SPE Major Project Report

Web-Application : Brain Tumor Analysis

- IMT2020094 Riddhi Chatterjee
- IMT2020034 Ankrutee Arora

Link to GitHub Repo: https://github.com/Riddhi-Chatterjee/Brain-Tumor-Analysis

Link to Frontend DockerHub Repo:
https://hub.docker.com/repository/docker/riddhich/brain-tumor-analysis-frontend/general

Link to Backend DockerHub Repo:
https://hub.docker.com/repository/docker/riddhich/brain-tumor-analysis-backend/general

About the Project:

The primary goal of this project is to develop an integrated system that accurately detects brain tumors, performs classification and segmentation to identify and locate brain tumors, predicts survival probabilities, and presents these functionalities through a user-friendly web interface.

Currently, only the brain tumor detection and classification tasks have been implemented. We have left the tasks like brain tumor segmentation, and survival prediction as future scope for this project.

Instructions to use our application:

- We can get our web application up and running by scheduling a build in our corresponding Jenkins project. It will run two containers – one for backend and one for frontend. After that, we can access our application at http://127.0.0.1:3000/
- We can manually run our application as well:
    - Clone our GitHub repo.
    - Switch to the backend directory.
    - Run the download_models.py script (Use command “python3 download_models.py”) to download the required ML models which we have trained on our own and stored in Google Drive.
    - Run the backend.py script (Use command “python3 backend.py”) to start our backend application.
    - Switch back to the parent directory and then cd into the frontend directory
    - Run the frontend.py script (Use command “python3 frontend.py no_cont”) to start our frontend application.
    - We can now access our application at http://127.0.0.1:3000/

Why This Project?

The choice of Brain Tumor detection, segmentation, and survival prediction stems from the critical need in healthcare for accurate and timely diagnosis. Early and accurate diagnosis of brain tumors is critical for timely treatment, and machine learning aids in achieving this by analyzing medical imaging data with high precision. Brain tumors are a severe health concern, and their early detection significantly improves treatment outcomes. Leveraging machine learning for this purpose not only enhances accuracy but also enables quicker and more precise diagnosis.

What is DevOps?

DevOps, a fusion of development and operations, revolutionizes software delivery by promoting collaboration, automation, and continuous feedback between traditionally siloed development and operations teams. In contrast to traditional practices, DevOps emphasizes seamless communication, automation of manual tasks through continuous integration and delivery, infrastructure as code for efficient management, and a culture of shared responsibility and continuous improvement. The focus on monitoring and feedback loops allows early issue detection, while the adoption of automation streamlines processes, leading to faster and more reliable software releases. DevOps represents a cultural and technical evolution, fostering a dynamic and collaborative environment for the entire software delivery lifecycle.

Understanding ML-Ops:

ML-Ops, short for Machine Learning Operations, refers to the amalgamation of practices, tools, and frameworks that streamline the lifecycle management of machine learning models. It encompasses the deployment, monitoring, versioning, and scaling of machine learning models in a production environment. ML-Ops ensures that machine learning systems are efficiently developed, deployed, and managed to deliver accurate and reliable results consistently.
It incorporates DevOps principles tailored specifically for machine learning systems, ensuring reproducibility, scalability, and reliability throughout the ML model lifecycle.

Why is MLOps needed for this project?

MLOps is essential for the success of our project as it provides a framework for deploying machine learning models into production environments, ensuring scalability, efficiency, and continuous monitoring. MLOps practices enable reproducibility, version control, and collaboration among diverse teams, including data scientists, software engineers, and healthcare professionals. Continuous Integration and Continuous Deployment (CI/CD) ensure quick testing and deployment of model updates, addressing issues promptly, while strict data security and compliance measures are implemented to protect sensitive healthcare information. MLOps also facilitates fault tolerance, disaster recovery, and the efficient handling of large volumes of medical imaging data, crucial for accurate and timely diagnosis. Overall, MLOps

plays a pivotal role in bridging the gap between model development and deployment, ensuring the reliability and ethical use of machine learning in critical healthcare applications.

Plan of Action being followed:

Data Collection -> Data Preprocessing ->Model Development ->Model Evaluation -> Web App Dev -> Backend Integration -> Testing -> Deployment

Building the Required Models

The initial phase involves the comprehensive development of machine learning models specialized in brain tumor detection, classification, segmentation, and survival prediction. This process encompasses data acquisition, preprocessing techniques, meticulous model selection, extensive training, thorough evaluation, and iterative optimization to ensure high accuracy and robust performance.

Integrating Models with the Web Interface

A critical aspect of the project is the integration of developed machine learning models into the web interface. This integration ensures user-friendly access to the models' functionalities, enabling medical practitioners or end-users to interact with the system effortlessly.

Configuring the Jenkins Pipeline

The project plan involves the incorporation of Continuous Integration/Continuous Deployment (CI/CD) tools, notably Jenkins, to automate the deployment process. Jenkins pipelines will be configured to facilitate the seamless deployment and testing of the developed models. However, detailed specifics regarding the tool's utilization will be provided in subsequent project updates.

Configuring the Web Interface

The web interface is pivotal in facilitating user interactions and visualizations related to the tasks at hand. This phase involves the strategic configuration of the user interface to seamlessly accommodate functionalities for tumor detection, segmentation visualization, and input/output mechanisms for survival prediction.

Models used:

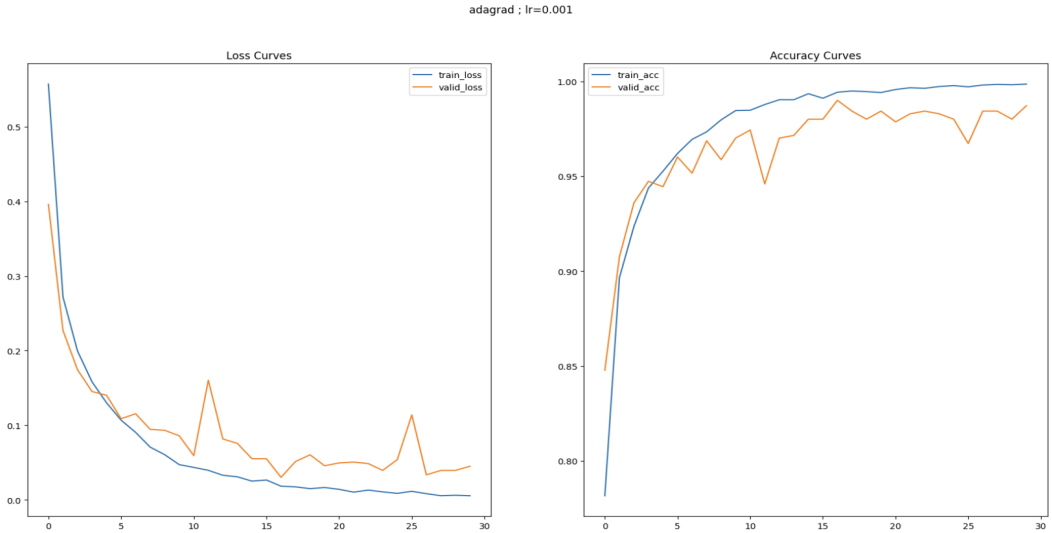
Tumor Classification:

The VGG16 brain tumor classification model is a deep convolutional neural network (CNN) specifically designed for the accurate identification of brain tumors in MRI images. This model is trained to classify images into four distinct classes: "glioma," "meningioma," "pituitary," and "no tumor". VGG16, known for its depth and simplicity, consists of 16 weight layers, including 13

convolutional layers and three fully connected layers. Its hierarchical structure enables the extraction of intricate features from the input MRI images, capturing both low and high-level patterns crucial for tumor classification. Our model gives 98.72% classification accuracy on unseen test data:

```
Best epoch: 16 with loss: 0.03 and acc: 98.72%
4150.07 total seconds elapsed. 138.34 seconds per epoch.
```

```
loss_acc_visuaize(history,optim="adagrad ; lr=0.001",path="../lr=0.001")
```



Link to our Kaggle notebook: https://www.kaggle.com/code/riddhich/brain-tumor-classification

Tumor Segmentation:

Though this feature is not yet integrated with our application, we have built a model which can perform Brain Tumor Segmentation. The tumor segmentation notebook utilizes a 2D U-Net architecture, a popular framework for semantic segmentation tasks. This architecture comprises an encoder-decoder structure with skip connections, allowing precise pixel-level delineation of tumors in MRI images. It takes an input MRI image and generates pixel-wise segmentation masks, outlining the tumor regions.

Link to our Kaggle notebook: https://www.kaggle.com/code/ankruteearora/tumor-segmentation

Datasets used:

Tumor Classification:

We have used the Brain Tumor MRI Dataset for Brain Tumor Classification. This dataset contains MRI images and their corresponding labels (i.e. tumor type). We have four distinct labels in this dataset: "glioma," "meningioma," "pituitary," and "no tumor".

Link to our Kaggle dataset:
https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/

Tumor Segmentation:

The dataset for tumor segmentation is curated from the BraTS2020 dataset, primarily focused on glioma segmentation. However, it is adaptable for general tumor segmentation tasks. It comprises MRI scans along with corresponding segmented masks, facilitating the training of segmentation models to accurately delineate tumor boundaries.

Link to our Kaggle dataset:
https://www.kaggle.com/datasets/awsaf49/brats20-dataset-training-validation/

Description of the Web App:

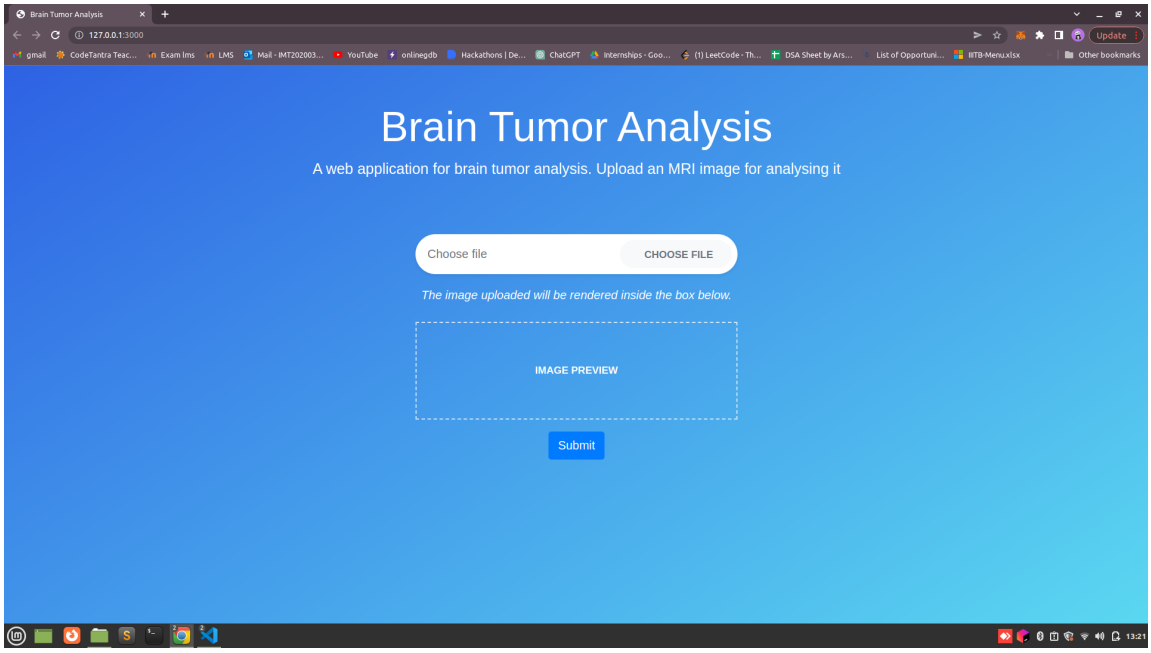
The web application is designed to provide a user-friendly interface for interacting with the developed machine learning models for brain tumor detection and segmentation. It will encompass several components:

Components of the Web App:

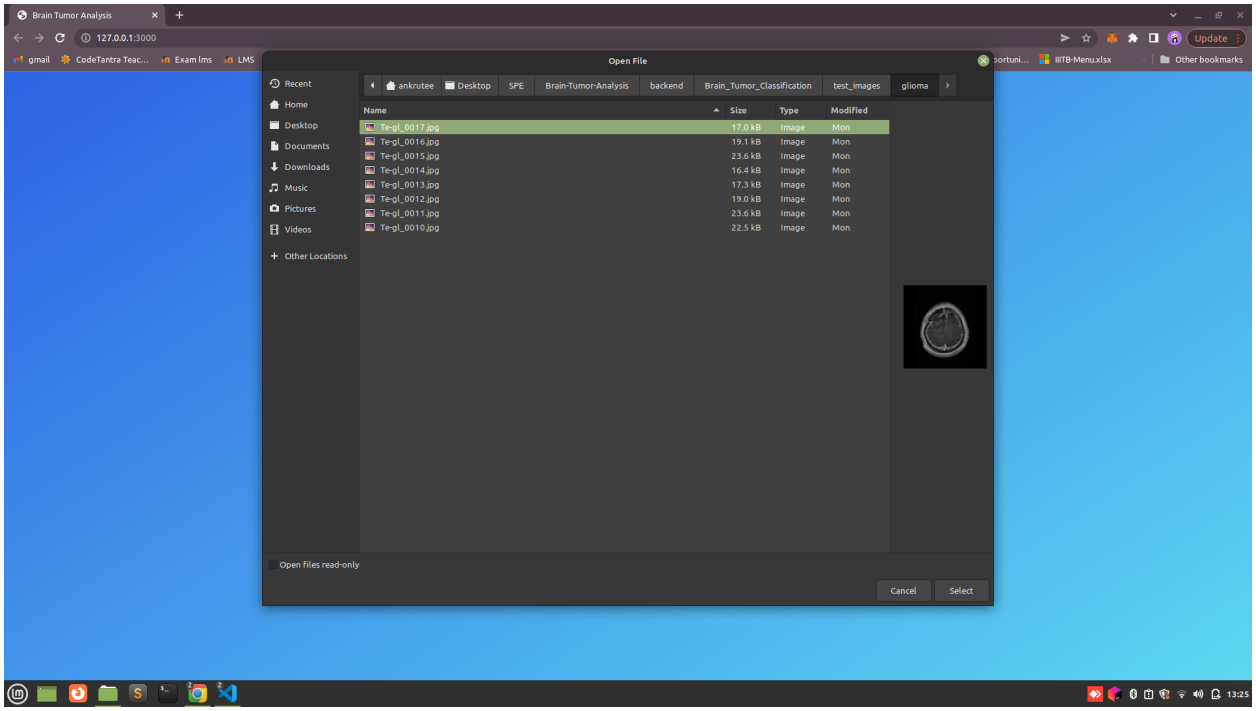
1. Homepage :This comprises of the main content where the user can add images and get their predicted outputs.
2. Tumor Classification Module: This module allows users to upload MRI images, which are then processed by the classification model. It displays the predicted tumor type or class based on the input image.
3. Tumor Segmentation Module: Users can upload MRI images to this module, and the segmentation model generates and displays masks outlining the tumor regions in the uploaded images.(Future scope)
4. Visualization Tools: Visual representations, such as image sliders, overlays of segmented masks on original images, or probability heatmaps, aiding in better understanding the model outputs.
5. User Inputs and Outputs: Interactive components enabling users to interact with the models, providing input images, and receiving model predictions or segmented outputs.
6. Result Display: A section to present the model predictions or segmented images in an easily interpretable format.

Application walk-through:

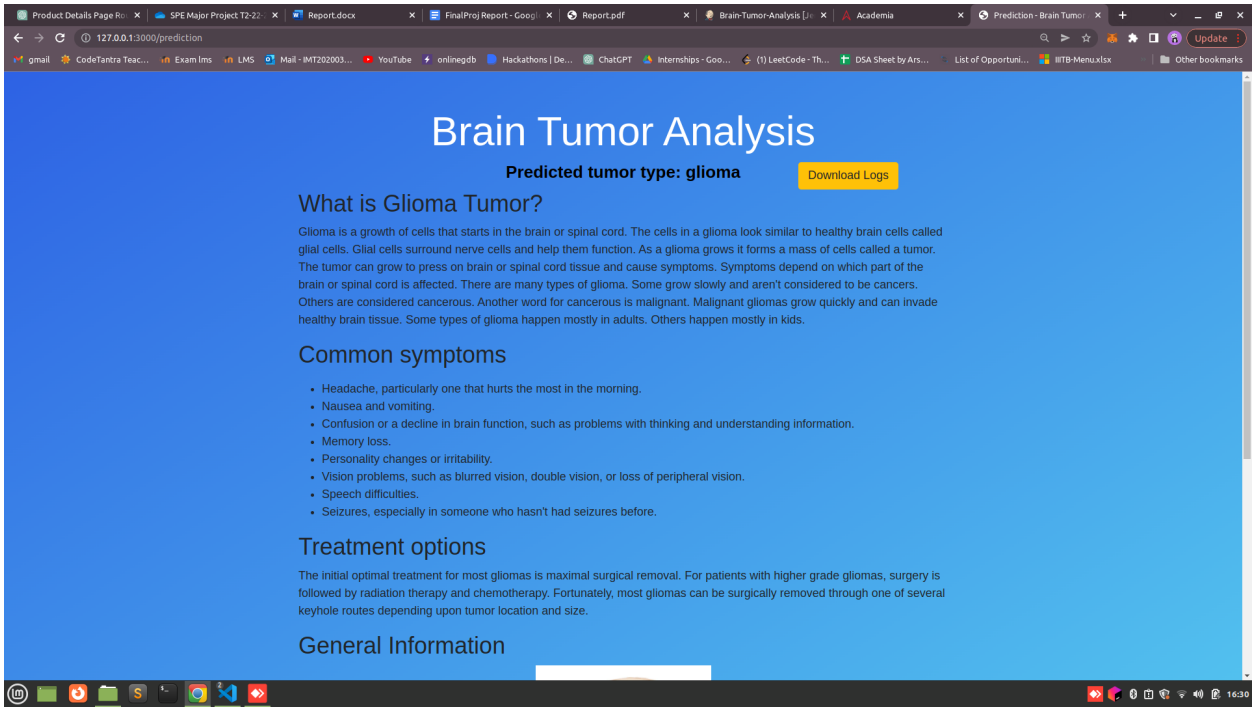
1. The main page:

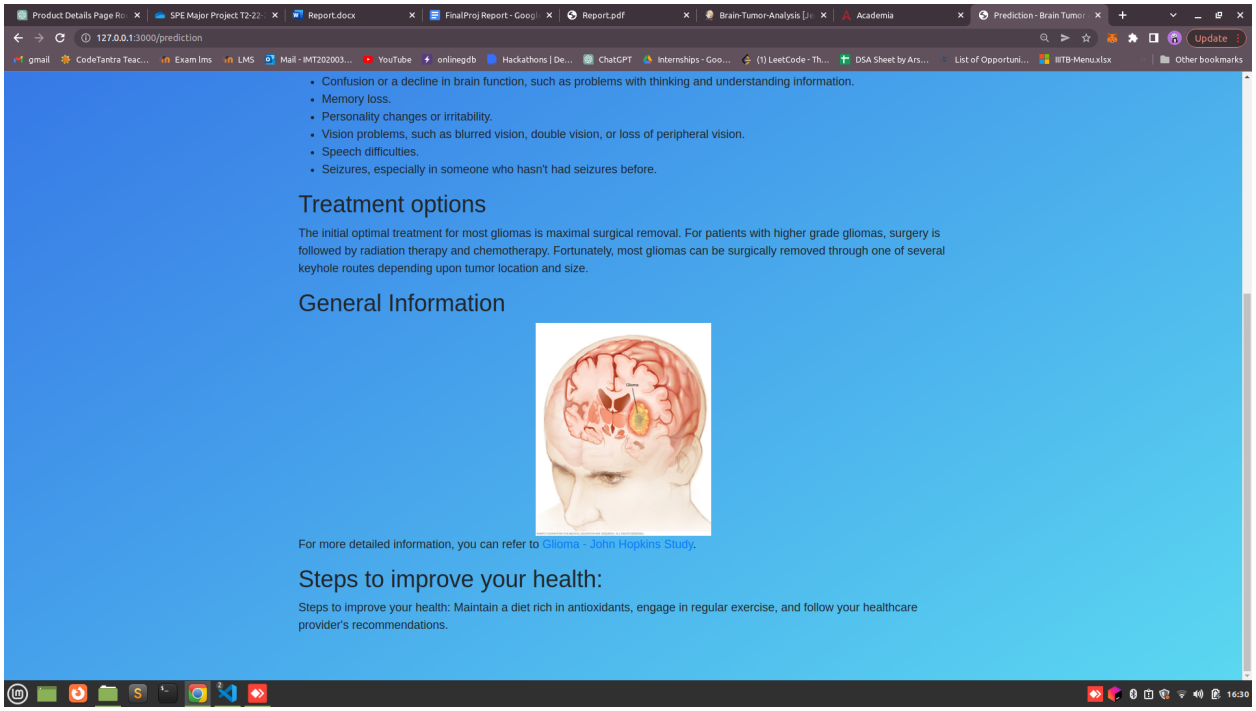


After clicking on the choose file button, we select images from our dataset.
Since we have a trained model, I have images of 3 types of tumors and below are the identification of each of them.
i) Glioma:
I select an image which has the glioma tumor.

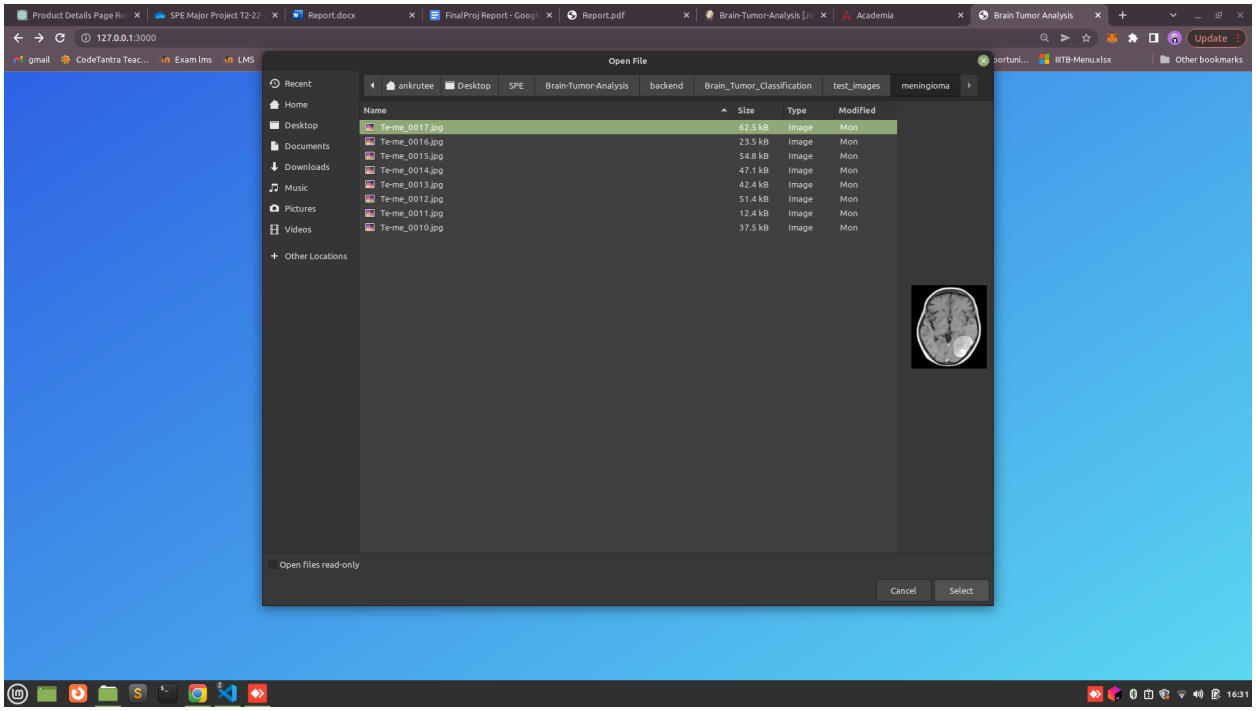


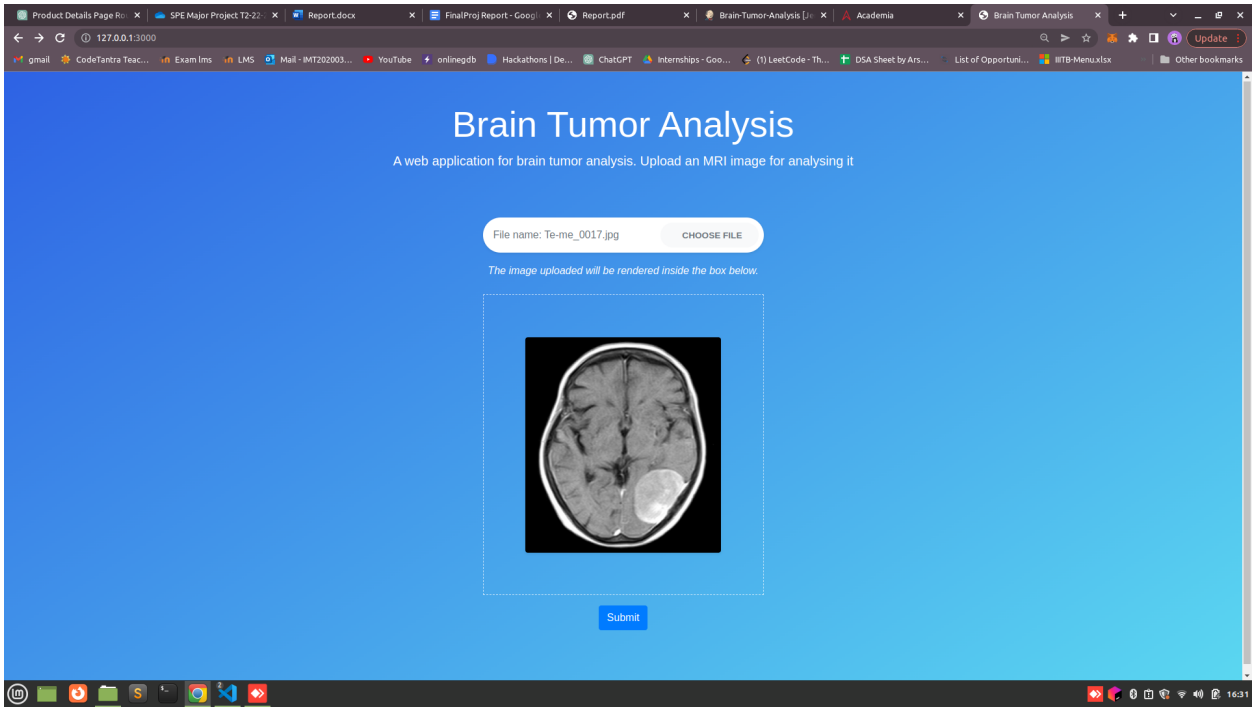
This is the result that is shown. It tells us about the Predicted tumor type, what is Glioma, its symptoms , treatment options and general information and safety precautions.



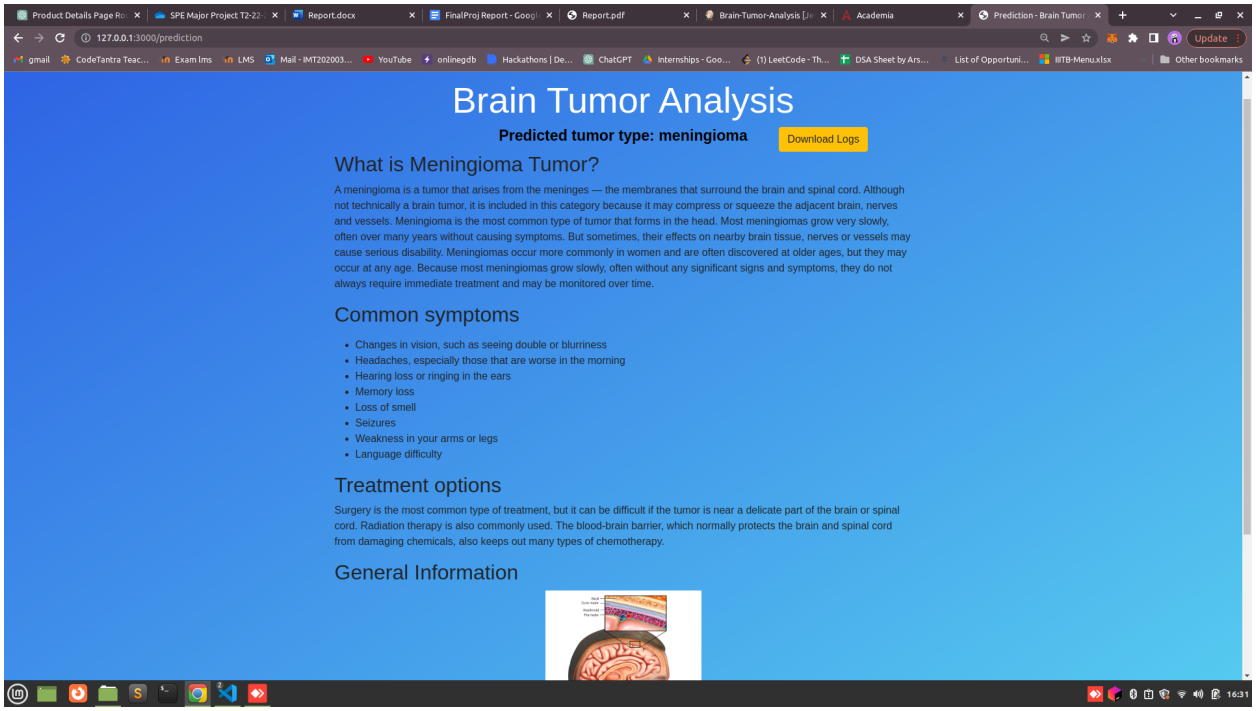


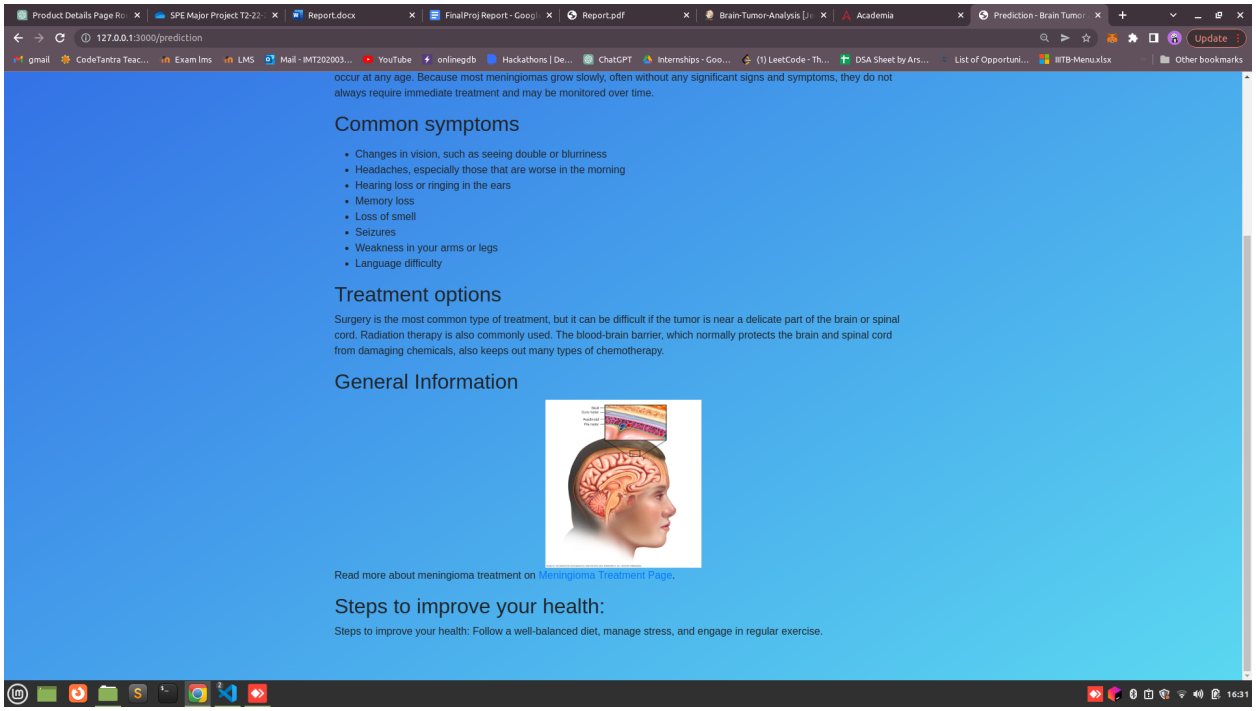
ii)Meningioma:



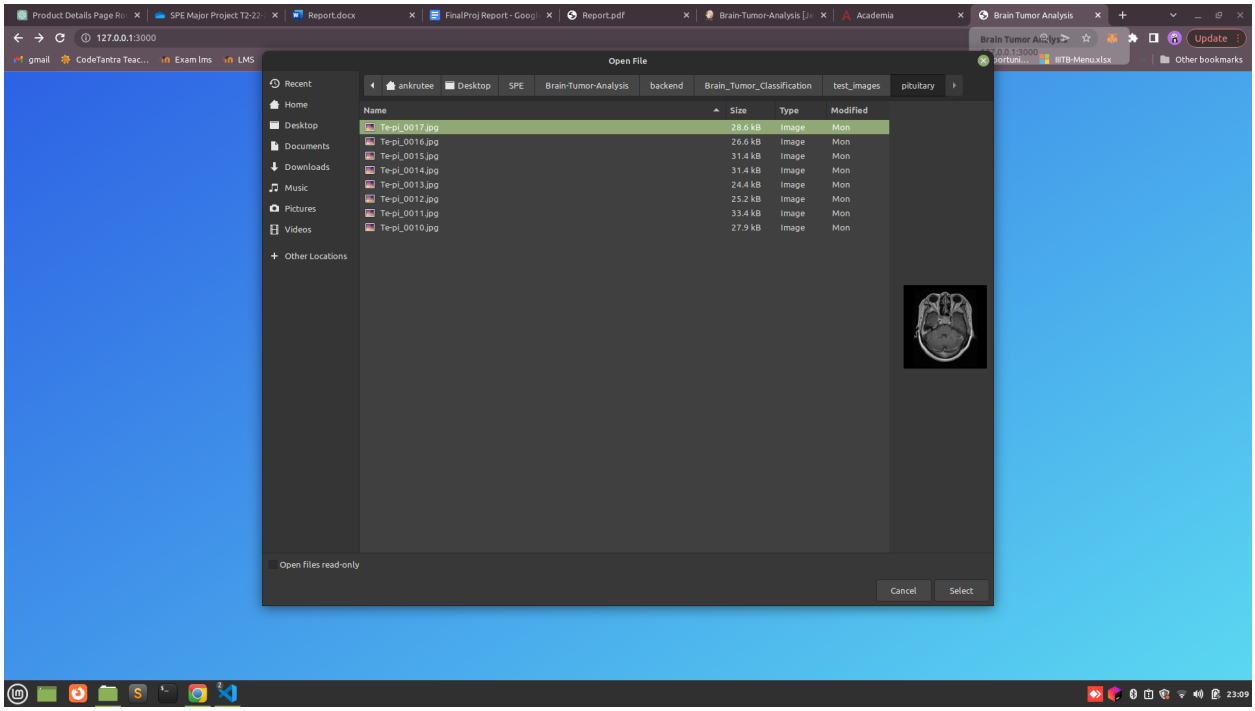


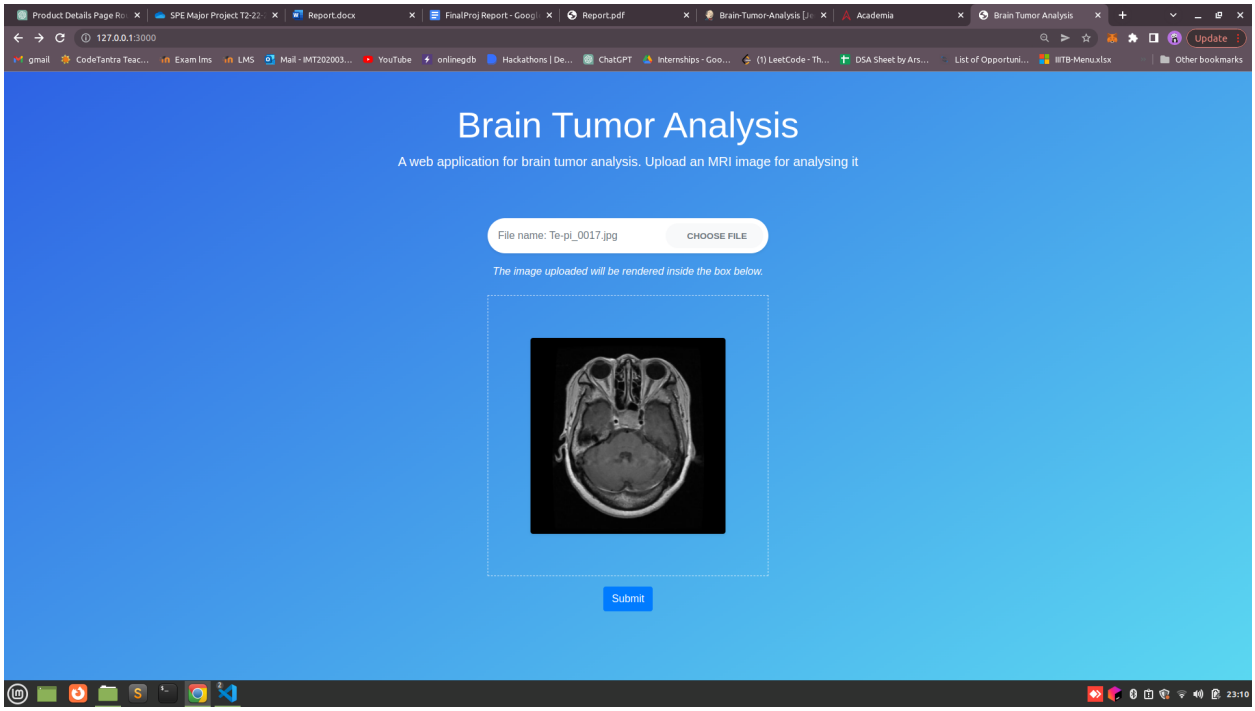
This is the result that is shown. It tells us about the Predicted tumor type, what is Meningioma, its symptoms , treatment options and general information and safety precautions.



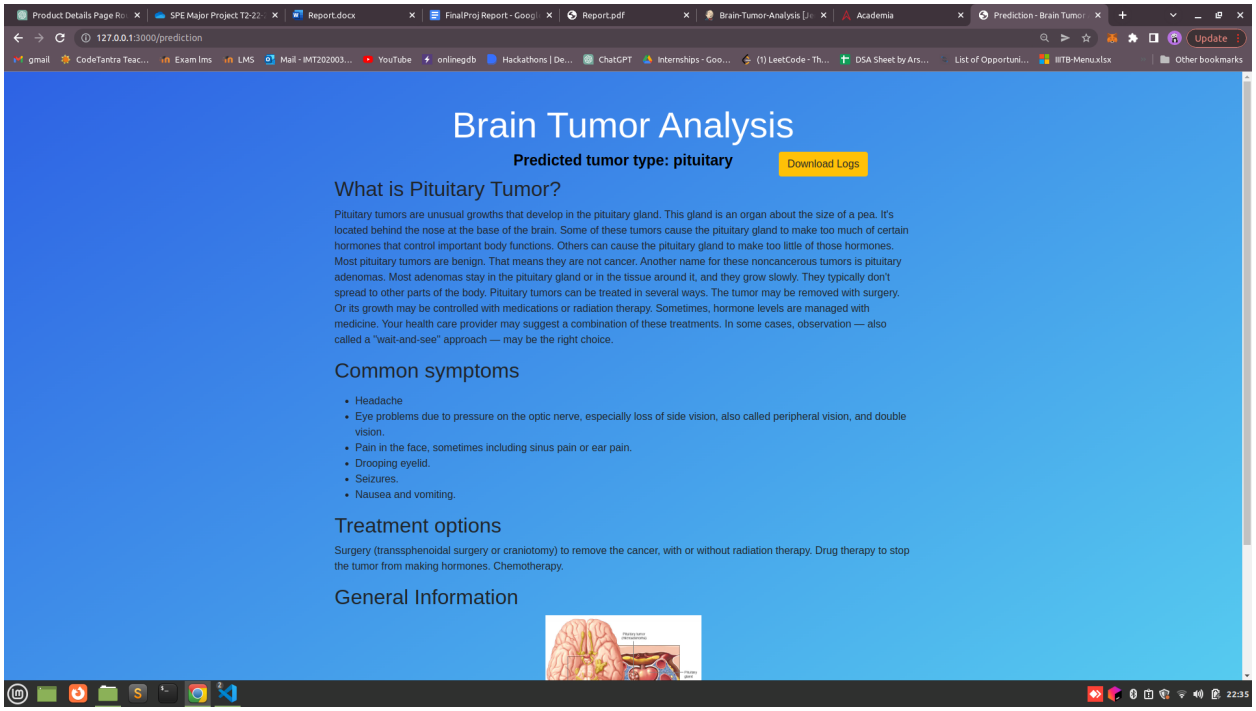


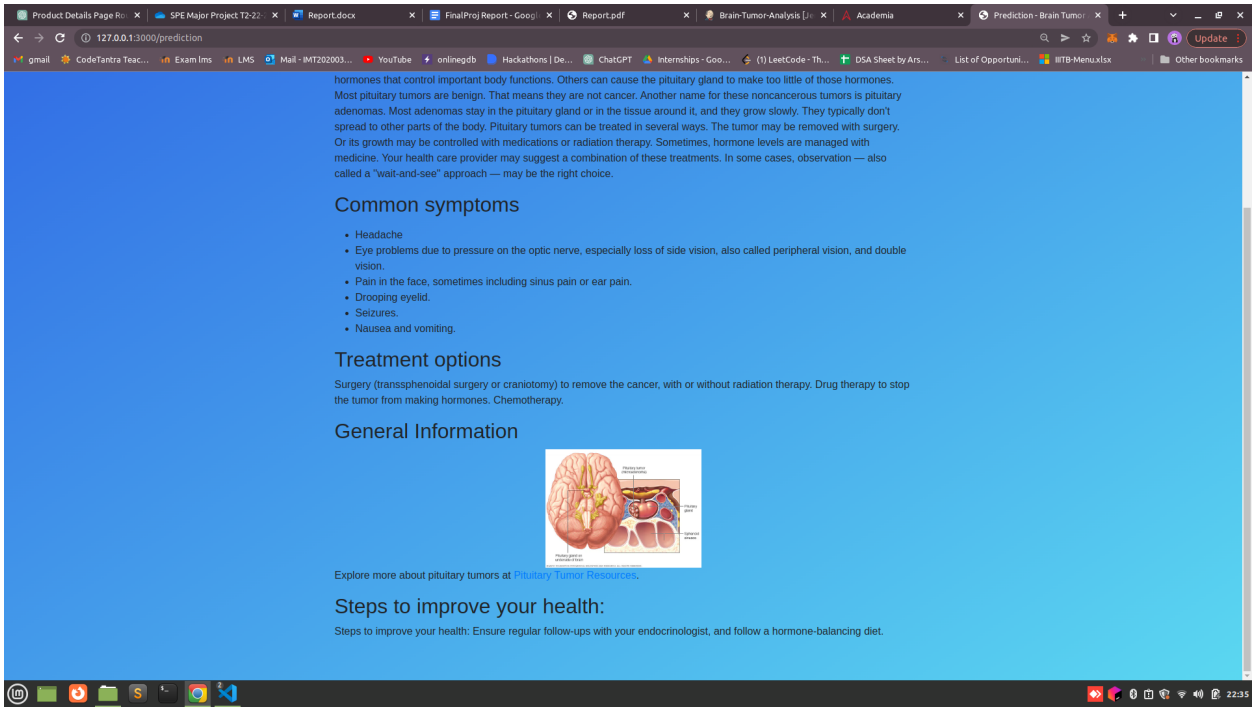
iii) pituitary



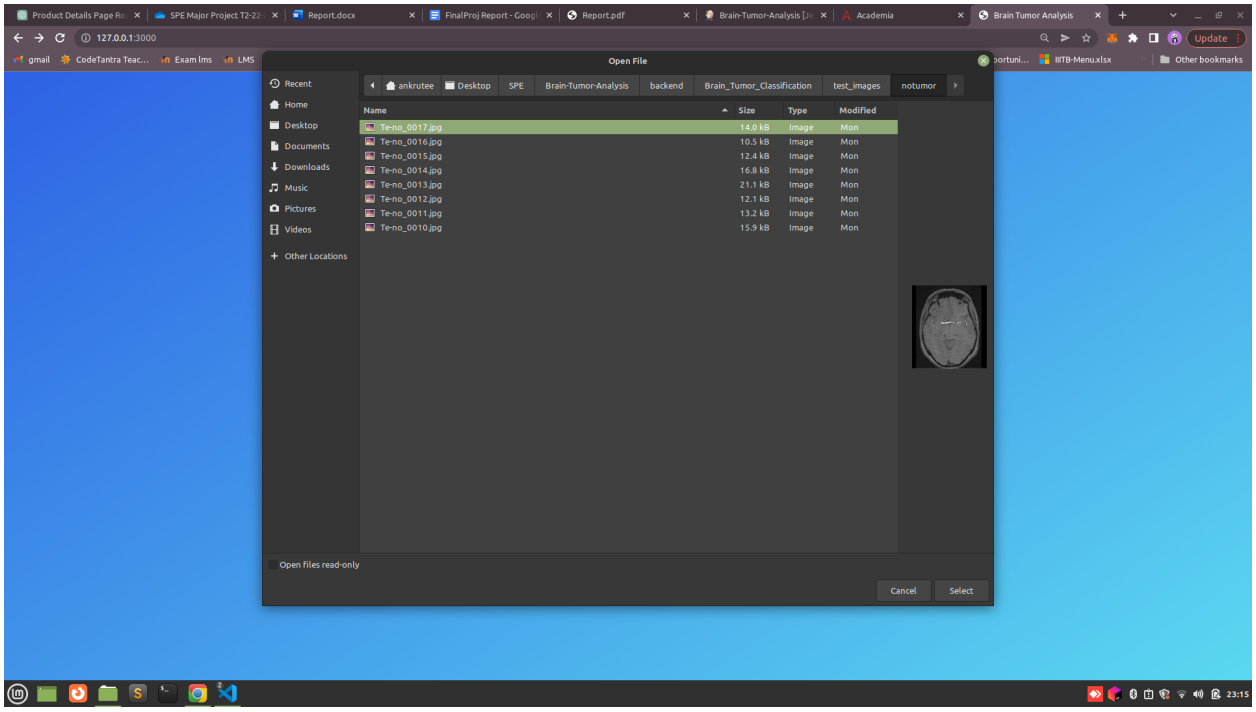


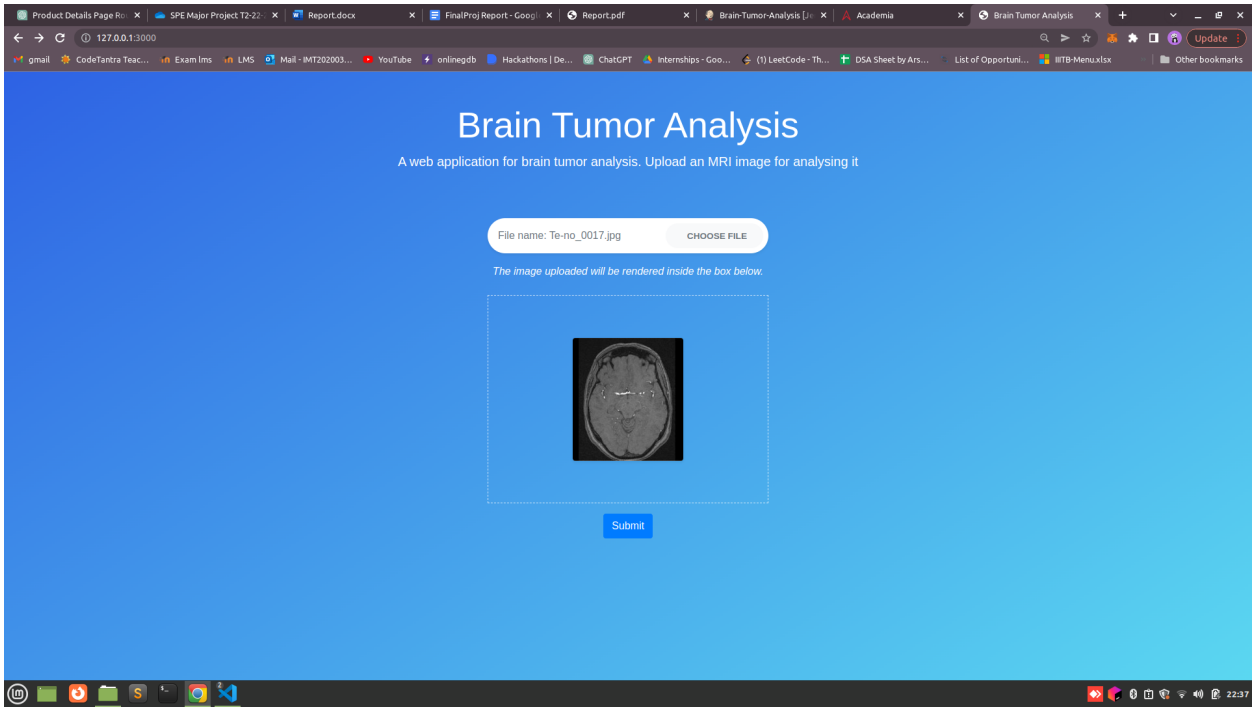
This is the result that is shown. It tells us about the Predicted tumor type, what is Pituitary tumor, its symptoms , treatment options and general information and safety precautions.

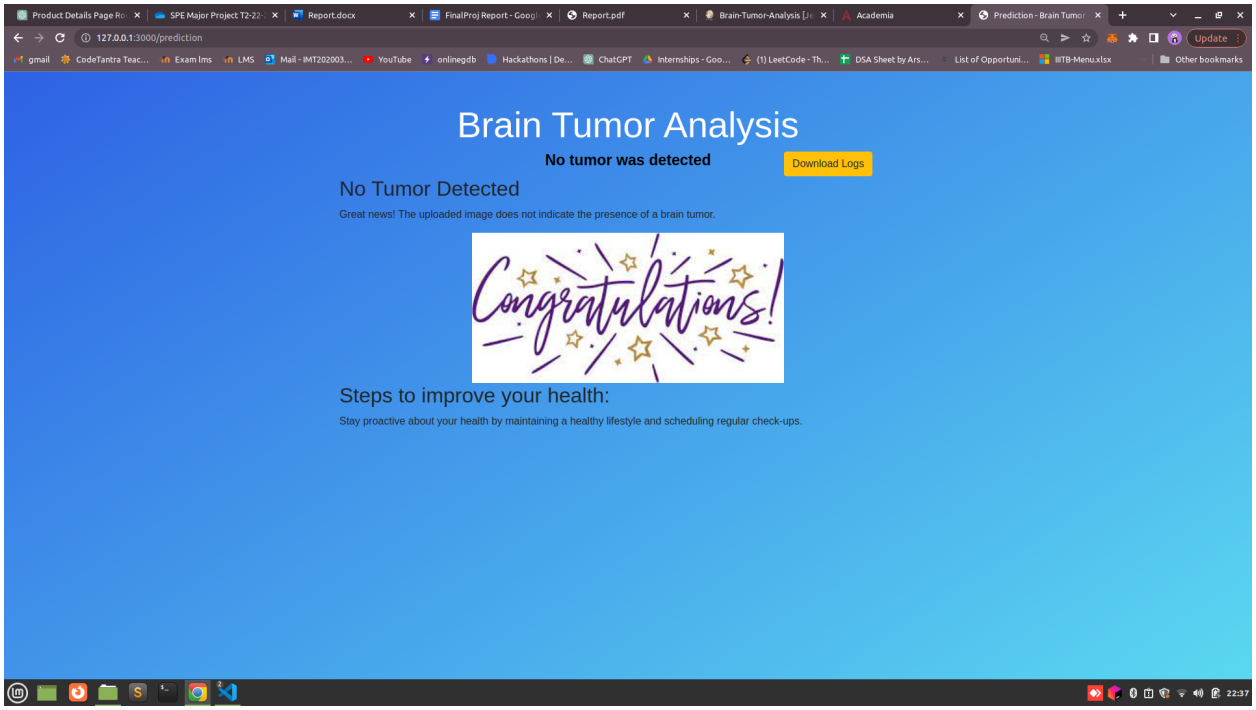




iv) No tumor detected:







Tools used for implementing MLOps:

A plethora of tools are available that help in building the CI/CD pipeline.

1. Version Control System- Git
2. CI/CD Pipeline- Jenkins
3. Testing Applications - unittest (testing tool for python code)
4. Containers and Deployment of Applications- Docker, Docker Compose
5. Operating on Services/ Infrastructure as Code- Ansible
6. Monitoring Services - ELK Stack

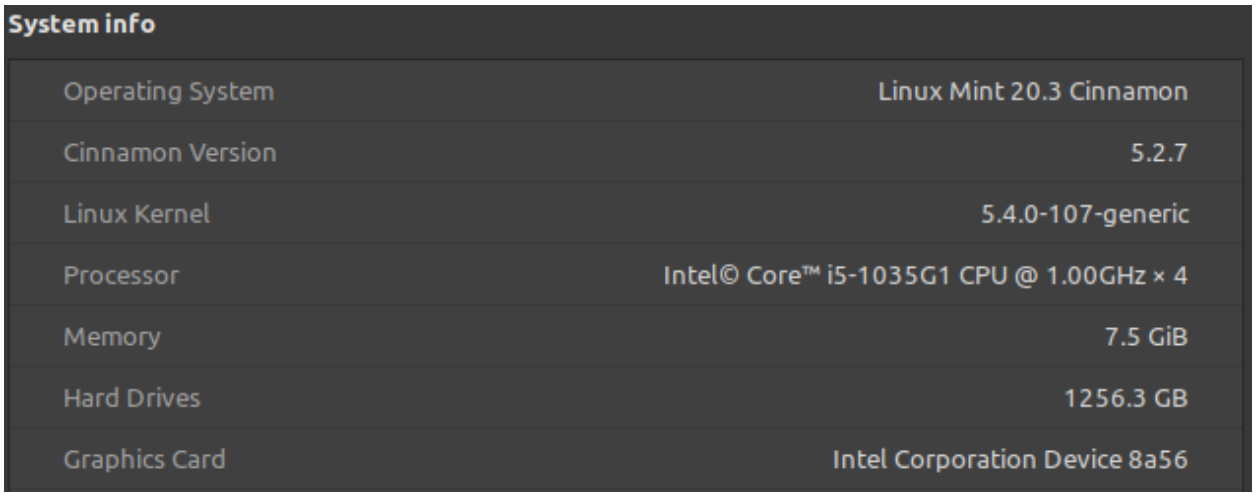
1. **Git**: A fantastic Version Control System, that allows us to track changes in code, and manage its different versions.
2. **GitHub**: GitHub acts as a Code Repository, and we can interact with it through Git. We used it for storing the code base. It also provides facilities such as WebHooks, that allow us to trigger builds remotely as soon as any changes to the code base are made.
3. **Jenkins**: An excellent tool for managing the CI/CD pipeline. It is free and open-source. We used it for creating the CI/CD pipeline for the project.
4. **unittest**: This is a testing tool for python code. We have used this library for testing our ML models. We have put our models through rigorous testing using varied test data and have verified their performance using the unittest testing tool.
4. **Docker**: Used for building containers, Docker is used to ensure that applications and services work as intended when deployed on the actual machine.We used Docker for creating images, and making containers which contain our ready-to-run application.

5. **Docker Compose**: A fantastic tool for managing multiple containers at once. Since we have multiple containers, we use Docker Compose to manage the working of these containers as a whole.
6. **Ansible**: The perfect Infrastructure as Code tool. It is used for automating a lot of tasks that are repetitive in nature, and tend to be tedious when done manually. We used Ansible for pulling Docker images and then running the containers.
7. **ELK Stack**: It is used for monitoring the applications and services that are running. We used it for creating visualizations of the logs generated by our web application.

Technology Stack
1. Backend: Javascript
2. Frontend: HTML + CSS, Java Servlet Pages
3. Container: Docker
4. Configuration Management: Ansible

System configuration:
For this project, we have used our local machines. We have used kaggle and our local machine for source code development (using VS code),for continuous integration and continuous deployment using Jenkins and
Docker image building.

System Configuration of the local machine:



Git - Source Code Management(SCM):

A source code management (SCM) tool is a piece of software that programmers use to keep track of their source code. For our project, both of us would clone the github source, make

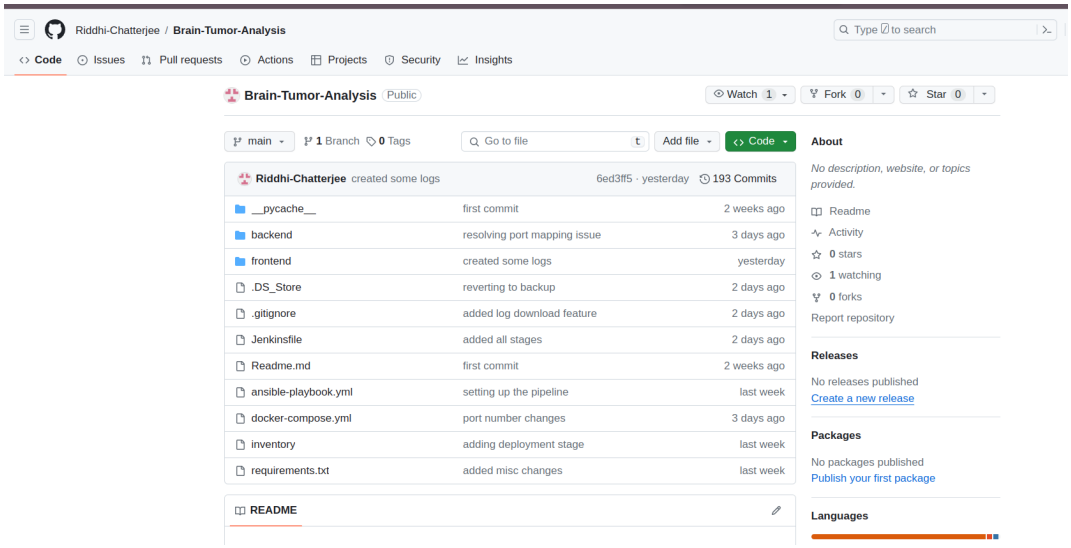
changes individually and then merge it with master. Every update repeats the loop of pulling the most recent code from git, resolving any conflicts, and then sending the changes to github.
Some of the important commands used by the team while managing
code using github:

- git add .
- git commit -m “message”
- git push
- git clone
- git pull
- git branch
- git merge
- git checkout

We added us as collaborators so that we can maintain the code alongside.

GitHub

We used GitHub to host our repository. After signing up on GitHub, we can create a new repository. We can synchronize our local Git repository with the GitHub repository. One can refer https://docs.github.com/en/migrations/importing-source-code/using-the-command-line-to-import-source-code/adding-locally-hosted-code-to-github to get the detailed steps explained in order to perform the synchronization. Below is the snapshot of the project’s GitHub repository:



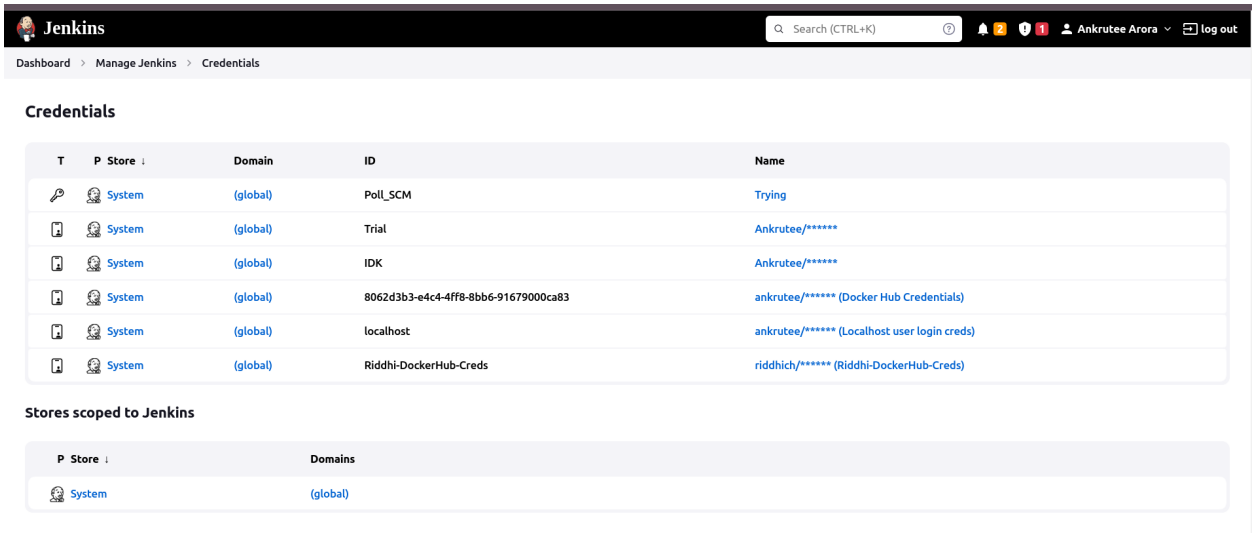
Jenkins:

Jenkins integration plays a crucial role in our project by automating the deployment pipeline. This integration enables us to create automated workflows that encompass model training, testing, deployment, and monitoring. By utilizing Jenkins, we aim to streamline these processes, making them more efficient, consistent, and manageable. This automation reduces the

likelihood of errors, accelerates deployment cycles, and enhances the overall reliability of our machine learning application.
By integrating Jenkins into our project, we aim to achieve a seamless pipeline for model deployment and ensure that our machine learning-based application operates efficiently, delivering accurate and reliable results to healthcare professionals or end-users through a user-friendly web interface.
This integration aligns with the broader objective of optimizing the development and deployment of machine learning models, ultimately contributing to more effective and accessible healthcare solutions.
However, there will be unmet dependencies for each stage, so we are going to resolve that by adding all the required plugins in one go. We will go to Dashboard > Manage Jenkins > Manage Plugins and add the necessary plugins by searching them in Available Plugins. The list of plugins that need to be present for this project to work apart from the default plugins are:

- Ansible
- Docker
- Docker Pipeline
- Docker Compose
- Git Client
- Git plugin
- GitHub

Managing All the credentials:



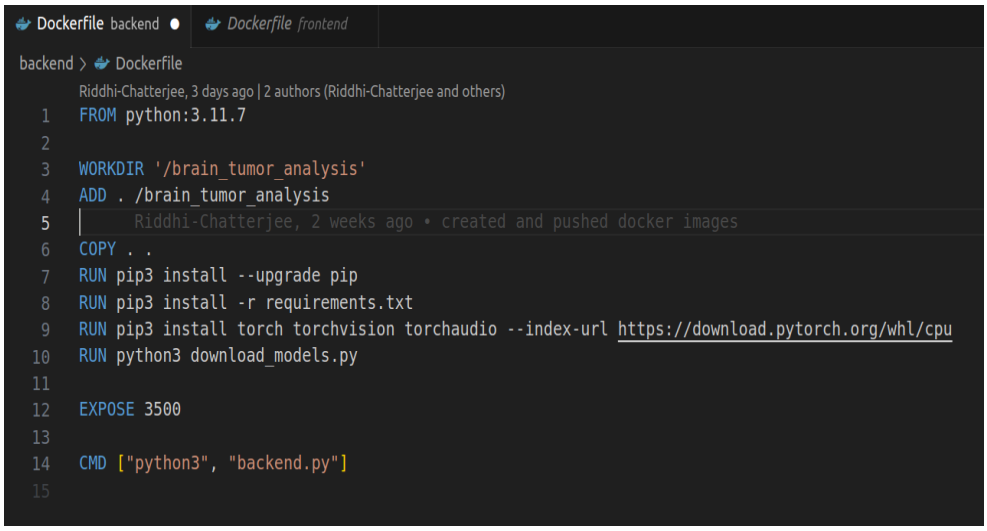
Docker :

Docker is an operating system virtualization platform that allows applications to be delivered in containers. As a result, rather than just supplying the software, the full environment is provided as a Docker image, including all the software dependencies. Since we now have a code base,

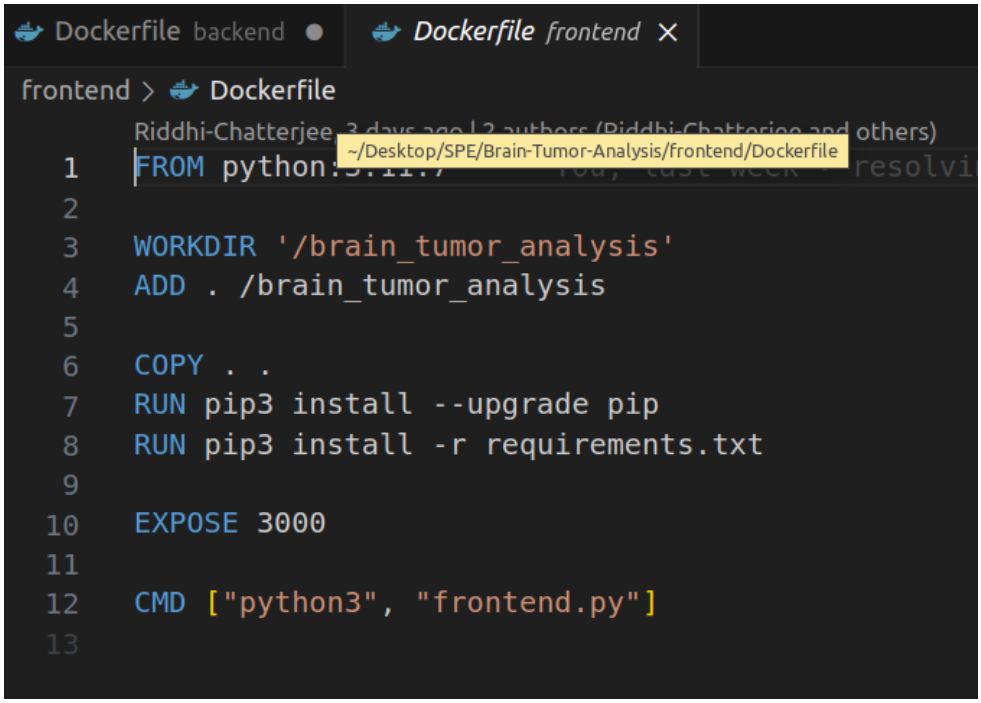
we need to put it inside a container so that it can be deployed easily. In order to do so, we install docker using:

sudo apt-get install docker.io

This Dockerfile sets up a Python environment, installs necessary dependencies, and prepares the container to run a backend component of a Brain Tumor Analysis application on port 3500.

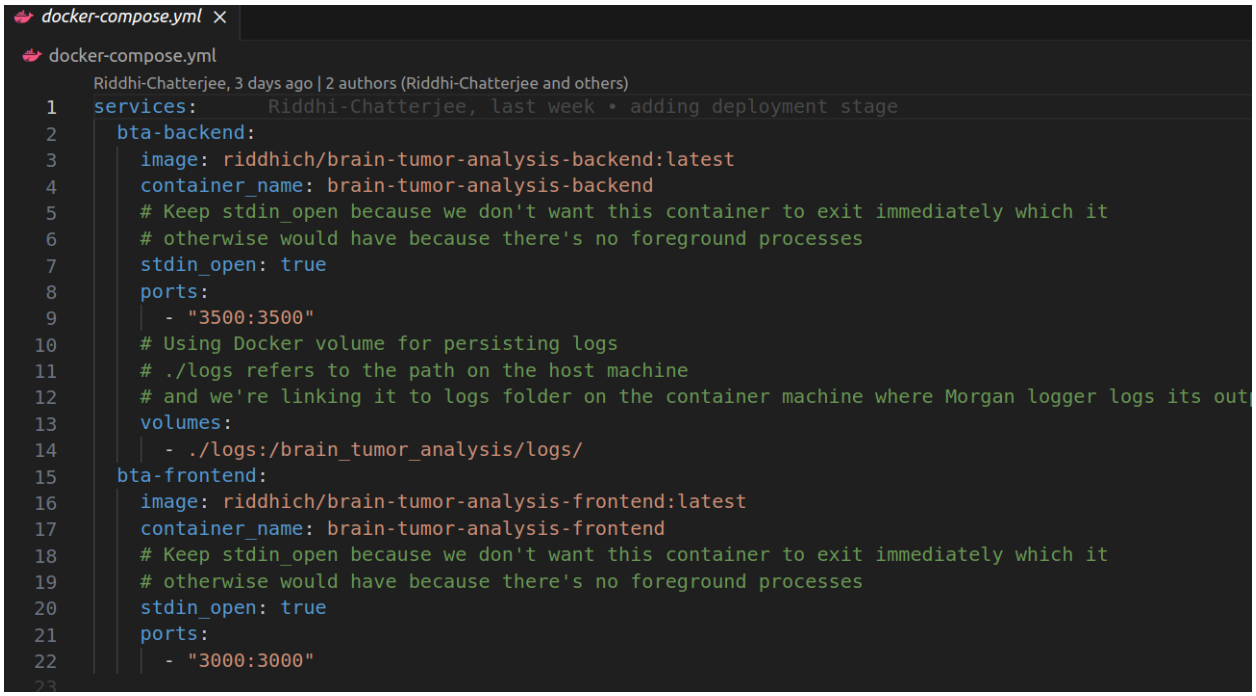


This Dockerfile creates a container environment configured to host a frontend component for a Brain Tumor Analysis application using Python and exposing port 3000 for the frontend's operation.



Docker compose

Docker Compose is a powerful tool that streamlines the development, deployment, and management of multi-container applications by providing a centralized and declarative way to define their configuration and dependencies.



docker-compose.yml

Riddhi-Chatterjee, 3 days ago | 2 authors (Riddhi-Chatterjee and others)

```yaml
services:      Riddhi-Chatterjee, last week • adding deployment stage
  bta-backend:
    image: riddhich/brain-tumor-analysis-backend:latest
    container_name: brain-tumor-analysis-backend
    # Keep stdin_open because we don't want this container to exit immediately which it
    # otherwise would have because there's no foreground processes
    stdin_open: true
    ports:
      - "3500:3500"
    # Using Docker volume for persisting logs
    # ./logs refers to the path on the host machine
    # and we're linking it to logs folder on the container machine where Morgan logger logs its out
    volumes:
      - ./logs:/brain_tumor_analysis/logs/
  bta-frontend:
    image: riddhich/brain-tumor-analysis-frontend:latest
    container_name: brain-tumor-analysis-frontend
    # Keep stdin_open because we don't want this container to exit immediately which it
    # otherwise would have because there's no foreground processes
    stdin_open: true
    ports:
      - "3000:3000"
```

- These service configurations define two containers for a Brain Tumor Analysis application: a backend and a frontend.
- The backend container exposes port 3500 on the host for communication, while the frontend container exposes port 3000.
- Both containers use stdin_open: true to prevent immediate exiting.
- The backend service additionally uses a Docker volume to persist logs generated by the application, ensuring log data is stored on the host machine's ./logs directory.

Ansible :

Ansible is used for Infrastructure as Code (IaC) service provider. It is very useful for automatically creating environments. We can install Ansible using the following commands:

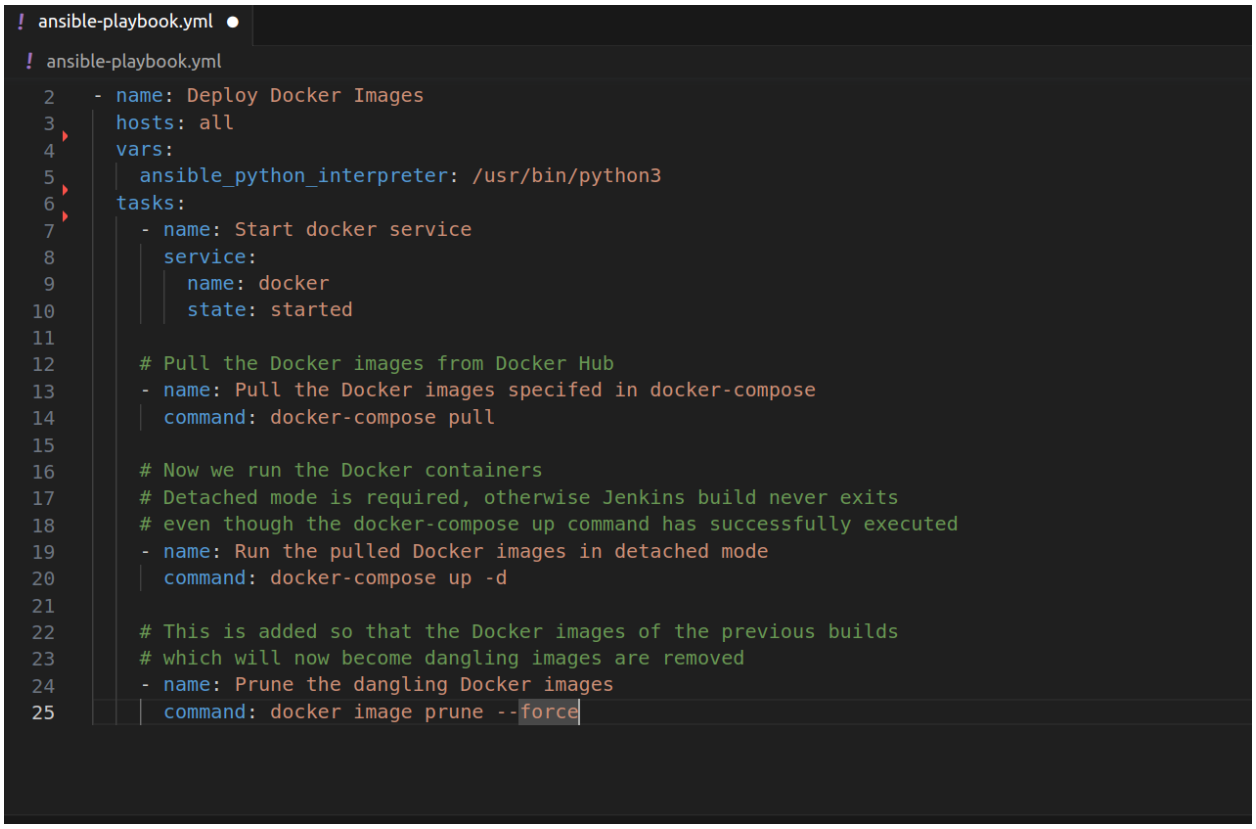
sudo apt install ansible

Ansible uses inventory files as reference, which tells it on which nodes the deployment needs to take place. Our inventory file (called inventory) is shown below:

[localhost]

127.0.0.1 ansible_connection=local ansible_user=riddhichatterjee

The above inventory file tells that the deployment machine is location at IP 127.0.0.1, whose user name is riddhichatterjee. Groups in Ansible can be used to easily organize different deployment machines.



We start docker service and pull docker image from docker hub .Now we run the Docker containers ,detached mode is required, otherwise Jenkins build never exits even though the docker-compose up command has successfully executed.Prune Dangling docker images is added so that the Docker images of the previous builds which will now become dangling images are removed.

ngrok

Instead of manually building jobs in Jenkins, we can build jobs automatically as and when our code repository gets updated (GitHub Webhooks implementation). In order to do so, we first need to install Ngrok, which exposes the Jenkins server running on our local machine to the internet.

In order to setup ngrok on our local system, we follow the below steps:

1. Sign up at Ngrok

2. Download the Ngrok file

3. Copy the ngrok authentication token from the dashboard.

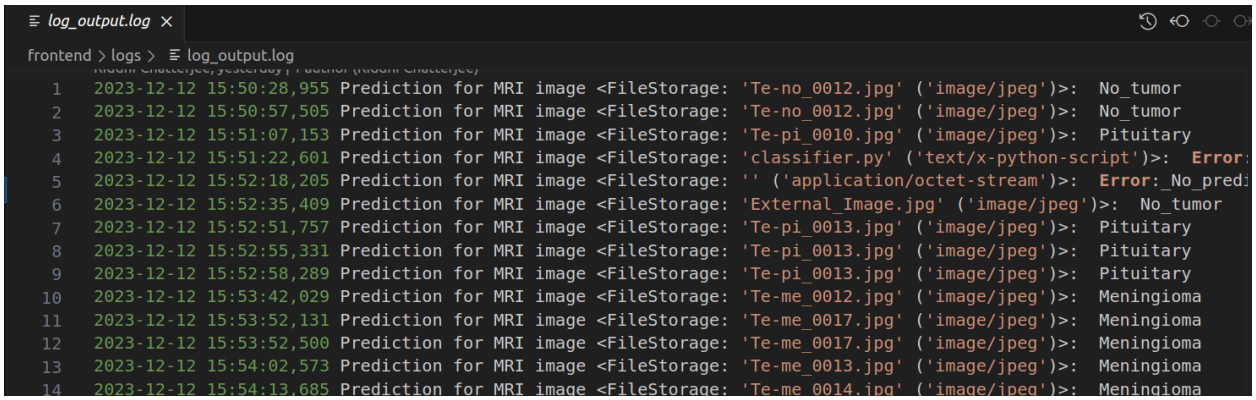
4. Add the authentication token to ngrok:

5. Expose the relevant port

Using this, we can expose our Jenkins Port number 8080 to our desired port number. We will use the new HTTP URL on which our Jenkins is exposed as the target URL for delivering webhooks.

ELK stack:

Now that we've set up the whole pipeline, we need to monitor it for any errors. In order to do so, we will use ELK Stack. Firstly, we create an account on the elk stack website. After we have logged in successfully, it will create a deployment for us. We can upload logs and visualize them. Below are some of the sample logs from the application:

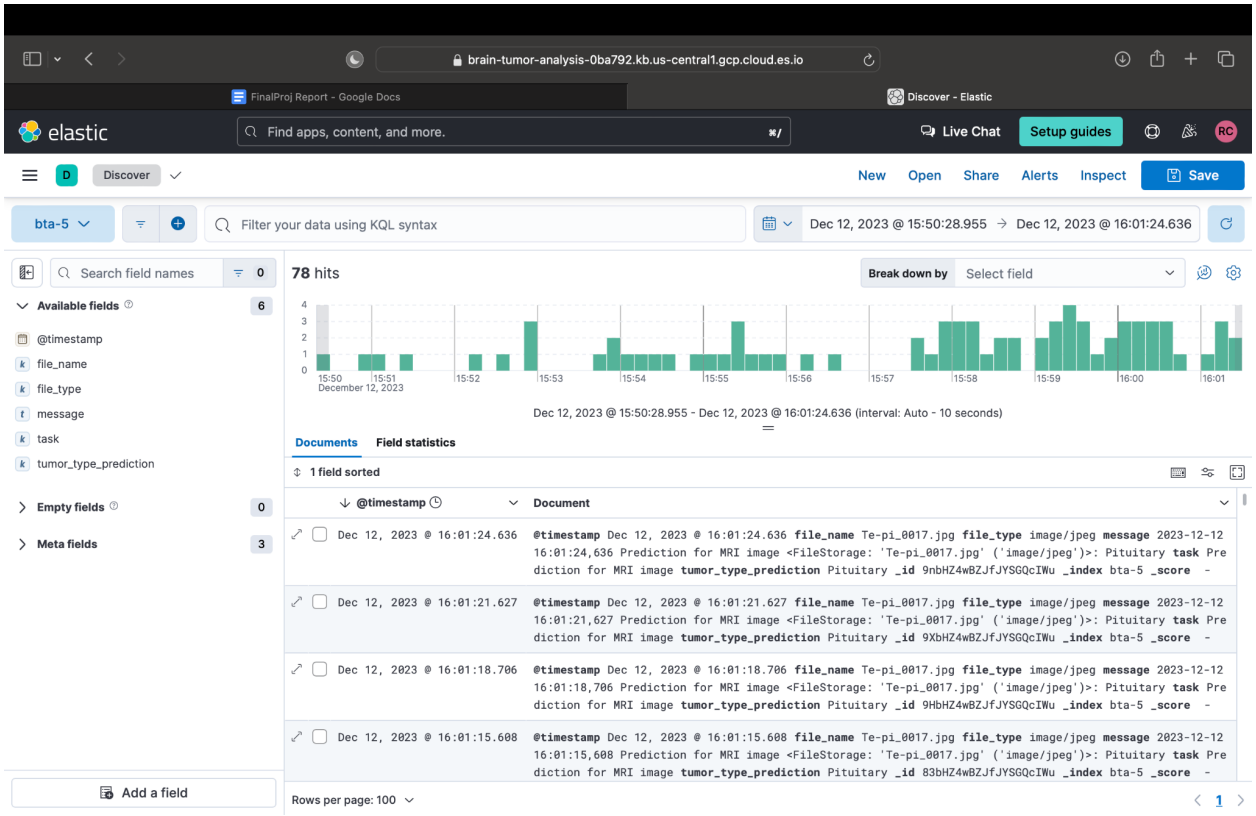


Grok pattern used to parse the logs:

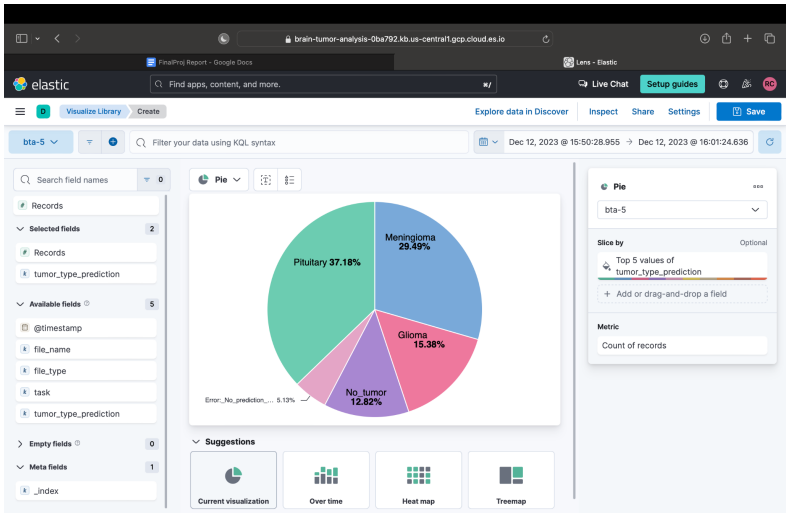
%{TIMESTAMP_ISO8601:timestamp} %{GREEDYDATA:task} <FileStorage: '%{DATA:file_name}' \('%{DATA:file_type}'\)>: %{GREEDYDATA:tumor_type_prediction}

The screenshots of ELK stack visualizing the logs is shown below:

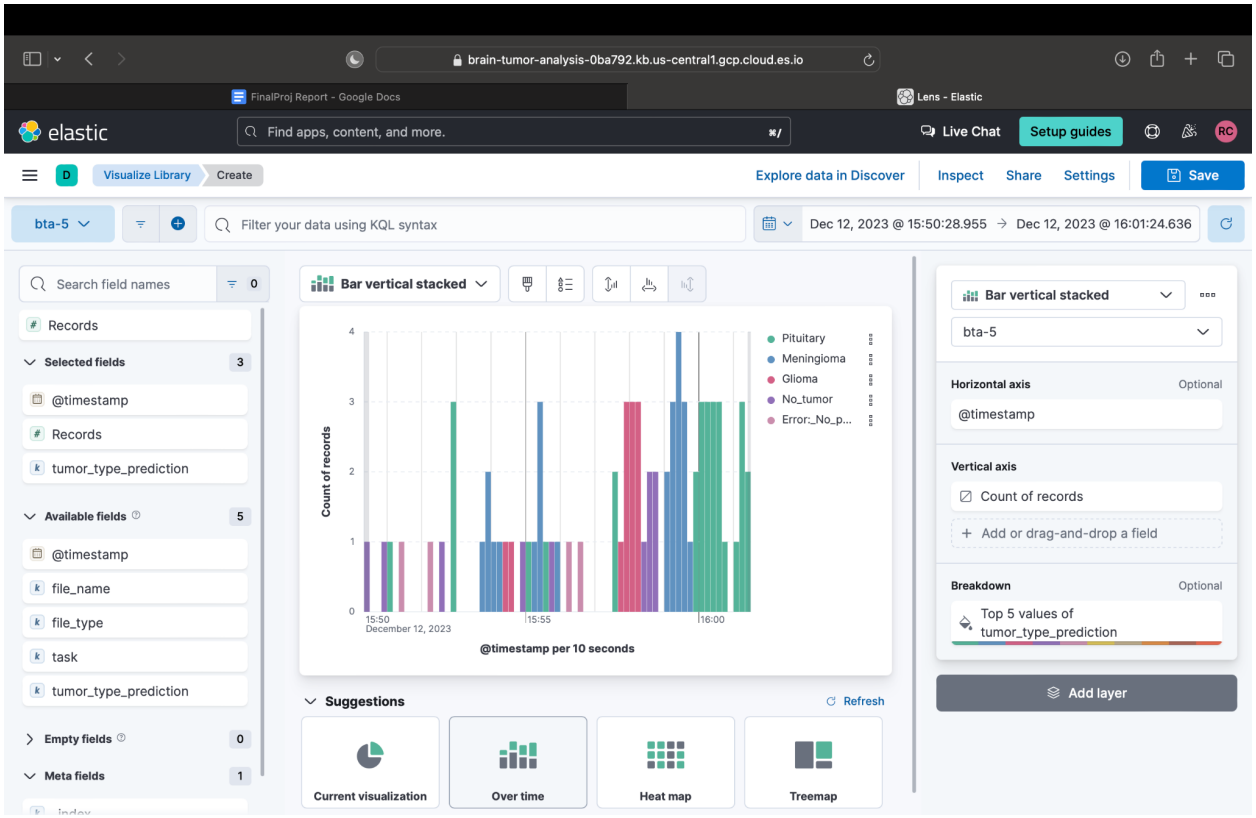
i) Graph showing the no. of hits over time:



ii) Pie chart showing the count of records for each type of prediction (including error messages):



iii) Graph showing the type of predictions made by the model over time:



Detailed Pipeline Explanation

We now describe in detail how we built the complete pipeline from scratch, that powers the complete CI/CD pipeline.

Git:

```
stages {
    stage('Git pull') {
        steps {
            git url: 'https://github.com/Riddhi-Chatterjee/Brain-Tumor-Analysis.git', branch: 'main'
        }
    }
```

This pipeline stage pulls the main branch of our GitHub repository. It simply invokes the Git plugin in Jenkins and pulls the relevant GitHub repository.This pipeline stage pulls the main branch of our GitHub repository. It simply
invokes the Git plugin in Jenkins and pulls the relevant GitHub repository.

Build

We utilise the requirements.txt file to build the Python project, then we use pip to install all of the project's dependencies. The whole requirements.txt may be found here.

$ python3 -m pip install -r requirements.txt

```
stage('Build') {
    steps {
        sh 'pip3 install --upgrade pip' //Upgrading pip3
        sh 'pip3 install -r requirements.txt' //Installing the required libraries
        sh '''cd backend
              python3 download_models.py''' //Downloading our models from google drive
    }
}
```

We use this to download our models and install the required libraries.

Test

```
stage('Test') {
    steps {
        sh '''cd backend
              python3 Brain_Tumor_Classification/test.py''' //Testing the classifier
    }
}
```

About test.py:

- Test Data Preparation: It load test images from a directory (./Brain_Tumor_Classification/test_images) and organizes them into a dictionary (test_images). It shuffles the keys of the dictionary randomly.
- Label Dictionary: There's a dictionary (label_dict) that maps abbreviations to full names of brain tumor types like 'gl' to 'glioma', 'me' to 'meningioma', 'no' to 'notumor', and 'pi' to 'pituitary'.
- Test Cases: The script defines multiple test cases (test1, test2, ..., test32), each handling a specific index of the shuffled test images dictionary.
  For each test case:
  It retrieves an image using the shuffled keys , calls the classify_image function on the image, compares the result with the expected label from the label_dict and prints the test result as SUCCESS if the predicted label matches the expected label, else FAILURE.
- Unit Test Execution: Finally, it runs all the defined test cases using unittest.main().

Build backend docker image

Script Block:

- Docker Image Building: The script uses the docker build command to create a Docker image tagged with a name derived from the variable registry appended with '-backend:latest'. This builds the backend component of the application and creates a Docker image for it using the Dockerfile located in the 'backend/' directory.
- Cleanup of Dangling Images: It tries to remove dangling (unused) Docker images using the docker rmi command. The command docker images -a -q -f "dangling=true" filters dangling images, and then docker rmi -f removes those images.
    - The try-catch block is used to handle any exceptions that might occur during the removal of images. If there are no dangling images or if an exception occurs during the removal process, it echoes "No images to delete... continuing with the pipeline..." and continues with the pipeline execution.

```
stage('Build backend docker image') {
    steps {
        script {
            sh 'docker build -t '+registry+'-backend:latest backend/'
            try {
                sh 'docker rmi -f $(docker images -a -q -f "dangling=true")'
            }
            catch (Exception e) {
                echo "No images to delete... continuing with the pipeline..."
            }
        }
    }
}
```

Build frontend docker image

Script Block:

- Docker Image Building: The script uses the docker build command to create a Docker image for the frontend component of an application. The image is tagged with a name derived from the variable registry appended with '-frontend:latest'. This command likely builds the frontend component using the Dockerfile located in the 'frontend/' directory. The other steps are similar to those mentioned in the build backend docker image.

```
stage('Build frontend docker image') {
    steps {
        script {
            sh 'docker build -t '+registry+'-frontend:latest frontend/'
            try {
                sh 'docker rmi -f $(docker images -a -q -f "dangling=true")'
            }
            catch (Exception e) {
                echo "No images to delete... continuing with the pipeline..."
            }
        }
    }
}
```

Login to Dockerhub

- It executes a shell command using sh.
- The command echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin is used to log in to DockerHub. It uses environment variables ($DOCKERHUB_CREDENTIALS_PSW and $DOCKERHUB_CREDENTIALS_USR) to provide credentials for authentication.

```
stage('Login to DockerHub') {
    steps {
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdi
    }
}
```

Push backend docker image to docker hub

It uses sh to execute the docker push command to push the Docker image tagged with the name derived from the registry variable appended with '-backend:latest' to DockerHub.

```
stage('Push backend docker image to DockerHub') {
    steps {
        sh 'docker push '+registry+'-backend:latest'
    }
}
```

Push frontend docker image to docker hub

Similar to the previous stage, it executes docker push to push the Docker image tagged with the name derived from the registry variable appended with '-frontend:latest' to DockerHub.

```
stage('Push frontend docker image to DockerHub') {
    steps {
        sh 'docker push '+registry+'-frontend:latest'
    }
}
```

Free local space

It executes docker rmi -f $(docker images -q) to remove all local Docker images from the system using the docker rmi command. The command docker images -q fetches the IDs of all images, and docker rmi -f removes them forcibly.

```
stage('Free local space') {
    steps {
        sh 'docker rmi -f $(docker images -q)'
    }
}
```

Ansible deploy

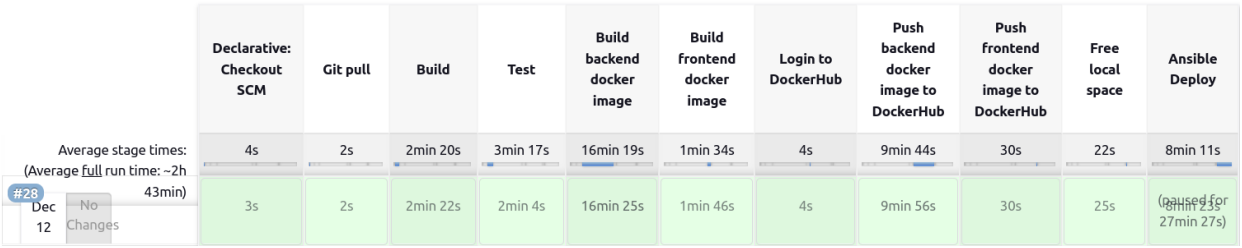
We are now ready to deploy our built application. Firstly, we run a script that ensures that the previous versions of the application are not running. Once that is done, we run the Ansible Playbook that contains commands to deploy the application.

This stage primarily involves deploying an Ansible playbook to a target machine ('localhost' in this case) using Jenkins credentials and settings, followed by manual intervention (user input) and cleanup tasks related to Docker containers and images.

```
stage('Ansible Deploy') {
    steps {
        ansiblePlaybook becomeUser: null,
        colorized: true,
        credentialsId: 'localhost',
        disableHostKeyChecking: true,
        installation: 'Ansible',
        inventory: 'inventory',
        playbook: 'ansible-playbook.yml',
        sudoUser: null

        input message: 'Finished using the web site? (Click "Proceed" to continue)'
        sh 'docker-compose down' //Stopping and removing the containers and networks linked to the projec
        sh 'docker rmi -f $(docker images -q)' //Cleaning up images
    }
}
}
}
```

That's it! We have successfully created our pipeline script. We mention that the Pipeline script is in the GitHub repository, as best practices, so that it can be version controlled accordingly. Below is an image showing the Pipeline in Jenkins:



	Declarative: Checkout SCM	Git pull	Build	Test	Build backend docker image	Build frontend docker image	Login to DockerHub	Push backend docker image to DockerHub	Push frontend docker image to DockerHub	Free local space	Ansible Deploy
Average stage times: (Average full run time: ~2h 43min)	4s	2s	2min 20s	3min 17s	16min 19s	1min 34s	4s	9min 44s	30s	22s	8min 11s
#28 Dec 12 No Changes	3s	2s	2min 22s	2min 4s	16min 25s	1min 46s	4s	9min 56s	30s	25s	8min 23s (paused for 27min 27s)

Future Scope:

1. Enhanced Model Accuracy: Continuously improving the accuracy and reliability of tumor detection algorithms through fine-tuning and incorporating advanced deep learning techniques.
2. Adding features: Model should be able to give the segmented image showing the tumor, its survival predictability and if it is treatable or not.
3. Expanded Dataset and Classes: Increasing the dataset with diverse and larger samples to improve the model's ability to detect various types, sizes, and locations of brain tumors.
4. Real-time Detection and Processing: Developing real-time or near-real-time processing capabilities for quicker diagnosis and intervention.
5. Integration with Medical Systems: Integrating the detection system with existing medical systems to assist healthcare professionals in diagnosing and monitoring brain tumor cases.
6. Interpretability and Explainability: Improving the interpretability of the model's decisions for better understanding by medical practitioners and patients.
7. Continuous Monitoring and Follow-up: Incorporating mechanisms for continuous monitoring and follow-up of detected cases to track changes and treatment progress.
8. Privacy and Ethical Considerations: Ensuring compliance with privacy regulations and maintaining ethical standards in handling patient data and diagnoses.