

# Project Overview

## Introduction:

The development of a dynamic College Website was carried using a robust tech stack comprising Django, HTML, and CSS. To ensure optimal performance and scalability, we leveraged AWS infrastructure, employing an autoscaling group with a load balancer to handle traffic efficiently. Our database is securely housed within a private subnet, aligning with industry best practices for data protection.

The deployment process was streamlined using Terraform, guaranteeing consistency and reliability across our infrastructure. Ansible playbooks were instrumental in provisioning, maintaining, and configuring our systems, enabling seamless management. Additionally, we implemented a series of shell scripts to automate repetitive tasks, enhancing efficiency and reducing manual intervention.

All the code for the project can be found at  
[https://github.com/harjotbasota/Project\\_CollegeWebsite](https://github.com/harjotbasota/Project_CollegeWebsite)

This project showcases my skills in cloud deployment, IaC tools such as terraform and ansible, python, Django, HTML and CSS and automation using shell scripting

## Goals:

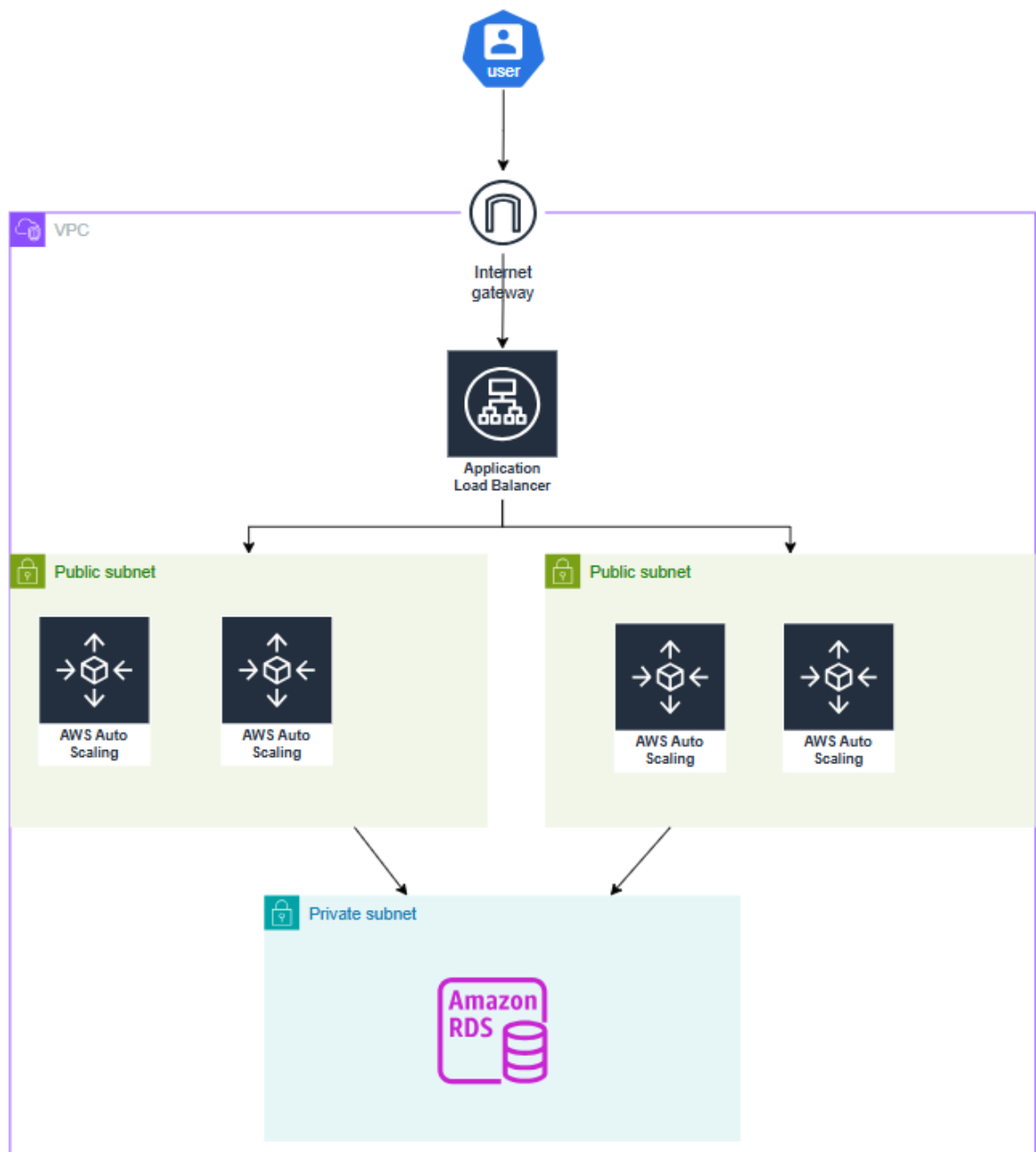
The Primary goal for this project is to build efficient and reliable infrastructure for our webapp which can be deployed with minimum interaction. On top of that project aimed at achieving **high availability and scalability** for seamless user experience. **IaC** tools are used to achieve **consistency** during the project. All the infrastructure is deployed according to security best practices.

## Technology Used:

- Ubuntu 22.04.4 LTS
- Django (Version 3.0)
- Ansible (Version 2.10.8)
- Terraform (Version 1.8..3)
- Boto3
- AWS VPC
- AWS EC2
- AWS Secrets Manager

- AWS RDS (Postgres)
- AWS AMI
- AWS Auto Scaling
- AWS Application Load Balancer
- Shell Scripting

Architecture:



## Deployment Overview:

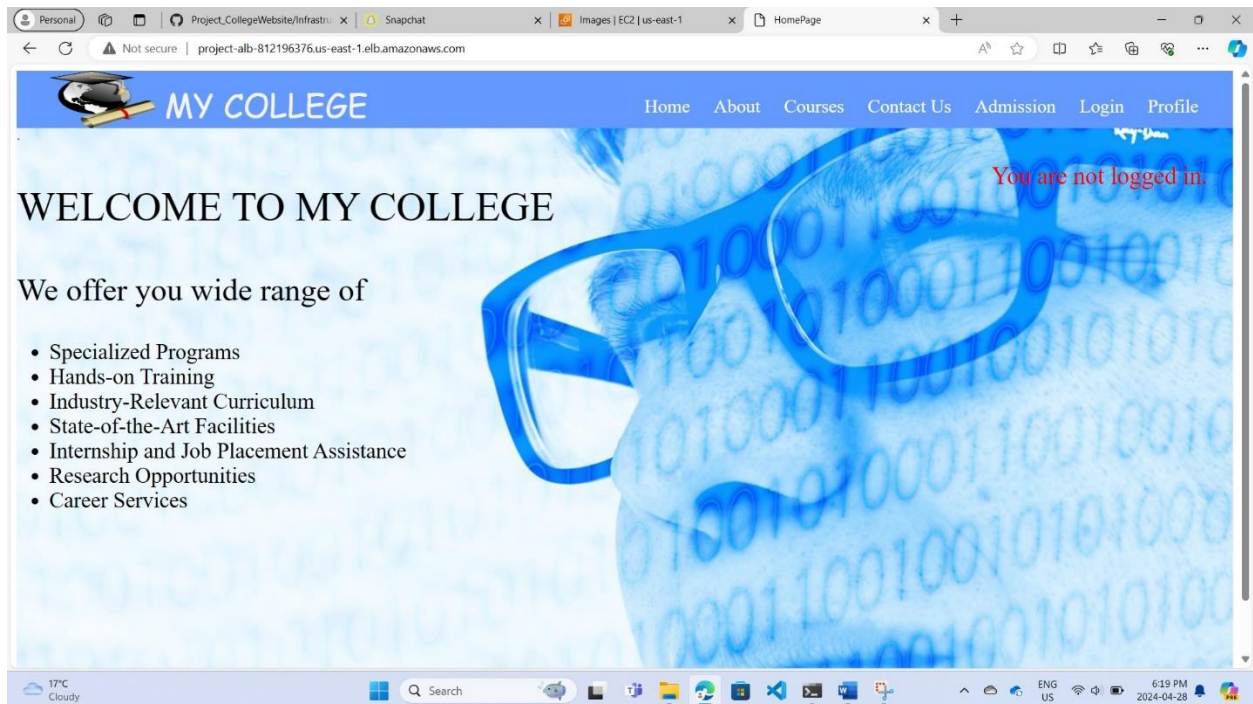
The Deployment is divided into seven phases

1. First step is to setup all the components of our VPC such as subnets, route tables, security groups and ACLs
2. Then we proceed to setup our database which will be used by our servers. This database will be provisioned to use AWS Secrets Manager for its credentials management. We will also setup script to dynamically send our database endpoint and secret name to Django application
3. Now we deploy single EC2 which we will use to make golden-ami that our Auto Scaling Group will so that we don't have to provision every instance in ASG. In this step we will also setup script which will build our inventory for ansible-playbook
4. In this step we will provision our EC2 instance using the ansible-playbook i.e, we will install all the required packages, copy the desired code and set up other necessary elements
5. Once our EC2 runs successfully and our application goes live we will build AMI from this instance
6. In this step we will build all the final components such as ALB and auto scaling group.
7. In the final step we will write a single script to run these step automatically one by one without requiring much interaction so that all our infrastructure can be deployed with single script. We will do the same to remove all resources from AWS

## Application Description:

This is rather a simple application which has few pages such as home, about page, contact information page, courses detail page. Along with that it has page for admission form which can be used take admission in the college. Once you submit the admission form you can login and access your profile from the profile page. After finishing you can logout. For security purposes I have used Mixins so that users can not access profiles of other users.

## End Result:



## Deployment and Infrastructure Details

1. As all the code is located on [https://github.com/harjotbasota/Project\\_CollegeWebsite](https://github.com/harjotbasota/Project_CollegeWebsite) so click here and go to readme.txt where I have written the steps to follow to deploy this application. We will follow these steps for our deployment

To launch this project

- This project needs aws region us-east-1 due to region name being hardcoded in django-app and provider block of all the terraform modules and other terraform code such as availability zones
  - Launch an ubuntu instance on aws( Ubuntu 22.04.4 LTS) which will serve as control node
  - Clone this github repo at /home/ubuntu/
  - Change the mod of setup.sh into executable( chmod +x setup.sh)
  - Set your aws credentials at ~/.aws/credentials as default profile
  - Go to Project\_CollegeWebsite/Infrastructure/ansible/secrets and write you credentials and sshkeys into the required files
  - execute the setup.sh to install all the requirements for this project
  - Before you launch the project you might also need to update the variable files in all terraform modules i.e, Project\_CollegeWebsite/Infrastructure/terraform.
- (NOTE -- some of the variables might need special attention such as ssh\_key\_name, ami for instance to use( use Ubuntu 22.04.4 LTS)
- Execute main.sh to launch this project
  - Use elb dns to access the website
  - Execute cleanup.sh to delete the resources created from this project

- Now we will go our AWS Console to build out control node. As mentioned in the README we will use us-east-1 region. We will deploy an ubuntu 22.04.4 LTS machine with public IP and keep port 22 open for ssh.

i-072c8ad4715482ee9 (control-node)

⚙️ ✕

Details

Status and alarms New

Monitoring

Security

Networking

Storage

Tags

▼ Instance summary [Info](#)

<div>Instance ID</div> <div>📄 i-072c8ad4715482ee9 (control-node)</div>	<div>Public IPv4 address</div> <div>📄 44.203.14.39   <a href="#">open address</a></div>	<div>Private IPv4 addresses</div> <div>📄 172.31.211.53</div>
<div>IPv6 address</div> <div>-</div>	<div>Instance state</div> <div>🕒 Pending</div>	<div>Public IPv4 DNS</div> <div>📄 ec2-44-203-14-39.compute-1.amazonaws.com   <a href="#">open address</a></div>
<div>Hostname type</div> <div>IP name: ip-172-31-211-53.ec2.internal</div>	<div>Private IP DNS name (IPv4 only)</div> <div>📄 ip-172-31-211-53.ec2.internal</div>	
<div>Answer private resource DNS name</div>	<div>Instance type</div>	<div>Elastic IP addresses</div>

- After our EC2 is ready we ssh

```

ubuntu@ip-172-31-211-53:~$ ls
ubuntu@ip-172-31-211-53:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.4 LTS
Release:        22.04
Codename:       jammy
ubuntu@ip-172-31-211-53:~$

```

4. As our control-node is ready we can now clone this github repository at ~  
(NOTE: make sure to clone repository only at ~ location as the paths have been defined according to this). In Project\_CollegeWebsite we have following:

**ConfigFiles/** : This contains configuration files for nginx and gunicorn

**Infrastructure/** : This directory contains all the IaC for our project divided into two sub directories named terraform and ansible

**Django/** : This directory contains all the django application code

**Setup.sh** : This is script to install the required packages on control-node

**main.sh** : This script builds all the infrastructure

**cleanup.sh** : This script destroys all the infrastructure

```

ubuntu@ip-172-31-211-53:~$ git clone https://github.com/harjotbasota/Project_CollegeWebsite.git
Cloning into 'Project_CollegeWebsite'...
remote: Enumerating objects: 381, done.
remote: Counting objects: 100% (381/381), done.
remote: Compressing objects: 100% (235/235), done.
remote: Total 381 (delta 148), reused 353 (delta 122), pack-reused 0
Receiving objects: 100% (381/381), 1.63 MiB | 28.29 MiB/s, done.
Resolving deltas: 100% (148/148), done.
ubuntu@ip-172-31-211-53:~$ ls
Project_CollegeWebsite
ubuntu@ip-172-31-211-53:~$

```

5. Go to Project\_CollegeWebsite directory and run `chmod +x setup.sh` and execute the script. This script will install the packages with required versions which we will use in this project. This script installs us terraform, ansible and boto3. This script will also make out main.sh and cleanup.sh executable. After this check that ansible and terraform has been installed successfully or not.

```

ubuntu@ip-172-31-211-53:~/Project_CollegeWebsite$ terraform --version
Terraform v1.8.2
on linux_amd64
ubuntu@ip-172-31-211-53:~/Project_CollegeWebsite$ ansible --version
ansible 2.10.8
  config file = None
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
ubuntu@ip-172-31-211-53:~/Project_CollegeWebsite$

```

6. Now we have to setup our access keys for terraform at ~/.aws/credentials . This is default location where terraform looks for credentials so we do not need any further configurations.

```

[default]
aws_access_key_id =
aws_secret_access_key =
~

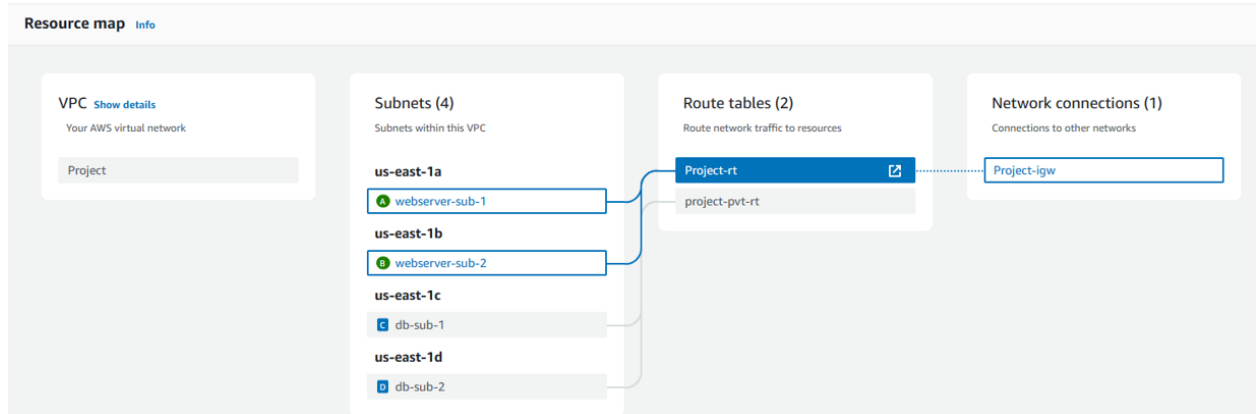
```

7. Go to Project\_CollegeWebsite/Infrastructure/ansible/secrets and write you credentials and sshkeys into the required files. The ssh keys file will be used by ansible to ssh into your instance which we will use to create our golden-ami. In this step you also need to update the ssh key name in Infrastructure/terraform/ami/terraform.tfvars with the name of key you copied in ansible/secrets.sshkey.pem . (NOTE: This is the key which we will attach to our ami instance not for the control-node) The credentials file will be copied to ~/.aws/credentials in your golden-ami instance
8. Have a look at all the Infrastructure/terraform/../../\*.tfvars files so that it is in accordance with you.
9. Now execute main.sh script. It will start the deployment process. It will deploy all the six phases all at once. Below is detailed breakdown of each phase

### **PHASE 1:**

- First thing we will create is our VPC itself. It will named as Project
- Then we will create four subnets for our VPC. Two subnets are public and two are private. Public subnets are to be used for autoscaling group and private subnets are to be used with RDS
- We will create one internet gateway and attach it to our VPC
- We will create to route tables one with route to internet gateway which will be for public subnets and other without route to IGW which will be for private subnets.





- We also create ACL for our VPC which allows all incoming traffic and all outgoing traffic
- For this project we need three different security groups
  - Alb-sg:** Which allows all incoming and outgoing traffic on port 80. This group will be attached with Load Balancer
  - Webserver-sg:** This group will accept traffic on port 80 from alb-sg and allow all outbound traffic on port 80 and port 443. Port 443 is used to connect with secrets manager but we are not receiving any traffic on port 443. By doing this our servers could be accessed only through DNS of load balancer can't be accessed directly from public IP.
  - Db-sg:** This group allows traffic on port 5432 which is default for postgres. It can accept and send traffic to this group itself but none others. So we attach it to database and webserver both

## PHASE 2:

- In this phase we will setup our database instance.
- First of all, we will create subnet group with our two private subnets created during the phase 1
- Then we will write a kms policy which will be attached to our secret and will be responsible for encryption, decryption, get secret and other kms related functions
- After this we will setup our database instance which will use Secrets Manager for storing credentials. We will setup our initial database as users and will make it publicly inaccessible
- Once our database instance is created we will execute `setvars.sh` script. This script will modify the terraform outputs for database name and secret ARN according to needs of Django application. After that it will write (over-write if previous values exist) the required values in Django directory under the name of `db_name.txt` and `secret_name.txt`. These values will be used by application to get endpoint of our rds instance and use secret name to retrieve the credentials from there.

Summary

DB identifier project-rds	Status Available	Role Instance	Engine PostgreSQL
CPU -	Class db.t3.micro	Current activity	Region & AZ us-east-1d

Connectivity & security

Monitoring

Logs & events

Configuration

Maintenance & backups

Tags

Recommendations


Connectivity & security

Endpoint & port

Networking

Security

Endpoint

 project-rds.cju0o3gckvpy.us-east-1.rds.amazonaws.com

Port

5432

Availability Zone

us-east-1d

VPC

Project (vpc-0abf3471cf726fa2d)

Subnet group

project-subnet-group

Subnets

[subnet-0a07e30fb247f1883](#)  
[subnet-09e916b580a60abd4](#)

Network type

IPv4

VPC security groups

[project-db-sg \(sg-065de6d95eeebecf\)](#)  
 Active

Publicly accessible

No

Certificate authority

[Info](#)  
 rds-ca-rsa2048-g1

Certificate authority date

May 25, 2061, 19:34 (UTC-04:00)

DB instance certificate expiration date

April 28, 2025, 16:56 (UTC-04:00)

### PHASE 3:

- In this phase we will create an ec2 instance which we will provision using ansible and then make AMI out of it
- Here we only create the ec2 instance and add an additional security group which opens port 22 so that ansible can connect with the instance. This will be created in our Project Vpc
- Once instance is created we will execute inventory.sh to rewrite our ansible inventory with IP of now created instance.

i-0728f9aef9e948d62 (ami-instance)

Details

Status and alarms New

Monitoring

Security


Networking

Storage

Tags

▼ Instance summary [Info](#)

Instance ID

 i-0728f9aef9e948d62 (ami-instance)

IPv6 address

-


Hostname type

IP name: ip-20-0-1-12.ec2.internal


Answer private resource DNS name

-

Auto-assigned IP address

 54.145.163.157 [Public IP]


Public IPv4 address

 54.145.163.157 | [open address](#)

Instance state

Running


Private IP DNS name (IPv4 only)

 ip-20-0-1-12.ec2.internal


Instance type

t2.micro

VPC ID

 vpc-0abf3471cf726fa2d (Project)

Private IPv4 addresses

 20.0.1.12


Public IPv4 DNS

-

Elastic IP addresses

-

AWS Compute Optimizer finding

 [Opt-in to AWS Compute Optimizer for recommendations.](#)  
[Learn more](#)

### Outputs:

```
ami-instance-id = "i-0728f9aef9e948d62"
ami-instance-public-ip = "54.145.163.157"
Ansible inventory has been updated to

[webservers]
54.145.163.157
```

### PHASE 4:

- Now we will provision our instance with help of ansible-playbook

```
TASK [Updating Cache] *****
changed: [54.145.163.157]

TASK [Copy django app to server] *****
changed: [54.145.163.157]

TASK [Creating an aws directory to store credentials] *****
changed: [54.145.163.157]

TASK [Adding credentials to server for aws access] *****
changed: [54.145.163.157]

TASK [Installing all the required packages] *****
changed: [54.145.163.157]

TASK [Migrating the database] *****
changed: [54.145.163.157]

TASK [Migrating the database] *****
changed: [54.145.163.157]

TASK [Creating the gunicorn service file] *****
changed: [54.145.163.157]

TASK [Creating directory for gunicorn sock file] *****
```

- This playbook will do the following things.
  - Copy our Django app to our server. It will only copy Django directory and all its files and sub directories but not other files and directory such as Infrastructure, main.sh etc as we don't need them on our server
  - It will put our credentials on ~/.aws/credentials in our ami-instance
  - It will install all the packages necessary for project i.e, Django, gunicorn , nginx, boto3 , psycopg2

4. It will run the Django database migration commands and will connect to our database using our RDS endpoint and secret name. We have a script running inside settings.py which will read the desired files to retrieve these values and also get the credentials from secrets manager
5. It will copy our gunicorn and nginx configuration files at the desired location
6. It will run commands to make static files available for nginx
7. It will restart nginx and gunicorn so that they are working with latest configurations
- Once done we can hit public IP of ami-instance to check if our application has gone live successfully

## PHASE 5:

- If our application is live we can create an golden-ami from our instance

AMI ID: ami-0535ccbec44ff1312			
Details	Permissions	Storage	Tags
AMI ID ami-0535ccbec44ff1312	Image type machine	Platform details Linux/UNIX	Root device type EBS
AMI name golden-ami	Owner account ID 801756726903	Architecture x86_64	Usage operation RunInstances
Root device name /dev/sda1	Status Available	Source 801756726903/golden-ami	Virtualization type hvm
Boot mode uefi-preferred	State reason -	Creation date Sun Apr 28 2024 17:02:35 GMT-0400 (Eastern Daylight Time)	Kernel ID -
Description -	Product codes -	RAM disk ID -	Deprecation time -
Last launched time	Block devices	Deprecation protection	

## PHASE 6:

- We will start this phase by creating a template which has to be used by our auto scaling group. This template is based on our golden-ami
- After that we will proceed to create a target group to which our load balancer will route traffic on port 80
- Now we will create a load balancer
- We will create an auto-scaling group which will scale based on CPU utilization of our instances.
- Once all the script is run we will get alb-dns as output

```
pp/project-alb/4b53e757a94457f9]
aws_lb_listener.listener: Creating...
aws_lb_listener.listener: Creation complete after 0s [id=arn:aws:elasticloadbalancing:us-east-1:123456789012:listener/app/project-alb/4b53e757a94457f9/af35d4544c220a2c]

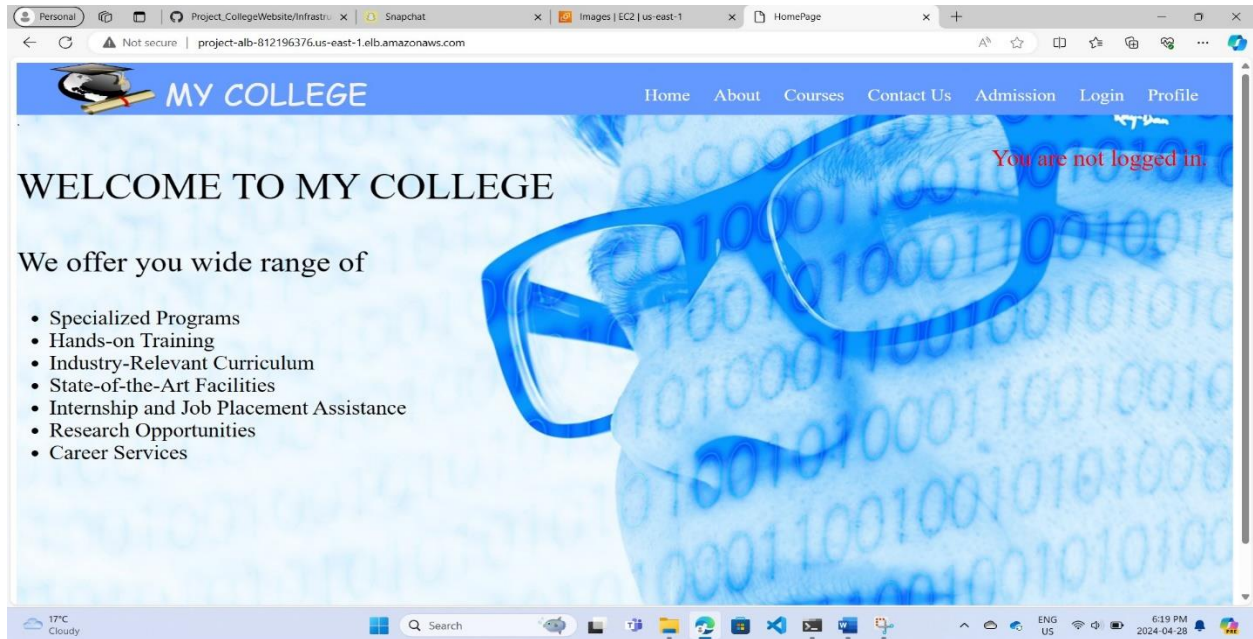
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:

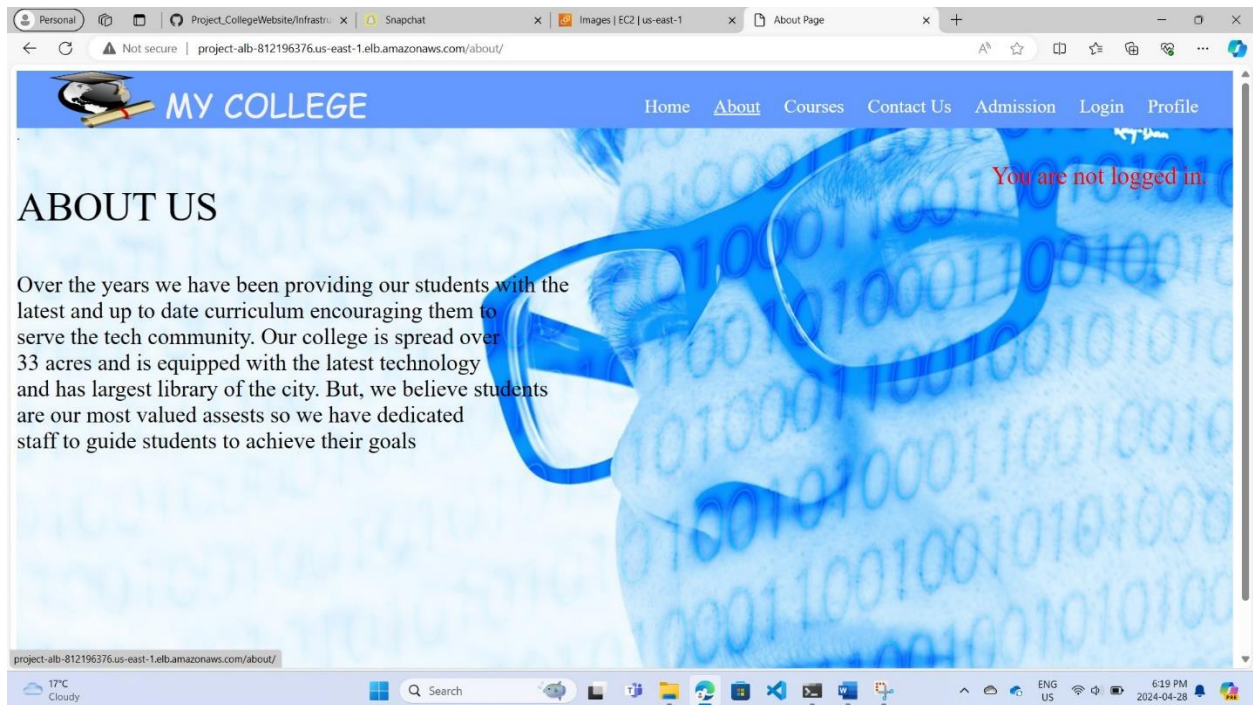
alb-dns = "project-alb-812196376.us-east-1.elb.amazonaws.com"
ubuntu@ip-172-31-211-53:~/Project_CollegeWebsite/Infrastructure/terraform/asg$
```

- When we hit this dns will be able to access our application

## HOME PAGE



## ABOUT PAGE





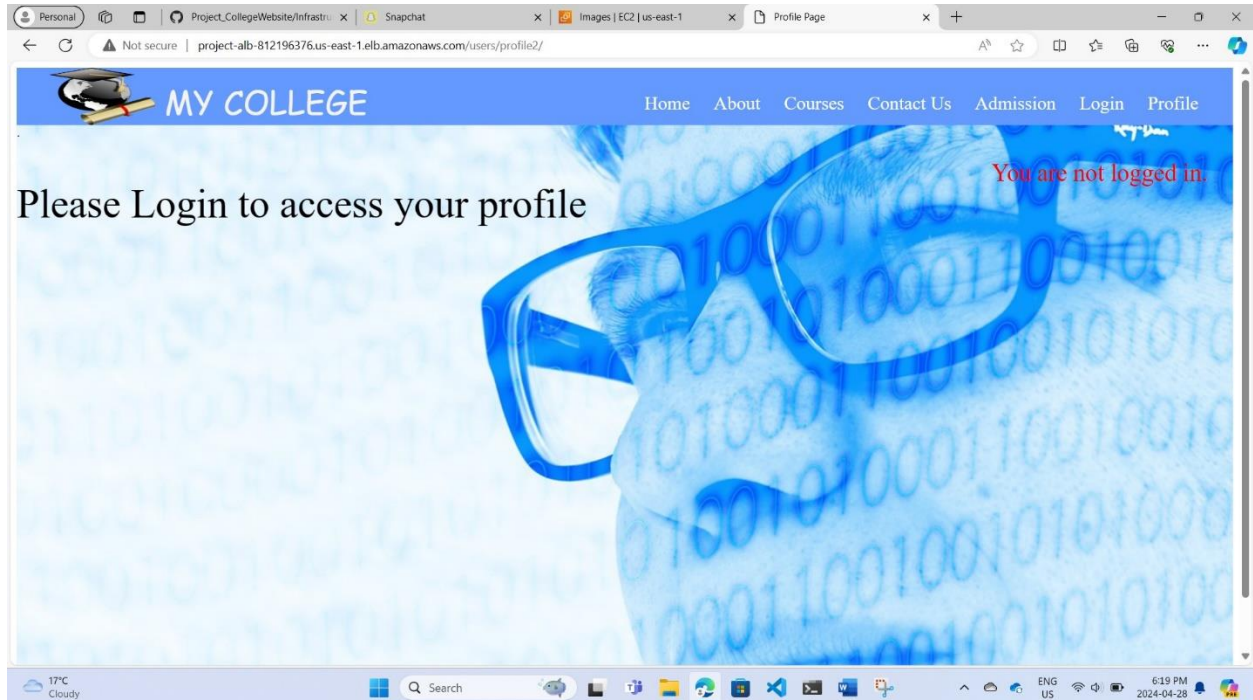
## COURSES PAGE



## CONTACT PAGE



## PROFILE PAGE( When not logged in)



## ADMISSION PAGE

The screenshot shows the "Admission Form" page of the MY COLLEGE website. The URL is `project-alb-812196376.us-east-1.elb.amazonaws.com/users/admissions/`. The page layout is consistent with the profile page, featuring the same header and background image. A red text overlay in the top right corner reads "You are not logged in". The main heading is "Fill Up details of Admission Form". The form includes the following fields and instructions:

- Username:**  Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.
- Course:**
- Email address:**
- Date of birth:**
- Password:**

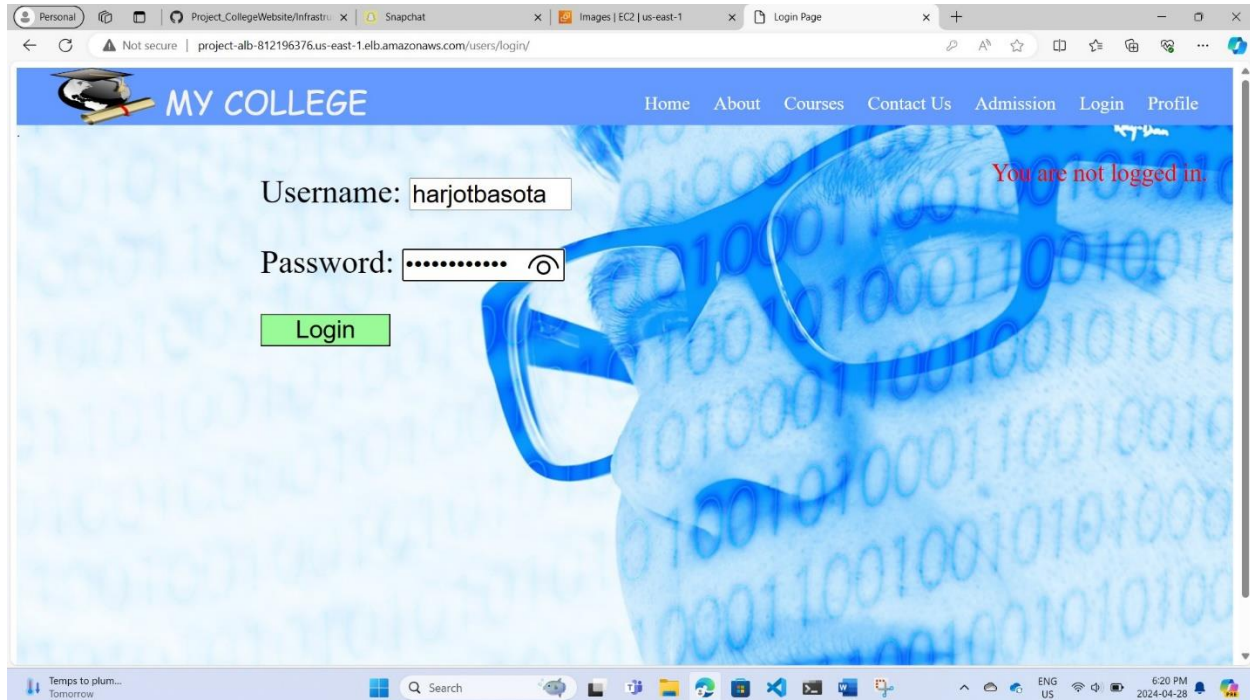
Below the password field, there are four bullet points providing password requirements:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

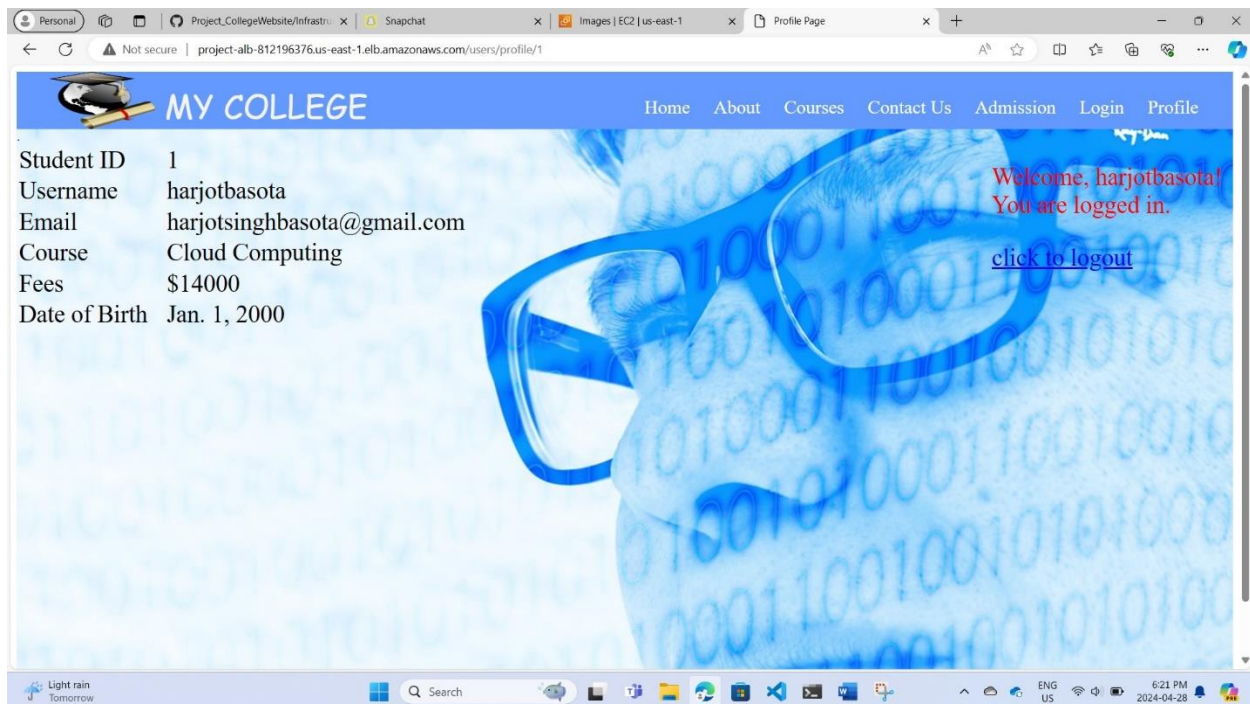
The form also includes a **Password confirmation:**  field with the instruction "Enter the same password as before, for verification." and a green **Save** button at the bottom.



## LOGGING IN



## PROFILE PAGE (When logged in)



NOTE : EXECUTE BASH CLEANUP TO DESTROY ALL THE RESOURCES IN END



