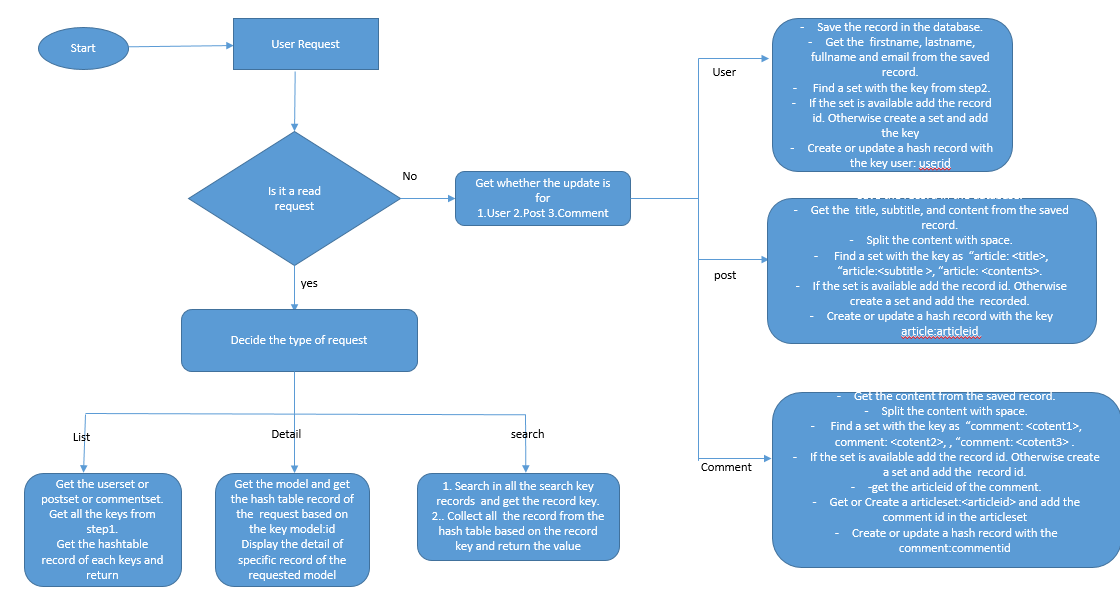
# Flow chart of the search engine



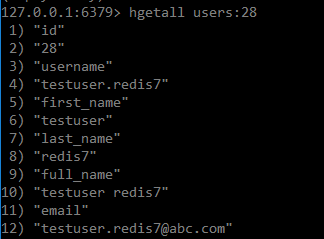
- The Application and the database structure remains the same.

- Along with the existing setup I have added a redis data structure where all the records in redis are created or updated every time a new record is created or updated in the actual database.

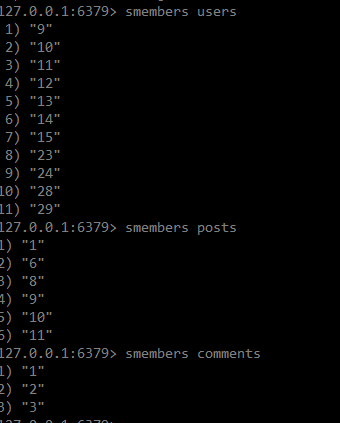
- For each record in the database the relevant record in Redis is stored in a hash table with

key = model:<record id from database> and value = {field1:value1, field2 : value2}

e.g (for user key user: <id> and value = {firstname: fnamevalue, lastname: lnamevalue, email: emailvalue)



- All the records id are stored in a model set with key= modelname and values = set(recordid’s from database)



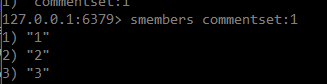
* Along with the above set, the posts created by the user are stored in another set with the key = article\_set:<user\_id> and value = articleID’s. This is to pick all the posts created by each user.

Below screen shot is the set of 2 articles 1 and 6 are stored in the articleset created by user with ID 2.



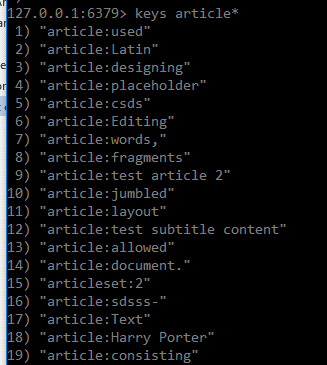
* We also create the comments\_set record. For each of the comments created for a post record we add it into the comments set with
* Key =comments\_set:<post\_id> and value = commentID’s

Below screen shot is the set of 3 comments 1,2 and 3 are stored in the commentset created for article with article ID 1.

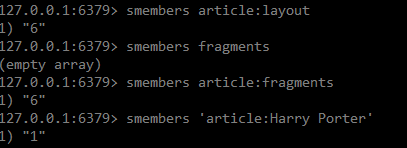


-For each of the field values in all the models except the content fields all the field values stored as key

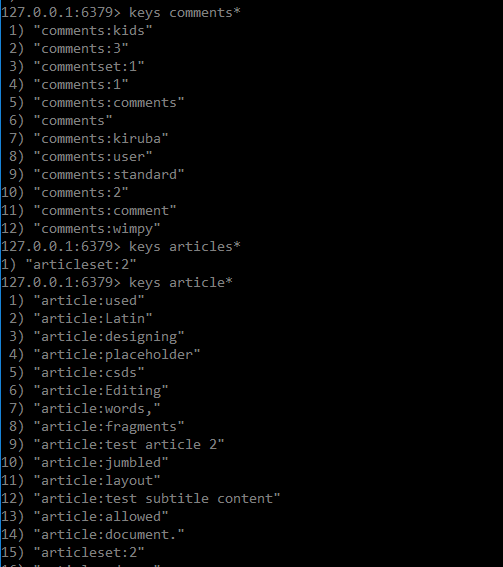
Below are the words from articles



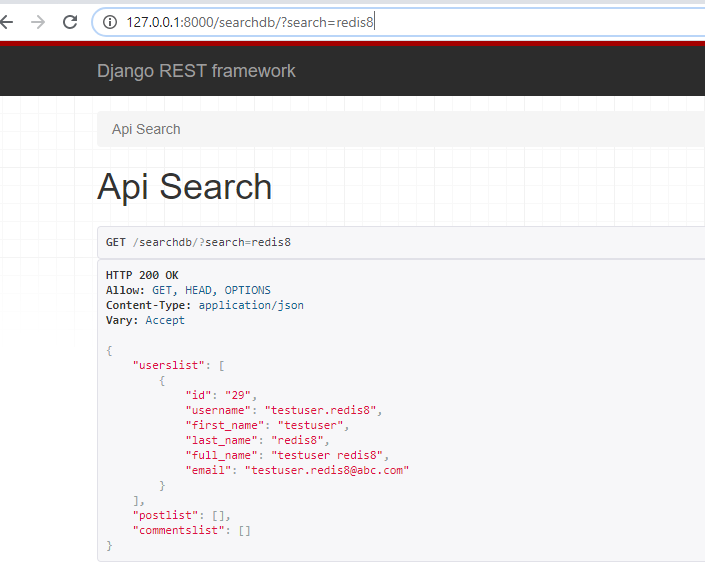
Under each keys the relevant article ID’s are stored.

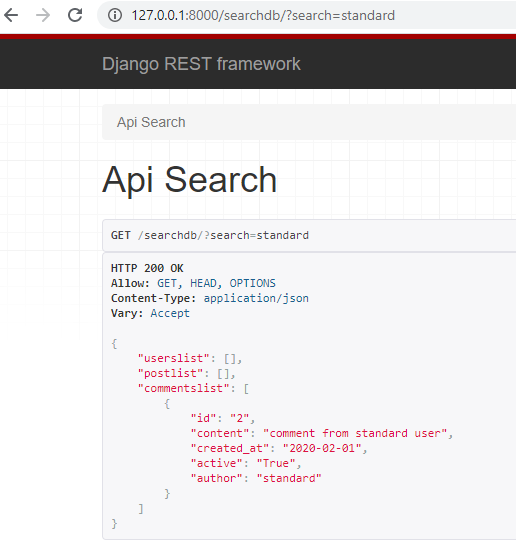


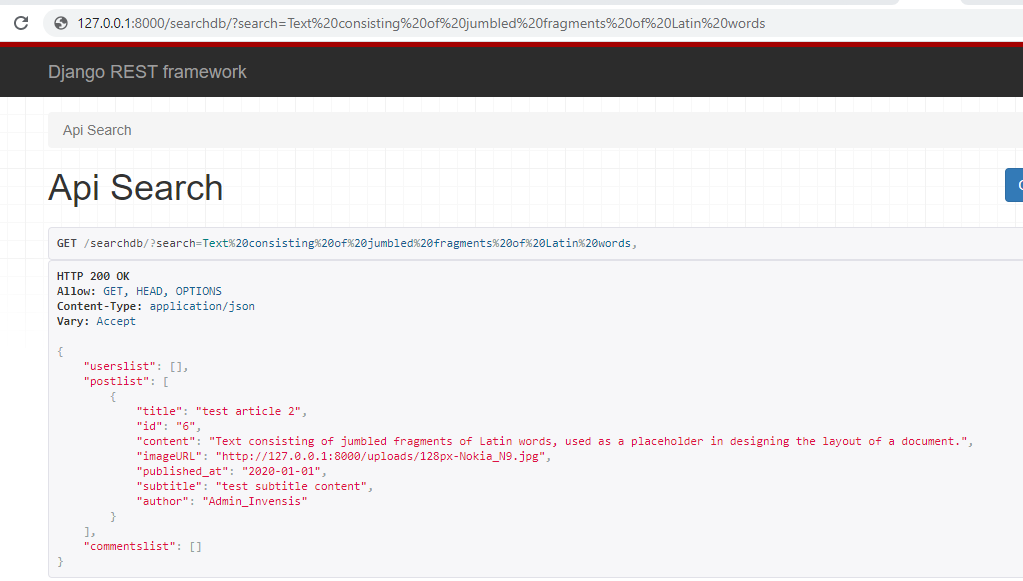
* Finally the content fields are split with space and created a list .
* All stop words are removed
* A new key (model:<contentword>) is created with each content words in the contentlist. And the values will be list of the record ID.



* When do the search the search key is split into different words and the search is done on the users key list, articles key list and comments key list.



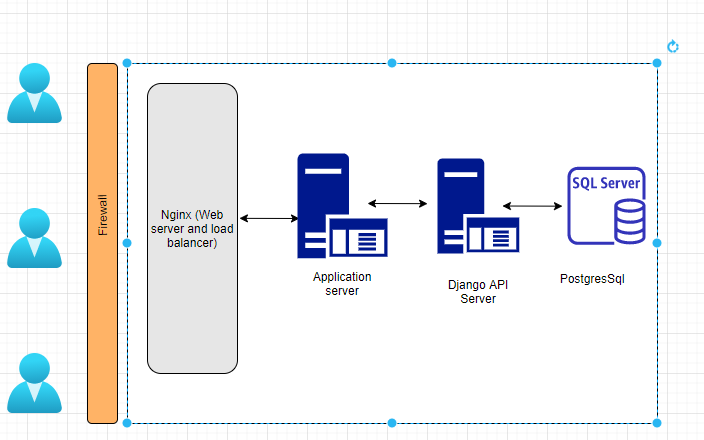




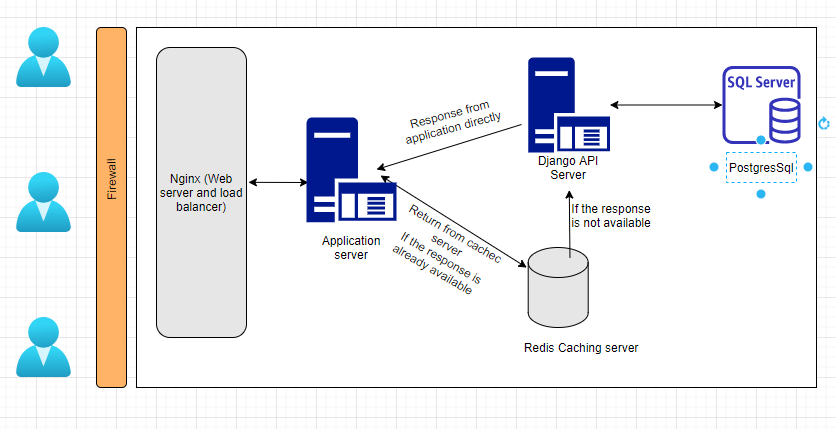
# Application Architecture

## Simple Search Engine Application using Django Rest Framework

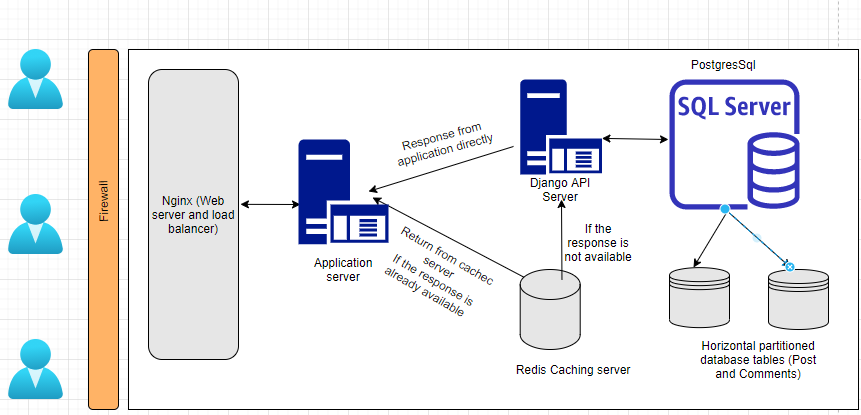
The below scenario can server up to 400000 hits per hour with Red hat server OS, 32 GB RAM and it can be optimized based on the number of core processor we are using.



Request with Caching Mechanism This can improve the performance by 50% minimum as most of the request are served from the cached server. We can set the caching time limit and we can overwrite the record in cache directly whenever save happens in the record.

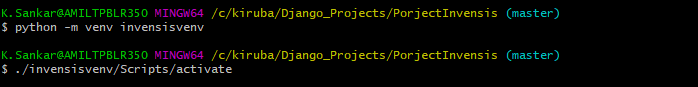


Third step of improvement is through horizontal partitioning the POST and COMMENTS table. This way we can reduce the load in indexing. This is useful when the search happens with large dataset.

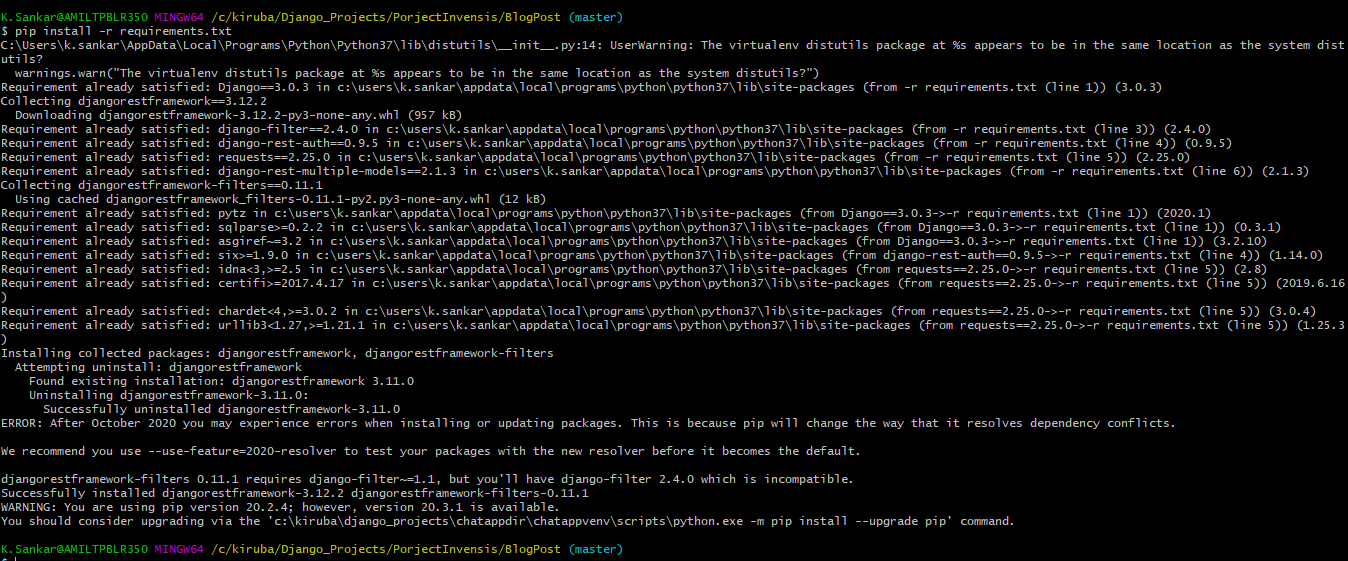


# Application Setup

* To Activate virtual environment
* Create the virtual environment using the command **python –m venv invensisvenv**
* Then type ./invensisvenv/Scripts/activate



* Installed the required packages using the command
* Pip install –r requirements.txt

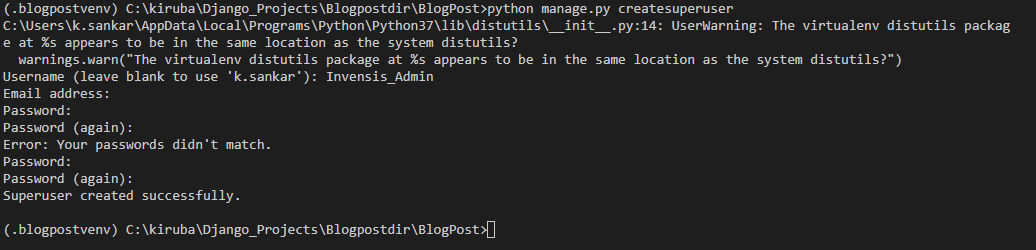


* Do the database setup for development server using the below command

python manage.py migrate

python manage.py createsuperuser

Now type the username and password when prompted



* Start the development server using the below command

python manage.py runserver

* Check whether the development server is running using the url

<http://127.0.0.1:8000/>

* Log in to the portal as admin

<http://127.0.0.1:8000/admin>

username: as provided above

password : as provided above

* Install Redis from <https://redis.io/download> or from docker using **docker pull redis**

Rest of all the users can be created using the API links.

**As session authentication is enabled so all the API’s can be accessed via browser itself.**

These API’s can only be used by Administrator

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Description** | **HTTP Method** | **API syntax** |
| User | List all user | GET/ POST | http://127.0.0.1:8000/users/ |
|  | Get and update specific user | PUT/ DELETE | [http://127.0.0.1:8000/users/<userid>](http://127.0.0.1:8000/users/%3cuserid) |
|  | Get specific user by email ID | GET | [http://127.0.0.1:8000/users/?email=<emailID>](http://127.0.0.1:8000/users/?email=%3cemailID) |
|  | Get specific user by first name | GET | [http://127.0.0.1:8000/users/?first\_name=<first\_name>](http://127.0.0.1:8000/users/?first_name=%3cfirst_name) |
|  | Get specific user by last name | GET | [http://127.0.0.1:8000/users/?last\_name=<last\_name>](http://127.0.0.1:8000/users/?last_name=%3clast_name) |
|  | Get all users records containing the seachvalue in fullname,username, email | GET | [http://127.0.0.1:8000/users/?search=<searchvalue>](http://127.0.0.1:8000/users/?search=%3csearchvalue) |

List of other API’s that can be used by all users.

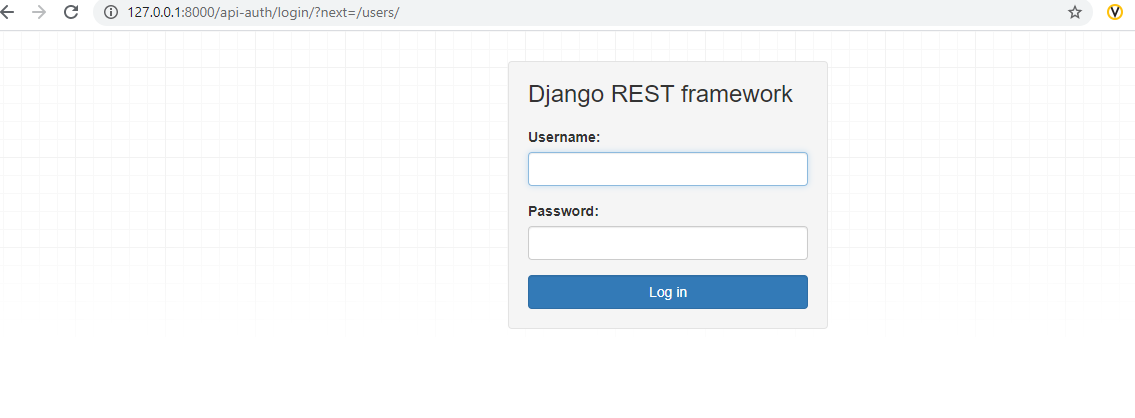
|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Description** | **HTTP Method** | **API syntax** |
| Post | List all Post | GET/ POST | http://127.0.0.1:8000/posts/ |
|  | Get and update specific post | PUT/ DELETE | http://127.0.0.1:8000/posts/<postid> |
|  | Get specific user by title | GET | http://127.0.0.1:8000/posts/?title=<title> |
|  | Get specific user by content | GET | http://127.0.0.1:8000/posts/?content=<content> |
|  | Get specific user by subtitle | GET | http://127.0.0.1:8000/posts/?subtitle=<subtitle> |
|  | Get all users records containing the seachvalue in title, subtitle, content | GET | http://127.0.0.1:8000/posts/?search=<searchvalue> |
|  |  |  |  |
| Comments | List all Comments for the Post | GET/ POST | http://127.0.0.1:8000/posts/<post:id>/comments/ |
|  | Get and update specific comment for a specific post | PUT/DELETE | http://127.0.0.1:8000/posts/<postid>/comments/<commnetid> |
|  | Get all comments for the post with the search critiria | GET | http://127.0.0.1:8000/posts/<postid>/comments/?search=<searchvalue |
|  |  |  |  |
| Search All | Search all the models with the specific search value where the current user has access | GET | http://127.0.0.1:8000/users-posts/?search=<searchvalue> |

# Navigation for testing the API’s

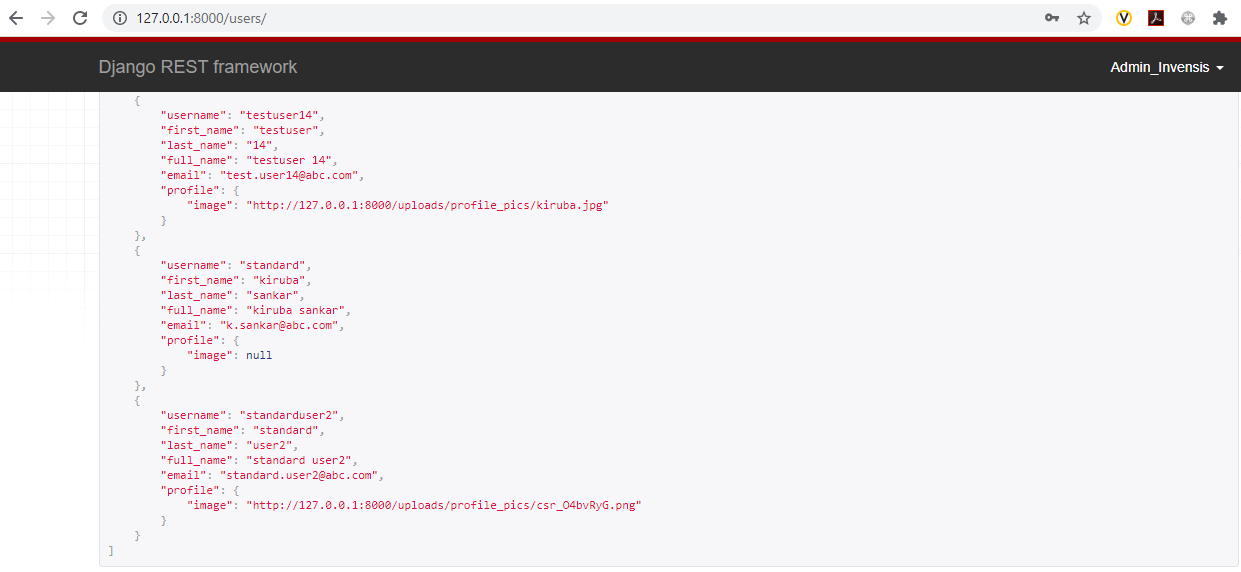
Once the development server is started you can login to the application using the URL below.

Login to the application using

http://127.0.0.1:8000/api-auth/login/?next=/users/



Sample screen shot of user request



Every GET request will display the option to POST a new document depending on the access. The POST request has both the Raw data format and HTML format.

Below is the screen shot for a user creation.

