

International Institute of Information Technology, Bangalore

CS816 Software Production Engineering Final Project

**Title:
HelloFriend**

**Under the Guidance of
Prof. B. Thangaraju**



Student Name 1: Brahma Kulkarni
Roll number: IMT2017011

Student Name 2: Swasti Shreya Mishra
Roll number: IMT2017043

•

Table of contents

- [Resources](#)
- [Abstract](#)
- [Introduction](#)
 - [Context](#)
 - [Our Application: HelloFriend](#)
- [What is DevOps and why it is used?](#)
- [Prerequisites](#)
- [System Configuration](#)
- [Planning](#)
 - [Functional Requirements](#)
 - [Non-functional Requirements](#)
 - [Design Diagram](#)
 - [Architecture Diagram](#)
- [Code Walk through](#)
 - [Frontend](#)
 - [Backend](#)
- [Source Control Management](#)
 - [Creating repository without having developed code](#)
 - [Creating a repository after having developed some amount of code](#)
 - [Frontend](#)
 - [Backend](#)
- [Building](#)
 - [Frontend](#)
 - [Backend](#)
- [Testing](#)
 - [Frontend](#)
 - [Backend](#)
- [Continuous Integration](#)
 - [Installing the required plugins](#)
 - [Creating a Jenkins pipeline project](#)
 - [Pipeline creation](#)
 - [Global Tool Configurations, Managing Credentials, and more](#)
 - [Generating pipeline syntax](#)
 - [Running the pipeline](#)
- [Docker Containerization and Image pushing to Docker Hub](#)
 - [Frontend](#)
 - [Backend](#)
- [Continuous Deployment using Ansible](#)
- [Logging](#)
- [Monitoring using ELK stack](#)
 - [Running the ELK stack](#)
- [Results](#)
- [Scope for future work](#)
- [Conclusion](#)
- [References](#)

Resources

- Git repository for frontend code: https://github.com/brahmakulkarni/SPE_Project_Frontend
- Git repository for frontend code: <https://github.com/swastishreya/HelloFriend>
- Dockerhub image for frontend: <https://hub.docker.com/r/brahma99/hellofriendfrontend>

- Dockerhub image for backend: <https://hub.docker.com/r/swastishreya/hellofriendbackend>
- Dockerhub image for neo4j: https://hub.docker.com/_/neo4j

Abstract

In this report, we will walk you through the entire pipeline of creating a web application from scratch. This would involve planning, actual development (coding), and the entire DevOps pipeline. This project was a part of a course on Software Production Engineering. Hence, in the course of this report, we will lay more emphasis on the DevOps pipeline. This entails steps taken to automate the building, testing, containerization, and deployment of the developed code. Later on, in this report, we talk about the importance of the DevOps methodology and why it is so popularly used. More information about the actual application that we built is provided in the next section.

Introduction

Context

The idea for this project came to us as we observed that there are a lot of people that don't find it very easy to make friends. In its current state, this project is intended for school/college/university environments. There are a lot of people that are not particularly comfortable with venturing out by themselves in order to try to mingle and connect with people who have some interests common to theirs. This might be because of various reasons, of which one is shyness. With this in mind, we set out with the intention of making this process a little easier for people.

Our Application: HelloFriend

Our application helps users find other users with similar interests to theirs (interests are gathered when a user is registering on our website). This is made very simple as we provide a similarity score between each user and all other users. We also show which interests the user shares with the other users (common interests). This information is shown upfront, making the process of finding people with similar interests simple and quick.

We also considered the fact that our target demographic is students. It is quite common for users of this age range to constantly change their interests. Hence, we also have a provision for users to edit/modify their profiles. Our application also refreshes in real-time. This means, if one user is logged in and at the same time, a new user registers, their details and similarity to the first user show up instantaneously on the homepage of the first user, and vice-versa.

What is DevOps and why it is used?

DevOps is just truncated form of development and operations. Historically, developers and operations teams have been separate entities. Developers generally want to make changes rapidly while the operations team seeks organizational stability. DevOps aims to bring all the teams close so that they can work together and be on the same page.

Before the advent of DevOps, the two teams functioned disjointly. Because of this, there used to be a lot of issues that came up due to a lack of communication. This would lead to pointing fingers at each other. So, instead of attempting to resolve the issue, more time is wasted on determining whose fault it was. Also, as a result of operating separately, the two teams might be using completely different tools for different stages. Such discrepancies will lead to further slowing down of the entire pipeline. As we can see, the earlier models seem pretty inefficient.

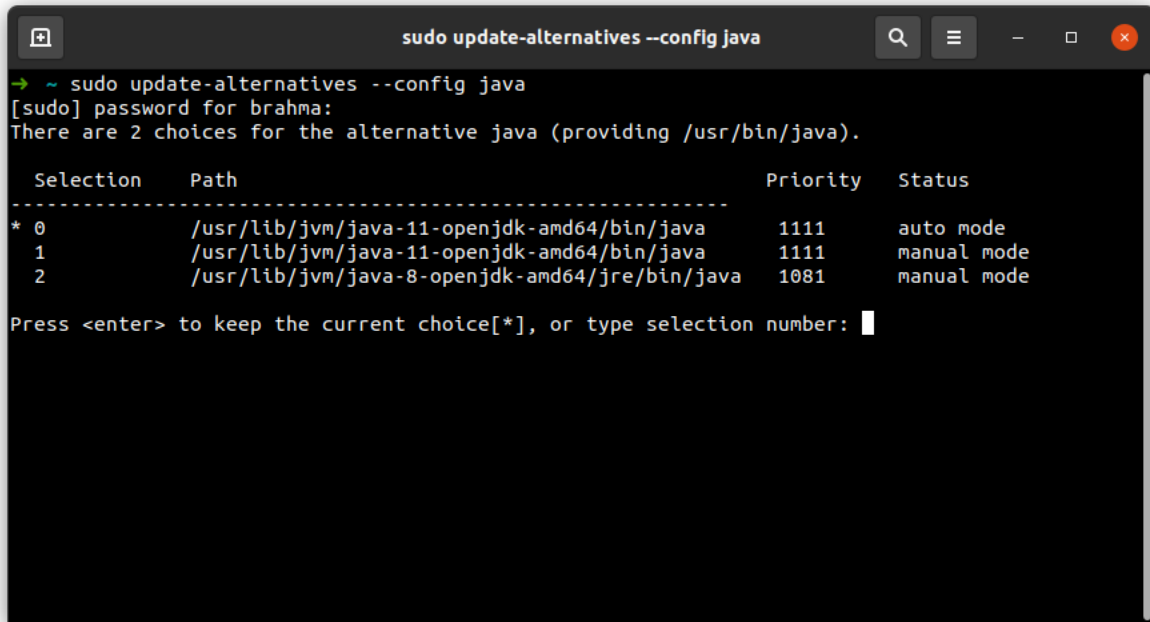
By employing the DevOps methodology, we can automate and quicken the whole process. After making an approved change in the codebase, we could set up the project in such a way that, a git push would trigger the rest of the pipeline (building, testing, deploying and monitoring). Also, due to continuous monitoring, the feedback is also brought back to the teams quickly, enabling bug fixes and new features to be coded quickly and efficiently.

Prerequisites

- Install git on your computer
- Install openjdk11. If you have multiple versions of java installed, you can switch to java11 using the following command on your terminal:

```
sudo update-alternatives --config java
```

This can be seen in the image below:

A terminal window titled 'sudo update-alternatives --config java' showing the command execution. The prompt is '~ sudo update-alternatives --config java'. The user is prompted for their password: '[sudo] password for brahma:'. The terminal then displays: 'There are 2 choices for the alternative java (providing /usr/bin/java).'. Below this is a table with columns: Selection, Path, Priority, and Status. The table lists two options: Option 0 (auto mode) and Option 1 (manual mode), both pointing to '/usr/lib/jvm/java-11-openjdk-amd64/bin/java' with priority 1111. Option 2 (manual mode) points to '/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java' with priority 1081. The prompt 'Press <enter> to keep the current choice[*], or type selection number:' is shown at the bottom with a cursor pointing to the asterisk in the first row.

Selection	Path	Priority	Status
* 0	/usr/lib/jvm/java-11-openjdk-amd64/bin/java	1111	auto mode
1	/usr/lib/jvm/java-11-openjdk-amd64/bin/java	1111	manual mode
2	/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java	1081	manual mode

- Follow this tutorial to install NodeJS in order to use ReactJS.
- Dependencies for React can be installed by cloning the frontend repository and running the following command in the project directory:

```
npm install
```

- Install Django and django_neomodel for Frontend-Django-Neo4j effective communication.
- The following commands can be run for the above step:

```
pip install Django
pip install django_neomodel
```

- Install Neo4j using this link.
- Install Jenkins. Use this link to do the same. After installing Jenkins, follow the steps given in this link to complete the setup of Jenkins.
- Install docker. Follow the following commands:

```

Installing Docker on Ubuntu 18/20
Step 1
sudo apt install apt-transport-https ca-certificates curl software-properties-common
Step 2
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
Step 3
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Step 4
sudo apt update
Step 5
sudo apt install docker-ce
Step 6 - Verify docker is running
sudo systemctl status docker

To add your user and Jenkins to docker group
sudo usermod -aG docker username
sudo usermod -aG docker jenkins (this needs to be done for seamless integration of docker and jenkins)

```

- Create an account on <https://hub.docker.com/>.
- Install and setup Ansible using the following commands:

```

sudo apt install openssh-server
ssh-keygen -t rsa

sudo apt update
sudo apt install ansible

check your ansible version using
ansible --version

check your ansible_python_version using
ansible -m setup localhost | grep ansible_python_version

```

- Use this link to download Elastic Search.
- Use this link to download Logstash.
- Use this link to download Kibana.

System Configuration

- Operating system: Ubuntu 20.04 LTS
- CPU: 4 cores
- RAM: 8GB (16GB is preferable)

Planning

Functional Requirements

- **Basic user details:** Basic details of the user like name, age, gender and email address are required for every user to have a basic information about every other user.
- **User interests:** We should be able to record user interests. This is a main factor in our application. Interests of a user are used to generate the similarity score of the user with every other user. These common interests and similarity score are what will be used by users to decide whether to approach other users.
- **Viewing other users, common interests and similarity scores:** This is the main point of our application. Each viewer should be able to see the interests they share with other users. What makes it even more simpler and quicker is that we will also display the similarity score. So, users can focus on users with higher similarity scores to them. This will help them narrow down the number of users they have to check up on.

- **Editing profiles:** As our target demographic is mostly students, we are aware of the fact that a lot of people in this age range have constantly changing interests. Some might even want to be addressed by a different name. Hence, we provide an option for every user to edit/modify their profiles.

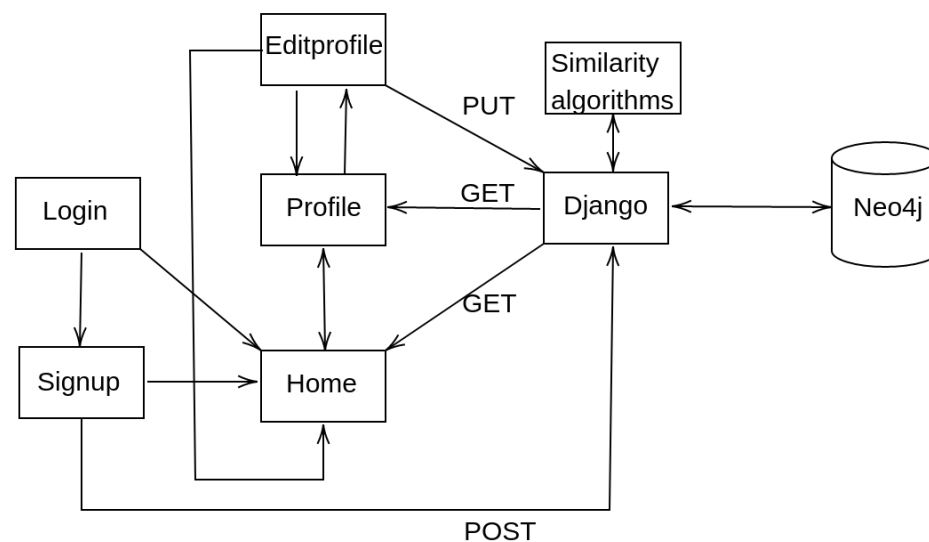
Non-functional Requirements

- **Security:** We never show sensitive information like passwords anywhere on the frontend. In fact, it isn't even sent from the backend to the frontend when there is a request for details of other users.
- **Simple, quick and user-friendly user interface:** This could be achieved in various ways. One way is to design the application such that the user has to have as few clicks as possible. The similarity information with all other users is directly shown on the home page and one can just scroll through them.
- **Seamless functioning across screen sizes:** Our application works well across different screen sizes. We didn't try running it on a smartphone, but it was working well with resizing the window size.
- **Concurrency:** Multiple instances of the app perform seamlessly. Information of new users added while a pre-existing user is logged in on another instance shows up in real-time on the pre-existing user's home screen and vice-versa.

Design Diagram

This section talks about the design of our application. As we can see in the provided design diagram, the application starts off on the Login page. If you already have an account, you can login and proceed to the home page. From the home page, you can go to the Profile page and vice-versa. From the Profile page, you can go to the Editprofile page. From the Editprofile page, you can go either the Profile page or the Home page.

The home page shows the information of other users and their similarity score and common interests to the logged in user. This is made possible by getting the information from the backend. The request is relayed to our database (Neo4j) and the data is sent back to the frontend to be displayed. The process is similar for the Profile page where the details of the logged in user are displayed. The Signup page needs to populate the user information in the database. Hence, the details are posted to the backend. A simple check is performed for the Login page. The edit profile page simply modifies the pre-existing details of the user in the database. Having edited a profile, the changes reflect immediately in the Profile page.

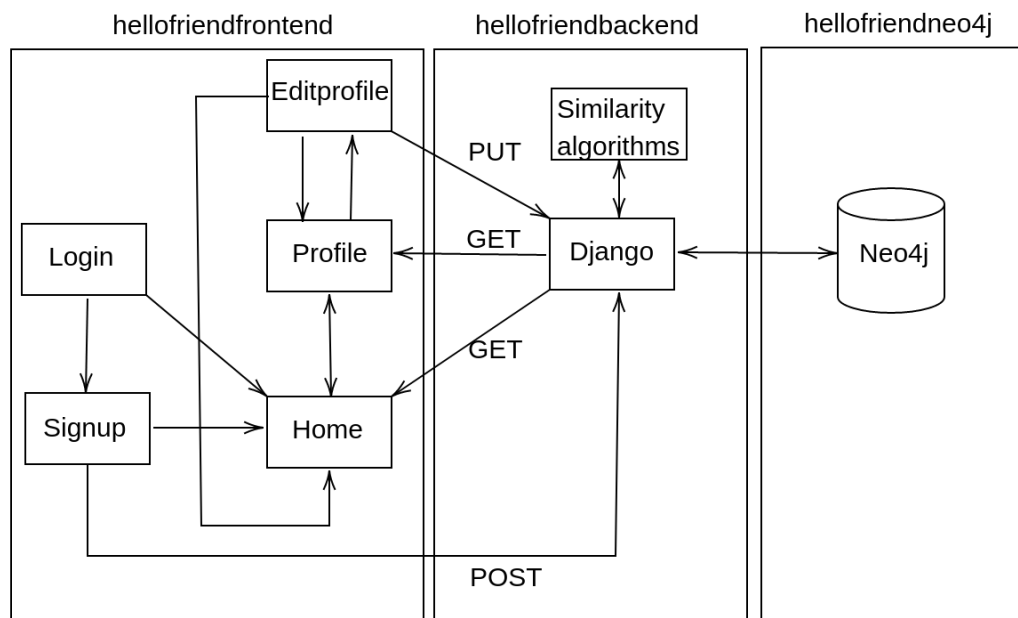


Architecture Diagram

Here we talk about the various Docker containers used for deployment and how they communicate with each other. We used 3 separate docker containers:

- One for the frontend (React): hellofriendfrontend
- One for the backend (Django): hellofriendbackend
- One for the database (Neo4j): hellofriendneo4j

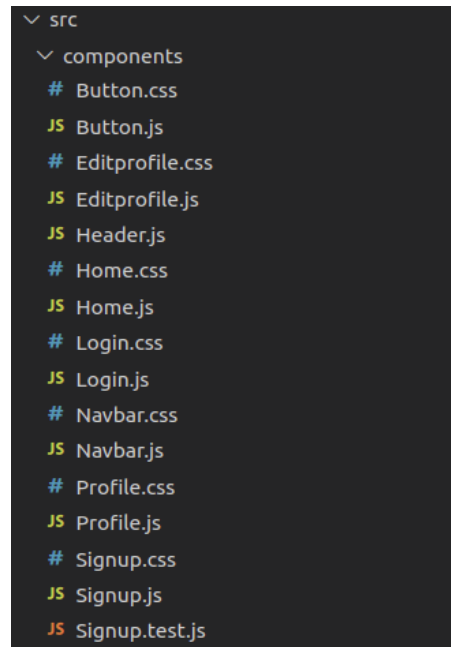
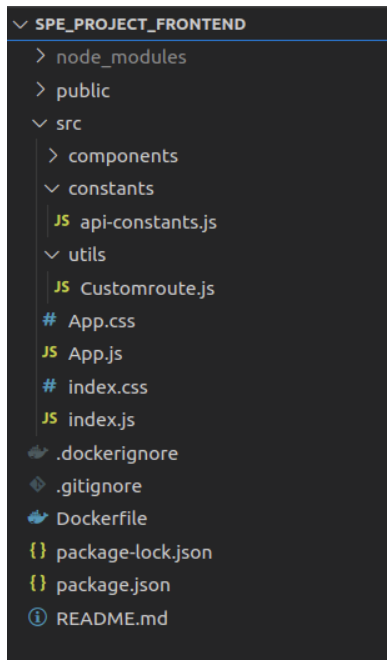
In the initial attempt to make these containers communicate with each other, we created a Docker network and ran all containers on that network. However, for reasons we weren't able to debug, we constantly got CORS errors when the frontend and backend were trying to communicate. Hence, instead, we used the inbuilt functionality of Docker to run all the containers on the "host" network. This meant, all the containers were run on the host computer's local network. Doing, this enabled the containers to communicate seamlessly.



Code Walk through

Frontend

A combination of React and CSS were used front the frontend. The React project was created by following the steps in this tutorial. Shown below is the directory structure for the frontend:



Here are a few things to note about the code implementation of the frontend:

- All components files have suffixes of .js and these files use reactjs.
- Most components have accompanying CSS files with suffixes of .css. These files are solely to add styling to the pages.
- App.js is the main file of the frontend implementation. This is what deals with switching between various pages.
- The package.json and package-lock.json files keep track of all the dependencies that the project uses.
- More information about the Dockerfile will be provided in this section.
- The .dockerignore file tells which files/directories must not be copied into the docker container. Here are the contents of the .dockerignore file:

```
node_modules
.git
```

- The .gitignore specifies which files/directories shouldn't be pushed to the remote git repository. Here are the contents of the .gitignore file:

```
# See https://help.github.com/articles/ignoring-files/ for more about ignoring files.

# dependencies
/node_modules
/.pnp
.pnp.js

# testing
/coverage

# production
/build

# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
```



```
yarn-debug.log*
yarn-error.log*
```

- The components directory houses all the components of the webapp.
- The constants folder has api-constants.js that houses all the API constants and some global variables used across all files.
- The most important parts of the frontend implementation are the parts where the requests are made to the backend. These can be seen in the images below:
- The utils directory contains Customroute.js that holds the logic for conditionally routing to a page that a logged in user has access to. If a login wasn't performed, it redirects to the Login page.

```
src > components > # Signup.js > @ Signup > @ sendDetailsToServer > @ then() callback
56 const sendDetailsToServer = (e) => {
57   if(state.name.length && state.age.length && state.gender.length && state.email.length && state.password.length) {
58     props.showError(null);
59     const payload={
60       "name":state.name,
61       "age":state.age,
62       "gender":state.gender,
63       "email":state.email,
64       "password":state.password,
65       "interests":state.interests,
66     }
67     axios.post(API_URL+"/user",payload)
68     .then(function (response) {
69       if(response.status === 200){
70         setState(prevState => ({
71           ...prevState,
72           'successMessage' : 'Registration successful. Redirecting to home page..'
73         }));
74         localStorage.setItem(ACCESS_TOKEN_NAME,response.data);
75         changeUserData(response.data);
76         console.log(response.data);
77         redirectToHome();
78         props.showError(null);
79         // console.log(response)
80       } else{
81         props.showError("Some error occurred");
82         console.log("Some error occurred")
83       }
84     })
85     .catch(function (error) {
86       console.log(error);
87     });
88   } else {
89     props.showError("Please enter valid username and password")
90     console.log("Please enter valid username and password")
91   }
92 }
93 }
```

```
src > components > # Login.js > @ Login
42 const sendDetailsToServer = (e) => {
43   e.preventDefault();
44   const payload={
45     "email":state.email,
46     "password":state.password,
47   }
48   axios.post(API_URL+"/login", payload)
49   .then(function (response) {
50     if(response.status === 200){
51       if (response.data.validLogin === 'True') {
52         setState(prevState => ({
53           ...prevState,
54           'successMessage' : 'Login successful. Redirecting to home page..'
55         }));
56         localStorage.setItem(ACCESS_TOKEN_NAME,response.data);
57         changeUserData(response.data);
58         console.log(response.data);
59         redirectToHome();
60         props.showError(null);
61       }
62     } else {
63       props.showError("Username and password do not match");
64     }
65   }
66   else if(response.code === 204){
67     props.showError("Username and password do not match");
68   }
69   else{
70     props.showError("Username does not exists");
71   }
72 }
73 .catch(function (error) {
74   console.log(error);
75 });
76 }
```

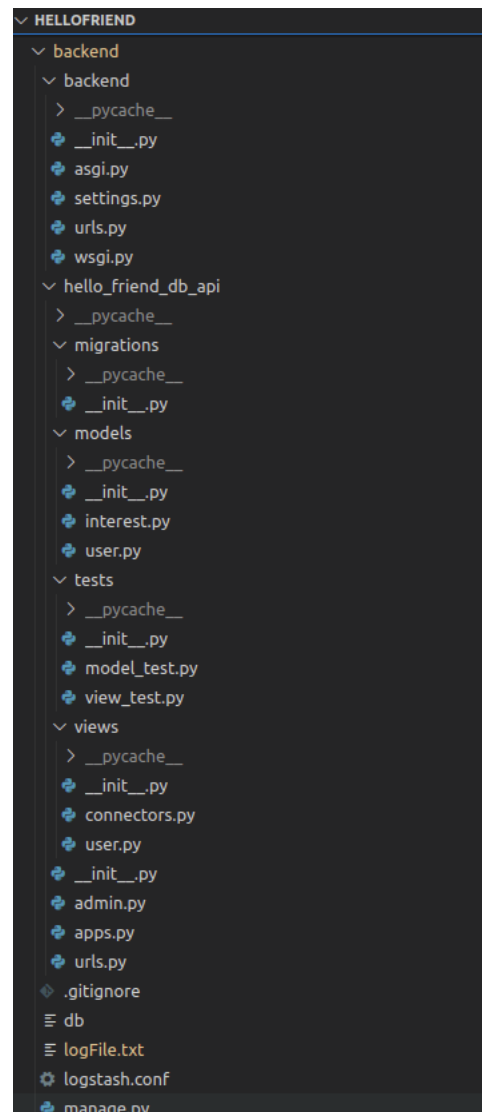
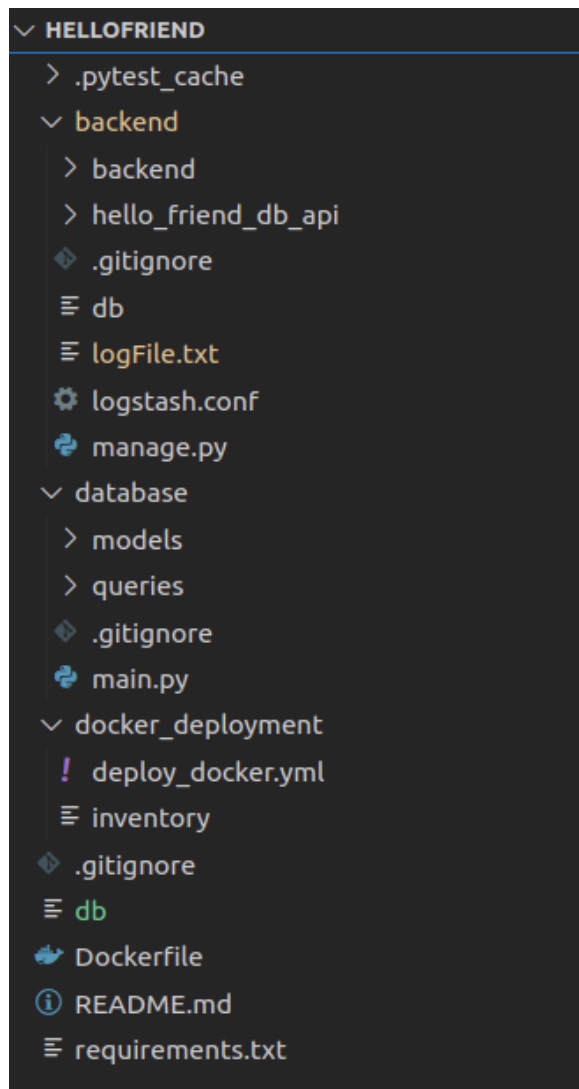
```

JS Editprofile.js X
src > components > JS Editprofile.js > Editprofile
48   const sendDetails = () => {
49     if(state.name.length && state.age.length && state.gender.length && state.email.length && state.password.length) {
50       props.showError(null);
51       const payload={
52         "name":state.name,
53         "age":state.age,
54         "gender":state.gender,
55         "email":state.email,
56         "password":state.password,
57         "interests":state.interests,
58       }
59       axios.put(API_URL+"/user",payload)
60       .then(function (response) {
61         if(response.status === 200){
62           setState(prevState => ({
63             ...prevState,
64             'successMessage' : 'Registration successful. Redirecting to home page..'
65           }))
66           localStorage.setItem(ACCESS_TOKEN_NAME,response.data);
67           changeUserData(response.data)
68           console.log("Hello")
69           console.log(response.data)
70           redirectToProfile();
71           props.showError(null)
72           // console.log(response)
73         } else{
74           props.showError("Some error occurred");
75           console.log("Some error occurred")
76         }
77       })
78       .catch(function (error) {
79         console.log(error);
80       });
81     } else {
82       props.showError('Please enter valid username and password')
83       console.log("Please enter valid username and password")
84     }
85   }

```

Backend

For the backend implementation, Python and Django were used. Shown below is the directory structure for the backend:



Here are a few things to note about the code implementation of the backend:

- The file backend/settings.py contains access information required for both the frontend and the Neo4j database. (For example - allowed hosts, database credentials, CORS configuration)
- The file hello_friend_db_api/urls.py contains the API endpoints that the frontend uses to send transaction requests to backend/database.
- The folder hello_friend_db_api/models/ contains the object-oriented class implementations of the nodes present in the database. We have used the User and Interest class/node to store the corresponding information regarding the same in our application. Multiple instances of these classes make up the data in our database. These nodes are connected using edges denoting relationships between them.
- The folder hello_friend_db_api/views/ contains files that can be used to create entries or make queries to the Neo4j database. The file user.py contains functions to create, modify and fetch user data. The file connectors.py contains functions to build relationships between the node instances.
- The folder hello_friend_db_api/tests/ contains tests which are described in detail in this section.
- More information about the Dockerfile will be provided in this section.

- The .gitignore specifies which files/directories shouldn't be pushed to the remote git repository. Here are the contents of the .gitignore file:

```
.vscode/
```

- The docker_deployment directory contains files used by Ansible to perform continuous deployment. This will be elaborated on in this section.
- The file logFile.txt is where all the logs of the application run are stored. Whereas, the file logstash.conf is the file used to configure logstash. Details regarding logging are mention in this section.

Source Control Management

Source control management is used to keep track of various versions of code under development. Commits (and their IDs) are used to keep track of each version. Additionally, using SCM, we can set up plugins in Jenkins (our continuous integration tool of choice), to pull code directly from our git repositories.

Creating repository without having developed code

- Go to your GitHub account.
- Click on the "+" symbol on the top-right.
- Select "New repository"
- Create a repository as shown in the image below. For this project, I made my repository public
- After clicking the "Create repository" button, follow the instructions on the resulting page to set a local repository for the created GitHub repository on your local computer.

Creating a repository after having developed some amount of code

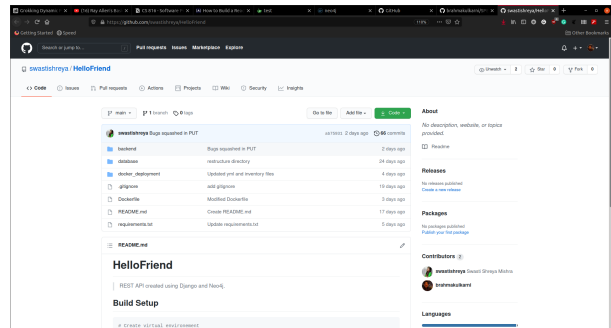
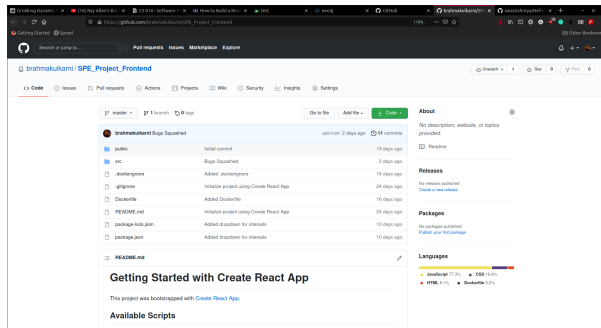
Follow the same process (as shown in the section above) to create a repository on GitHub. After that, follow the following commands by opening a terminal in the directory of the project that you're developing:

```
git init ./
git remote add origin <git_repo_url>
git add .
git commit -m "Initial commit"
git push -u origin master (or main depending on whether you renamed your master branch)
```

After this, whenever you make any changes to your code and want to push the changes to the remote repository (GitHub repository), use the following commands:

```
git add <the_files_that_you_changed>
git commit -m "<commit_message>"
git push -u origin master (or main depending on whether you renamed your master branch to main)
```

You can visit this link for the frontend git repository and this link for the backend git repository. Shown below are images of the frontend and backend repositories:



Frontend

Here is the step in the Jenkins Pipeline that corresponds to pulling the git code from this repository:

```
stage('Step1: Git pull:Frontend') {
  steps {
    git 'https://github.com/brahmakulkarni/SPE_Project_Frontend'
  }
}
```

The way to generate this pipeline syntax is shown in this section.

Backend

Here is the step in the Jenkins Pipeline that corresponds to pulling the git code from this repository:

```
stage('Step6: Git pull:Backend') {
  steps {
    git branch: 'main', url: 'https://github.com/swastishreya/HelloFriend'
  }
}
```

The way to generate this pipeline syntax is shown in this section.

Building

Building involves taking all the source code, installing all dependencies and compiling it. This would make the environment ready and capable to directly run the developed code and see the outputs.

Frontend

To build the react code, all we need to do is have pre-populated package.json and package-lock.json files. Having this, we just need to run:

```
npm install
```

Here is the step in the Jenkins Pipeline that corresponds to setting up the environment for the frontend:

```
stage('Step2: Install npm dependencies:Frontend') {
    steps {
        sh 'npm install'
    }
}
```

There is no specific way to generate the pipeline syntax for this step. Hence, we just use shell commands as shown in the image above.

Backend

To build the backend code, all we need to do is have pre-populated requirements.txt file. This file houses all the dependencies that the python project requires. This file can be generated from a virtualenv using the following command:

```
pip freeze > requirements.txt
(or)
pip3 freeze > requirements.txt
```

Having this, we just need to run:

```
pip install -r requirements.txt
(or)
pip3 install -r requirements.txt
```

Here is the step in the Jenkins Pipeline that corresponds to setting up the environment for the frontend:

```
stage('Step7: Set up environment:Backend') {
    steps {
        sh 'pip3 install -r requirements.txt'
    }
}
```

There is no specific way to generate the pipeline syntax for this step. Hence, we just use shell commands as shown in the image above.

Testing

It is important to test a codebase thoroughly before deploying. This ensures that the application is working the way it's supposed. If a codebase isn't tested properly, the product could hit the market with behavior that wasn't intended. This could have dire consequences as it might harm a large number of people.

Frontend

For testing the React code, we used the in-built React Testing Library. A simple testcase was written to demonstrate how to write unit testcases for react code. This is code for this test case is shown below:

```

JS Signup.test.js X
src > components > JS Signup.test.js > ...
1  import {BrowserRouter as Router} from 'react-router-dom'
2  import { render, screen } from '@testing-library/react';
3  import Signup from './Signup';
4
5  test("renders app title element", () => {
6    <Router>
7      render(<Signup />);
8      const titleElement = screen.getByText(/Signup/i);
9      expect(titleElement).toBeInTheDocument();
10   </Router>
11 });

```

Here is the console output for successfully running the tests for the frontend:

```
+ CI=true npm run test a
```

```
> frontend@0.1.0 test
```

```
> react-scripts test "a"
```

```
PASS src/components/Signup.test.js (12.39 s)
```

```
✓ renders app title element (4 ms)
```

```
Test Suites: 1 passed, 1 total
```

```
Tests:      1 passed, 1 total
```

```
Snapshots:  0 total
```

```
Time:       20.354 s
```

```
Ran all test suites matching /a/i.
```

Here is the step in the Jenkins Pipeline that corresponds to testing for the frontend:

```

stage('Step3: Test:Frontend') {
    steps {
        sh 'CI=true npm run test a'
    }
}

```

"npm run test" goes into an interactive mode that we can't come out of while automating things in Jenkins. Hence, we used CI=true. This doesn't enter the interactive mode. Also, the "a" at the end was added to run all test cases. By default "npm run install" runs only the testcases that were added in the latest commit. Hence, adding the "a" at the end ensures that all the testcases are run.

There is no specific way to generate the pipeline syntax for this step. Hence, we just use shell commands as shown in the image above.

Backend

We used pytest to perform the tests for the backend. To run the tests we need to run the following command in the main project directory:

```

pytest
(or)
python -m pytest
(or)
python3 -m pytest

```

Shown below is model_test.py:

```

model_test.py 9+ X
backend > hello_friend_db_api > tests > model_test.py > ...
1  from django.test import TestCase
2  from neomodel import db, clear_neo4j_database
3  from hello_friend_db_api.models import *
4  import os
5  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'backend.settings')
6  import django
7  django.setup()
8
9  class ModelTestClass(TestCase):
10     @classmethod
11     def setUpTestData(cls):
12         print("setUpTestData: Run once to set up non-modified data for all class methods.")
13         clear_neo4j_database(db)
14         user1 = User(name='Swasti', age=22, gender='Female', email='swamishr@adobe.com', password='swasti123', interests=['Anime', 'Painting', 'Dancing', 'Computer Vision', 'Dogs'])
15         user1.save()
16         user2 = User(name='Gurleen', age=21, gender='Female', email='ginni@goldmansachs.com', password='ginni123', interests=['Painting', 'Computer Vision', 'Dogs', 'Writing'])
17         user2.save()
18         res = user1.friends.connect(user2)
19
20     def test_save_model_user(self):
21         print("Method: test_save_model_user.")
22         fetch_user1 = User.nodes.get(name='Swasti')
23         for interest in fetch_user1.interests:
24             interest_node = None
25             try:
26                 interest_node = Interest.nodes.get(name=interest)
27             except:
28                 interest_node = Interest(name=interest)
29                 interest_node.save()
30             fetch_user1.interestedIn.connect(interest_node)
31         assert fetch_user1 is not None
32         fetch_user2 = User.nodes.get(name='Gurleen')
33         for interest in fetch_user2.interests:
34             interest_node = None
35             try:
36                 interest_node = Interest.nodes.get(name=interest)
37             except:
38                 interest_node = Interest(name=interest)
39                 interest_node.save()
40             fetch_user2.interestedIn.connect(interest_node)
41         assert fetch_user2 is not None

```



```

42
43     def test_unique_uid(self):
44         print("Method: test_unique_uid.")
45         fetch_user1 = User.nodes.get(name='Swasti')
46         fetch_user2 = User.nodes.get(name='Gurleen')
47         self.assertEqual(fetch_user1.uid, fetch_user2.uid)
48
49     def test_model_user_relation(self):
50         fetch_user1 = User.nodes.get(name='Swasti')
51         fetch_user2 = User.nodes.get(name='Gurleen')
52         self.assertTrue(fetch_user1.friends.is_connected(fetch_user2))
53
54     def test_model_user_similarity(self):
55         sim, common_interest = User.getSimilarity('Swasti', 'Gurleen')
56         self.assertEqual(sim, 3)
57
58     def clearTestData(self):
59         clear_neo4j_database(db)
60         fetch_user1 = User.nodes.get(name='Swasti')
61         fetch_user2 = User.nodes.get(name='Gurleen')
62         self.assertIsNone(fetch_user1)
63         self.assertIsNone(fetch_user2)
64

```

The following functions test the given functionalities:

- setUpTestData - Here we clear the database and set up some dummy data to continue testing further.
- test_save_model - We test whether the data that we set up in the step above got saved to the database.
- test_unique_id - In Neo4j, all the elements must contain a unique id (uid) which is generated automatically and assigned to the database instances when they are created. This should be unique for all the elements as the failure of this step will lead to buggy fetch query results.
- test_model_user_similarity - In this function we test whether the similarity function works the way we want it to work.
- clearTestData - Here we clear all the dummy data we had created, so that the database is free to be used by the application.
- One additional step that can be taken here is, instead of clearing the entire database in the beginning, we can store it in a file before testing, and reload the data back to the database from the file after testing is over.

Shown below is view_test.py:

```

view_test.py x
backend > hello_friend_db_api > tests > view_test.py > ...
1  from django.test import TestCase
2  from neomodel import db, clear_neo4j_database
3  from hello_friend_db_api.models import User
4  import os
5  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'backend.settings')
6  import django
7  django.setup()
8
9  class ViewTestClass(TestCase):
10
11      def test_view_user(self):
12          print("Method: test_view_user.")
13          pass
14

```

Here is the console output for successfully running the tests for the backend:

```
+ python3 -m pytest
===== test session starts =====
platform linux -- Python 3.8.5, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /var/lib/jenkins/workspace/HelloFriend
collected 5 items

backend/hello_friend_db_api/tests/model_test.py .... [ 80%]
backend/hello_friend_db_api/tests/view_test.py . [100%]

===== warnings summary =====
../.local/lib/python3.8/site-packages/django/apps/registry.py:91
/var/lib/jenkins/.local/lib/python3.8/site-packages/django/apps/registry.py:91: RemovedInDjango41Warning: 'django_neomodel' defines default_app_config =
'django_neomodel.apps.NeomodelConfig'. Django now detects this configuration automatically. You can remove default_app_config.
    app_config = AppConfig.create(entry)

-- Docs: https://docs.pytest.org/en/stable/warnings.html
===== 5 passed, 1 warning in 12.45s =====
```

Here is the step in the Jenkins Pipeline that corresponds to testing for the backend:

```
stage('Step8: Test:Backend') {
    steps {
        sh 'python3 -m pytest'
    }
}
```

There is no specific way to generate the pipeline syntax for this step. Hence, we just use shell commands as shown in the image above.

Continuous Integration

Continuous Integration is the process of various members working on a project continuously/frequently integrate their work with the work of others. The continuous integration tool that I used was Jenkins. Jenkins has various plugins for git, GitHub, Maven, Docker and Ansible, all of which we are using in this project. These plugins help with the seamless integration of all these tools and enable Jenkins to automate the whole pipeline with the tap of a single button.

Installing the required plugins

Plugins can be installed by going to Manage Jenkins → Manage Plugins ("Available" tab). Here are a list of Jenkins plugins that need to be installed for this project:

- Ansible plugin
- Docker Pipeline
- Git
- GitHub
- GitHub Integration
- NodeJS Plugin


Creating a Jenkins pipeline project

- On the main Jenkins Dashboard, click on "New Item"


- Select Pipeline and provide a project name to create a new Pipeline project. This is shown below:

Enter an item name


» Required field


Freestyle project


This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.


Maven project


Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.


Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.


Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.


Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Pipeline creation

- If you just created the pipeline project, scroll down to the Pipeline section.
- This is where we write the pipeline script. The pipeline script elaborates on the stages and steps that need to be followed to integrate various stages of the whole DevOps methodology.

Shown below is our pipeline script:

```

pipeline {
    environment {
        dockerImage = ""
    }
    agent any

    stages {
        stage('Step1: Git pull:Frontend') {
            steps {
                git 'https://github.com/brahmakulkarni/SPE_Project_Frontend'
            }
        }
        stage('Step2: Install npm dependencies:Frontend') {
            steps {
                sh 'npm install'
            }
        }
        stage('Step3: Test:Frontend') {
            steps {
                sh 'CI=true npm run test a'
            }
        }
        stage('Step4: Building docker image:Frontend') {
            steps {

```

```

        script {
            dockerImage = docker.build "brahma99/hellofriendfrontend:latest"
        }
    }
}
stage('Step5: Push Docker image to Docker Hub:Frontend') {
    steps {
        script {
            docker.withRegistry('', 'docker-jenkins') {
                dockerImage.push()
            }
        }
    }
}
stage('Step6: Git pull:Backend') {
    steps {
        git branch: 'main', url: 'https://github.com/swastishreya/HelloFriend'
    }
}
stage('Step7: Set up environment:Backend') {
    steps {
        sh 'pip3 install -r requirements.txt'
    }
}
stage('Step8: Test:Backend') {
    steps {
        sh 'python3 -m pytest'
    }
}
stage('Step9: Building docker image:Backend') {
    steps {
        script {
            dockerImage = docker.build "swastishreya/hellofriendbackend:latest"
        }
    }
}
stage('Step10: Push Docker image to Docker Hub:Backend') {
    steps {
        script {
            docker.withRegistry('', 'docker-jenkins-swasti') {
                dockerImage.push()
            }
        }
    }
}
stage('Step11: Ansible pull image') {
    steps {
        ansiblePlaybook becomeUser: null, colored: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'docke
    }
}
}
}

```

Global Tool Configurations, Managing Credentials, and more

- To globally configure Ansible in Jenkins, go to Dashboard → Manage Jenkins → Global Tool Configuration.
- Scroll down to the Ansible section and select add Ansible.
- Fill it in as shown below:

Ansible

Ansible installations

Add Ansible



Ansible

Name

Ansible

Path to ansible executables directory

/usr/bin/

☐ Install automatically



Delete Ansible

- To fill the field "Path to ansible executables directory", open a terminal on your computer and enter the command:

whereis ansible

- To ensure smooth operation between Docker Hub and Jenkins, we must save our Docker Hub credentials on Jenkins. To do so, go to Dashboard → Manage Jenkins → Manage Credentials.
- Now under "Stores scoped to Jenkins", click on global and then on Add credentials.
- Then select "Username with password" in the "Kind" field.
- Then fill in the details as shown below:

Scope: Global (Jenkins, nodes, items, all child items, etc) ?

Username: brahma99 ?

Password: Concealed Change Password ?

ID: docker-jenkins ?

Description: Docker Hub credentials ?

Save

- We also need to make it possible for Jenkins to be able to login into our remoteuser without the use of a password. For this, follow the commands given below:

```
sudo su - jenkins (this logs us into the jenkins user)
ssh-keygen -t rsa (generates ssh keys)
ssh-copy-id REMOTEUSER@<REMOTE IP ADDRESS> (eg. ssh-copy-id brahma@localhost)
(This last line will copy the remote ssh keys to the known hosts in Jenkins, enabling Jenkins to login to the remoteuser without needing a
```

Generating pipeline syntax

Pipeline

Definition: Pipeline script

Script

```
1 pipeline {
2   environment {
3     dockerImage = ""
4   }
5   agent any
6
7   stages {
8     stage('Git pull') {
9       steps {
10        git 'https://github.com/brahmakulkarni/DevOpsCalc.git'
11      }
12    }
13    stage('Build and test') {
14      steps {
15        sh 'mvn clean package'
16      }
17    }
18    stage('Building docker image') {
19      steps {
20
```

☒ Use Groovy Sandbox ?

Pipeline Syntax

- For the git pipeline syntax, press the "Pipeline Syntax" as shown above. In "Sample Step", select the "git: Git" option in the pull-down menu and fill in the other details and click on generate pipeline syntax. If your repository is private, you can add your git credentials here too. This is shown below:

Steps

Sample Step git: Git

Repository URL https://github.com/brahmakulkarni/DevOpsCalc

Branch master

Credentials - none - Add

☒ Include in polling?

☒ Include in changelog?

[Generate Pipeline Script](#)

```
git 'https://github.com/brahmakulkarni/DevOpsCalc'
```

- The pipeline syntax for the building, testing, Docker, and Docker Hub stages was written manually and can be seen in the pipeline syntax. The docker credentials we created is used in the Docker Hub stage as reflected in the pipeline syntax and is also shown below:

```
stage('Push Docker image to Docker Hub') {  
  steps {  
    script {  
      docker.withRegistry('', 'docker-jenkins') {  
        dockerImage.push()  
      }  
    }  
  }  
}
```

- To generate the pipeline syntax for the ansible stage, press the "Pipeline Syntax" button. In "Sample Step", select the "ansiblePlaybook: Invoke an ansible playbook" option in the pull-down menu and fill in the other details and click on generate pipeline syntax. This is shown below:

Sample Step

ansiblePlaybook: Invoke an ansible playbook

Ansible tool

Ansible

Playbook file path in workspace

docker_deployment/depoy_docker.yml

Inventory file path in workspace

docker_deployment/inventory

SSH connection credentials

None

Vault credentials

None

Use become

Become username

Use sudo (deprecated)

Sudo username (deprecated)

Host subset

Tags

Tags to skip

Task to start at

Number of parallel processes to use

Disable the host SSH key check

Colorized output

Extra parameters

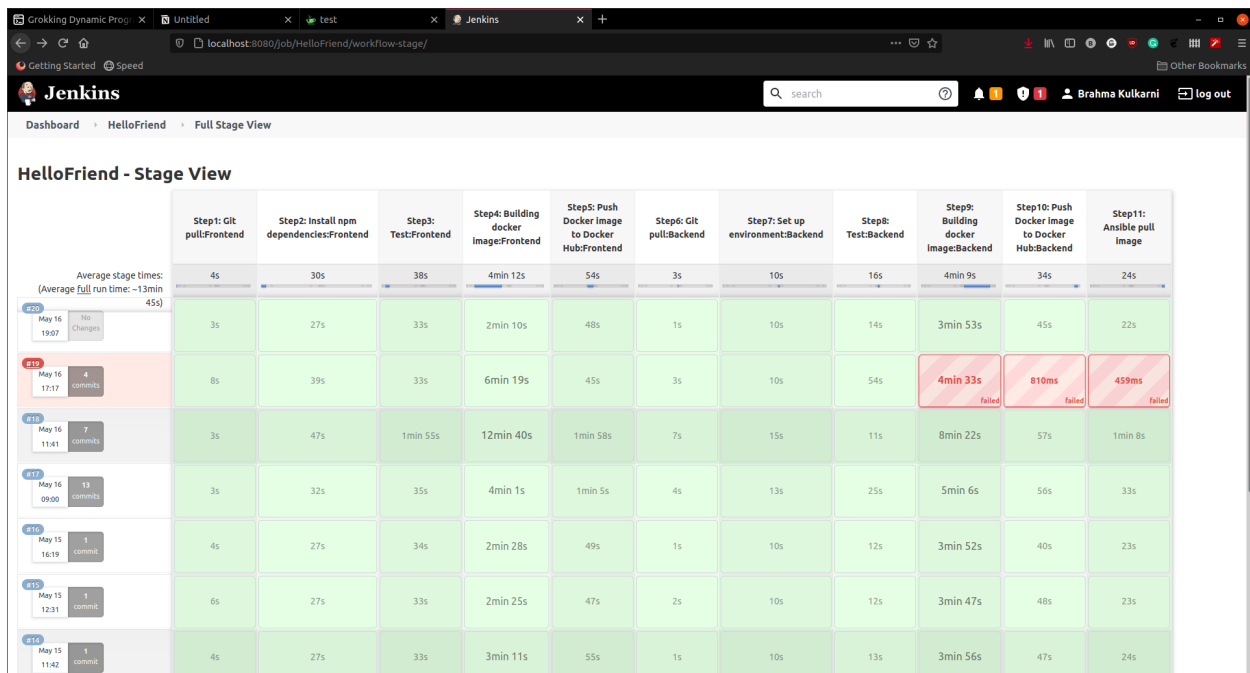
Generate Playbook Script

ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true, installation: 'ansible', inventory: 'docker_deployment/inventory', playbook: 'docker_deployment/depoy_docker.yml', sudoUser: null

As you can see, the playbook file (.yml file) and inventory file are inside a docker-deployment folder in the project directory. This was also seen in the directory structure shown before.

Running the pipeline

The entire pipeline can be run by pressing the build now button on the sidebar. Shown below are the various stages of our Jenkins pipeline:



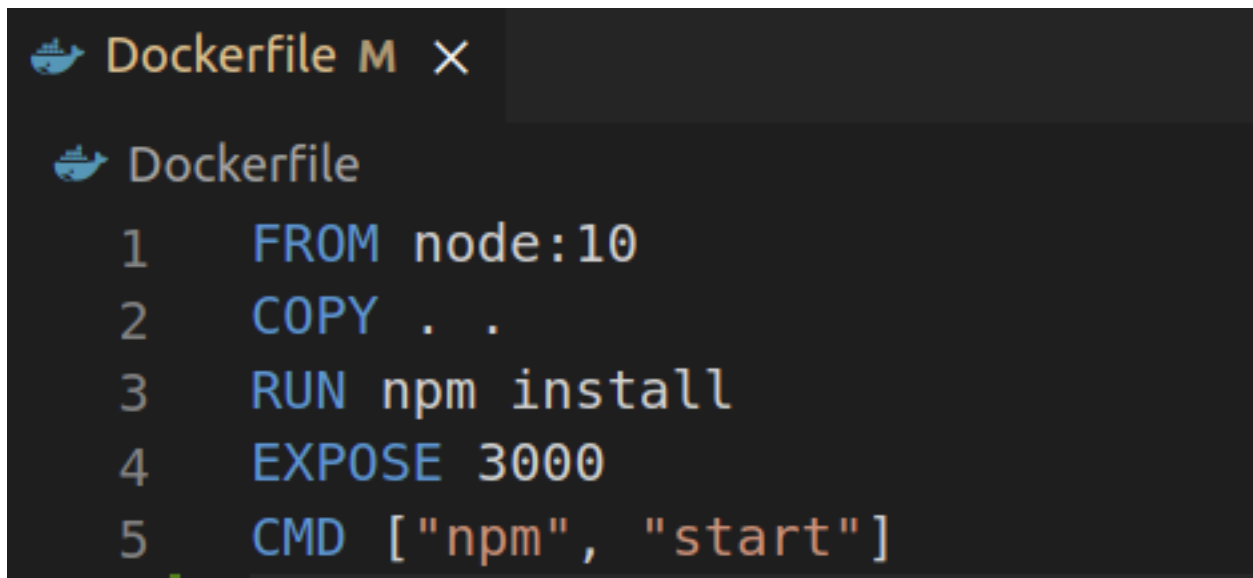
Docker Containerization and Image pushing to Docker Hub

The containerization tool of choice used was Docker. Docker creates containers that are basically visualizations at the Operating System level. Docker containers can be thought of as light-weight virtual machines. They have smaller memory footprints when compared to virtual machines as VMs have an entire copy of an operating system while docker containers have the bare only bare minimum facilities like a root file system and an interactive terminal. Docker containers help in isolation. Whatever is put within a docker container is completely independent, isolated, and can be tended to separately.

Note: You will observe that both docker files have an "**EXPOSE**" instruction. This instruction tells docker that the container listens on the specified network port number at runtime.

Frontend

Here is a screenshot of the Dockerfile used for the frontend of this project:

A screenshot of a code editor showing a Dockerfile. The editor has a dark background with light blue and orange text. The title bar of the editor shows a Docker logo, the filename 'Dockerfile M', and a close button. The content of the Dockerfile is as follows:

```
1 FROM node:10
2 COPY . .
3 RUN npm install
4 EXPOSE 3000
5 CMD ["npm", "start"]
```

The run command ensures that the dependencies for the environment are installed in the container, where as the CMD instruction is what is run at runtime. "npm start" is the command used to start the react webapp. This is hosted by default on port number 3000.

Here are the steps in the Jenkins Pipeline that corresponds to building the Docker Image for the frontend and pushing it to DockerHub:

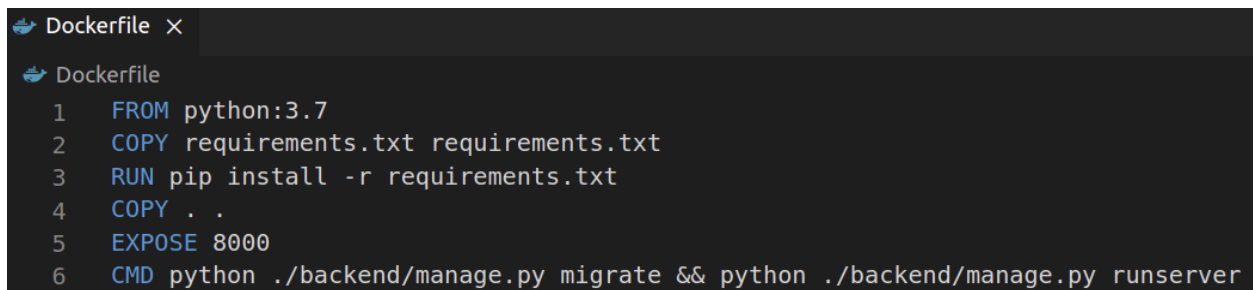

```

stage('Step4: Building docker image:Frontend') {
    steps {
        script {
            dockerImage = docker.build "brahma99/hellofriendfrontend:latest"
        }
    }
}
stage('Step5: Push Docker image to Docker Hub:Frontend') {
    steps {
        script {
            docker.withRegistry('', 'docker-jenkins') {
                dockerImage.push()
            }
        }
    }
}
}

```

Backend

Here is a screenshot of the Dockerfile used for the backend of this project:



```

Dockerfile
Dockerfile
1 FROM python:3.7
2 COPY requirements.txt requirements.txt
3 RUN pip install -r requirements.txt
4 COPY . .
5 EXPOSE 8000
6 CMD python ./backend/manage.py migrate && python ./backend/manage.py runserver

```

The run command ensures that the dependencies for the environment are installed in the container, whereas the CMD instruction is what is run at runtime. The command in the CMD instruction initializes and starts the API server.

Here are the steps in the Jenkins Pipeline that corresponds to building the Docker Image for the backend and pushing it to DockerHub:

```

stage('Step9: Building docker image:Backend') {
    steps {
        script {
            dockerImage = docker.build "swastishreya/hellofriendbackend:latest"
        }
    }
}
stage('Step10: Push Docker image to Docker Hub:Backend') {
    steps {
        script {
            docker.withRegistry('', 'docker-jenkins-swasti') {
                dockerImage.push()
            }
        }
    }
}
}

```

Continuous Deployment using Ansible

Continuous deployment is the process of automating the process of deploying applications to multiple client machines. Continuous deployment enables developers to quickly make the application available to their clients in its current state. Their feedback is quickly brought back to developers and they can fix bugs/add new features, and the cycle continues.

Ansible is actually a configuration management tool. However, it can also be used to push anything to multiple "managed hosts". According to Ansible, there are two types of machines: control nodes and managed hosts. Ansible is installed and run from the control node. A control node houses all the copies of your Ansible project files and configuration information. Managed hosts are systems to which the application/infrastructure as code is pushed. Ansible acts as a central distributor from the control node to all the managed hosts. The managed hosts are listed in the inventory file.

In this project, we used Ansible to pull the Docker images from Docker Hub to the local system we used. For the purpose of this project, we only used one managed host, which was the local machine we used. Shown below is the ansible-playbook file (deploy_docker.yml):

```
! deploy_docker.yml X
docker_deployment > ! deploy_docker.yml
1  ---
2  - name: Pull Docker image of HelloFriend
3    hosts: all
4    vars:
5      ansible_python_interpreter: /usr/bin/python3
6    tasks:
7      - name: Delete a network, disconnecting all containers
8        docker_network:
9          name: network_one
10         state: absent
11         force: yes
12
13      - name: Create a network
14        docker_network:
15          name: network_one
16
17      - name: Pull image HelloFriendFrontend
18        docker_image:
19          name: brahma99/hellofriendfrontend:latest
20          source: pull
21
22      - name: Pull image HelloFriendBackend
23        docker_image:
24          name: swastishreya/hellofriendbackend:latest
25          source: pull
26
27      - name: Pull neo4j Official Docker Image
28        docker_image:
29          name: neo4j
30          source: pull
```

As mentioned earlier, creating our own network and running all three containers on that network was causing problems and we ended up using the host network. Hence, the steps shown below, weren't used:

```
- name: Delete a network, disconnecting all containers
  docker_network:
    name: network_one
    state: absent
    force: yes

- name: Create a network
  docker_network:
    name: network_one
```

These steps deleted the network if already present and recreated the network.

Observe the following line:

```
vars:
  ansible_python_interpreter: /usr/bin/python3
```

This line was needed to point to the correct python version (when there are multiple versions of python installed). To know which, python version to point to, check the ansible_python_version using:

```
ansible -m setup localhost | grep ansible_python_version
```

If the version is 3.x, specify /usr/bin/python3 (as in my case). Otherwise, if the ansible_python_version is 2.x, specify /usr/bin/python instead.

The inventory file is shown below:

```
localhost ansible_user=brahma
```

Logging

Logging is an important part of any development process. It keeps track of all the operations that were performed in the application, errors, warnings, debugging information, etc. To generate the logs, we used the standard logging module of python. Shown below is how we specified the format of the log file that we wanted to generate:

```
import logging

logging.basicConfig(filename="logFile.txt",
                    filemode='a',
                    format='%(asctime)s %(levelname)s-%(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S')
```

Various logging messages like INFO, WARNING, DEBUG, ERROR etc. can be used to analyse the workflow of the application, and monitor and remove bugs whenever they appear.

A snippet from a log file is shown below:

```
2021-05-15 01:05:56 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/hello_friend_db_api/views/connectors.py changed, reloading.
2021-05-15 01:05:56 INFO-Watching for file changes with StatReloader
2021-05-15 16:18:04 INFO-Watching for file changes with StatReloader
```

```

2021-05-15 19:07:45 INFO-Watching for file changes with StatReloader
2021-05-15 19:08:39 ERROR-Invalid HTTP_HOST header: '17fbdabf443b.ngrok.io'. You may need to add '17fbdabf443b.ngrok.io' to ALLOWED_HOSTS.
2021-05-15 19:08:41 WARNING-Bad Request: /
2021-05-15 19:11:43 ERROR-Invalid HTTP_HOST header: '17fbdabf443b.ngrok.io'. You may need to add '17fbdabf443b.ngrok.io' to ALLOWED_HOSTS.
2021-05-15 19:11:43 WARNING-Bad Request: /user
2021-05-15 19:12:20 ERROR-Invalid HTTP_HOST header: '17fbdabf443b.ngrok.io'. You may need to add '17fbdabf443b.ngrok.io' to ALLOWED_HOSTS.
2021-05-15 19:12:20 WARNING-Bad Request: //getAllUsers
2021-05-15 19:12:56 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/backend/settings.py changed, reloading.
2021-05-15 19:12:56 INFO-Watching for file changes with StatReloader
2021-05-15 19:12:58 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/backend/settings.py changed, reloading.
2021-05-15 19:12:58 INFO-Watching for file changes with StatReloader
2021-05-15 19:13:04 ERROR-Error occurred while executing userDetails POST method...
2021-05-15 19:14:32 ERROR-Error occurred while executing userDetails POST method...
2021-05-15 19:21:10 ERROR-Internal Server Error: /user
2021-05-15 19:22:34 ERROR-Error occurred while executing userDetails POST method...
2021-05-15 19:26:52 INFO-Watching for file changes with StatReloader
2021-05-15 19:27:14 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/backend/settings.py changed, reloading.
2021-05-15 19:27:14 INFO-Watching for file changes with StatReloader
2021-05-15 19:29:25 ERROR-Internal Server Error: /user
2021-05-15 19:33:00 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/hello_friend_db_api/views/user.py changed, reloading.
2021-05-15 19:33:00 INFO-Watching for file changes with StatReloader
2021-05-15 19:33:05 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/hello_friend_db_api/views/user.py changed, reloading.
2021-05-15 19:33:05 INFO-Watching for file changes with StatReloader
2021-05-15 19:33:08 ERROR-Internal Server Error: /user
2021-05-15 19:34:21 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/hello_friend_db_api/views/user.py changed, reloading.
2021-05-15 19:34:21 INFO-Watching for file changes with StatReloader
2021-05-15 19:34:23 INFO-/home/swasti/Documents/SPE/HelloFriend/backend/hello_friend_db_api/views/user.py changed, reloading.
2021-05-15 19:34:23 INFO-Watching for file changes with StatReloader

```

Now just try out the application and perform some operations to generate the log file. However, after we close the application, the container automatically exits and the log file remains in the docker container. Hence, we need to copy the log file to our laptop. Follow the following commands to do the same:

```

docker ps -a (This command lists all the latest containers. Use the container id of the most recently closed container for the next step)
docker cp <container-id>:<path-to-log-file-in-container> <dest-path-on-local-machine>

```

Having copied the log file to our local machine, we can now use this with the ELK stack for monitoring.

Monitoring using ELK stack

ELK stands for Elasticsearch, Logstash and Kibana. These three form a stack of sorts, hence the name ELK "stack". They follow the following pipeline:

1. Logstash: collects logs from various sources and user filters like grok to parse the logs and organize them in fields defined by us and how we want to parse the logs.
2. Elasticsearch: is a search and analytics engine that takes input from Logstash and sifts through the preprocessed logs.
3. Kibana: is a visualization engine that runs on top of Elasticsearch. This is used to visualize various aspects of the logs and gain insights that we might not make by just looking at the logs in plaintext form.

The download and set up of the ELK stack was discussed in the Prerequisites section. Before actually running the ELK stack it is important to note that all three of these tools are pretty heavy. For example, Elasticsearch itself uses up around 8.3 GB of RAM when running. Hence, it is recommended that you run it on the cloud or by increasing your swap space (videos for the same can be found on YouTube). We were using 8 GB of RAM. Just to be safe, we made the swap space 8 GB too leaving us with a total of around 16 GB to play with.

Logstash requires a configuration file (.conf file). This file majorly has three sections: Input, Filter, and Output. Input deals with providing the log files as input. In our case, we just provide the path to our log file. The filter section is where the parsing happens. For example, the grok plugin uses regular expressions to match expressions in the logs to patterns we are looking for and map them in fields defined by us. The output section sends the preprocessed logs to a particular destination (ElasticSearch in my case) and also sets an index name that will be useful for Elasticsearch and Kibana. Our configuration file is shown below (calculator.conf):

```

input{
  file{
    path => "/home/swasti/Documents/SPE/HelloFriend/backend/logFile.txt"
    start_position => "beginning"
  }
}
filter
{
  grok{
    match => {"message" => "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:log-level}-%{GREEDYDATA:message}"}
  }
  date {
    match => ["timestamp", "ISO8601"]
  }
}
output{
  elasticsearch{
    hosts => ["localhost:9200"]
    index => "index_name"
  }
  stdout{
    codec => rubydebug
  }
}
}

```

Running the ELK stack

Let's first run ElasticSearch using the following command:

```
<path-to-elastic-search-dir>/bin/elasticsearch
```

ElasticSearch runs on <http://localhost:9200>. Verify that it is running by going to the URL.

Next run Kibana using the following command (on a separate terminal):

```
<path-to-kibana-dir>/bin/kibana
```

Kibana runs on <http://localhost:5601>. Verify that it is running by going to the URL.

Next run Logstash using the following command (on a separate terminal):

```
<path-to-logstash-dir>/bin/logstash -f <path-to-conf-file>
```

When everything is up and running, on the terminal on which we ran Logstash, the console output should look similar to this (please note that the example shown below was from an older project and is shown for illustration purposes only):

```

{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:10 957",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:10 957 [Calculator.java] [INFO] Calculator Performing square root operation on 64.0",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Performing square root operation on 64.0",
  "@timestamp" => 2021-03-17T00:37:10.957Z,
  "log_level" => "INFO"
}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:15 289",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:15 289 [Calculator.java] [INFO] Calculator Performing factorial operation on 3",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Performing factorial operation on 3",
  "@timestamp" => 2021-03-17T00:37:15.289Z,
  "log_level" => "INFO"
}

```

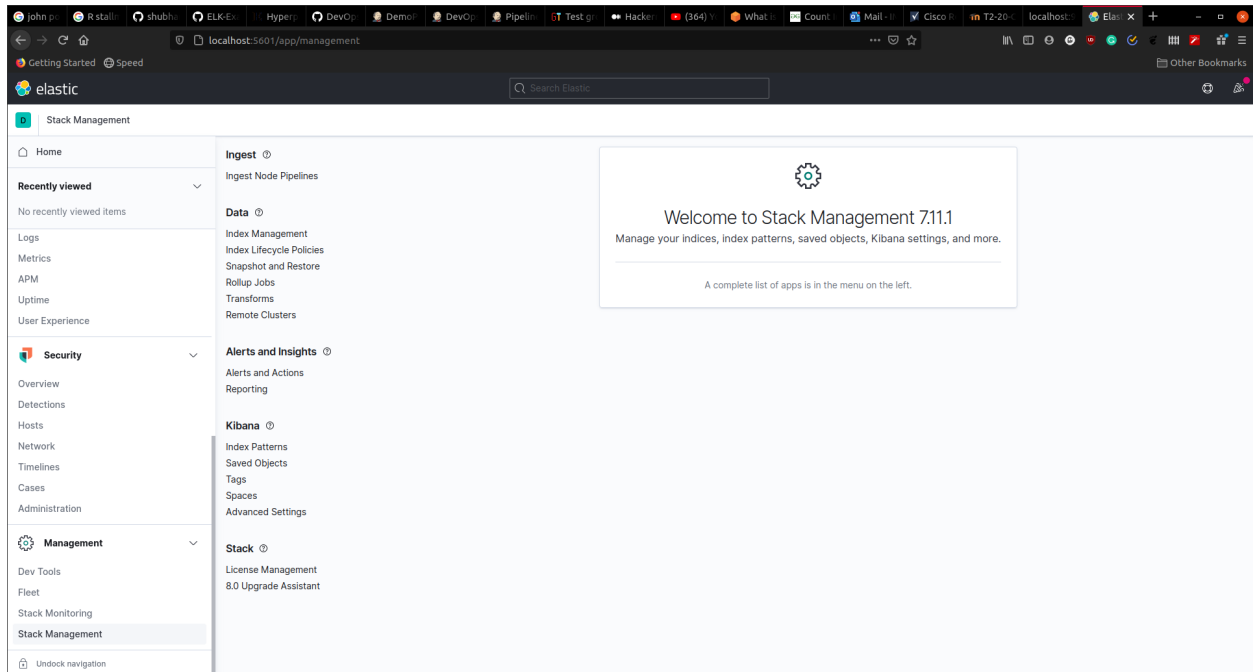
```

}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:15 290",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:15 290 [Calculator.java] [INFO] Calculator Result of factorial operation on 3 is: 6",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Result of factorial operation on 3 is: 6",
  "@timestamp" => 2021-03-17T00:37:15.290Z,
  "log_level" => "INFO"
}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:18 061",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:18 061 [Calculator.java] [INFO] Calculator Performing natural logarithm operation on 10.0",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Performing natural logarithm operation on 10.0",
  "@timestamp" => 2021-03-17T00:37:18.061Z,
  "log_level" => "INFO"
}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:34 631",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:34 631 [Calculator.java] [INFO] Calculator Performing power operation on 2.0,3.0",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Performing power operation on 2.0,3.0",
  "@timestamp" => 2021-03-17T00:37:34.631Z,
  "log_level" => "INFO"
}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:10 980",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:10 980 [Calculator.java] [INFO] Calculator Result of square root operation on 64.0 is: 8.0",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Result of square root operation on 64.0 is: 8.0",
  "@timestamp" => 2021-03-17T00:37:10.980Z,
  "log_level" => "INFO"
}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:18 062",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:18 062 [Calculator.java] [INFO] Calculator Result of natural logarithm operation on 10.0 is: 2.3",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Result of natural logarithm operation on 10.0 is: 2.302585092994046",
  "@timestamp" => 2021-03-17T00:37:18.062Z,
  "log_level" => "INFO"
}
{
  "@version" => "1",
  "timestamp" => "17/Mar/2021:06:07:34 632",
  "host" => "rahul-Lenovo-Y520-15IKBN",
  "message" => "17/Mar/2021:06:07:34 632 [Calculator.java] [INFO] Calculator Result of power operation on 2.0,3.0 is: 8.0",
  "Main_File" => "Calculator.java",
  "path" => "/home/rahul/Downloads/calculator.log",
  "logger_message" => "Calculator Result of power operation on 2.0,3.0 is: 8.0",
  "@timestamp" => 2021-03-17T00:37:34.632Z,
  "log_level" => "INFO"
}
}

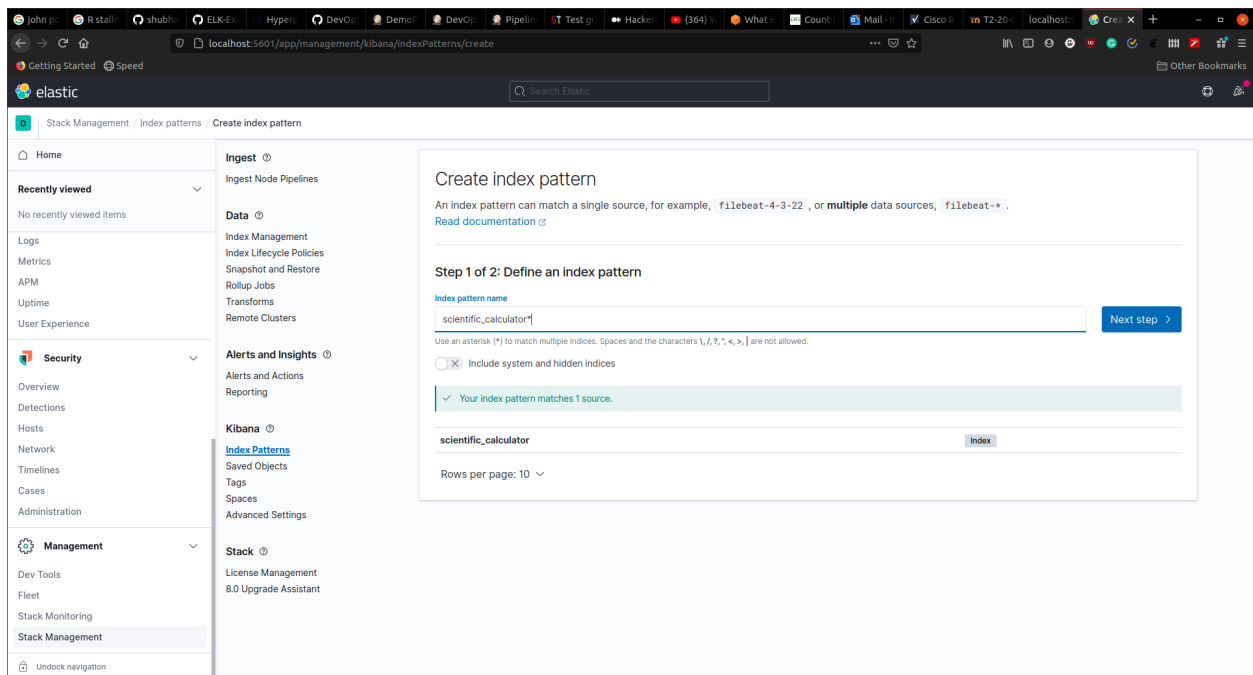
```

Now open a browser and go to <http://localhost:5601>(Kibana) and follow the following steps:

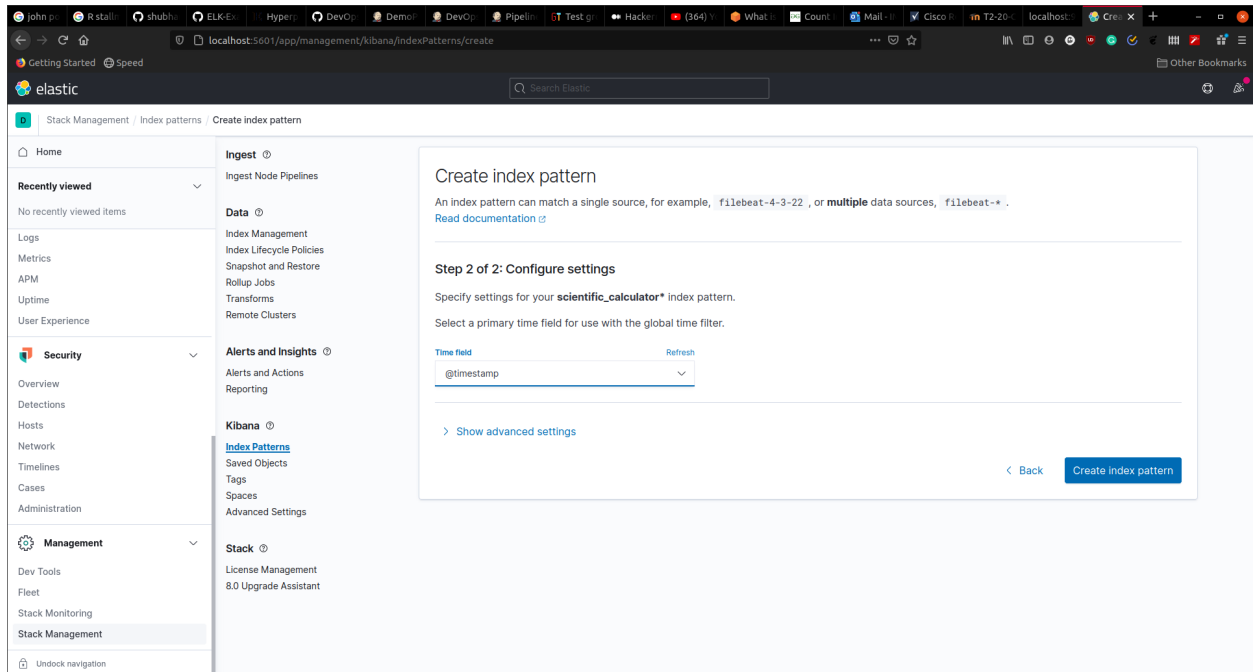
- Go to the sidebar and go to Management → Stack Management
- Then go to Kibana → Index Patterns in the secondary side menu as shown below:



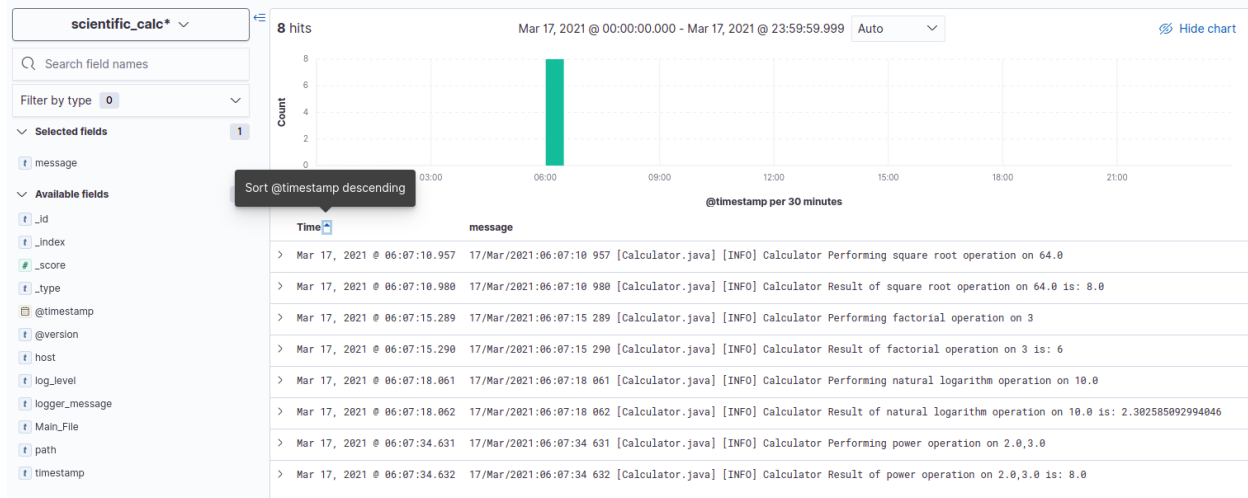
- Then type in a matching index pattern as shown in the image below and click on "Next step" (please note that the example shown below was from an older project and is shown for illustration purposes only).



- On the next page select @timestamp as the option in the "Time field" as shown below and click on "Create index pattern".

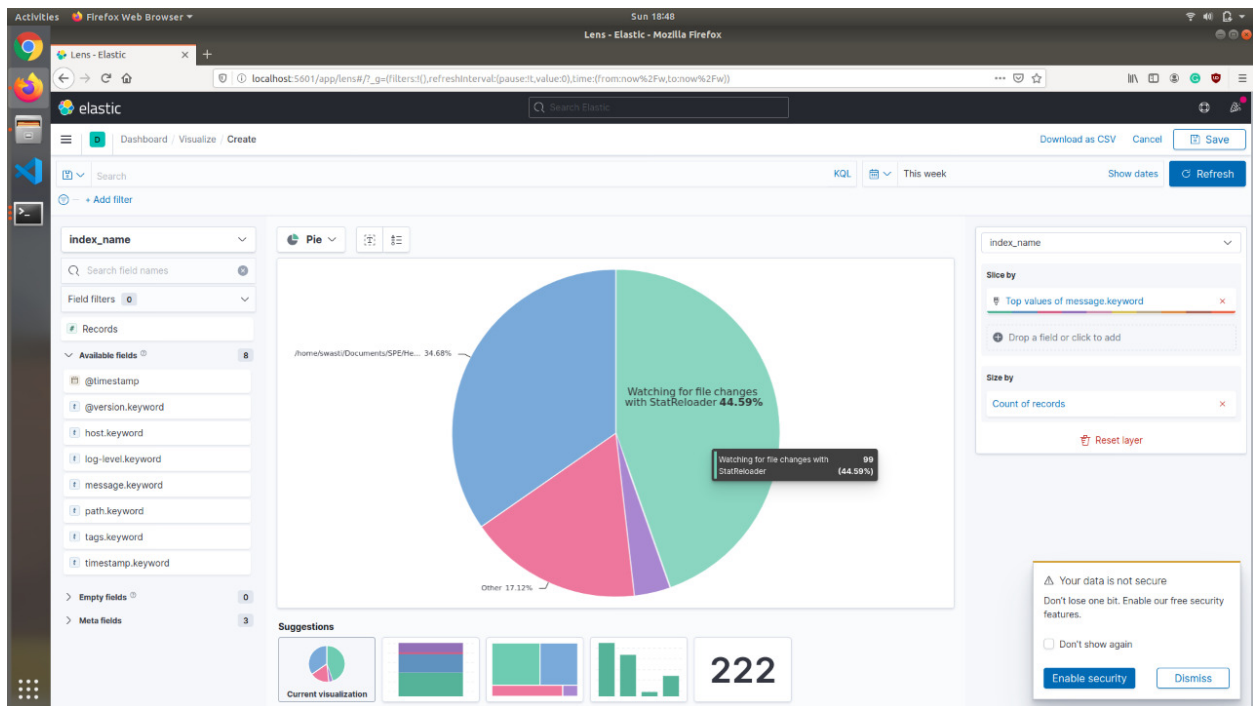
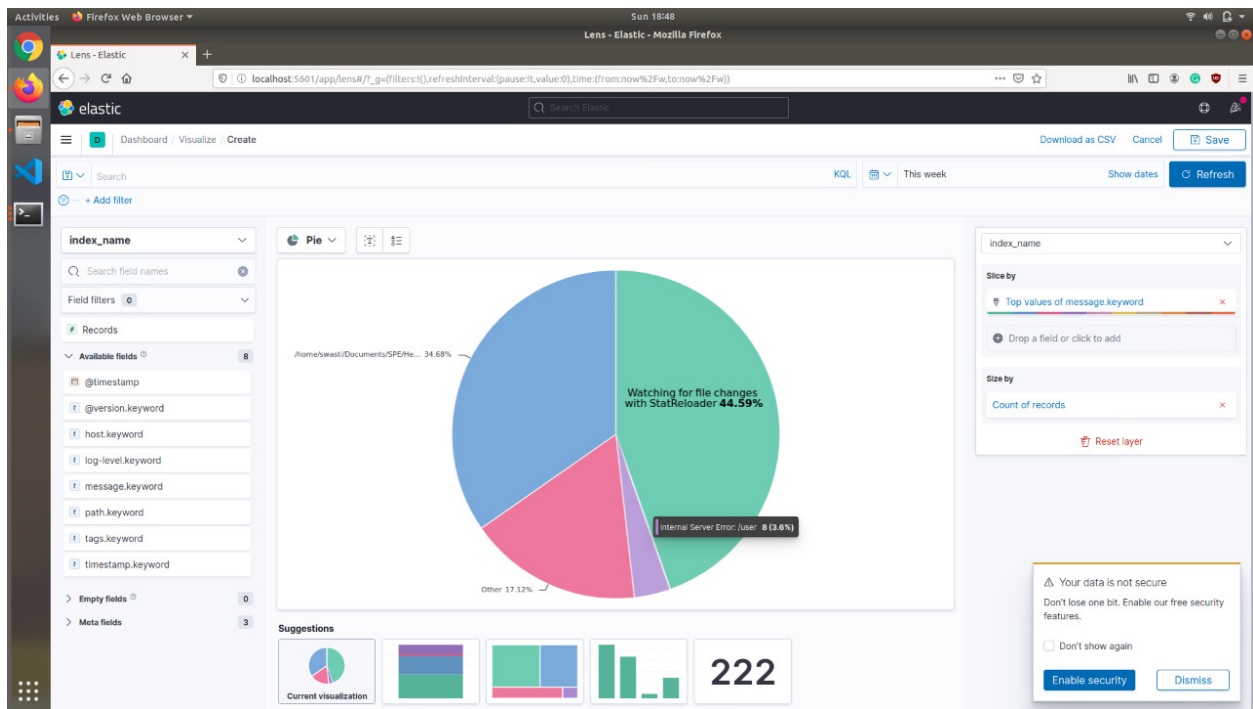


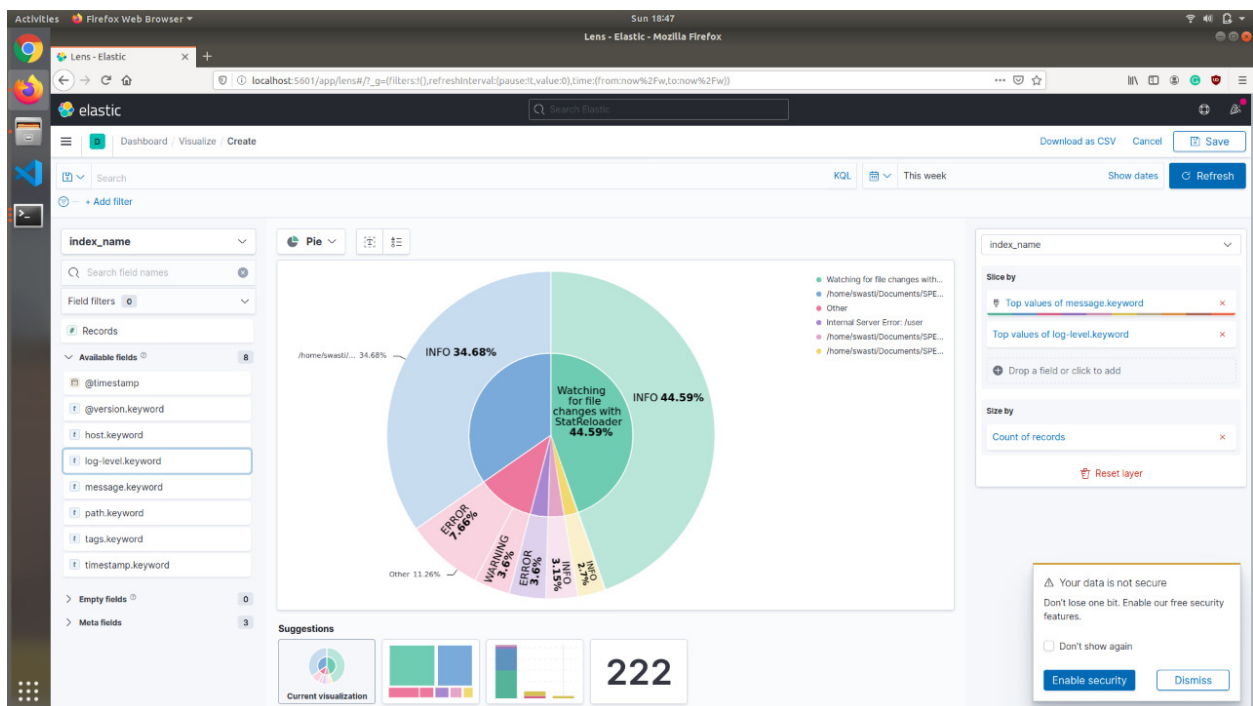
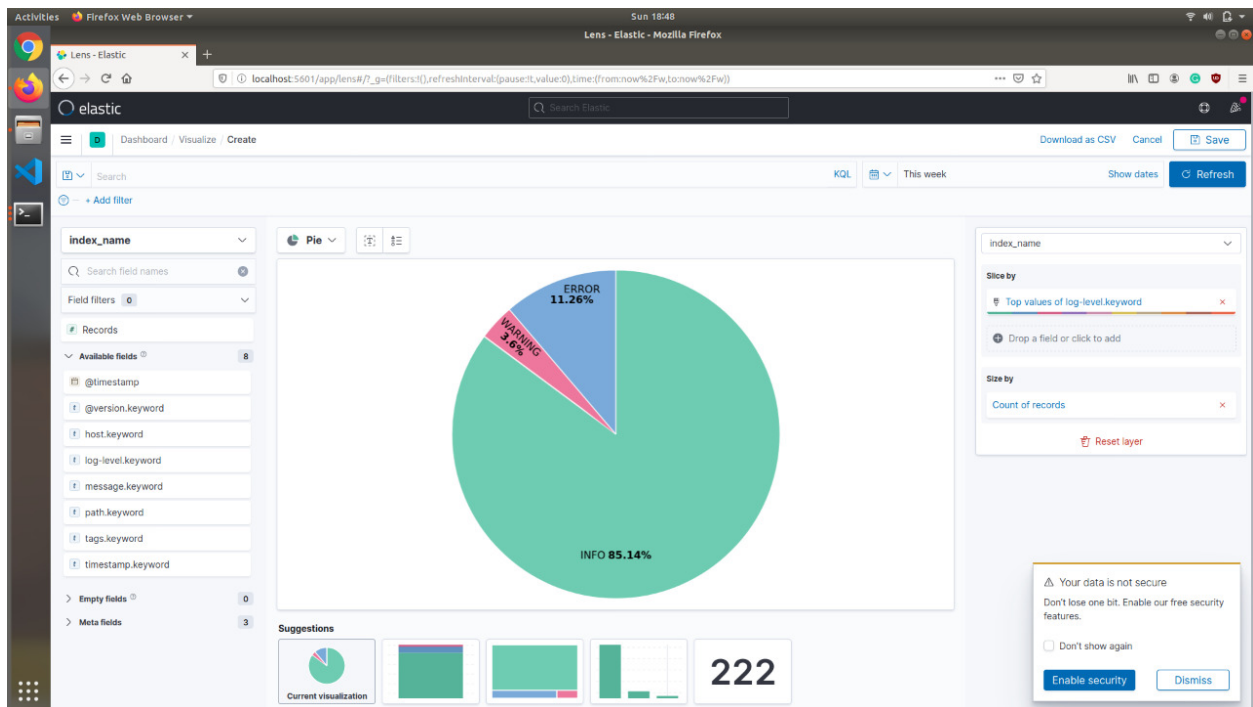
- Now go to Analytics → Discover in the primary side pane to and select the index pattern that we just created in the pull-down menu and adjust the date by clicking on the calendar icon in the top bar to whenever your logs were generated. This page will show us various aspects of the logs and we can use the Elasticsearch powered search to view specific fields, and search for sub content within some fields. This is shown in the image below (please note that the example shown below was from an older project and is shown for illustration purposes only):



- Now go to Analytics → Dashboard in the primary side pane.
- Click on "Create new dashboard"
- Click on "Create panel". This is used to make new visualizations. There are various different kinds of visualizations that are possible. In this project, I used the "Lens" visualization.
- After clicking on Lens, you can just drag and drop the fields you want on to the central portion of the page (where it asks you to drop the attribute) and select a visualization from a plethora visualization options.

Shown below are some of the visualizations that we made:





Results

Screenshots of all the pages of our application are shown below. We will not be delving into the working of the application here as more details of this were given in this section.

- Signup page

Sign Up

Name
Swasti

Age
22

Gender
Female

Email Address
swasti@gmail.com

Password

Confirm Password

Select your interests
Dancing x Anime x Painting x Travelling x x

Signup

Have an account already? Click here to login

- Login page

Login

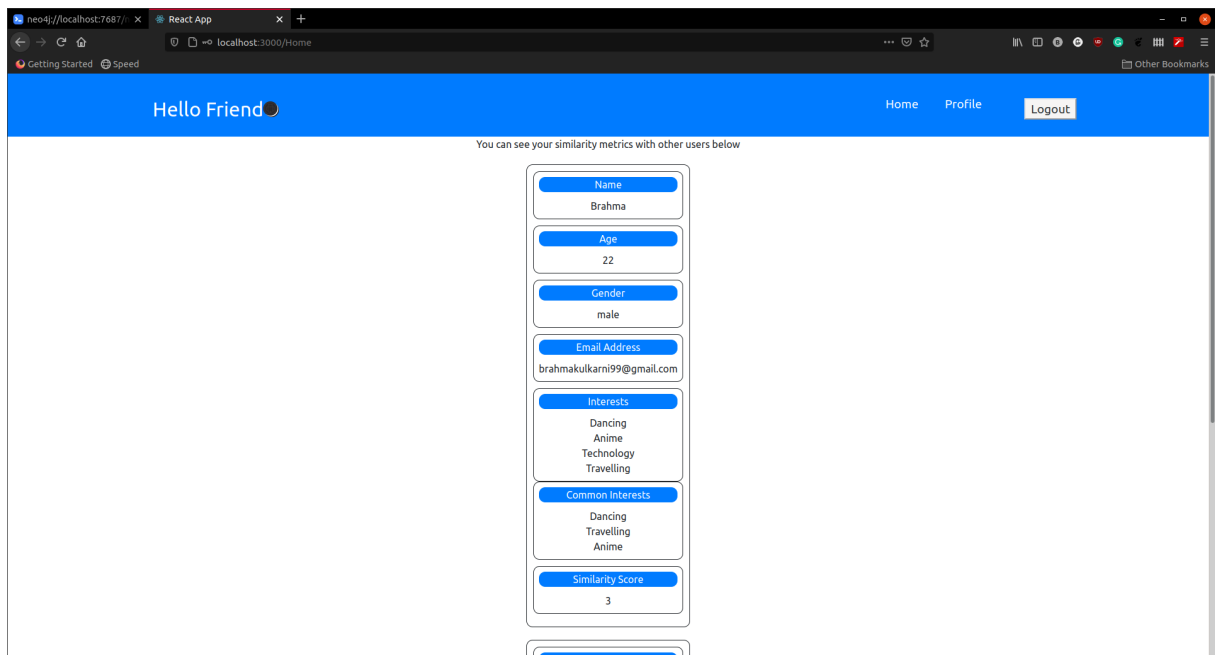
Email
swasti@gmail.com

Password

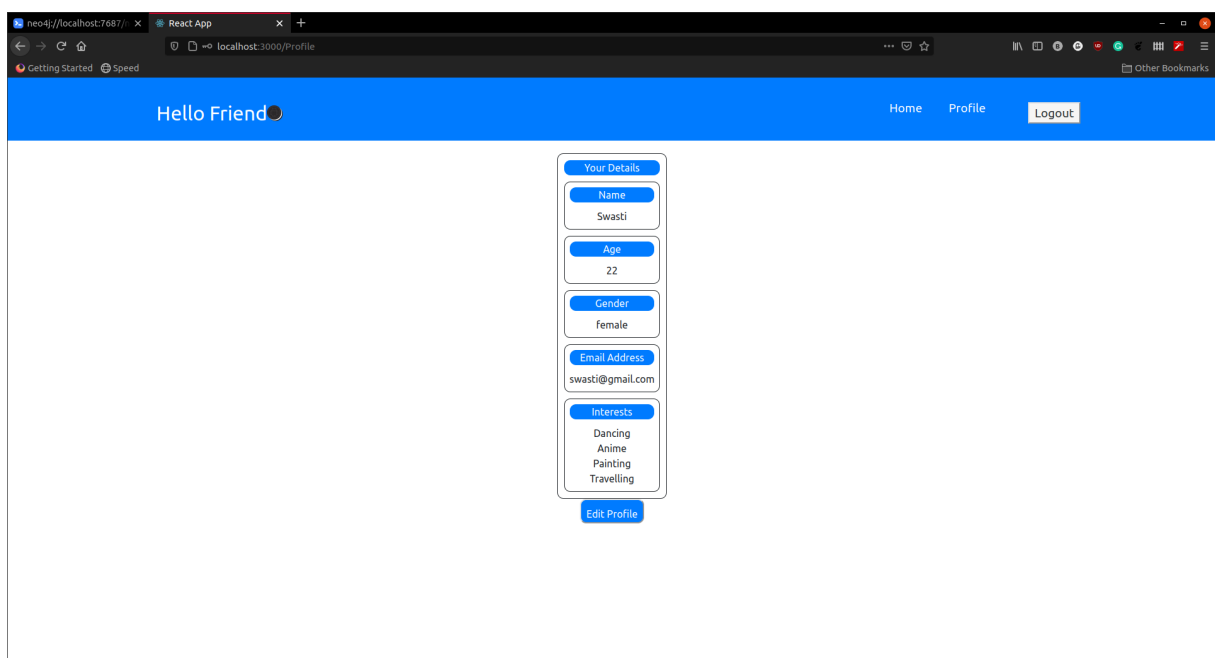
Login

Don't have an account yet? Click here to Sign Up

- Home page



- Profile page



- Edit profile page

ne04j://localhost:7687/ x React App x +

Getting Started Speed

localhost:3000/Editprofile

Hello Friend

Home Profile Logout

Edit Details

Name

Enter name

Age

Enter age

Gender

Select an option

Email Address

Enter email

Password

Enter password

Confirm Password

Confirm password

Select your interests

Select...

Edit

Scope for future work

Our application, HelloFriend is in a rudimentary state at the moment. We are aware that there can be some improvements and some new features that can be added to it. Some of the same are listed below:

- **Powerful Searching capabilities:** We would like to add a powerful search for our application. This would involve searching using various filters making the process of finding similar people that much more simple.
- **Dynamic interests:** At the moment, we have a fixed set of pre-defined interests hard coded in our application. We wish to make it possible for users to enter new interests, making the experience with our application more rich and realistic.
- **Broader deployment:** We would like to deploy our application onto cloud based platform like heroku. This would make it possible for a much larger number of people to use our application.

Conclusion

Through the course of this project we picked up on how to work with technologies/tools like React, Django and Neo4j. That, however, is only about the development side. This project has also made us appreciate the importance of a DevOps pipeline. By doing everything in the DevOps pipeline, from scratch, by ourselves has given us a practical understanding of how a DevOps pipeline benefits a project in practice. If you consider all the steps in this project that we discussed:

1. Planning
2. Coding
3. Building
4. Testing
5. Containerizing
6. Delivering
7. Deploying

As we have seen in this report, steps 3 to 7 were automated. The actual value of this is seen when we consider the time taken for one stretch of automation from building to deployment. In our project, the average time taken for this was only 13 minutes! Looking at the number of steps, we think that is a very short time. This whole process involves rebuilding the code from scratch (when some changes are made to the code base), testing, building docker images, pushing the images to

DockerHub and pulling these images to the deployment machines. Hence, we think such a project helps us really appreciate the importance of DevOps. We can't understand and appreciate this to the same level when it is taught to us as theory.

References

- Neo4j installation: <https://neo4j.com/docs/operations-manual/current/installation/>
- Jenkins installation: <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-18-04>
- Jenkins setup: <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-18-04#step-4---setting-up-jenkins>
- Elasticsearch installation: <https://www.elastic.co/downloads/elasticsearch>
- Logstash installation: <https://www.elastic.co/downloads/logstash>
- Kibana installation: <https://www.elastic.co/downloads/kibana>
- NodeJS installation for ReactJS and reference for frontend testing: <https://www.freecodecamp.org/news/how-to-build-a-react-project-with-create-react-app-in-10-steps/>