***TRAVEL-MEMORY***

# Cloning the repository-

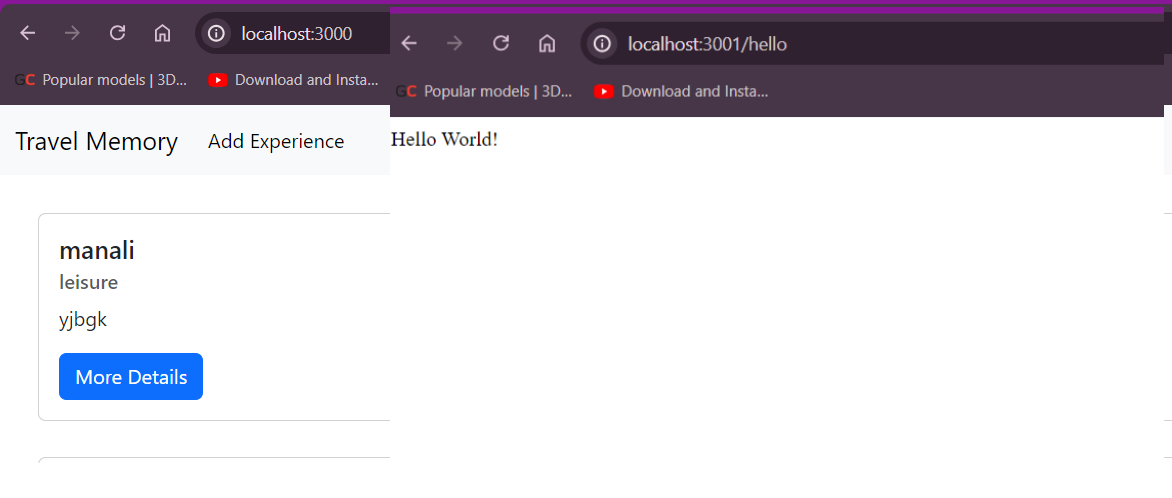
1. Fork the “TravelMemory” repository in your github.
2. Create a folder in your local machine and clone the “TravelMemory” repository. git clone https://github.com/Ravikishans/TravelMemory.git

# Configuring MongoDB-

1. Create an account in MongoDb atlas
2. Create a free cluster m0, shared, aws
3. Create username and password
4. Allow all access 0.0.0.0/0
5. Click on connect and download MongoDb Compass
6. Copy the connection string and paste it in the compass with your password and press connect.
7. Now, create a database named ‘travelmemory’
8. Copy the connection string and add ‘/<databasename>’ in to it “mongodb+srv://ravikishan:Cluster0@cluster0.y9zohpu.mongodb.net/travelmemory”

# Setting up the Application on local host-

1. Go to the cloned TM application and create a ‘.env’ in backend folder.
2. Put the details inside ‘.env’. MONGO\_URI='<copied connection string>' PORT=3001
3. Now install the node module inside backend and frontend folders using command “npm install”
4. Run the “node index.js” command for executing the backend.
5. Now go to “TravelMemory/frontend/src/url.js” in another window of consol and change the url to <http://localhost:3001> and run “npm start”.

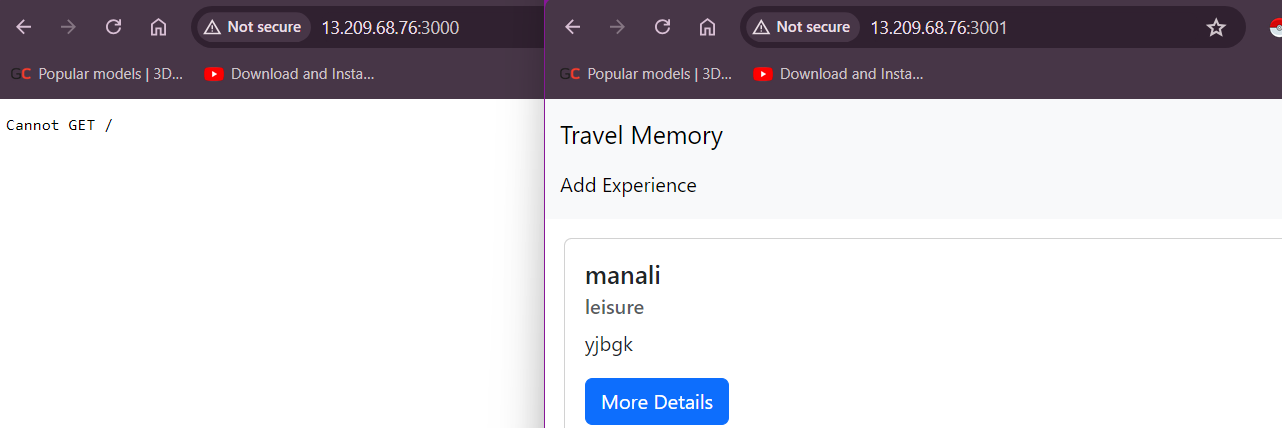


# Setting up the Application on AWS-

1. Push the TravelMemory on your GitHub repository from your local machine.
2. Now create an EC2 instance on your AWS and clone the Repository.
3. Install Nginx and node in both the front end and backend folder.
4. Create an “. env” file in backend folder and enter the values as we did before.

**MONGO\_URI='<copied connection string>'**

**PORT=3000**

1. Run the “node index.js” command for executing the backend.
2. Now go to “TravelMemory/frontend/src/url.js” in another window of consol and change the url to **“<http://publicIP:3000>”**. And also we have to change port no. to 3001 in “sudo nano /home/ubuntu/TravelMemory/frontend/node\_modules/react-scripts/scripts/start.js” for directing frontend to port no. to 3001.
3. Now we can run “npm start &”
4. **We have reversed the port numbers as given in the Assignment.** 

# Scaling of the application-

### Configuring the backend-

1. For this we have to configure nginx.
2. We have to edit and configure reverse proxy in “/etc/nginx/sites-available/default”

server {

        listen 80;

        listen [::]:80;

        server\_name \_;

#       root /var/www/example.com;

#       index index.html;

        location / {

                #try\_files $uri $uri/ =404;

                proxy\_pass http://localhost:3000;

                proxy\_http\_version 1.1;

                proxy\_set\_header Upgrade $http\_upgrade;

                proxy\_set\_header Connection 'upgrade';

                proxy\_set\_header Host $host;

                proxy\_cache\_bypass $http\_upgrade;

        }

}

1. Now restart the nginx, go to the backend folder and run command “node index.js &” for running in the background.
2. Now create AMI for backend application. And from that AMI create multiple instances.
3. Similarly change the nginx config for frontend

server {

        listen 80;

        listen [::]:80;

        server\_name \_;

#       root /var/www/example.com;

#       index index.html;

        location / {

                #try\_files $uri $uri/ =404;

                proxy\_pass http://localhost:3001;

                proxy\_http\_version 1.1;

                proxy\_set\_header Upgrade $http\_upgrade;

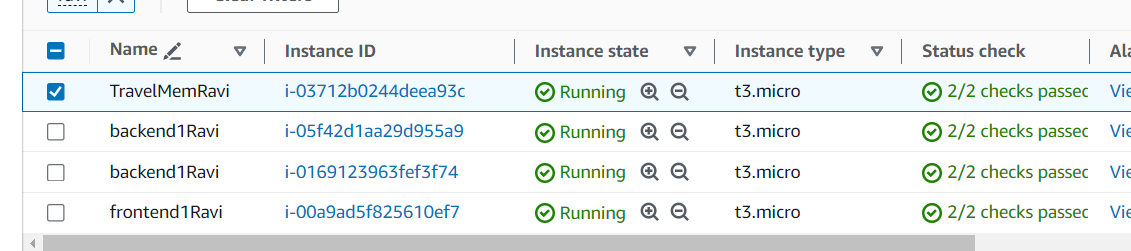
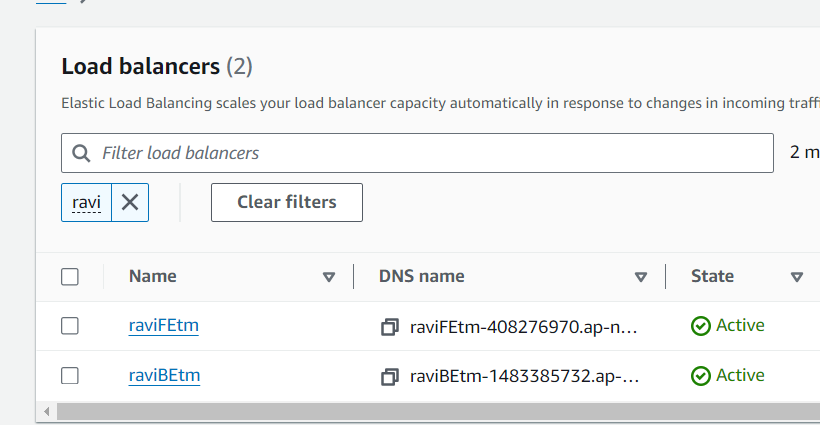
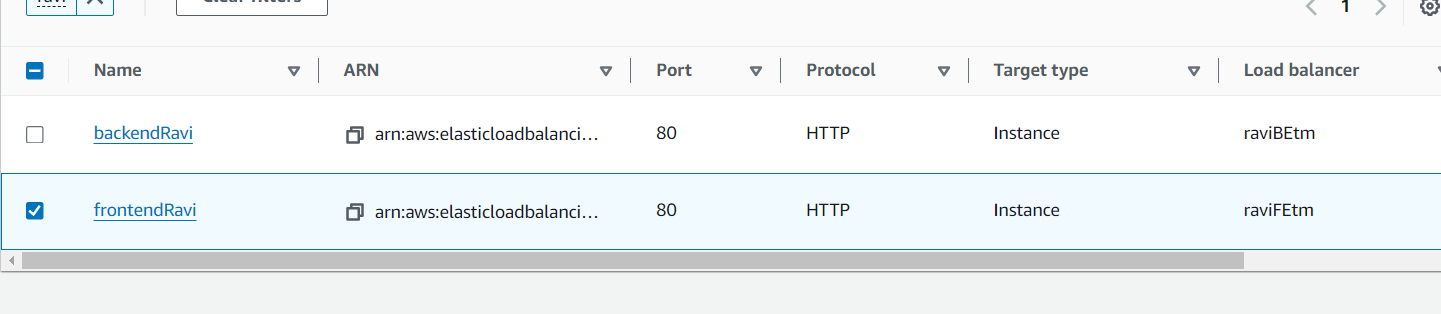
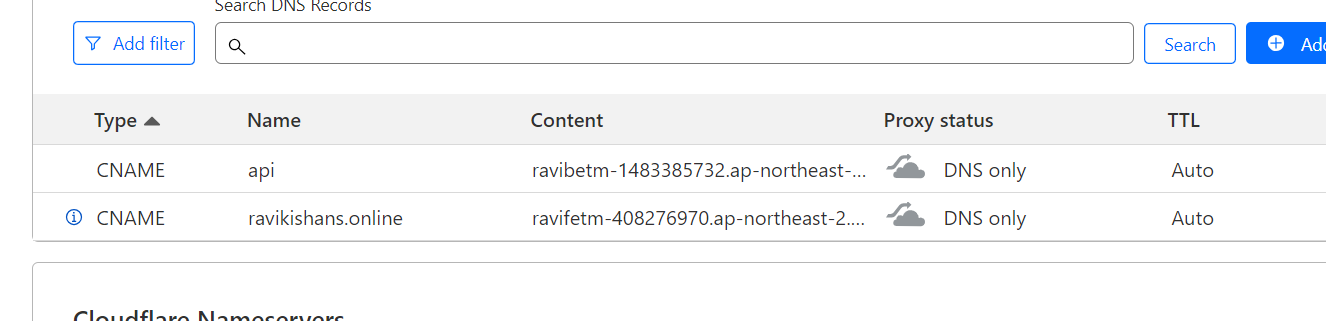
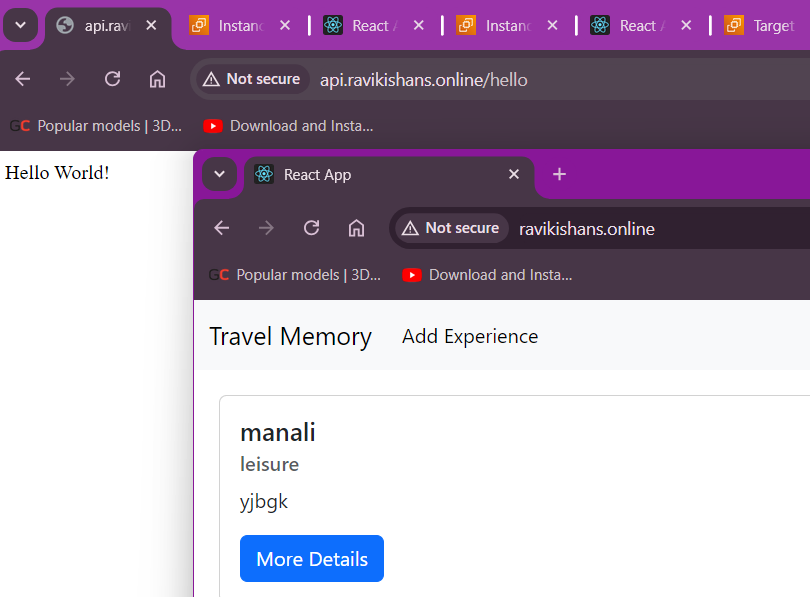
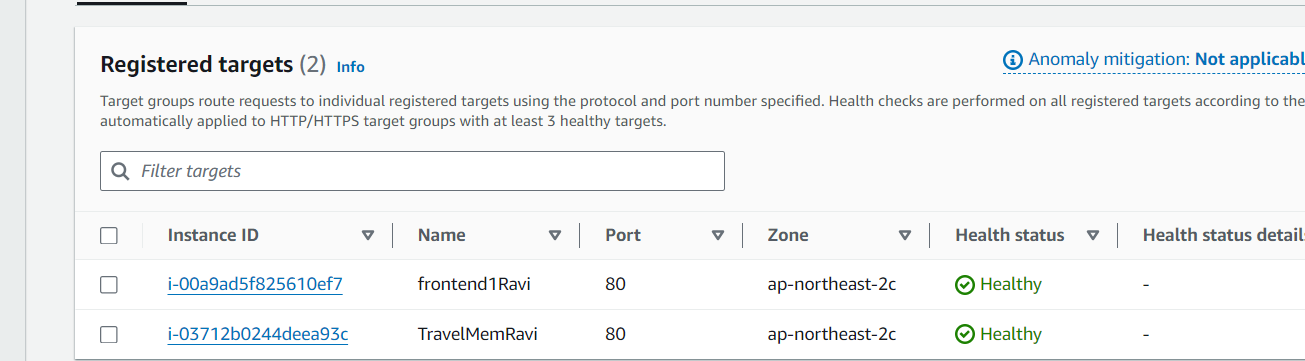
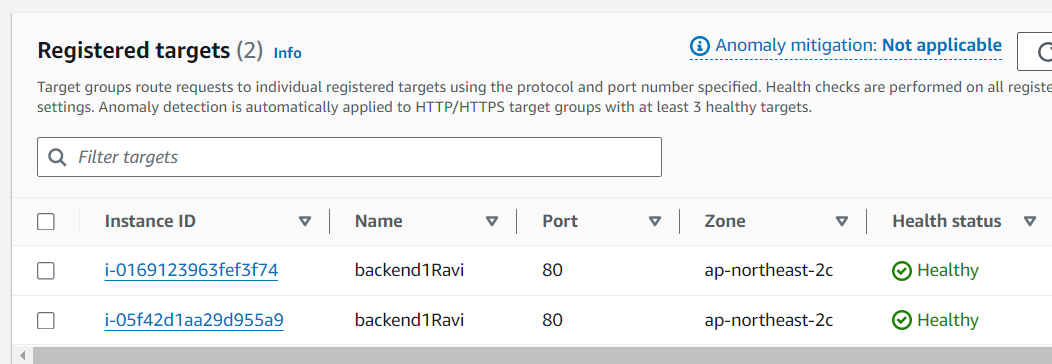
                proxy\_set\_header Connection 'upgrade';

                proxy\_set\_header Host $host;

                proxy\_cache\_bypass $http\_upgrade;

        }

}

1. Now similarly restart nginx, go to the frontend and run “npm start &”.
2. Now create AMI and launch multiple instances.
3. Now we have to create load balancers for frontend and backend. For connecting them to instances we have to create two target groups one for frontend instances and other for backend instances.
4. Now we have to configure DNS on cloudflare with load balancer;s “url” using CNAME.
5. Now we can see both the frontend and background working fine.
6. We can also do the health checks in both the target groups.
7. Adding draw.io workflow
8. 