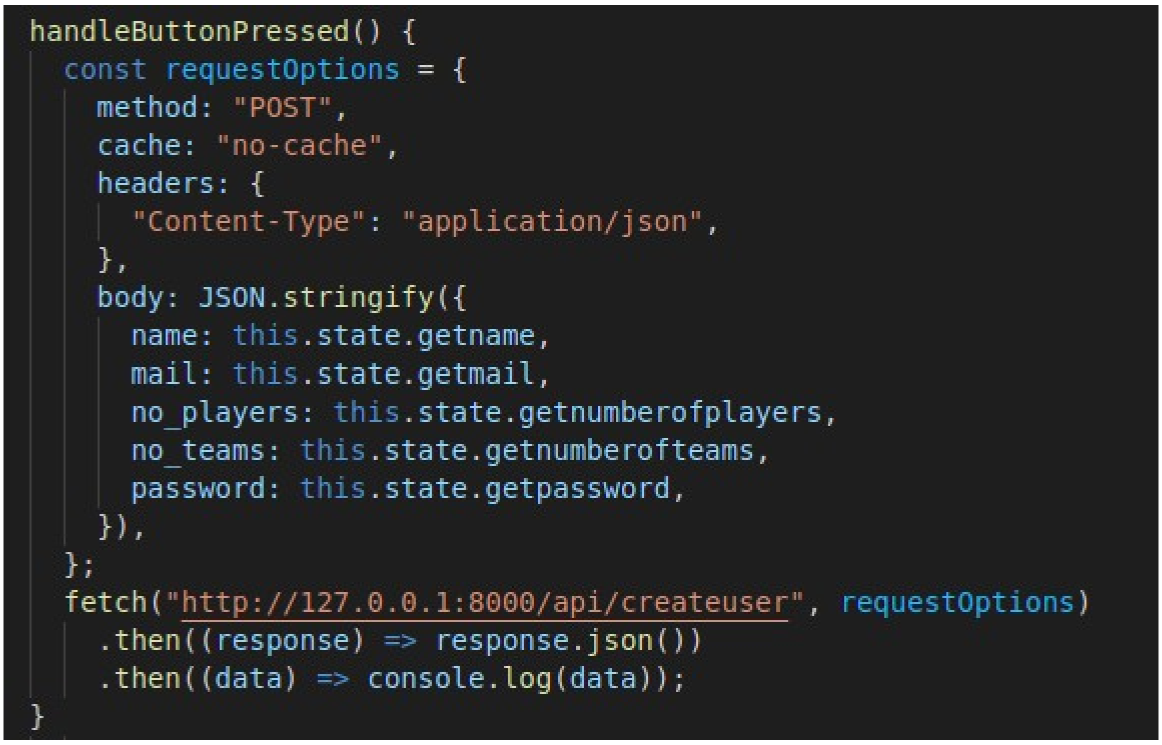
Password *

☐ I accept all terms and conditions

SIGN UP

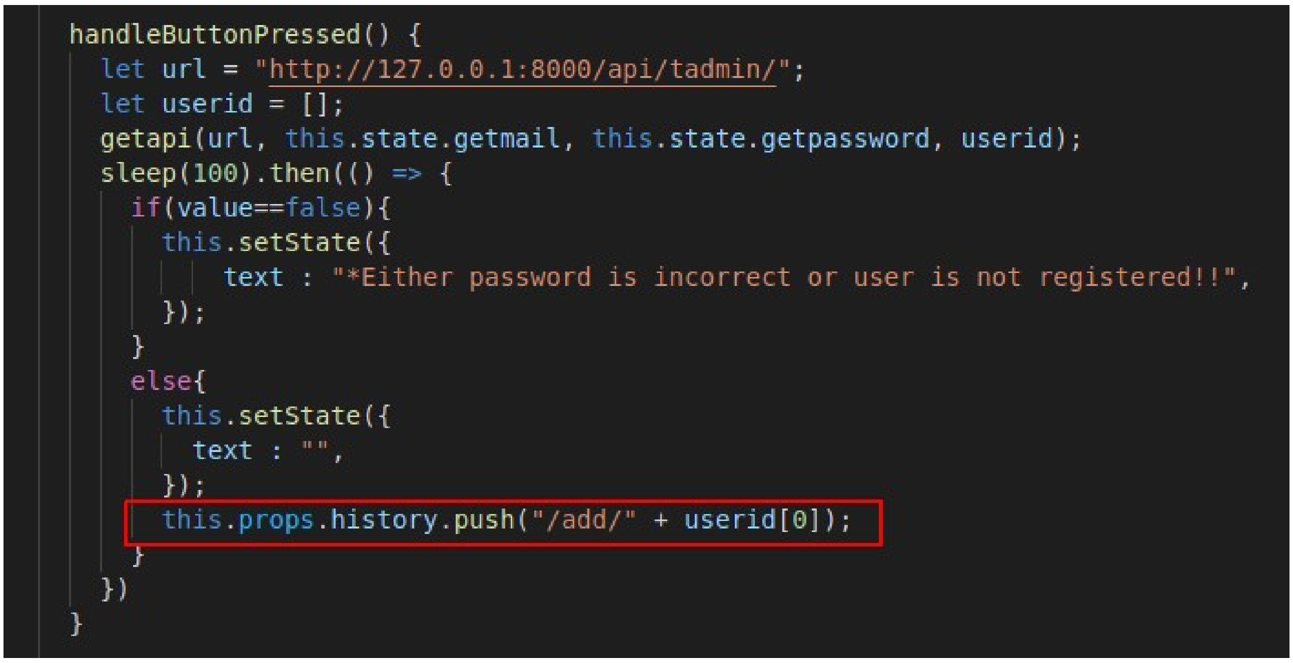
Already have an account? Sign in

We are using POST request to store the data that is registered by the organization. At first, we are using the /createuser api request that can handle all the data and send the data as json response to the django view. Then we are creating a user instance of the data and store the data in TAdmin table (the CreateUserView is shown in the views.py section). Then we are sending a response to javascript that the user is created successfully. The POST request handle in js file is as follows,



```
handleButtonPressed() {
  const requestOptions = {
    method: "POST",
    cache: "no-cache",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      name: this.state.getname,
      mail: this.state.getmail,
      no_players: this.state.getnumberofplayers,
      no_teams: this.state.getnumberofteams,
      password: this.state.getpassword,
    }),
  };
  fetch("http://127.0.0.1:8000/api/createuser", requestOptions)
    .then((response) => response.json())
    .then((data) => console.log(data));
}
```

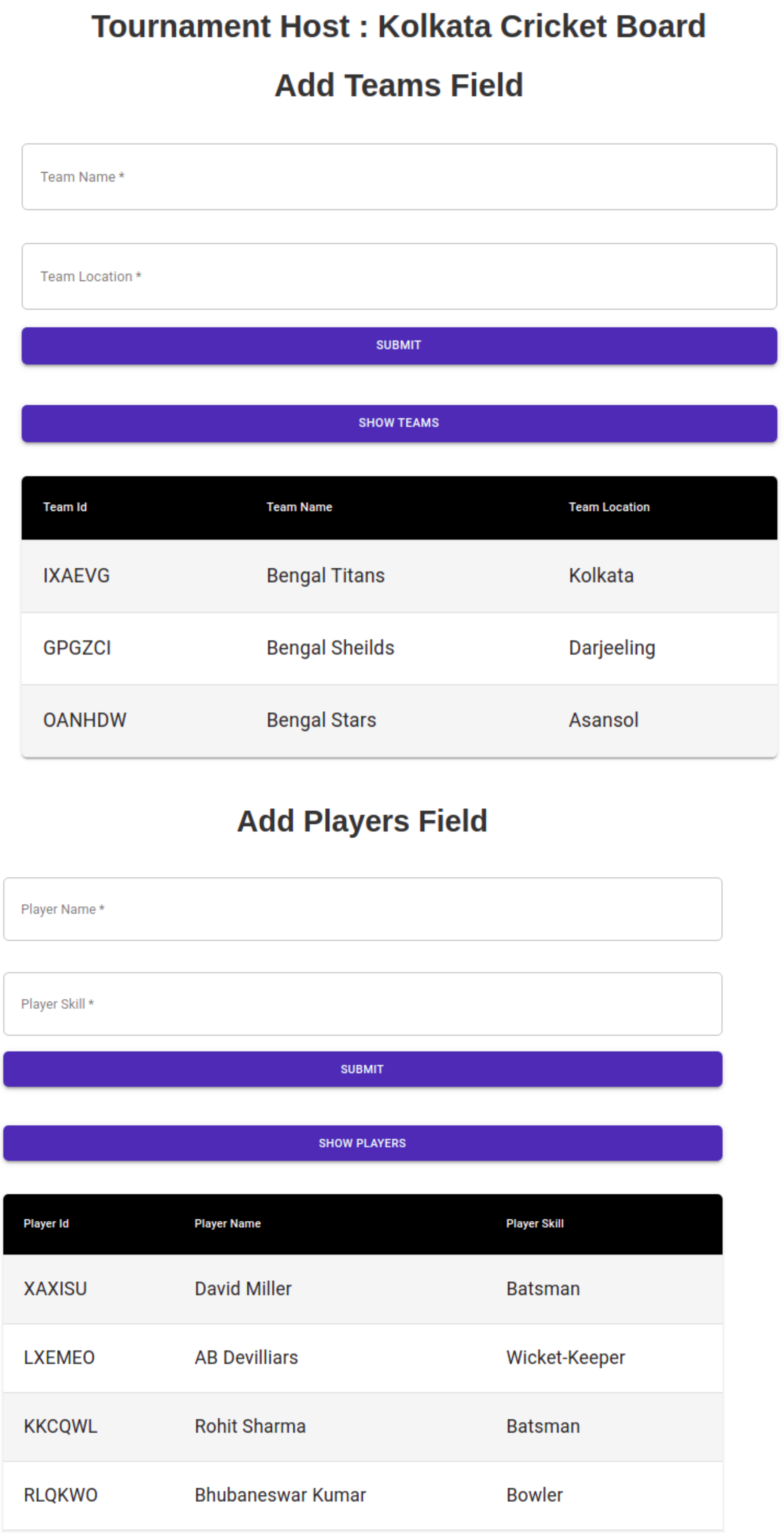
For sign in we are using GET request to get the data from database that is handled by djangoview. After getting the request we are matching usermail and password given by user and if it is matched then we are use the org_id as token and forward the user in the team and played add page.



```
handleButtonPressed() {
  let url = "http://127.0.0.1:8000/api/tadmin/";
  let userid = [];
  getapi(url, this.state.getmail, this.state.getpassword, userid);
  sleep(100).then(() => {
    if(value==false){
      this.setState({
          text : "*Either password is incorrect or user is not registered!!",
      });
    }
    else{
      this.setState({
        text : "",
      });
      this.props.history.push("/add/" + userid[0]);
    }
  })
}
```

The token is handled by the the code boxed in red rectangle.

iv. After login organiser can add teams and players in its tournament. There are separate database of players and teams that is used to store data of players and teams.



Tournament Host : Kolkata Cricket Board

Add Teams Field

Team Id	Team Name	Team Location
IXAEVG	Bengal Titans	Kolkata
GPGZCI	Bengal Sheilds	Darjeeling
OANHDW	Bengal Stars	Asansol

Add Players Field

Player Id	Player Name	Player Skill
XAXISU	David Miller	Batsman
LXEMEO	AB Devilliars	Wicket-Keeper
KKCQWL	Rohit Sharma	Batsman
RLQKWO	Bhubaneswar Kumar	Bowler

5. Auction functionality :

After adding all teams and players organiser can visit the auction page directly. All the registered teams need to login to auction page to start the auction. The teams are also separated in login using token that is team id.

In the auction page organiser can start the auction, sold the player after bidding is done, bring next player and see ongoing and completed players' details. Teams can bid for a player and see the ongoing and completed players' details.

The functionality is shown using following diagram

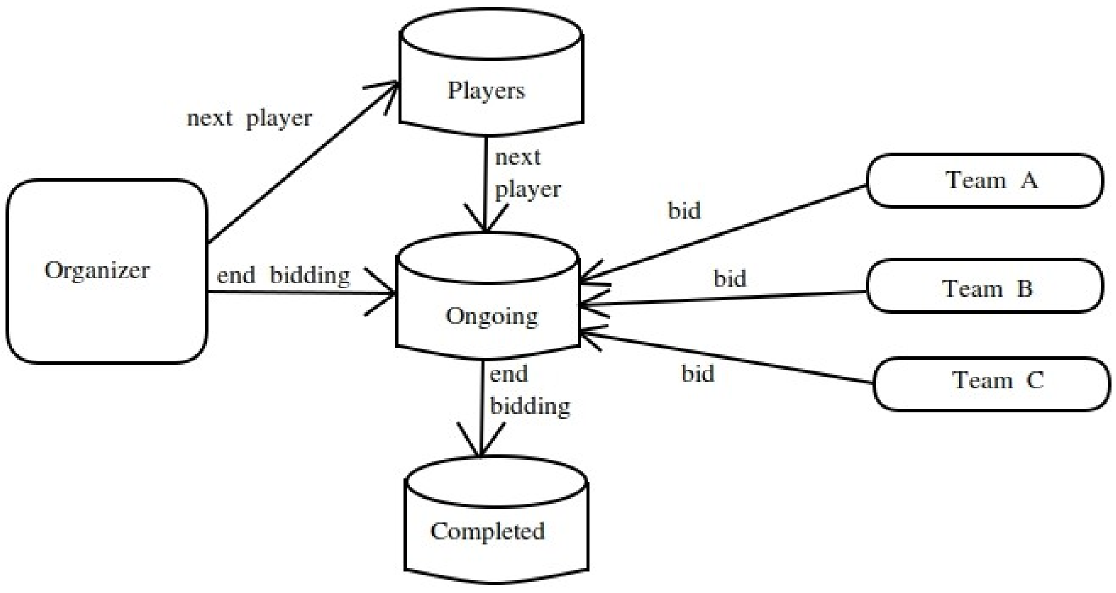
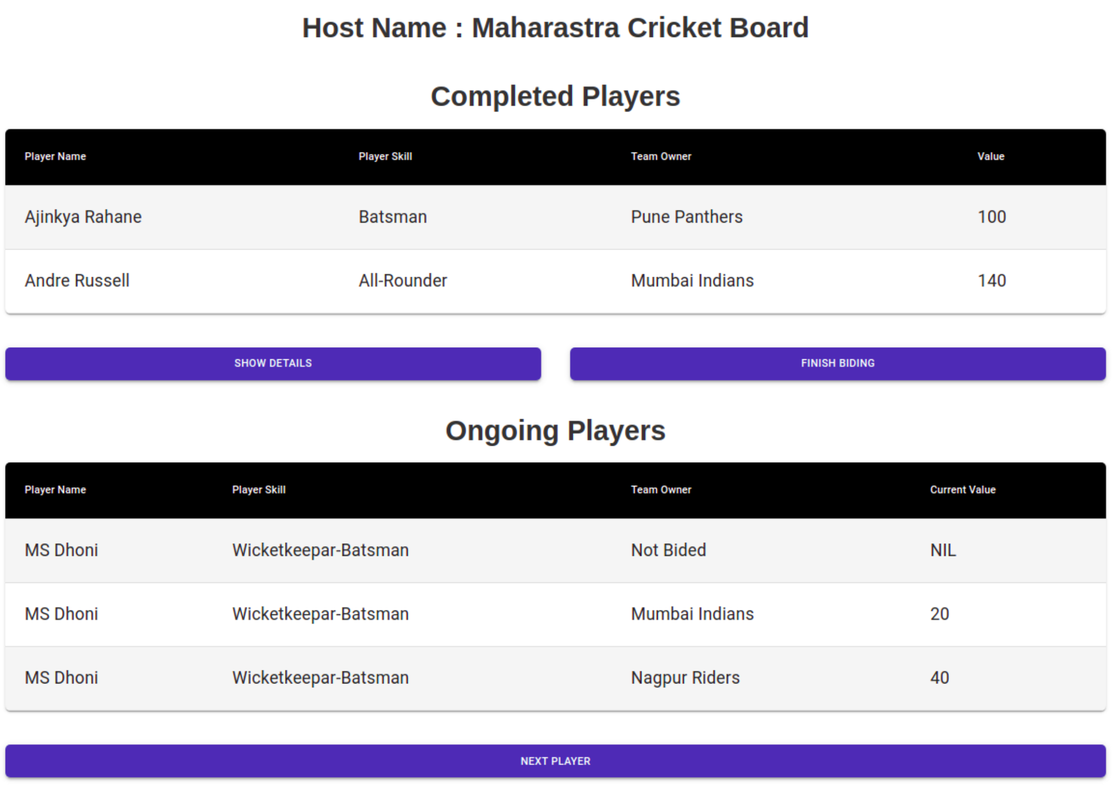


FIG : Auction functionality

In organizer page the auction page will look as follows



Host Name : Maharastra Cricket Board

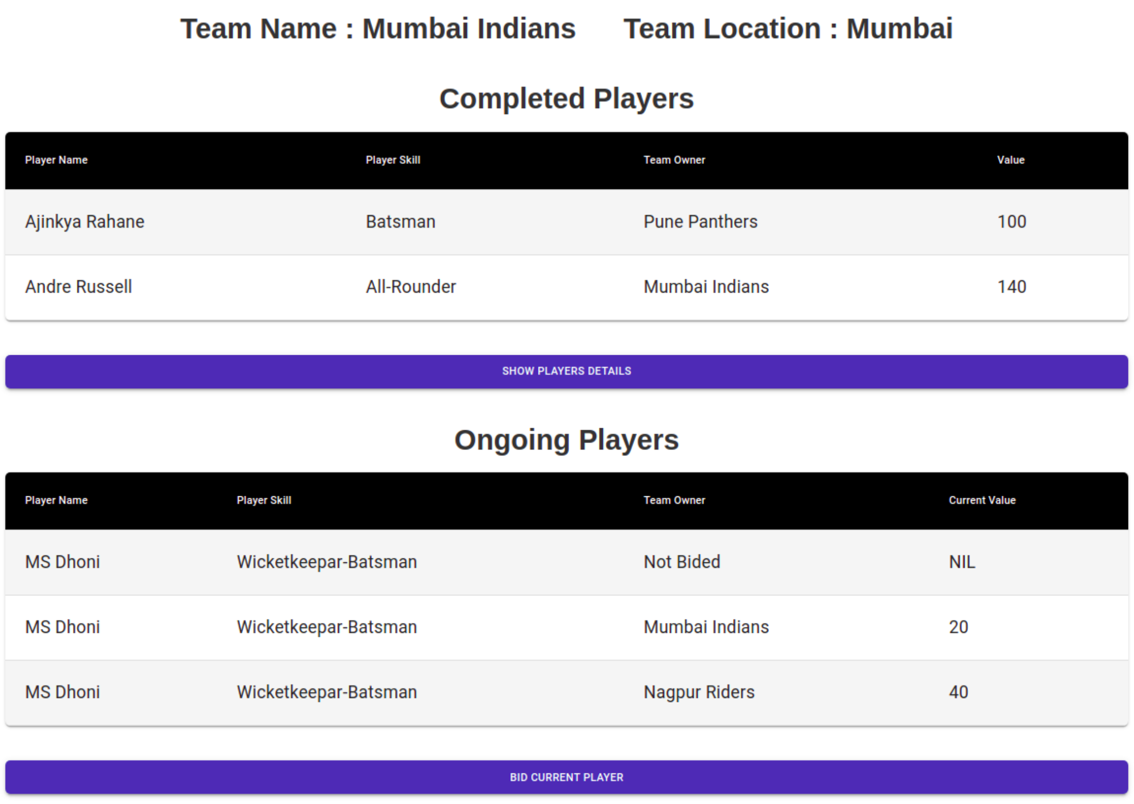
Completed Players

Player Name	Player Skill	Team Owner	Value
Ajinkya Rahane	Batsman	Pune Panthers	100
Andre Russell	All-Rounder	Mumbai Indians	140

Ongoing Players

Player Name	Player Skill	Team Owner	Current Value
MS Dhoni	Wicketkeeper-Batsman	Not Bided	NIL
MS Dhoni	Wicketkeeper-Batsman	Mumbai Indians	20
MS Dhoni	Wicketkeeper-Batsman	Nagpur Riders	40

In teams page the auction page will look as follows



Team Name : Mumbai Indians   Team Location : Mumbai

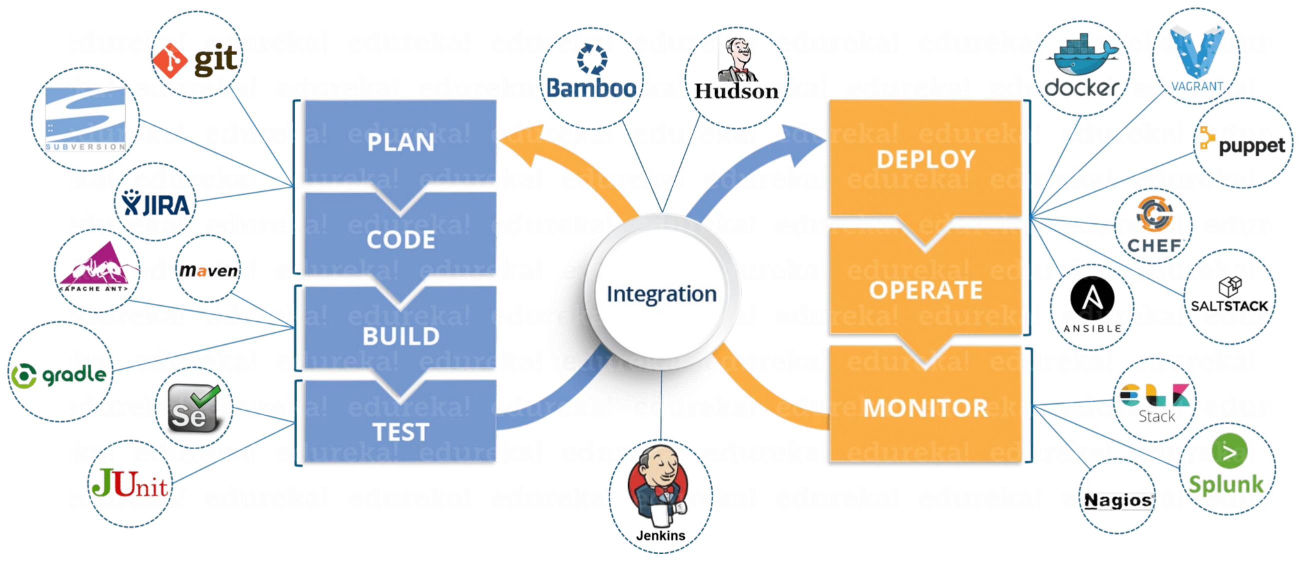
Completed Players

Player Name	Player Skill	Team Owner	Value
Ajinkya Rahane	Batsman	Pune Panthers	100
Andre Russell	All-Rounder	Mumbai Indians	140

Ongoing Players

Player Name	Player Skill	Team Owner	Current Value
MS Dhoni	Wicketkeeper-Batsman	Not Bided	NIL
MS Dhoni	Wicketkeeper-Batsman	Mumbai Indians	20
MS Dhoni	Wicketkeeper-Batsman	Nagpur Riders	40

II. Ops Walkthrough :



DevOps Pipeline :

The project includes automating the integration and deployment chain which includes source control management, continuous integration and continuous deployment. The cycle would include pushing latest changes of codes to git (GitHub), pushing build docker image on to docker hub, build project from docker image, automate all functionality (Jenkins) and continuous monitoring (ELK Stack). Tools used includes :

1. SCM : Git and Github

(https://github.com/SubhodeepSahoo/auction_web)

2. Building Docker Image

(https://hub.docker.com/repository/docker/sahoosubhodeep/auction_web)

3. Continuous Integration : Jenkins

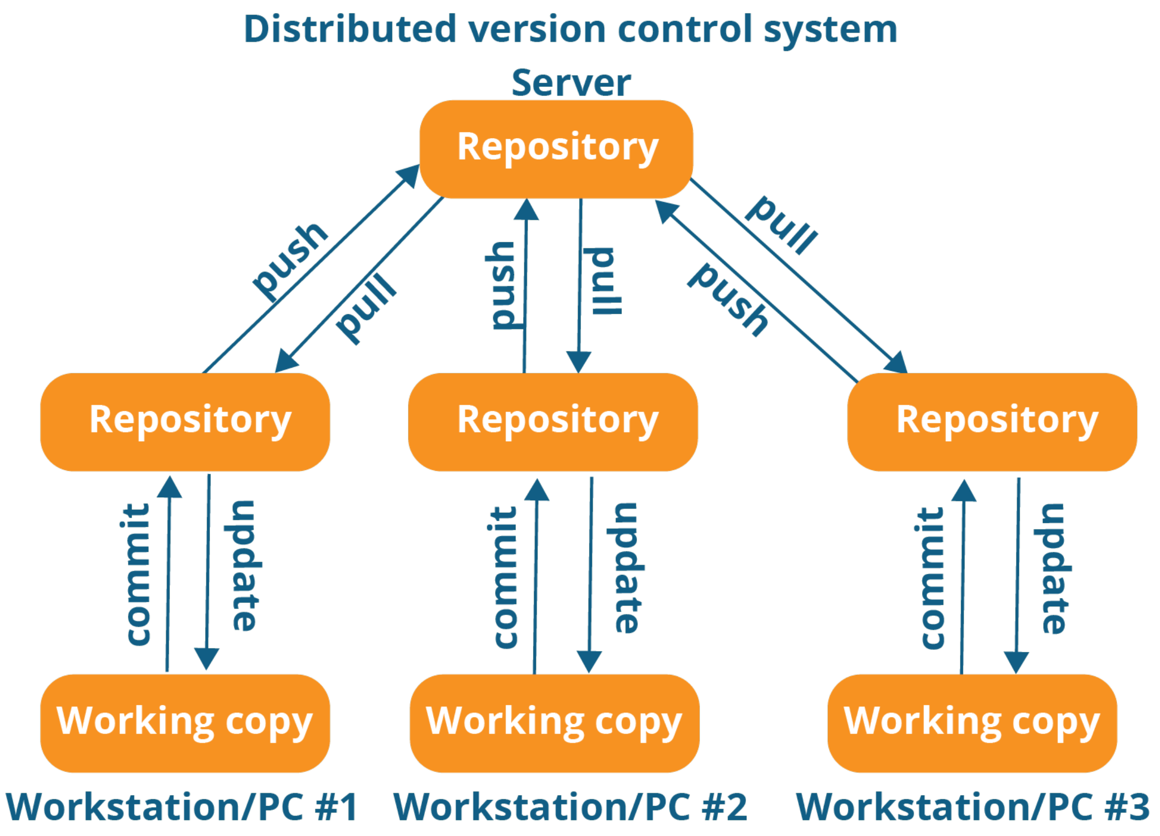
4. Building Project from Docker Image

5. Continuous Deployment : Ansible

6. Continuous Monitoring : ELK Stack

1. Source Control Management :

We are using Git and Github as distributed version control system to mantain the code and manage the project. The architectecture looks as follows



The steps are as belows to maintain Git and Github :

1. Create a new repository on to https://www.github.com. This includes adding repository name and description. The repository name should be unique to the signed in user. I made a repository named calculator_devops.git.

2. Now we will maintain a local repository to maintain code of the project. There are following steps to maintain git repository–

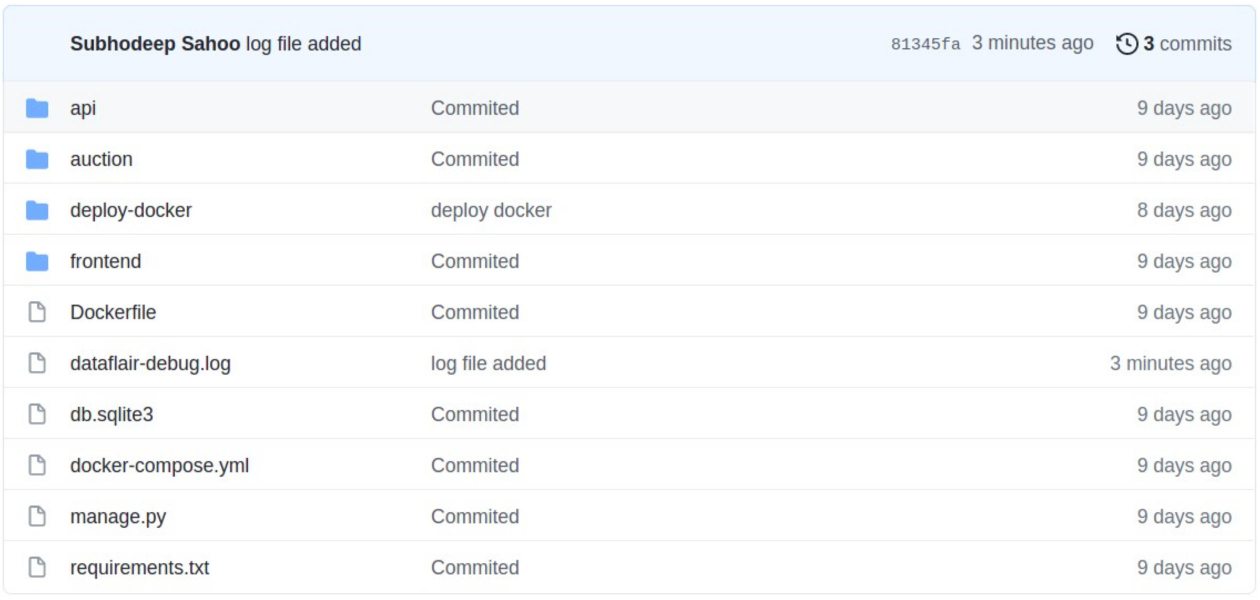
i. git init ./ (To initialize current directory as a repository)

ii. git add <changed_files_path> (To add file in the repository)

iii. git commit -m “Commit message name” (To commit changes)

iv. git remote add origin <git_repo_url> (To get access of github repository)

v. git push origin <branch_name> (Push a branch to your remote repository)



Subhodeep Sahoo log file added 81345fa 3 minutes ago 3 commits

api	Commited	9 days ago
auction	Commited	9 days ago
deploy-docker	deploy docker	8 days ago
frontend	Commited	9 days ago
Dockerfile	Commited	9 days ago
dataflair-debug.log	log file added	3 minutes ago
db.sqlite3	Commited	9 days ago
docker-compose.yml	Commited	9 days ago
manage.py	Commited	9 days ago
requirements.txt	Commited	9 days ago

2. Building Docker Image :

Docker is an open platform for developing, shipping, and running applications. Docker enables us to separate our applications from our infrastructure so we can deliver software quickly. With Docker, we can manage our infrastructure in the same ways we manage our applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, we can significantly reduce the delay between writing code and running it in production.

We are using docker to create a docker image of jar file created after build process and push the latest image on to docker hub. The pushed procedure would be done after every build which would include removing preciously pushed docker image and replace with latest built one.

Following are the commands for seting up docker :

1. To the group of docker which can be done using

   $ sudo usermod –aG docker <username>

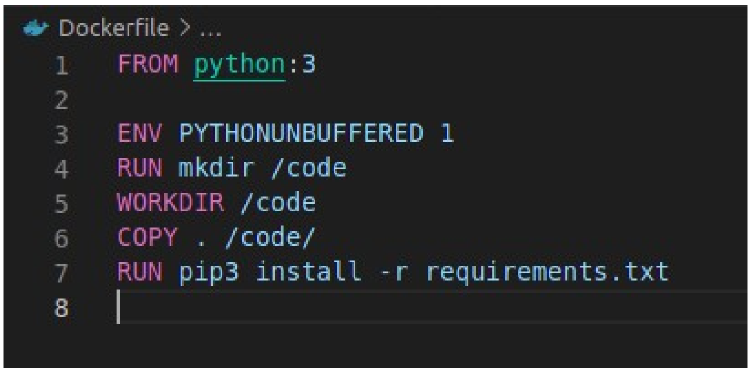
2. Run the docker on to terminal via command

   $ sudo systemctl start docker

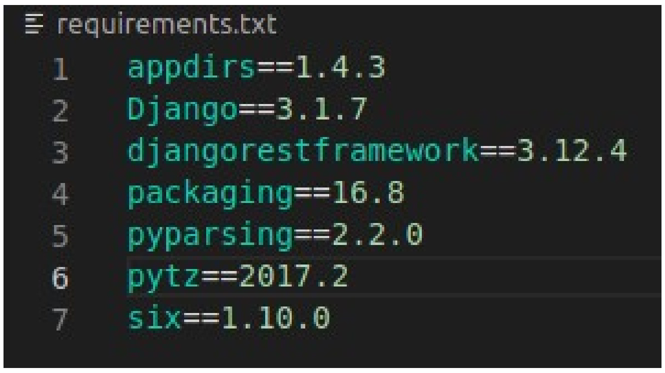
3. Now we will create a Dockerfile in the auction directory. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession. In this dockerfile we are installing all the required environments using the following command

   RUN pip3 install -r requirements.txt

   Here requirement.txt file has all the packages with versions that needs to run the project.



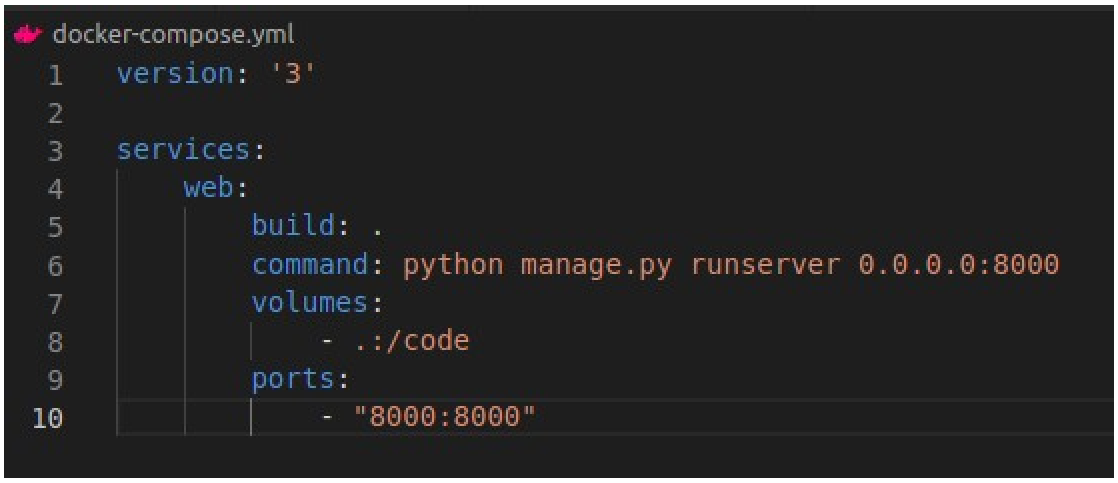
```
Dockerfile > ...
FROM python:3

ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
COPY . /code/
RUN pip3 install -r requirements.txt
```



```
requirements.txt
appdirs==1.4.3
Django==3.1.7
djangorestframework==3.12.4
packaging==16.8
pyparsing==2.2.0
pytz==2017.2
six==1.10.0
```

4. Now we will create a docker-compose file that helps us to define and run multi-container Docker applications. In this file we first define the version of docker which we are using. Then we will create the services that our container can perform. In services we define the dockerfile path in build and then write

the command that needs to run our project. Finally we write the port number in which the server will run.



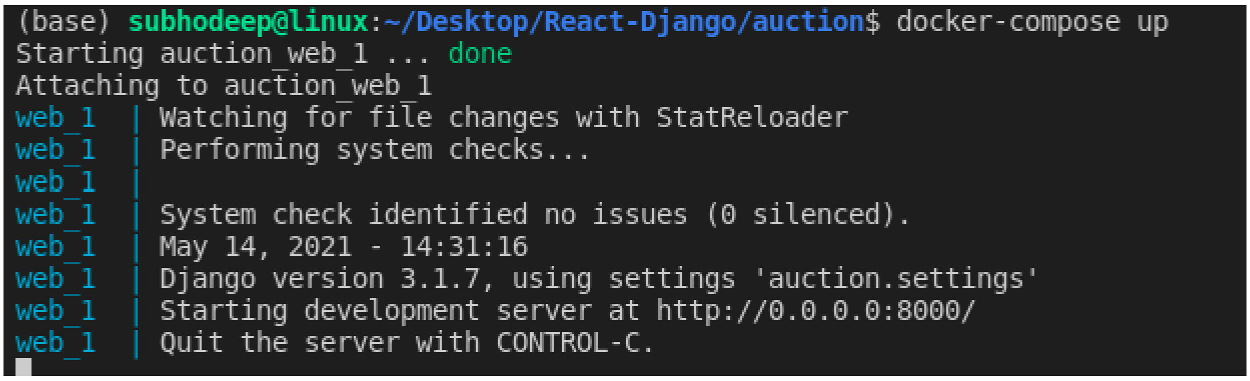
```
docker-compose.yml
version: '3'

services:
    web:
        build: .
        command: python manage.py runserver 0.0.0.0:8000
        volumes:
            - .:/code
        ports:
            - "8000:8000"
```

5. Now we will build our build our docker image using

   $ docker-compose build

6. To run the docker image we will use the following command

   $ docker-compose up



```
(base) subhodeep@linux:~/Desktop/React-Django/auction$ docker-compose up
Starting auction_web_1 ... done
Attaching to auction_web_1
web_1  | Watching for file changes with StatReloader
web_1  | Performing system checks...
web_1  |
web_1  | System check identified no issues (0 silenced).
web_1  | May 14, 2021 - 14:31:16
web_1  | Django version 3.1.7, using settings 'auction.settings'
web_1  | Starting development server at http://0.0.0.0:8000/
web_1  | Quit the server with CONTROL-C.
```

7. To see all docker images we will use

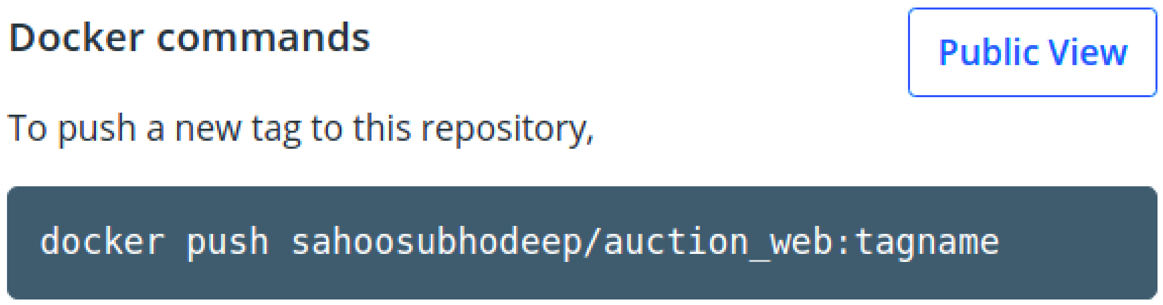
   $ docker images

8. To pull the docker image use command

   $ docker pull <docker_image_name:tag>

9. To push the docker image use command

   $ docker push <username>/<repository_name>:tagname



Docker commands

Public View

To push a new tag to this repository,

```
docker push sahoosubhodeep/auction_web:tagname
```

3. Continuous Integration :

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed. The following are the steps to do so –

1. Make sure to add Jenkins to docker group, so that Jenkins can use docker for build docker image. Following command is used to do so –

   $ sudo usermod –aG docker Jenkins

2. We can verify using –

   $ sudo grep jenkins /etc/gshadow

3. We can now start Jenkins with, by command

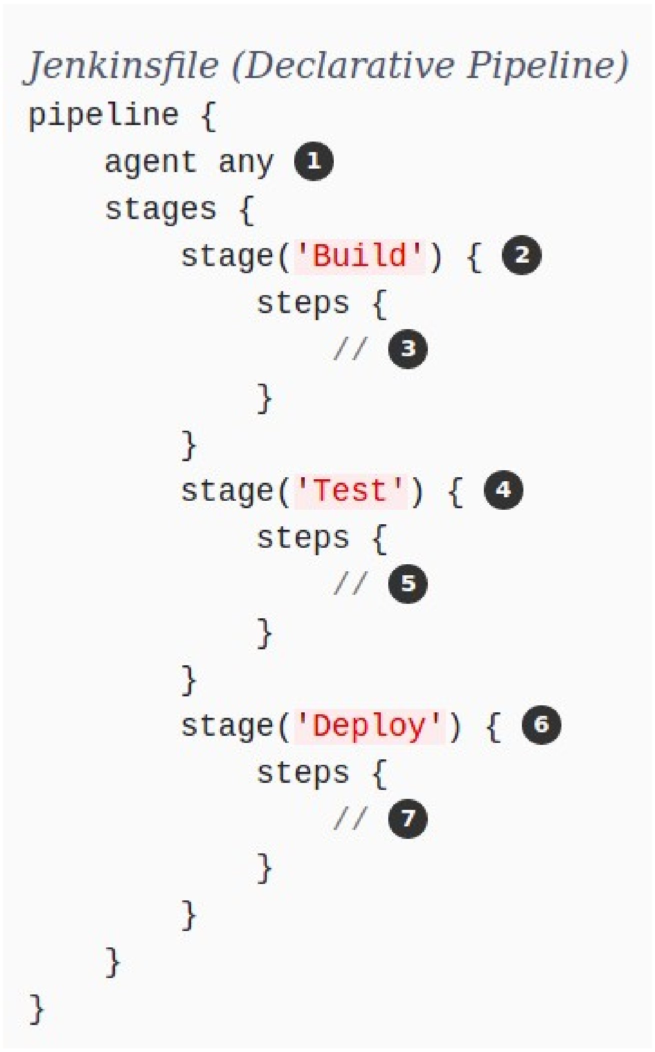
   $ sudo systemctl start jenkins

4. Now we can login to jenkins in our browser with port 8080,

   http://localhost:8080

Jenkins Pipeline :

1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

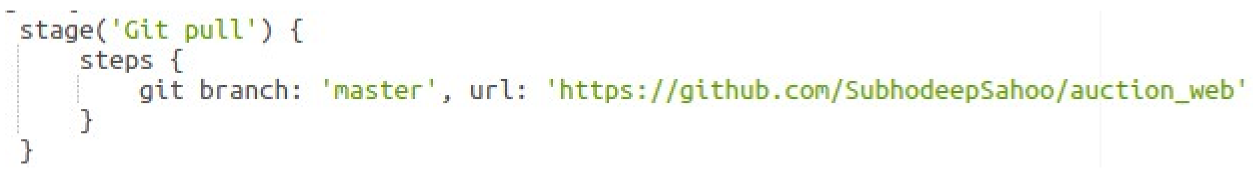


Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any ❶
    stages {
        stage('Build') { ❷
            steps {
                // ❸
            }
        }
        stage('Test') { ❹
            steps {
                // ❺
            }
        }
        stage('Deploy') { ❻
            steps {
                // ❼
            }
        }
    }
}
```

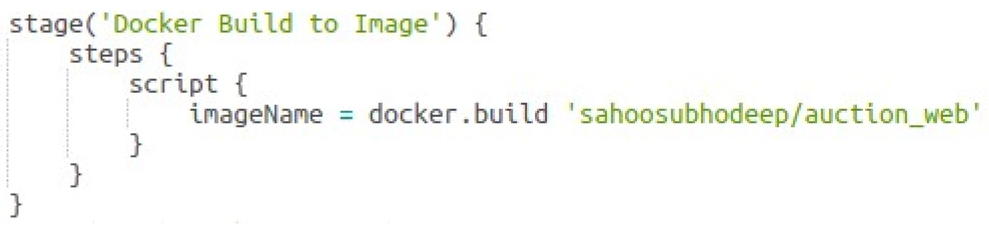
Creating Jenkins Pipeline for our project :

1. In first stage we will pull our project from github



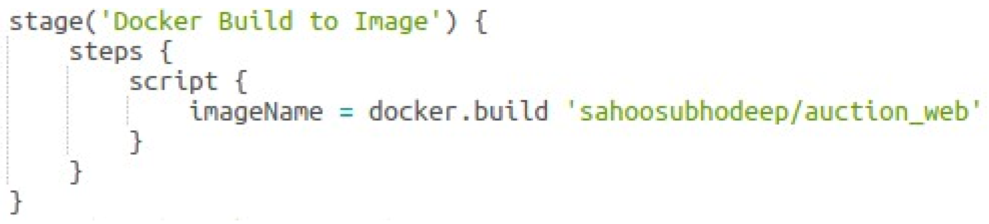
```
stage('Git pull') {
    steps {
        git branch: 'master', url: 'https://github.com/SubhodeepSahoo/auction_web'
    }
}
```

2. Then we will build our dockerimage that we created



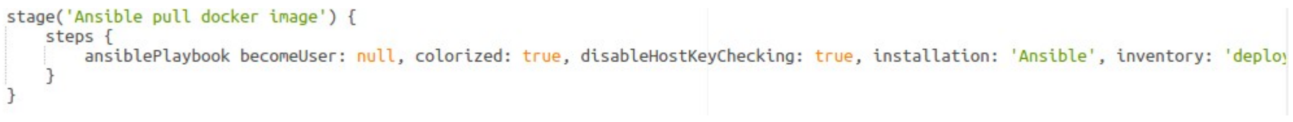
```
stage('Docker Build to Image') {
    steps {
        script {
            imageName = docker.build 'sahoosubhodeep/auction_web'
        }
    }
}
```

3. Next wepush our docker image



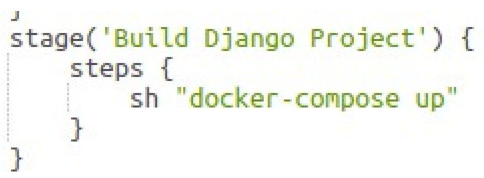
```
stage('Docker Build to Image') {
    steps {
        script {
            imageName = docker.build 'sahoosubhodeep/auction_web'
        }
    }
}
```

4. Now we will deploy our docker image using ansible



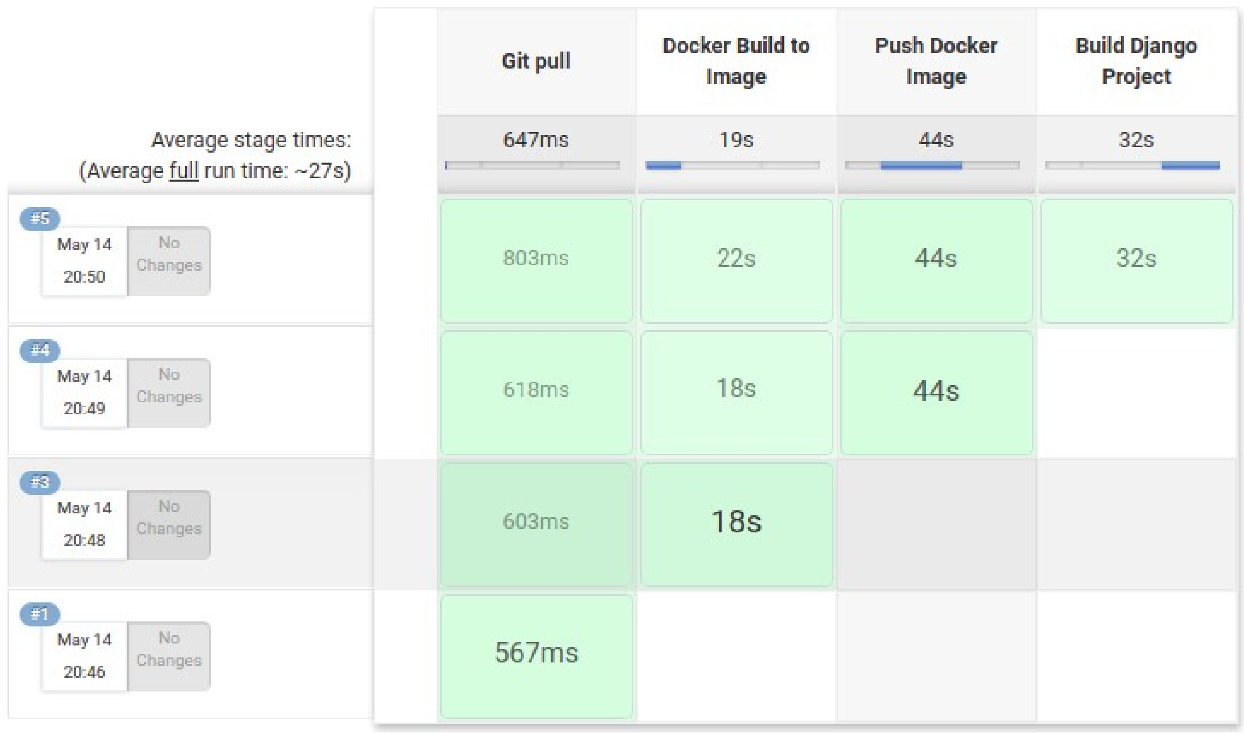
```
stage('Ansible pull docker image') {
    steps {
        ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'deploy
    }
}
```

5. Finally we run our docker image to run the project



```
stage('Build Django Project') {
    steps {
        sh "docker-compose up"
    }
}
```

Stage View :

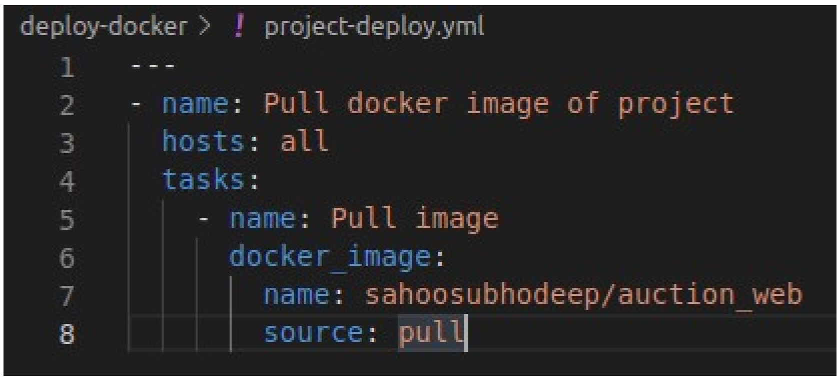


	Git pull	Docker Build to Image	Push Docker Image	Build Django Project
Average stage times: (Average full run time: ~27s)	647ms	19s	44s	32s
#5 May 14 20:50 No Changes	803ms	22s	44s	32s
#4 May 14 20:49 No Changes	618ms	18s	44s	
#3 May 14 20:48 No Changes	603ms	18s		
#1 May 14 20:46 No Changes	567ms			

5. Continuous Deployment :

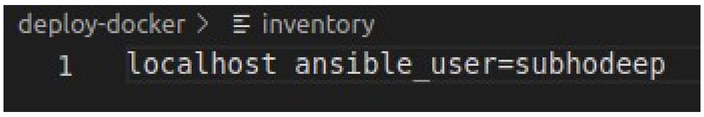
Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates. In this step we install Ansible on the controller node and deploy our docker image in all our hosts. In this step we create a directory named deploy-docker in which we first create a file named project-deploy.yml which contains the contains the instructions required to deploy the docker image to our hosts.

Our project-deploy.yml will have the following code



```
deploy-docker > ! project-deploy.yml
---
- name: Pull docker image of project
  hosts: all
  tasks:
    - name: Pull image
      docker_image:
        name: sahoosubhodeep/auction_web
        source: pull
```

After that in the same directory we add our inventory file which contains the details of our host i.e. there name and ip address

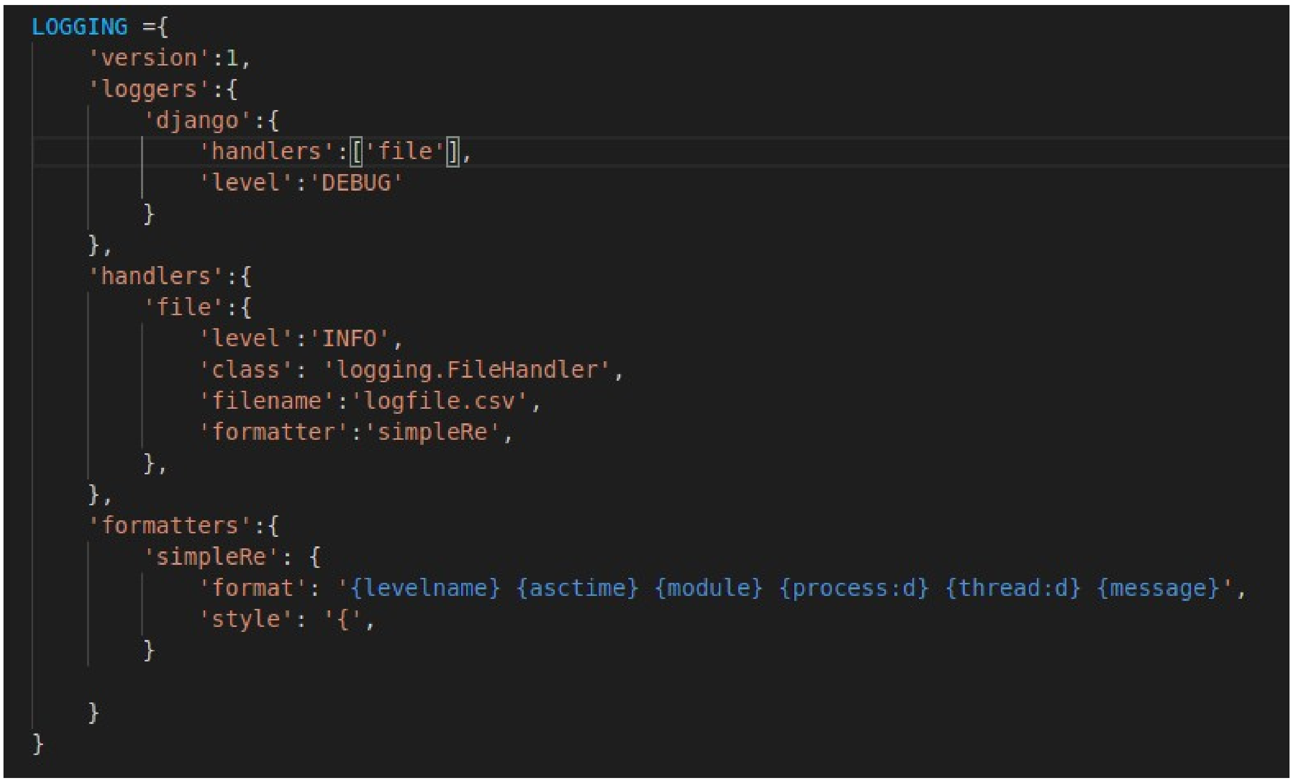


```
deploy-docker > inventory
localhost ansible_user=subhodeep
```

6. Continuous Monitoring :

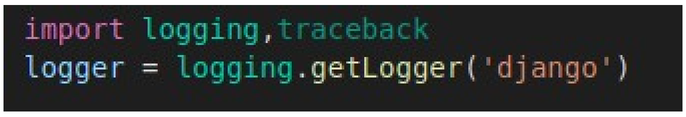
"ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch.

To generate log file of our project we add following code in setting.py

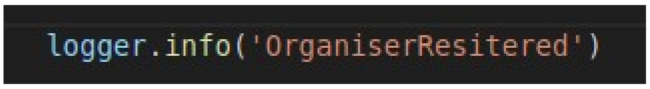


```python
LOGGING ={
    'version':1,
    'loggers':{
        'django':{
            'handlers':['file'],
            'level':'DEBUG'
        }
    },
    'handlers':{
        'file':{
            'level':'INFO',
            'class': 'logging.FileHandler',
            'filename':'logfile.csv',
            'formatter':'simpleRe',
        },
    },
    'formatters':{
        'simpleRe': {
            'format': '{levelname} {asctime} {module} {process:d} {thread:d} {message}',
            'style': '{',
        }

    }
}
```

And also we add some log info in views.py file to

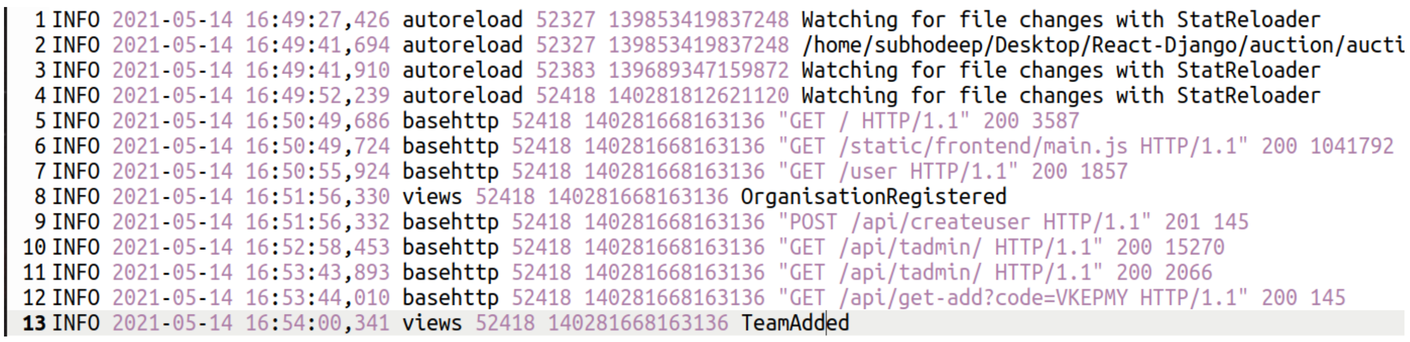


```python
import logging,traceback
logger = logging.getLogger('django')
```



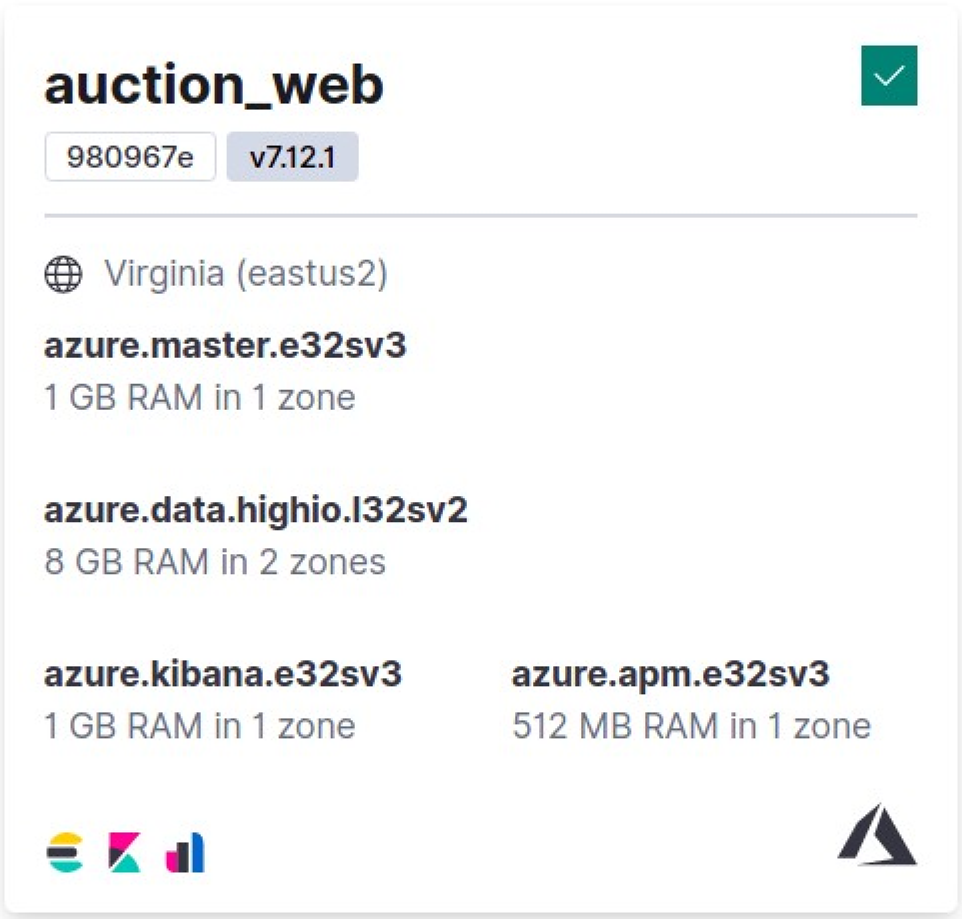
```python
logger.info('OrganiserResitered')
```

Our logfile is saved in logfile.csv



```
INFO 2021-05-14 16:49:27,426 autoreload 52327 139853419837248 Watching for file changes with StatReloader
INFO 2021-05-14 16:49:41,694 autoreload 52327 139853419837248 /home/subhodeep/Desktop/React-Django/auction/aucti
INFO 2021-05-14 16:49:41,910 autoreload 52383 139689347159872 Watching for file changes with StatReloader
INFO 2021-05-14 16:49:52,239 autoreload 52418 140281812621120 Watching for file changes with StatReloader
INFO 2021-05-14 16:50:49,686 basehttp 52418 140281668163136 "GET / HTTP/1.1" 200 3587
INFO 2021-05-14 16:50:49,724 basehttp 52418 140281668163136 "GET /static/frontend/main.js HTTP/1.1" 200 1041792
INFO 2021-05-14 16:50:55,924 basehttp 52418 140281668163136 "GET /user HTTP/1.1" 200 1857
INFO 2021-05-14 16:51:56,330 views 52418 140281668163136 OrganisationRegistered
INFO 2021-05-14 16:51:56,332 basehttp 52418 140281668163136 "POST /api/createuser HTTP/1.1" 201 145
INFO 2021-05-14 16:52:58,453 basehttp 52418 140281668163136 "GET /api/tadmin/ HTTP/1.1" 200 15270
INFO 2021-05-14 16:53:43,893 basehttp 52418 140281668163136 "GET /api/tadmin/ HTTP/1.1" 200 2066
INFO 2021-05-14 16:53:44,010 basehttp 52418 140281668163136 "GET /api/get-add?code=VKEPMY HTTP/1.1" 200 145
INFO 2021-05-14 16:54:00,341 views 52418 140281668163136 TeamAdded
```

Now we create a deployment named ‘auction_web’ in elastic search cloud.



auction_web

980967e v7.12.1

Virginia (eastus2)

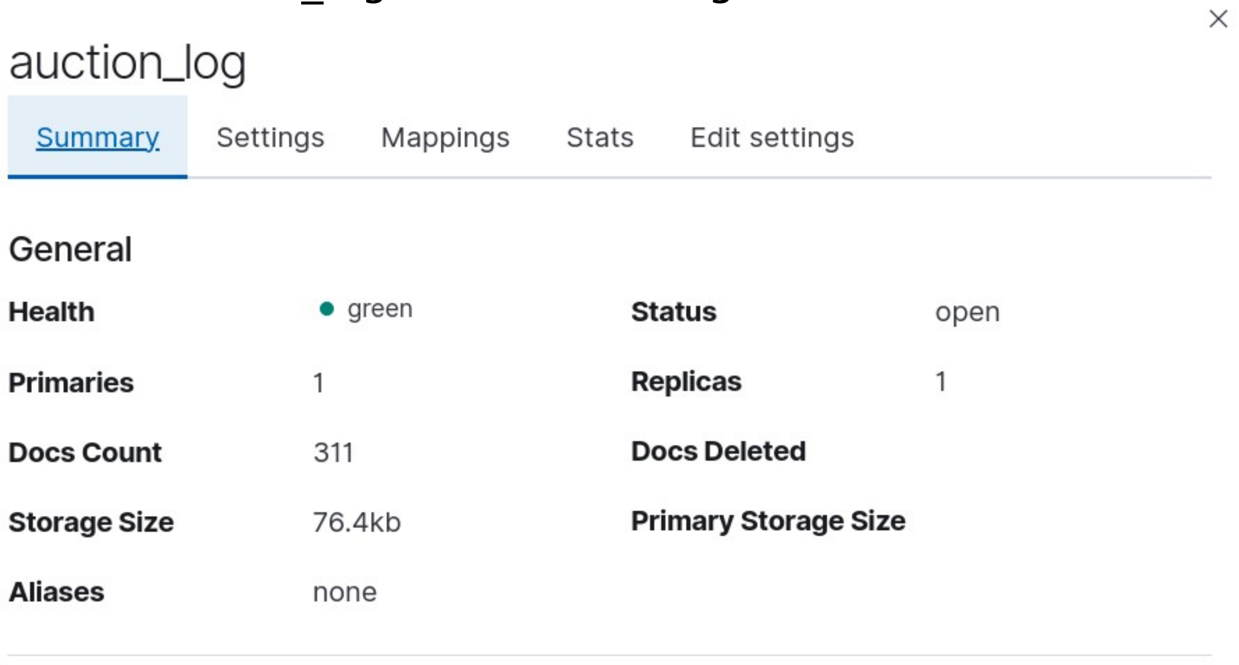
azure.master.e32sv3
1 GB RAM in 1 zone

azure.data.highio.l32sv2
8 GB RAM in 2 zones

azure.kibana.e32sv3
1 GB RAM in 1 zone

azure.apm.e32sv3
512 MB RAM in 1 zone

We then upload the logfile.csv and generate an index named ‘auction_log’ to visualize log data.

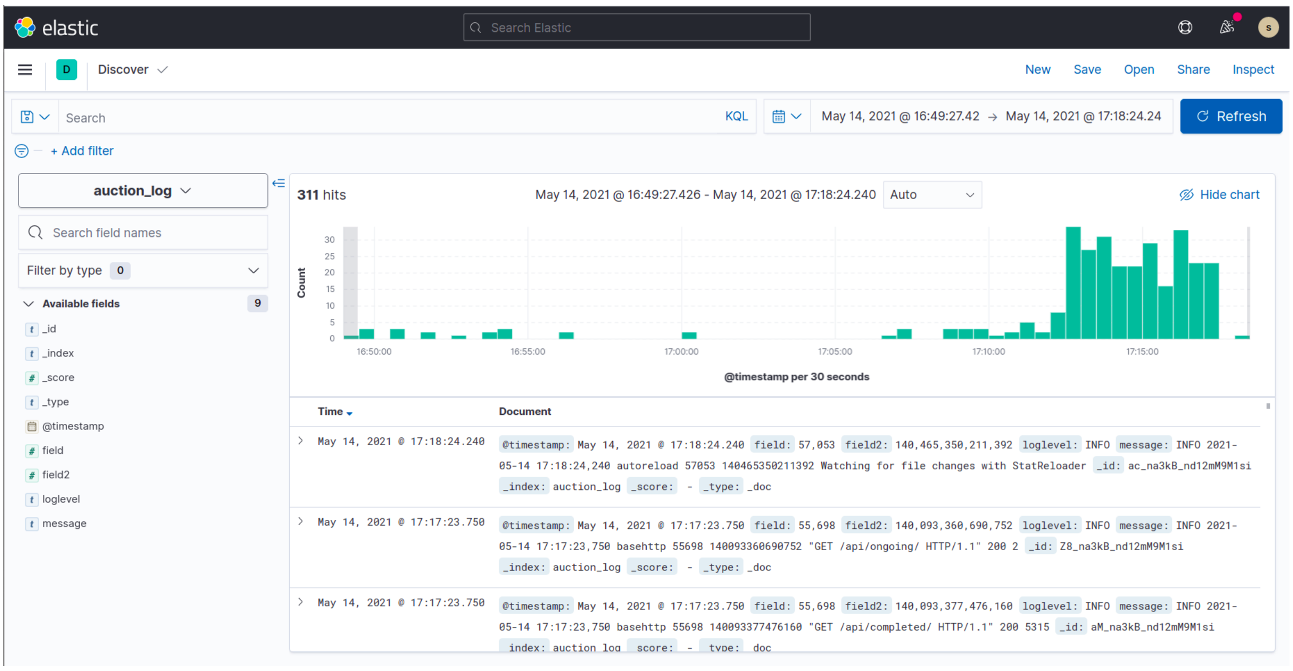


auction_log

General

Health	● green	**Status**	open
Primaries	1	**Replicas**	1
Docs Count	311	**Docs Deleted**	
Storage Size	76.4kb	**Primary Storage Size**	
Aliases	none		

Our visualiztion in kibana as follows



311 hits

May 14, 2021 @ 16:49:27.426 - May 14, 2021 @ 17:18:24.240 Auto

```
Time      Document
May 14, 2021 @ 17:18:24.240  @timestamp: May 14, 2021 @ 17:18:24.240 field: 57,053 field2: 140,465,350,211,392 loglevel: INFO message: INFO 2021-
05-14 17:18:24,240 autoreload 57053 140465350211392 Watching for file changes with StatReloader _id: ac_na3kB_nd12mM9M1si
_index: auction_log _score: - _type: _doc

May 14, 2021 @ 17:17:23.750  @timestamp: May 14, 2021 @ 17:17:23.750 field: 55,698 field2: 140,093,360,690,752 loglevel: INFO message: INFO 2021-
05-14 17:17:23,750 basehttp 55698 140093360690752 "GET /api/ongoing/ HTTP/1.1" 200 2 _id: Z8_na3kB_nd12mM9M1si
_index: auction_log _score: - _type: _doc

May 14, 2021 @ 17:17:23.750  @timestamp: May 14, 2021 @ 17:17:23.750 field: 55,698 field2: 140,093,377,476,160 loglevel: INFO message: INFO 2021-
05-14 17:17:23,750 basehttp 55698 140093377476160 "GET /api/completed/ HTTP/1.1" 200 5315 _id: aM_na3kB_nd12mM9M1si
_index: auction_log _score: - _type: _doc
```

Chalenges Faced :

1. **Integration Django and React :**

   It takes several days to install react packages to in django framework basically in ‘frontend’ django app. There is very less tutorials that discussed about connecting react app in same port that django uses i.e. ‘8080’.

2. **Registration Issue :**

   We are registering for many users without logging out the session in our browser. It takes several hours to figure out how to release the session before registering for another user.

3. **Login Issue :**

   We had to login in for several teams at same time to perform parallel auction process. But we could’t login for multiple teams at same time as it gives security error for trying to get several session at a time. So, we disabled cookies to handle that problem.

4. **Differentiate Multiple Teams :**

   We had a big challange to show all the teams only related data and to find a way so that they can use all functionality seperately. For that we use team_id as tag in url index to separate all teams.

5. **Genrating Proper Log File :**

   We had a challenge to generate a good log file so that elasticsearch cloud can read and visualize data properly. We had changes the Logging dictionary in settings.py several times to generate suitable log file.

6. Ansible Setup :

We faced issue to setup ansible in Jenkins pipeline. We got the error shown below and unable to find any solution for the issue.

```
TASK [Pull image] ****************************************************************
¤[0;31mfatal: [localhost]: FAILED! => {"changed": false, "msg": "Failed to import the required Python library (Docker SDK for Python: docker (Python >= 2.7) or docker-py (Python 2.
6)) on linux's Python /usr/bin/python3. Please read module documentation and install in the appropriate location. If the required library is installed, but Ansible is using the wro
ng Python interpreter, please consult the documentation on ansible_python_interpreter, for example via `pip install docker` or `pip install docker-py` (Python 2.6). The error was:
cannot import name '__version__' from 'docker' (unknown location)"}¤[0m
```

References :

1. https://www.djangoproject.com/start/
2. https://reactjs.org/tutorial/tutorial.html
3. https://www.youtube.com/watch?v=JD-age0BPVo
4. https://docs.docker.com/get-started/
5. https://www.youtube.com/watch?v=Y_rh-VeC_j4&t=881s
6. https://www.youtube.com/watch?v=-vVml7cpWzY
7. https://www.jenkins.io/doc/tutorials/
8. https://www.elastic.co/learn