V2 Cloud Home Assignment – Full Stack Developer

By – Harjot Singh Saggu (harjot9868@gmail.com)

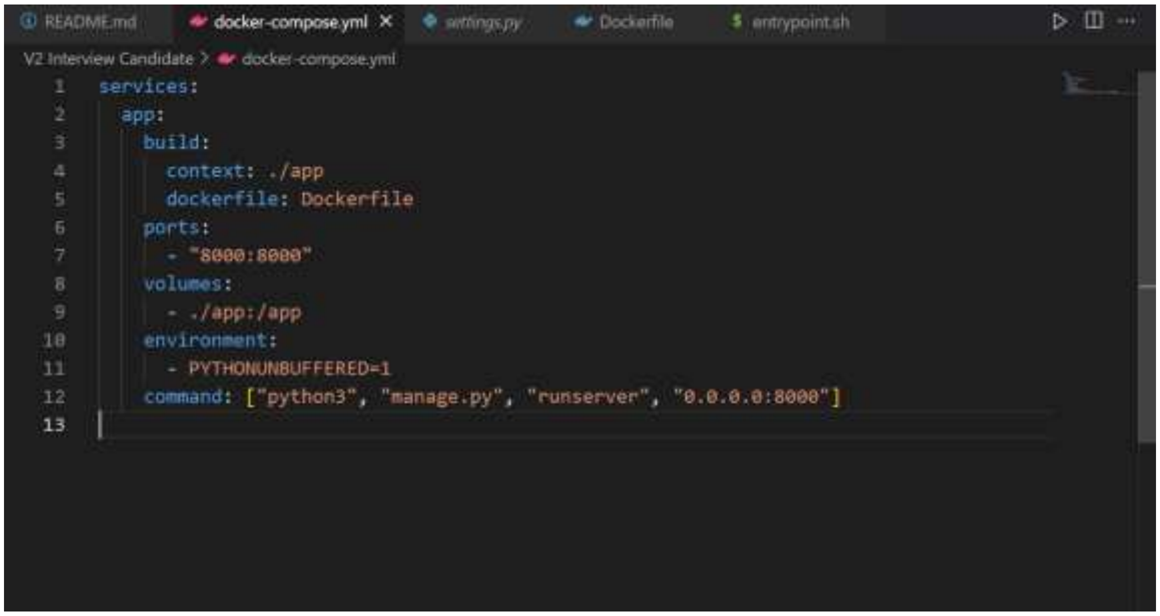
All code are written by me on my local desktop server. Zip file contains the final code.

Answer 1.

As I walked though the code found Django application setup in the /app directory. First task is to write docker compose.yml file.

- My approach to write the docker-compose.yml file is first to look services we need to set up the containers. First service I setup is "app" for running Django server.
- The context specifies the directory containing the Dockerfile and application code. Here, ./app indicates that the Dockerfile is located in the app folder.
- The dockerfile directive tells Docker which Dockerfile to use
- I choose the ports localhost8000 for django server
- The volumes section mounts the ./app directory from the host into the /app directory in the container. This enables live reloading of code changes during development.
- The environment section sets environment variables for the container. PYTHONUNBUFFERED=1 is used to ensure that Python outputs logs in real time.
- Then I wrote the command python manage.py runserver to run the server with the port number

Screen Shot of my PC screen for the **Docker-compose.yml**

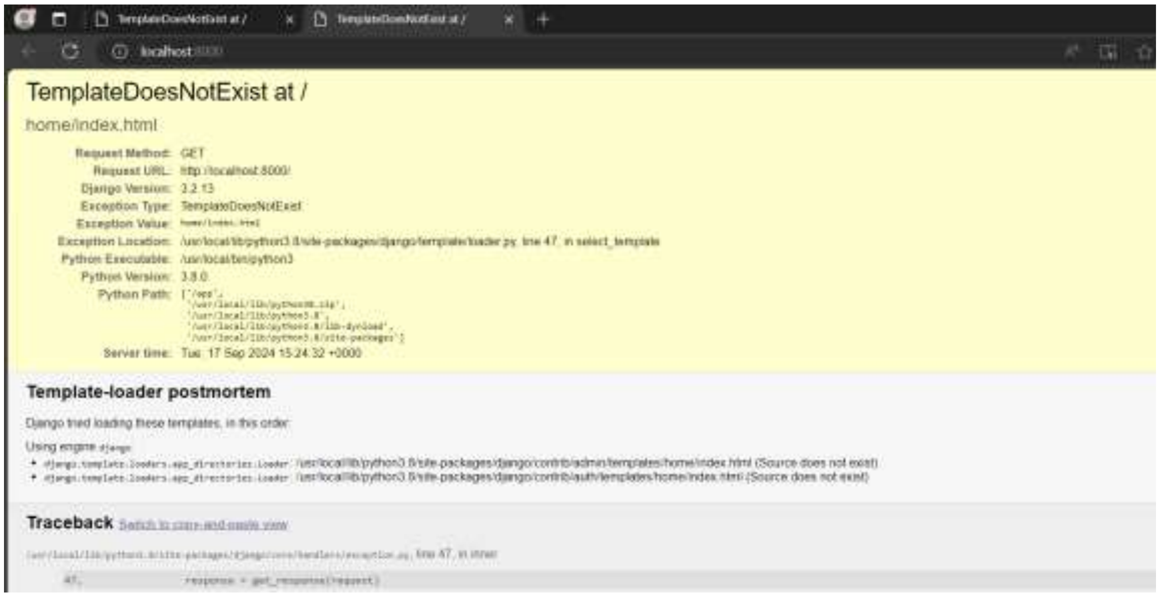


```
services:
  app:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ./app:/app
    environment:
      - PYTHONUNBUFFERED=1
    command: ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
```

After that I run on the command **"docker compose up".**

Answer 2.

During the startup process, an error was encountered indicating that a **template does not exist** as shown below. The specific error message was related to Django's template loading system.

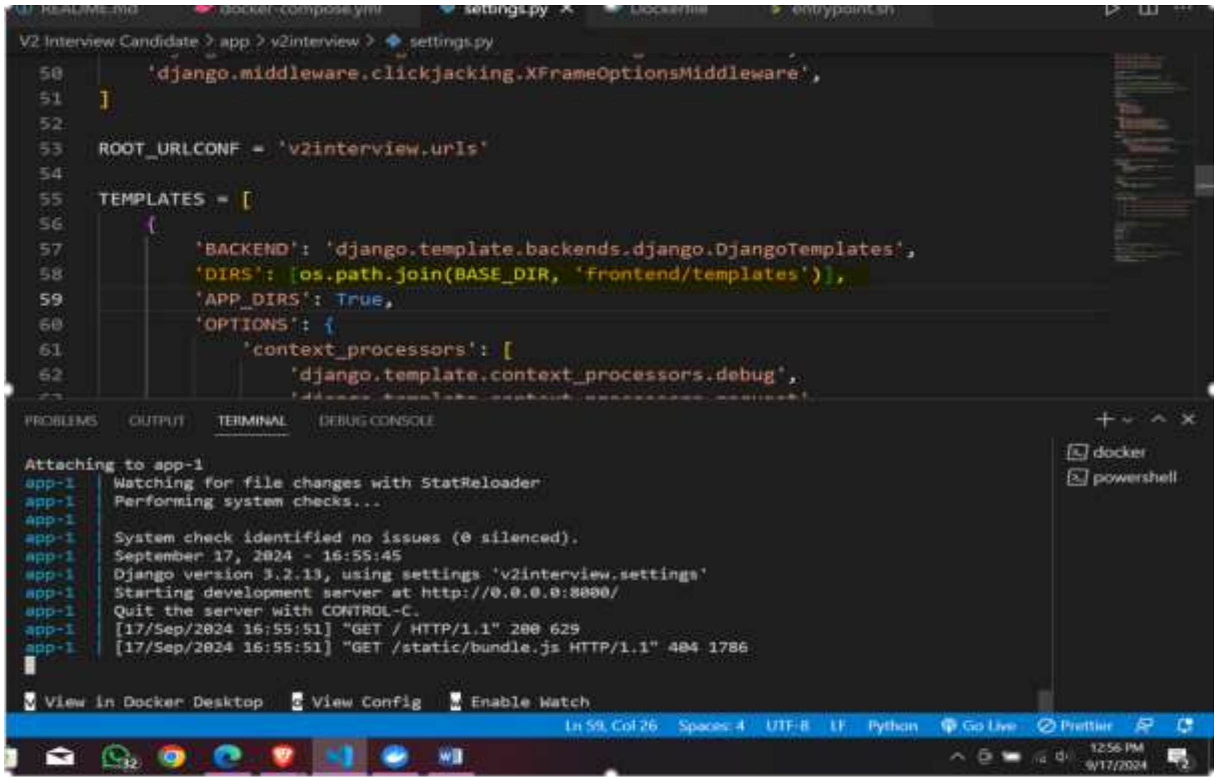


```
TemplateDoesNotExist at /
home/index.html

     Request Method: GET
        Request URL: http://localhost:8000/
     Django Version: 3.2.13
     Exception Type: TemplateDoesNotExist
    Exception Value: home/index.html
 Exception Location: /usr/local/lib/python3.8/site-packages/django/template/loader.py, line 47, in select_template
  Python Executable: /usr/local/bin/python3
     Python Version: 3.8.0
        Python Path: ['/app',
                      '/usr/local/lib/python38.zip',
                      '/usr/local/lib/python3.8',
                      '/usr/local/lib/python3.8/lib-dynload',
                      '/usr/local/lib/python3.8/site-packages']
        Server time: Tue, 17 Sep 2024 15:24:32 +0000

Template-loader postmortem
Django tried loading these templates, in this order:
Using engine django:
  • django.template.loaders.app_directories.Loader: /usr/local/lib/python3.8/site-packages/django/contrib/admin/templates/home/index.html (Source does not exist)
  • django.template.loaders.app_directories.Loader: /usr/local/lib/python3.8/site-packages/django/contrib/auth/templates/home/index.html (Source does not exist)

Traceback Switch to copy-and-paste view
/usr/local/lib/python3.8/site-packages/django/core/handlers/exception.py, line 47, in inner
    47.                 response = get_response(request)
```

The error message indicated that the application was unable to locate the template files.

Upon reviewing the configuration, it became clear that the template directory path was incorrectly specified in the Django settings.

To fix this issue, I updated the **TEMPLATES setting** in the Django settings.py file. The correction involved modifying the 'DIRS' list to include the correct path for the frontend templates



```python
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'v2interview.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'frontend/templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
```

```
Attaching to app-1
app-1  | Watching for file changes with StatReloader
app-1  | Performing system checks...
app-1  |
app-1  | System check identified no issues (0 silenced).
app-1  | September 17, 2024 - 16:55:45
app-1  | Django version 3.2.13, using settings 'v2interview.settings'
app-1  | Starting development server at http://0.0.0.0:8000/
app-1  | Quit the server with CONTROL-C.
app-1  | [17/Sep/2024 16:55:51] "GET / HTTP/1.1" 200 629
app-1  | [17/Sep/2024 16:55:51] "GET /static/bundle.js HTTP/1.1" 404 1786
```

Answer 3.

I install all dependencies listed in the package.json file from using npm install.

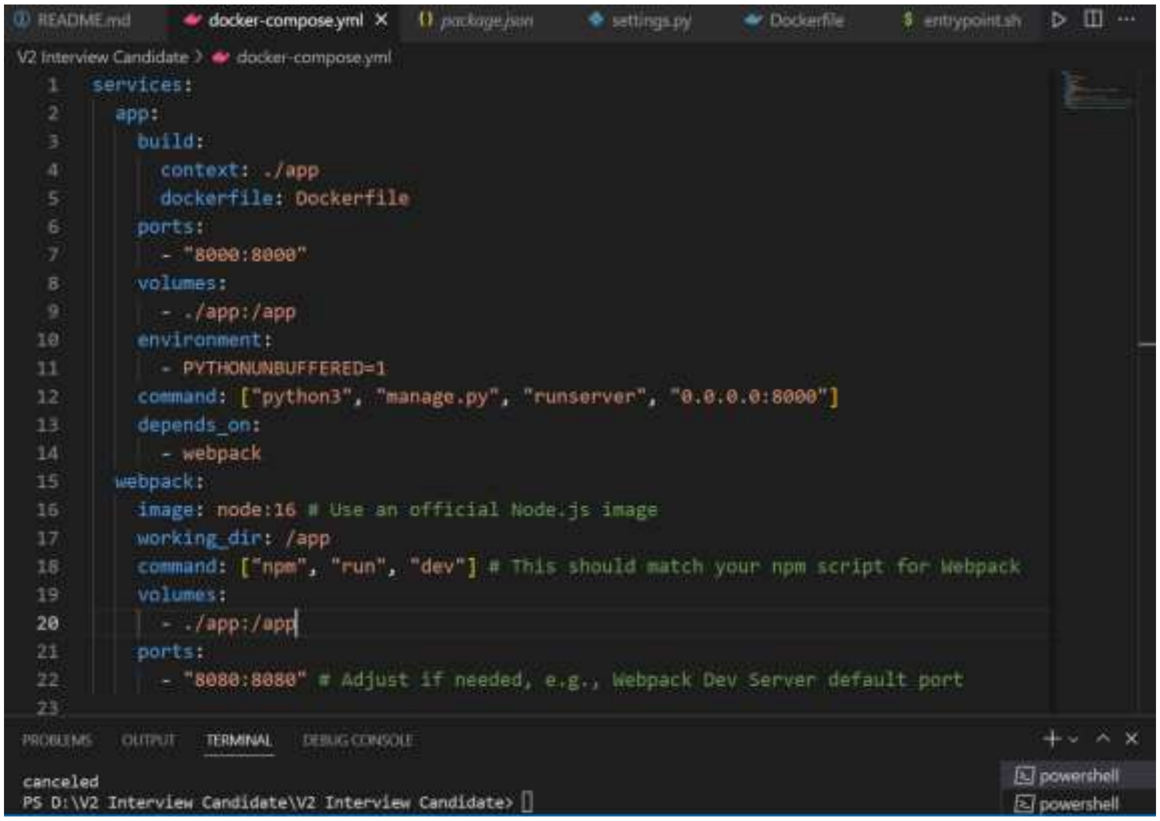
I added a separate service for Webpack in the docker-compose.yml file. This way, both the Django app and Webpack can run in parallel when starting the containers.

I defined the service name as webpack. This service is responsible for running the Webpack bundler.

I set up a volume mount that links my local app directory to the /app path in the container. This allows any changes I make to the local files to be reflected in the container immediately.

I specified the command that runs when the Webpack service starts npm run webpack.

I indicated that the webpack service depends on the Django server. This means Docker Compose ensures that the app service starts before the webpack service

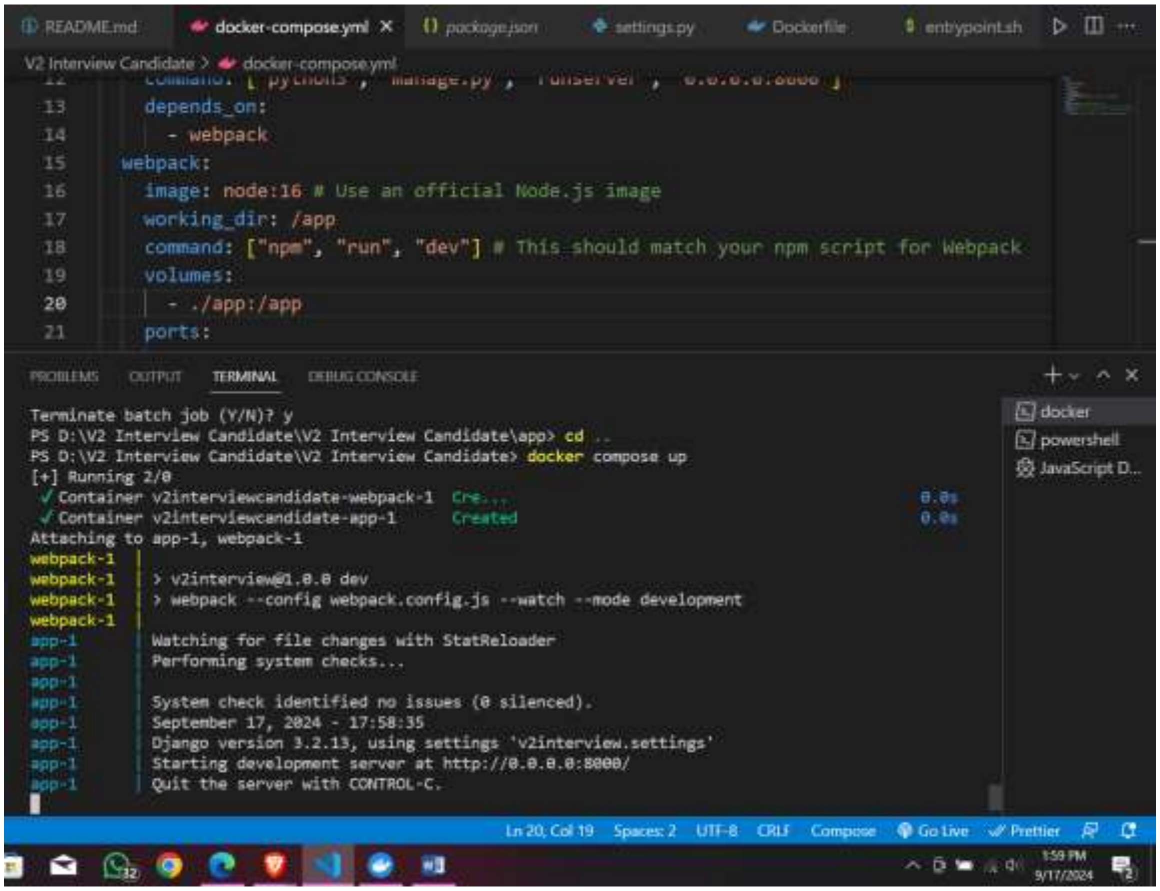


```yaml
services:
  app:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ./app:/app
    environment:
      - PYTHONUNBUFFERED=1
    command: ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
    depends_on:
      - webpack
  webpack:
    image: node:16 # Use an official Node.js image
    working_dir: /app
    command: ["npm", "run", "dev"] # This should match your npm script for Webpack
    volumes:
      - ./app:/app
    ports:
      - "8080:8080" # Adjust if needed, e.g., Webpack Dev Server default port

```

```
canceled
PS D:\V2 Interview Candidate\V2 Interview Candidate>
```

Running “docker compose up” as shown below.

And we can see webpack-1 and app-1 containers running successfully.



```
    command: ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
    depends_on:
      - webpack
  webpack:
    image: node:16 # Use an official Node.js image
    working_dir: /app
    command: ["npm", "run", "dev"] # This should match your npm script for Webpack
    volumes:
      - ./app:/app
    ports:
```

```
Terminate batch job (Y/N)? y
PS D:\V2 Interview Candidate\V2 Interview Candidate\app> cd ..
PS D:\V2 Interview Candidate\V2 Interview Candidate> docker compose up
[+] Running 2/0
 ✓ Container v2interviewcandidate-webpack-1  Cre...                      0.0s
 ✓ Container v2interviewcandidate-app-1      Created                     0.0s
Attaching to app-1, webpack-1
webpack-1  |
webpack-1  | > v2interview@1.0.0 dev
webpack-1  | > webpack --config webpack.config.js --watch --mode development
webpack-1  |
app-1      | Watching for file changes with StatReloader
app-1      | Performing system checks...
app-1      |
app-1      | System check identified no issues (0 silenced).
app-1      | September 17, 2024 - 17:58:35
app-1      | Django version 3.2.13, using settings 'v2interview.settings'
app-1      | Starting development server at http://0.0.0.0:8000/
app-1      | Quit the server with CONTROL-C.
```

Answer 4.

I made sure that Django Rest Framework was installed in my project. “pip install djangorestframework”

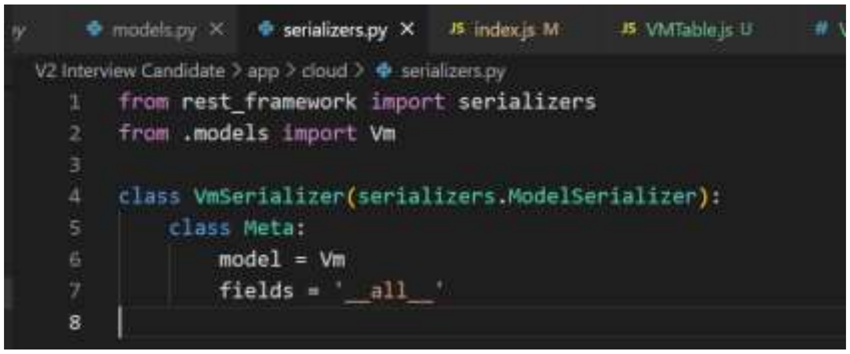
I added **'rest_framework'** and **‘cloud’** app to the **INSTALLED_APPS** list in my settings.py file

Model.py is already defined.

a) Creating a Serializer

I created a serializer to convert VM instances into JSON format and validate incoming data. I added a new file named serializers.py in the app directory, file path **: /app/cloud/ serializers.py**

The class VMSerializer inherits from serializers.ModelSerializer, linking it directly to the VM model through its inner Meta class. By using fields = '__all__', it includes all fields defined in the model. This setup simplifies the handling of API data for the VM model.

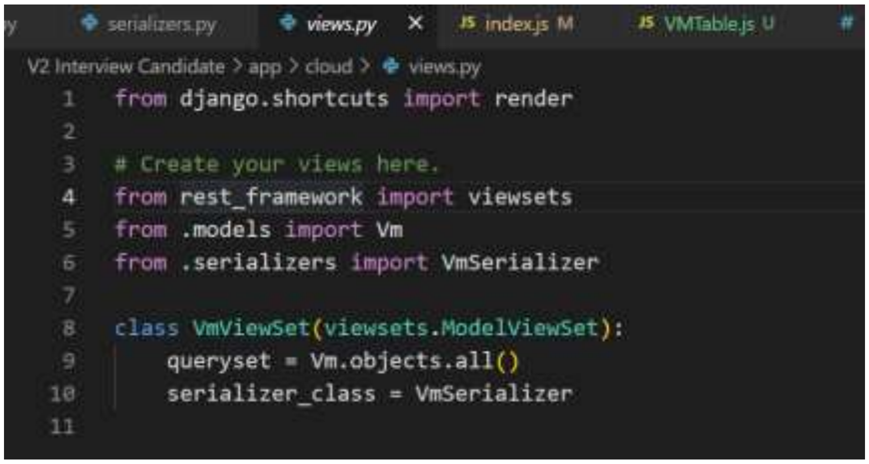


```python
from rest_framework import serializers
from .models import Vm

class VmSerializer(serializers.ModelSerializer):
    class Meta:
        model = Vm
        fields = '__all__'
```

b) Creating a ViewSet

I created a viewset in views.py that handles the CRUD operations for the VM model. I used DRF's ModelViewSet for simplicity. File path : **/app/cloud/ views.py**

By inheriting from viewsets.ModelViewSet, it automatically provides methods for listing, creating, retrieving, updating, and deleting VM instances. The queryset attribute retrieves all VM objects, and the serializer_class specifies that VMSerializer will be used for data serialization and validation.

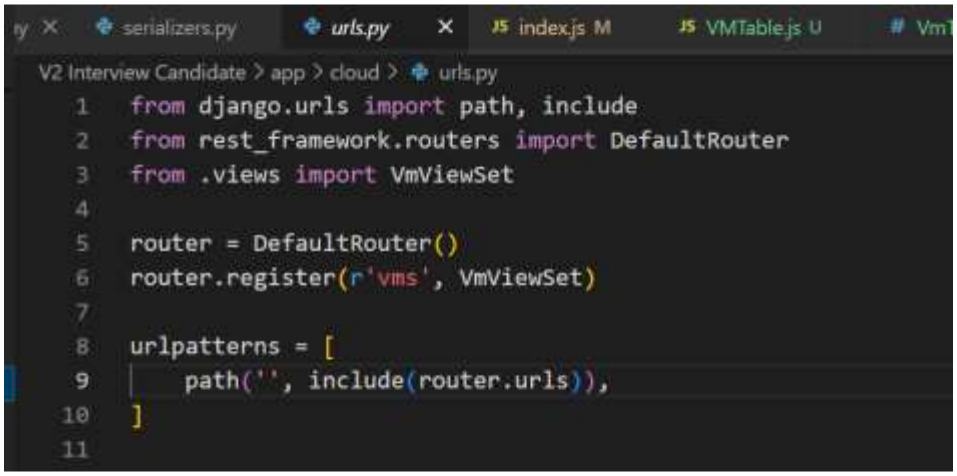


```python
from django.shortcuts import render

# Create your views here.
from rest_framework import viewsets
from .models import Vm
from .serializers import VmSerializer

class VmViewSet(viewsets.ModelViewSet):
    queryset = Vm.objects.all()
    serializer_class = VmSerializer
```

c) Setting Up URL Routing

I created a new file named urls.py in the app directory to define the API endpoints. I included the router for the VMViewSet. File path: **/app/cloud/urls.py**

This code sets up the URL routing for the API. It imports path and include from Django's URL module, along with DefaultRouter from Django Rest Framework. A router instance is created, and the VMViewSet is registered with the route vms, which maps API requests to the viewset. Finally, urlpatterns includes the router's URLs, allowing the defined endpoints to be accessible under the API path.



```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import VmViewSet

router = DefaultRouter()
router.register(r'vms', VmViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

d) Included App URLs

In the main urls.py file of the project, I included the app's URLs

```python
path('api/',  include('cloud.urls')),
```

e) Testing apis

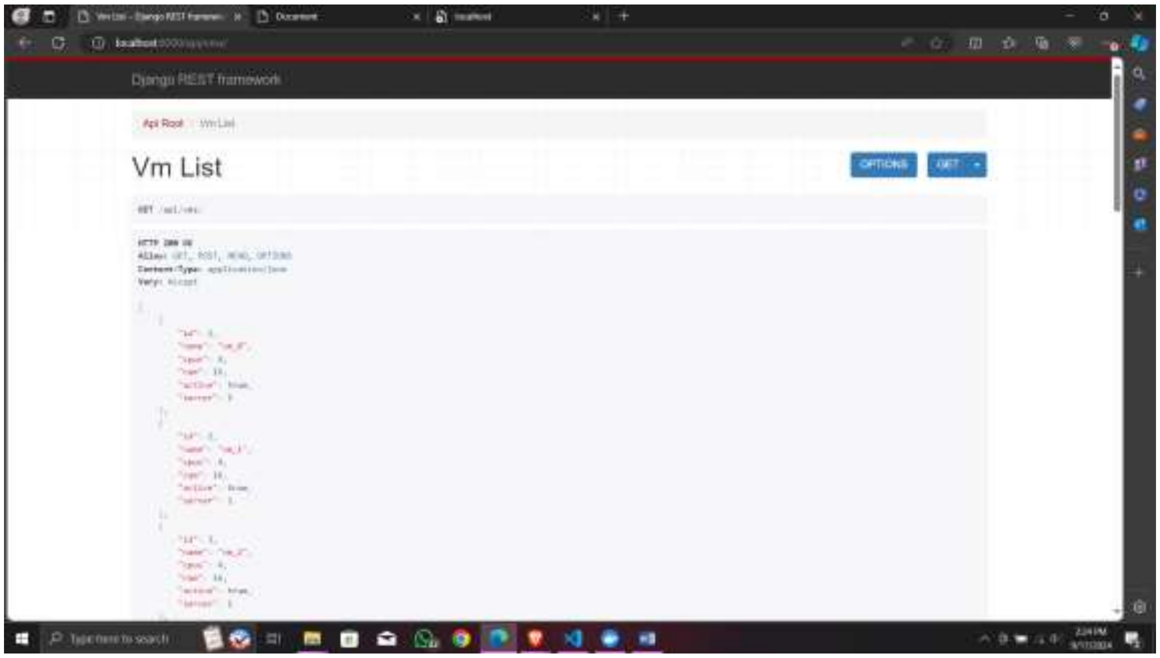
List VMs: http://localhost:8000/api/vms/ (GET request)

Create VM: http://localhost:8000/api/vms/ (POST request with JSON body)

Retrieve VM: http://localhost:8000/api/vms/<id>/ (GET request with specific VM ID)

Update VM: http://localhost:8000/api/vms/<id>/ (PUT or PATCH request with updated JSON body)

Delete VM: http://localhost:8000/api/vms/<id>/ (DELETE request)



Vm List

```
GET /api/vms/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[illegible]
```

Answer 5.

I setup the react project in **app/frontend/static/reactapp** directory. Moved Components and service folder from static/react dir.

VMTable.js component

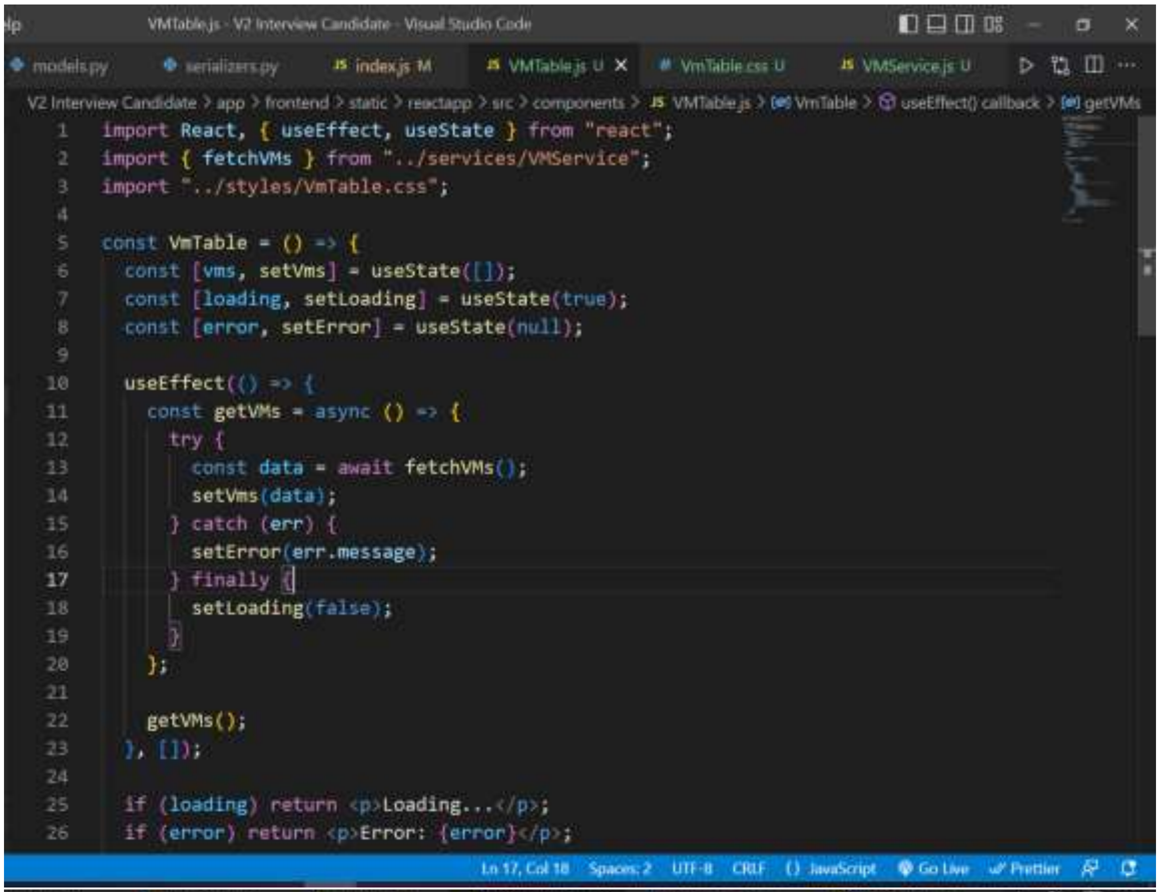
This component will handle fetching and displaying the VM data.

It employs React hooks to manage state and side effects, with useState creating three state variables: vms for storing fetched VM data, loading to indicate if data is being loaded, and error to hold any error messages.

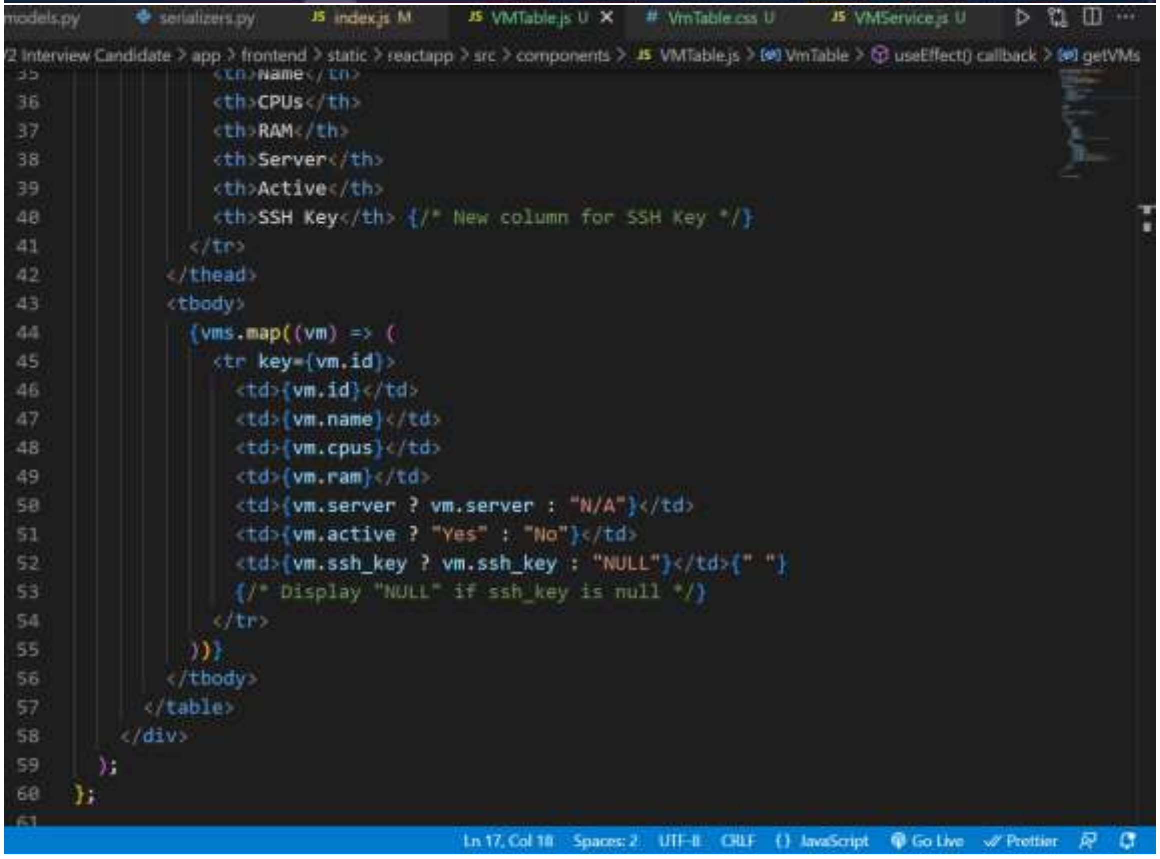
The useEffect hook fetches VM data when the component mounts, using an asynchronous function that calls the fetchVMs function from the service.

If the fetch is successful, the vms state updates with the retrieved data; if an error occurs, the error message is stored in the error state, and the loading state is set to false.

Once loading is complete and no error has occurred, the component renders a table with headers for various VM attributes, mapping over the vms array to create rows for each VM.


VMTable.js - V2 Interview Candidate - Visual Studio Code
models.py
serializers.py
index.js M
VMTable.js U
VmTable.css U
VMService.js U
V2 Interview Candidate > app > frontend > static > reactapp > src > components > VMTable.js > VmTable > useEffect() callback > getVMs
Ln 17, Col 18
Spaces: 2
UTF-8
CRLF
{} JavaScript
Go Live
Prettier

```javascript
import React, { useEffect, useState } from "react";
import { fetchVMs } from "../services/VMService";
import "../styles/VmTable.css";

const VmTable = () => {
  const [vms, setVms] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const getVMs = async () => {
      try {
        const data = await fetchVMs();
        setVms(data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    getVMs();
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;
```

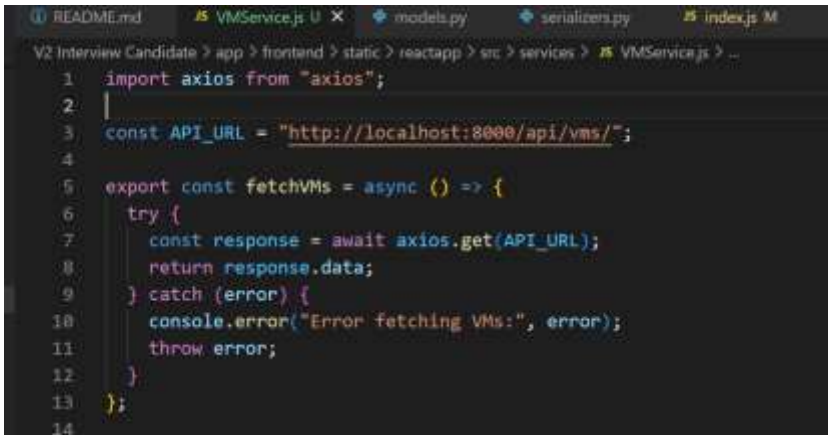

models.py
serializers.py
index.js M
VMTable.js U
VmTable.css U
VMService.js U
V2 Interview Candidate > app > frontend > static > reactapp > src > components > VMTable.js > VmTable > useEffect() callback > getVMs
Ln 17, Col 18
Spaces: 2
UTF-8
CRLF
{} JavaScript
Go Live
Prettier

```javascript
            <th>Name</th>
            <th>CPUs</th>
            <th>RAM</th>
            <th>Server</th>
            <th>Active</th>
            <th>SSH Key</th> {/* New column for SSH Key */}
          </tr>
        </thead>
        <tbody>
          {vms.map((vm) => (
            <tr key={vm.id}>
              <td>{vm.id}</td>
              <td>{vm.name}</td>
              <td>{vm.cpus}</td>
              <td>{vm.ram}</td>
              <td>{vm.server ? vm.server : "N/A"}</td>
              <td>{vm.active ? "Yes" : "No"}</td>
              <td>{vm.ssh_key ? vm.ssh_key : "NULL"}</td>{" "}
              {/* Display "NULL" if ssh_key is null */}
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};
```

VMServices.js

I created VmServices.js file path: **./static/reactapp/src/services/VmServices.js.** The VMService module is responsible for making API calls, defining an API_URL for the backend endpoint and providing the fetchVMs function that sends a GET request to retrieve VM data. This function returns the data upon success and includes error handling to log and rethrow errors.

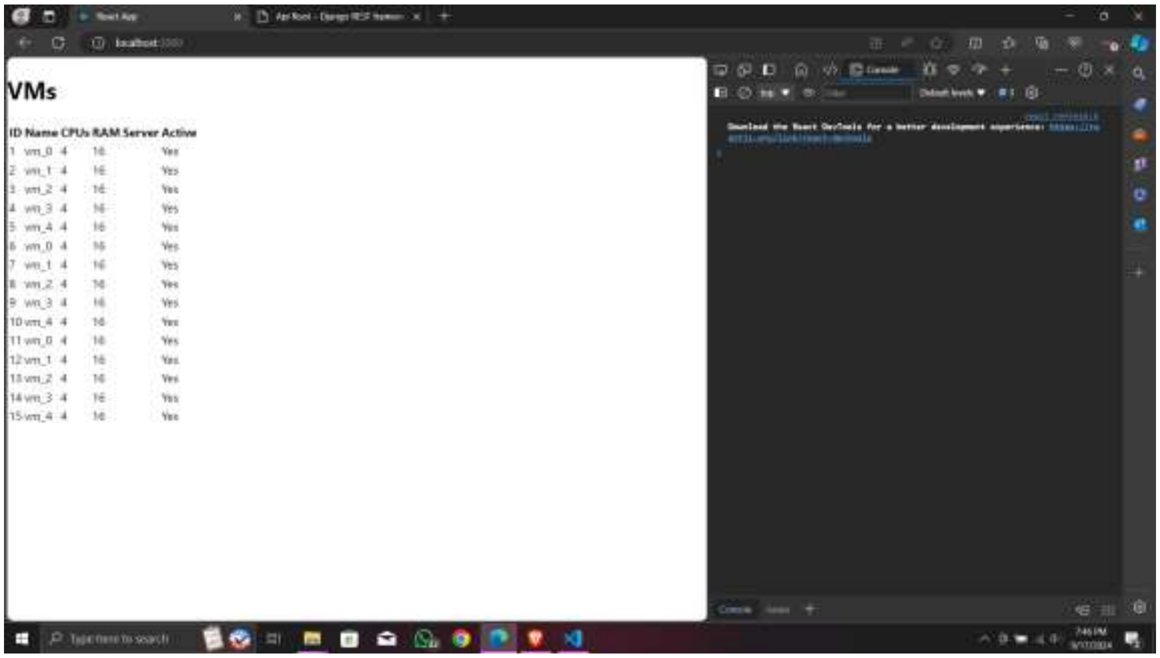
I installed Axios using npm for handling api requests.



README.md
VMService.js U
models.py
serializers.py
index.js M
V2 Interview Candidate > app > frontend > static > reactapp > src > services > VMService.js > ...

```javascript
import axios from "axios";

const API_URL = "http://localhost:8000/api/vms/";

export const fetchVMs = async () => {
  try {
    const response = await axios.get(API_URL);
    return response.data;
  } catch (error) {
    console.error("Error fetching VMs:", error);
    throw error;
  }
};
```

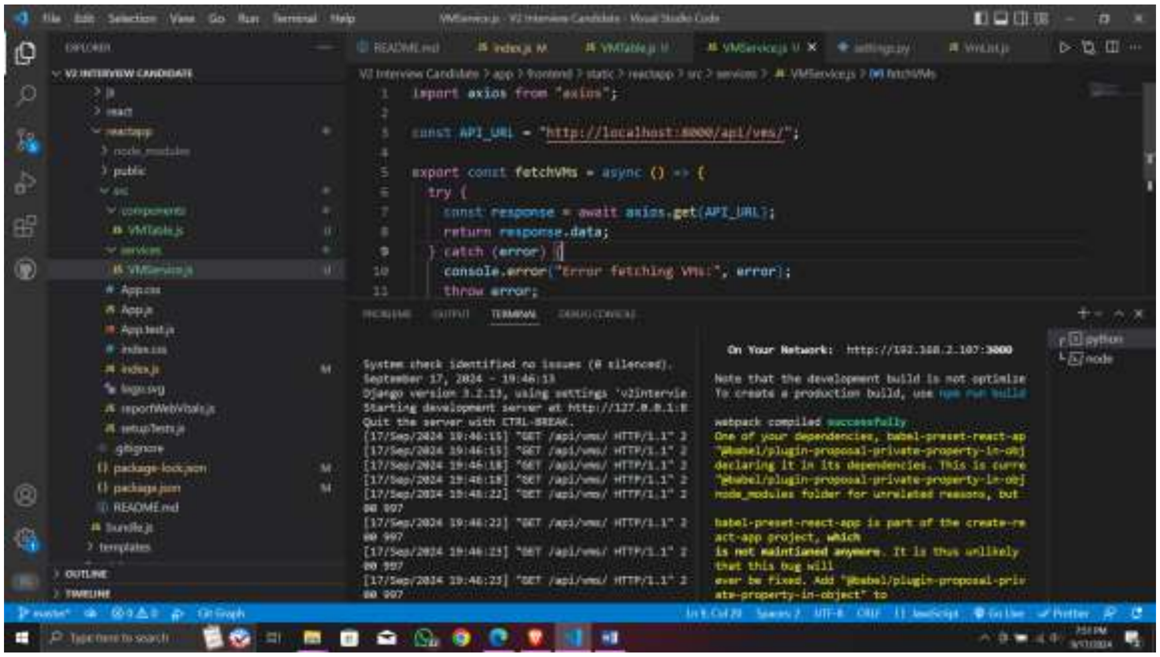
Output on localhost



VMs

ID	Name	CPUs	RAM	Server	Active
1	vm_0	4	16		Yes
2	vm_1	4	16		Yes
3	vm_2	4	16		Yes
4	vm_3	4	16		Yes
5	vm_4	4	16		Yes
6	vm_0	4	16		Yes
7	vm_1	4	16		Yes
8	vm_2	4	16		Yes
9	vm_3	4	16		Yes
10	vm_4	4	16		Yes
11	vm_0	4	16		Yes
12	vm_1	4	16		Yes
13	vm_2	4	16		Yes
14	vm_3	4	16		Yes
15	vm_4	4	16		Yes

```
Download the React DevTools for a better development experience: [illegible]
```



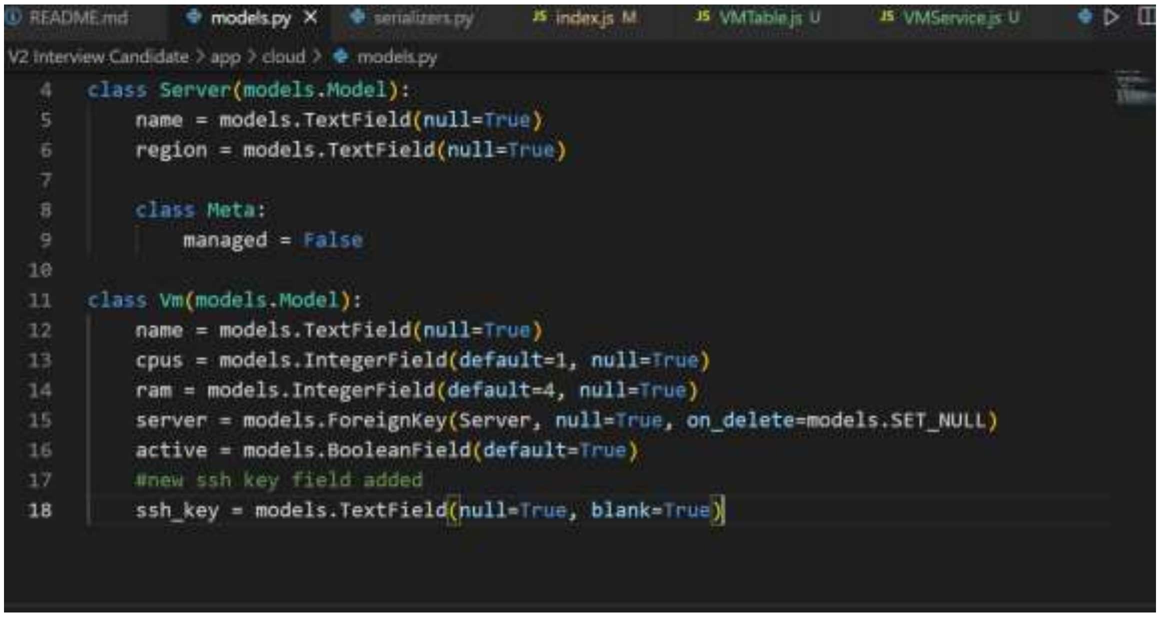
```
import axios from "axios";

const API_URL = "http://localhost:8000/api/vms/";

export const fetchVMs = async () => {
  try {
    const response = await axios.get(API_URL);
    return response.data;
  } catch (error) {
    console.error("Error fetching VMs:", error);
    throw error;
```

```
System check identified no issues (0 silenced).
September 17, 2024 - 19:46:13
Django version 3.2.13, using settings 'v2intervie
Starting development server at http://127.0.0.1:8
Quit the server with CTRL-BREAK.
[17/Sep/2024 19:46:15] "GET /api/vms/ HTTP/1.1" 2
[17/Sep/2024 19:46:15] "GET /api/vms/ HTTP/1.1" 2
[17/Sep/2024 19:46:18] "GET /api/vms/ HTTP/1.1" 2
[17/Sep/2024 19:46:18] "GET /api/vms/ HTTP/1.1" 2
[17/Sep/2024 19:46:22] "GET /api/vms/ HTTP/1.1" 2
00 997
[17/Sep/2024 19:46:22] "GET /api/vms/ HTTP/1.1" 2
00 997
[17/Sep/2024 19:46:23] "GET /api/vms/ HTTP/1.1" 2
00 997
[17/Sep/2024 19:46:23] "GET /api/vms/ HTTP/1.1" 2
00 997
```

```
  On Your Network:  http://192.168.2.107:3000

Note that the development build is not optimize
To create a production build, use npm run build

webpack compiled successfully
One of your dependencies, babel-preset-react-ap
"@babel/plugin-proposal-private-property-in-obj
declaring it in its dependencies. This is curre
"@babel/plugin-proposal-private-property-in-obj
node_modules folder for unrelated reasons, but

babel-preset-react-app is part of the create-re
act-app project, which
is not maintianed anymore. It is thus unlikely
that this bug will
ever be fixed. Add "@babel/plugin-proposal-priv
ate-property-in-object" to
```

Answer 6.

To add a new field called ssh_key to the Vm model in Django, I started by modifying the model to include this new field. I assumed that the ssh_key would be stored as a string, so I used a TextField.



```python
class Server(models.Model):
    name = models.TextField(null=True)
    region = models.TextField(null=True)

    class Meta:
        managed = False

class Vm(models.Model):
    name = models.TextField(null=True)
    cpus = models.IntegerField(default=1, null=True)
    ram = models.IntegerField(default=4, null=True)
    server = models.ForeignKey(Server, null=True, on_delete=models.SET_NULL)
    active = models.BooleanField(default=True)
    #new ssh key field added
    ssh_key = models.TextField(null=True, blank=True)
```

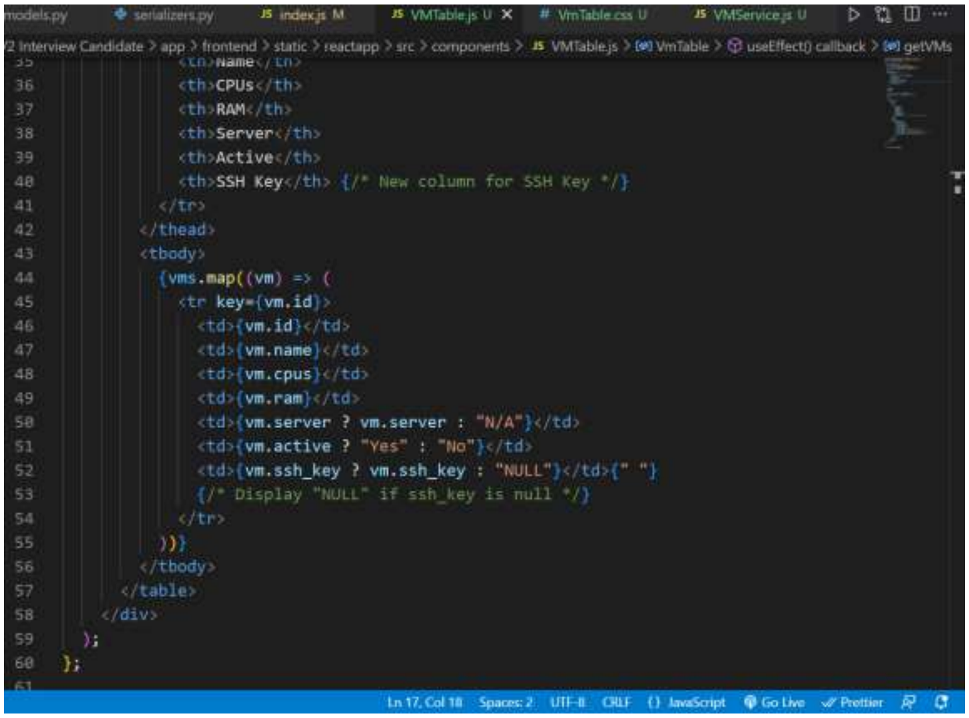
created a migration file to apply this change to the database

- python manage.py makemigrations

After that, I applied the migration to the database

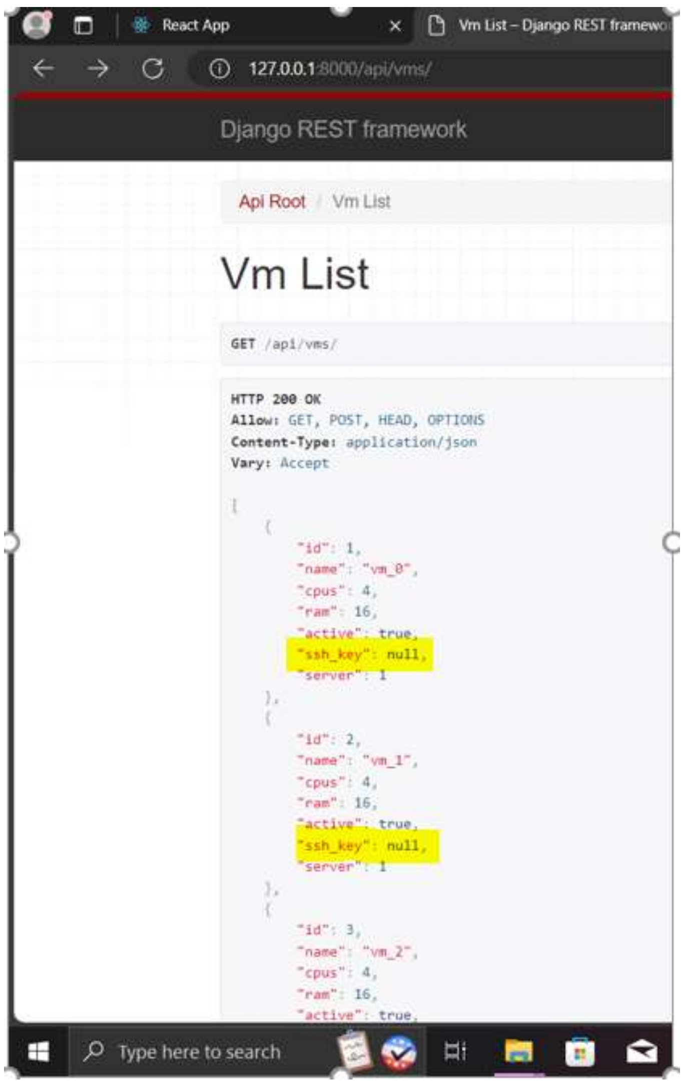
- python manage.py migrate

To update the frontend component to display the new ssh_key field in the VM table, I will modify the VmTable component accordingly


models.py
serializers.py
index.js M
VMTable.js U
VmTable.css U
VMService.js U
2 Interview Candidate > app > frontend > static > reactapp > src > components > VMTable.js > VmTable > useEffect() callback > getVMs
Ln 17, Col 18 Spaces: 2 UTF-8 CRLF {} JavaScript Go Live Prettier

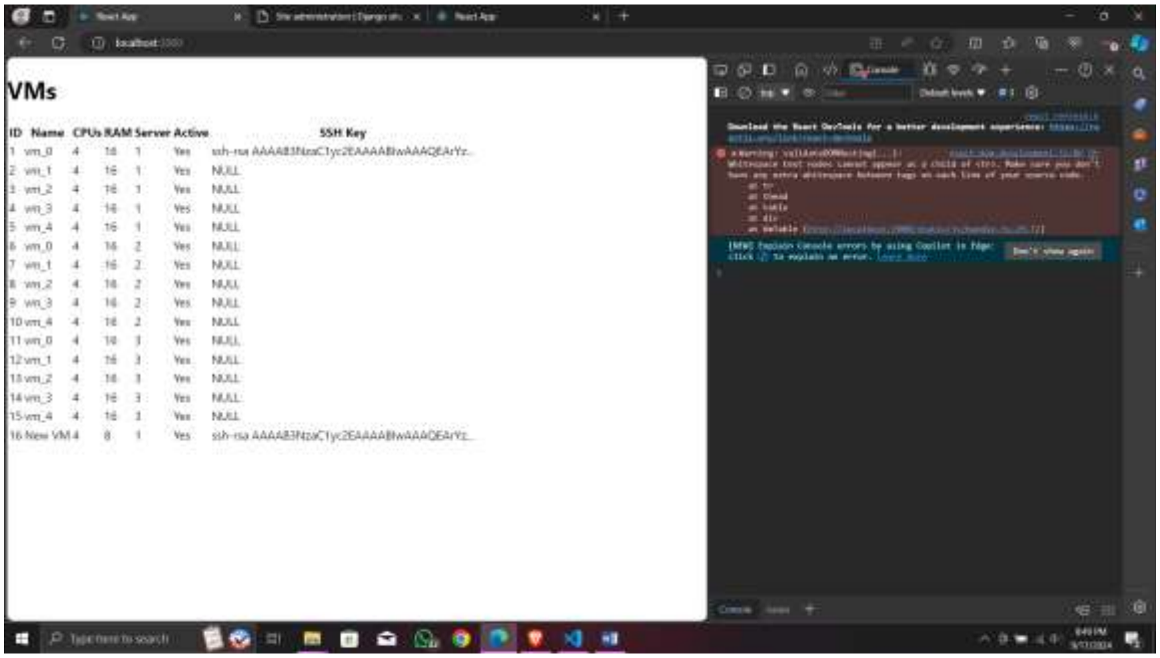
```
            <th>Name</th>
            <th>CPUs</th>
            <th>RAM</th>
            <th>Server</th>
            <th>Active</th>
            <th>SSH Key</th> {/* New column for SSH Key */}
          </tr>
        </thead>
        <tbody>
          {vms.map((vm) => (
            <tr key={vm.id}>
              <td>{vm.id}</td>
              <td>{vm.name}</td>
              <td>{vm.cpus}</td>
              <td>{vm.ram}</td>
              <td>{vm.server ? vm.server : "N/A"}</td>
              <td>{vm.active ? "Yes" : "No"}</td>
              <td>{vm.ssh_key ? vm.ssh_key : "NULL"}</td>{" "}
              {/* Display "NULL" if ssh_key is null */}
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};
```

Output showing ssh_key



```
GET /api/vms/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "name": "vm_0",
        "cpus": 4,
        "ram": 16,
        "active": true,
        "ssh_key": null,
        "server": 1
    },
    {
        "id": 2,
        "name": "vm_1",
        "cpus": 4,
        "ram": 16,
        "active": true,
        "ssh_key": null,
        "server": 1
    },
    {
        "id": 3,
        "name": "vm_2",
        "cpus": 4,
        "ram": 16,
        "active": true,
```

Frontend output



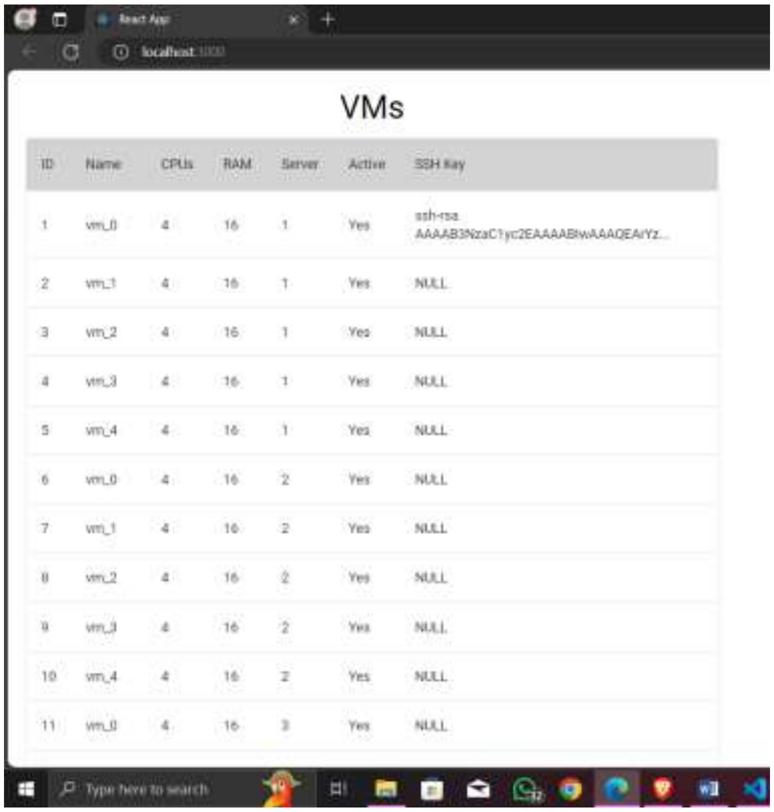
VMs

ID	Name	CPUs	RAM	Server	Active	SSH Key
1	vm_0	4	16	1	Yes	ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEArYz...
2	vm_1	4	16	1	Yes	NULL
3	vm_2	4	16	1	Yes	NULL
4	vm_3	4	16	1	Yes	NULL
5	vm_4	4	16	1	Yes	NULL
6	vm_0	4	16	2	Yes	NULL
7	vm_1	4	16	2	Yes	NULL
8	vm_2	4	16	2	Yes	NULL
9	vm_3	4	16	2	Yes	NULL
10	vm_4	4	16	2	Yes	NULL
11	vm_0	4	16	3	Yes	NULL
12	vm_1	4	16	3	Yes	NULL
13	vm_2	4	16	3	Yes	NULL
14	vm_3	4	16	3	Yes	NULL
15	vm_4	4	16	3	Yes	NULL
16	New VM	4	8	1	Yes	ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEArYz...

```
Download the React DevTools for a better development experience: [illegible]
▸ Warning: validateDOMNesting(...): [illegible]
Whitespace text nodes cannot appear as a child of <tr>. Make sure you don't
have any extra whitespace between tags on each line of your source code.
    at tr
    at thead
    at table
    at div
    at VmTable ([illegible])
[NEW] Explain Console errors by using Copilot in Edge:
click [illegible] to explain an error. Learn more
```

Extra Additional updates I was trying

1. Modified UI : I enhanced the user interface of the VmTable component by implementing the Material-UI Table component for improved aesthetics and usability. This change ensures a more polished and responsive design, making it easier for users to navigate and view the virtual machines. Although I aimed to implement further modifications to enhance the UI, I ran out of time to complete those additional improvements.



VMs

ID	Name	CPUs	RAM	Server	Active	SSH Key
1	vm_0	4	16	1	Yes	ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEArYz...
2	vm_1	4	16	1	Yes	NULL
3	vm_2	4	16	1	Yes	NULL
4	vm_3	4	16	1	Yes	NULL
5	vm_4	4	16	1	Yes	NULL
6	vm_0	4	16	2	Yes	NULL
7	vm_1	4	16	2	Yes	NULL
8	vm_2	4	16	2	Yes	NULL
9	vm_3	4	16	2	Yes	NULL
10	vm_4	4	16	2	Yes	NULL
11	vm_0	4	16	3	Yes	NULL