Workspace WellBeing Hub - SPE Final Project

By:-

Shaurya Agrawal - IMT2020539

Pushkar Pawar - IMT2020015

Link to Github :- SPE_FINAL_PROJECT

Overview

The Workspace Wellbeing Hub stands as a dedicated platform designed to nurture a positive and collaborative work environment, placing the well-being of employees at its core. By seamlessly integrating a multifaceted discussion forum, the project facilitates open communication, support, and collaboration among team members. The emphasis lies in creating a space where employees can share insights, seek assistance, and engage in discussions that contribute to their overall well-being.

Tech Stack Used :-

In building our project, we leveraged a powerful set of technologies known as the MERN stack. This combination of MongoDB, Express.js, React.js, and Node.js played a crucial role in shaping the architecture and functionality of our web application.

1. MongoDB:

- MongoDB served as our data hub, efficiently storing and managing the essential information for our Application.

2. Express.js:

- Express.js took charge of the server-side tasks, handling the logic and routing of our application.

3. React.js:

- The frontend of our project was crafted with React.js. This library is responsible for creating a user-friendly interface, making it easy and enjoyable for users to interact with the website.

4. Node.js:

- Node.js powered the backend of our website, serving as the engine that makes everything run. It efficiently managed requests, ensuring a responsive and dynamic user experience.

DevOps Tools Used In Our Project :-

1. Continuous Integration and Deployment (CI/CD):

- We implemented Jenkins to establish a robust CI/CD pipeline. This allowed us to automate the building, testing, and deployment of our web application. Jenkinsfile in the source code contains the pipeline script.
- We have used the GitHub hook trigger for GITScm polling and github webhook along with vscode port forwarding to make port 8080 accessible to github.

2. Version Control:

- Git, a widely adopted distributed version control system, was our go-to tool for managing source code

3. Containerization:

- We utilized Docker to containerize our application, ensuring consistent deployment across various environments. Docker significantly reduced environment-related issues. To automate deployment, scaling, and container management, we have used Docker Compose.

- FrontEnd

```
FROM node:latest

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the entire current directory into the container
COPY . .

# Install app dependencies
RUN npm install

# Expose the port the app runs on
EXPOSE 3000

# Define the command to run the application
CMD ["npm", "start"]
```

- Backend

```
# Use an official Node.js runtime as a parent image
FROM node:latest

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the entire current directory into the container
COPY . .

# Install app dependencies
RUN npm install

# Expose the port the app runs on
EXPOSE 8000

# Define the command to run the application
CMD ["npm", "start"]
```

- Database : We have used official mongodb image on dockerhub

4. Ansible Deployment:

- Ansible, an open-source automation tool, played a crucial role in deploying, configuring, and managing systems and applications across multiple servers. Using a declarative language called YAML, Ansible simplified the automation of complex deployment processes.
- Inventory

```
[localhost]
127.0.0.1 ansible_connection=local ansible_user=pushkar
```

- Ansible yml file

```yaml
---
- name: Deploy MERN Application
  hosts: all
  # vars:
  #   ansible_python_interpreter: /usr/bin/python3
  # become: true
  # vars:
  #   ansible_become_pass: "hello"

  tasks:
    - name: Pull Client Image
      docker_image:
        name: pushkar015/client:latest
        source: pull

    - name: Pull Server Image
      docker_image:
        name: pushkar015/server:latest
        source: pull

    - name: Pull Mongo Image
      docker_image:
        name: mongo
        source: pull

    - name: Run Docker Compose
      command: docker compose up -d
```

5. Docker Compose:

Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to describe your entire application stack, including services, networks, and volumes, in a single YAML file called `docker-compose.yml`. This file serves as a configuration blueprint for your application, outlining how different containers interact and function within the overall system.

```yaml
version: '3'
services:
  database:
    image: mongo:latest
    ports:
      - '27017:27017'

  backend:
    image: pushkar015/server
    ports:
      - '8000:8000'
    environment:
      ACCESS_SECRET_KEY: de37bbab03b0380009e4b2f9c6fa6dcf95ae3a
      REFRESH_SECRET_KEY: a60327222ffaa884573490fc0574d69f3f5d1
      DB: mongodb://database:27017/t5
  frontend:
    image: pushkar015/client
    ports:
      - '3000:3000'
```

6. Monitoring and Logging:

- For effective monitoring, we employed the ELK stack (Elasticsearch, Logstash, Kibana).
- For logging we have used Winston tool : `winston` is a popular logging library for Node.js applications. It provides a flexible and extensible logging framework that allows developers to log messages with different levels of severity, format log entries, and transport logs to various destinations.

Following is the configuration code.

```
import winston from "winston";
const { combine, timestamp, json } = winston.format;

const logger = winston.createLogger({
    level: process.env.LOG_LEVEL || 'info',
    format: combine(
        timestamp({ format: "HH:mm:ss" }),
        json()
    ),
    transports: [
        new winston.transports.Console(),
        new winston.transports.File({ filename: 'logs/server.log' })
    ],
});

export default logger;
```

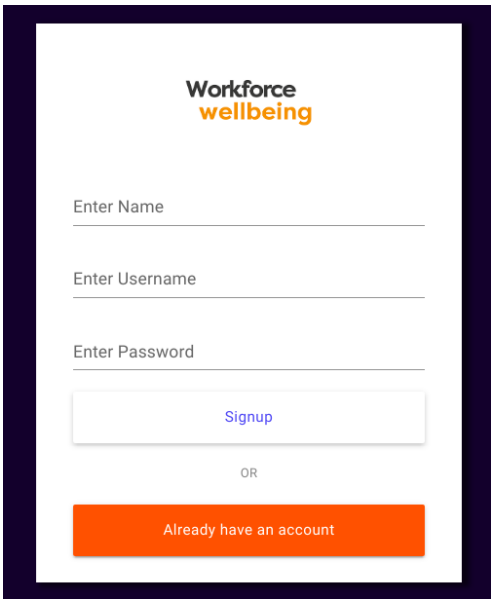
7. Chai and Mocha for testing:

- Mocha and Chai are both popular JavaScript testing libraries/frameworks, commonly used in conjunction for testing applications, especially in the context of Node.js and browser-based JavaScript.
- Mocha is a flexible and feature-rich testing framework for JavaScript. It provides a structure for organizing test suites and test cases, along with various hooks and options for customizing the testing process.
- Chai is an assertion library that works seamlessly with testing frameworks like Mocha. It provides a set of expressive and chainable methods for making assertions about the behavior of code. Chai allows you to write clear and human-readable test specifications.

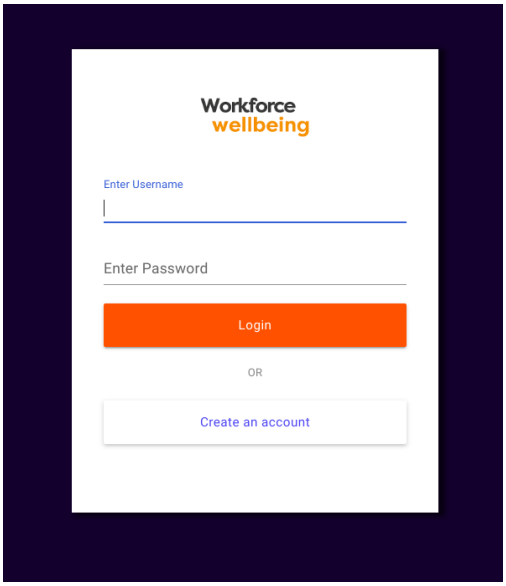
```
const signupData = {
  "name": "temp",
  "username": "temp",
  "password": "temp",
};

describe('POST /signup', () => {
  it('should sign up the user and return a success message', (done) => {
    chai.request(app)
      .post('/signup')
      .send(signupData)
      .end((err, res) => {
        expect(res).to.have.status(200);
        expect(res.body).to.have.property('msg', 'Signup successfull'); // Adjust property name
        done();
      });
  });
});
```

Features:

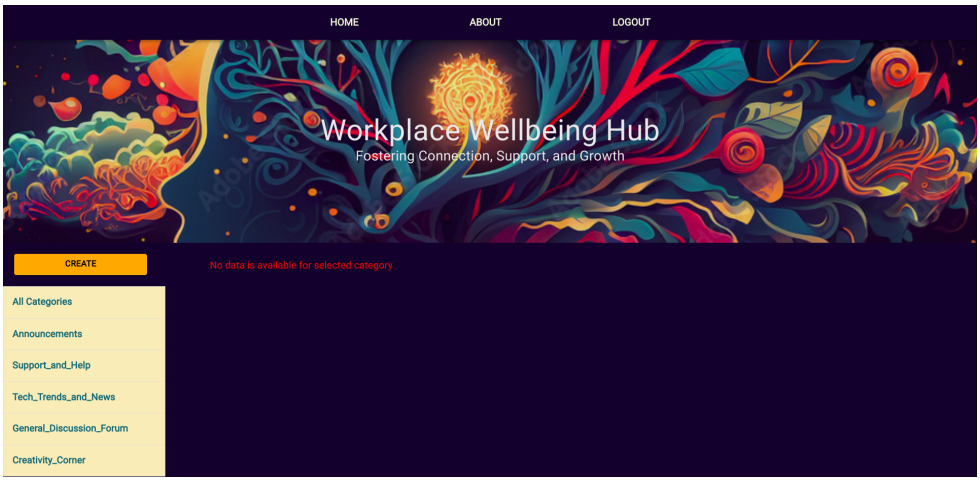
1. Easy Log In:
    a. The users land on the home page, from where they can choose to sign-up if they are a new user.



    b. They can choose to sign-in if they are an existing user.



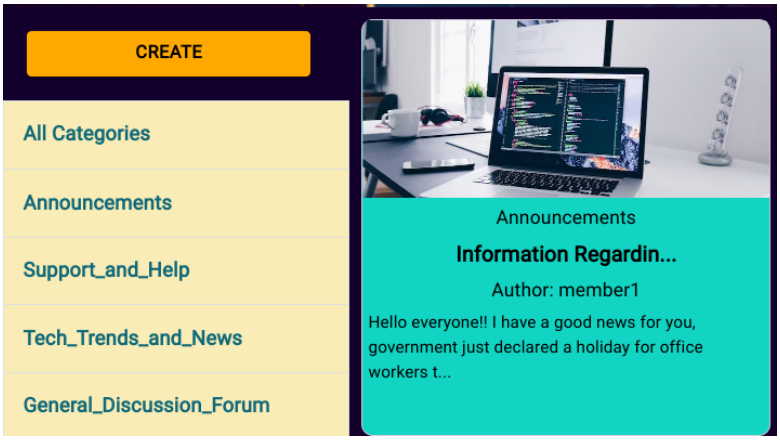
2. Diverse Discussion Sections:
    a. There are 5 different discussion sections but a user can view all posts of various section under All Categories .



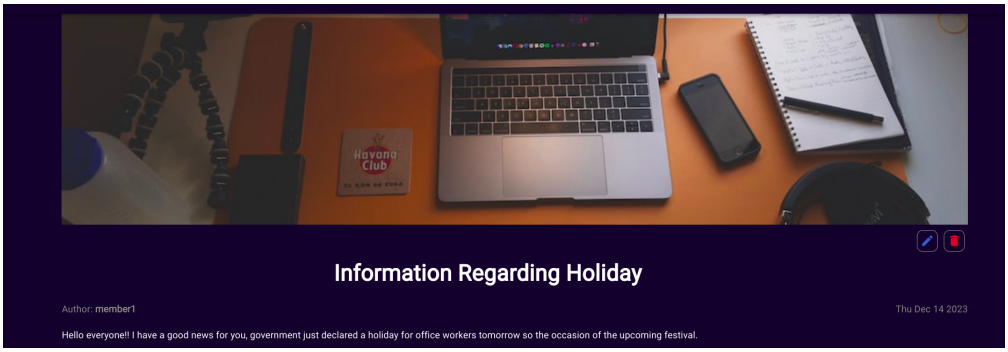
    b. Announcements: Share important updates and celebrate team achievements.
    c. Support and Help: Seek assistance and guidance on work-related challenges.
    d. Tech Trends and News: Engage in discussions about the latest tech happenings.
    e. General Discussion Forum: Create a laid-back space for casual conversations and team bonding.

f. Creativity Corner: Showcase and discuss creative ideas and personal hobbies.

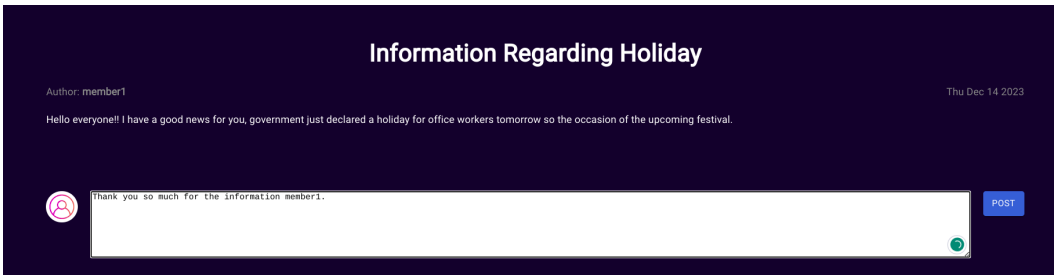
3. Flexible Post Management:
   a. Users can create posts to share information and updates.

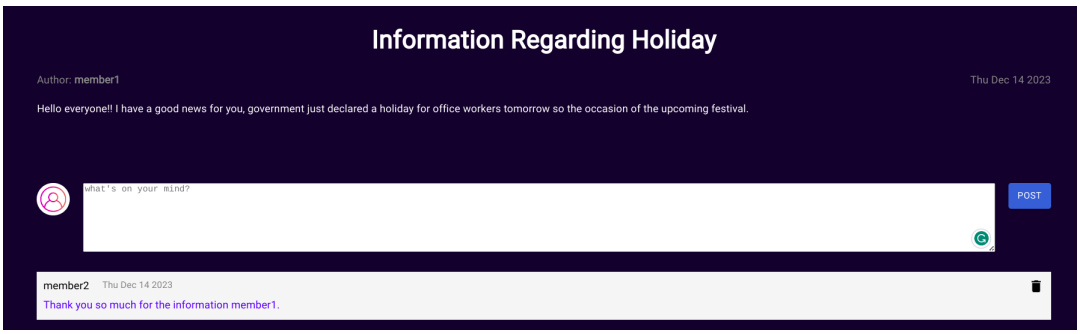


   b. Edit and delete options empower individuals to keep content relevant and up-to-date.



4. Interactive Commenting:
    a. Team members can leave comments on posts, encouraging a supportive and collaborative environment.





    b. Comments can only be deleted, ensuring respectful and constructive interactions.
5. Emphasis on Well-being:
    a. The platform is intentionally designed to enhance overall employee well-being.
    b. Encourages positive communication and interaction to strengthen team bonds.

Extended Use Cases:

Team Building Initiatives:

- Create posts or discussions aimed at team-building activities, fostering a sense of unity.

Mentorship Opportunities:

- Dedicate a space for mentorship discussions, allowing team members to seek guidance and advice.

Employee Recognition Corner:

- Celebrate achievements and milestones to boost morale and acknowledge team efforts.

Wellness Resources:

- Share resources related to employee wellness, such as health tips, mindfulness practices, and stress management techniques.

Feedback and Improvement Discussions:

- Facilitate discussions for constructive feedback and improvement ideas to enhance team dynamics.

WorkspaceWellbeing API Documentation:

User Authentication

Login

- Endpoint: `/login`
- Method: `POST`
- Description: Allows a user to log in and obtain an authentication token.
- Request Body:
  - `username` (string): User's username
  - `password` (string): User's password
- Response:
  - Successful login: 200 OK with a access and refresh token

    - Failed login: 400 Password does not match
    - Failed login: 500 error while login the user

Signup

- Endpoint: `/signup`
- Method: `POST`
- Description: Allows a new user to sign up and create an account.
- Request Body:
    - `username` (string): New user's desired username
    - `password` (string): New user's desired password
- Response:
    - Successful signup: 200 Signup successfull
    - Failed signup: 500 Error while signing up user

Logout

- Endpoint: `/logout`
- Method: `POST`
- Description: Allows a user to log out by invalidating the authentication token.
- Request Body:
    - None
- Response:
    - Successful logout: 204 Logout Successfull

Posts

Create Post

- Endpoint: `/create`
- Method: `POST`
- Description: Allows a user to create a new post.
- Request Body:
    - `title` (string): Title of the post
    - `content` (string): Content of the post
- Response:
    - Successful post creation: 200 Post saved successfully

- Failed post creation: 500 error

Update Post

- Endpoint: `/update/:id`
- Method: `PUT`
- Description: Allows a user to update an existing post.
- Request Params:
  - `id` (string): ID of the post to be updated
- Request Body:
  - `title` (string): New title for the post
  - `content` (string): New content for the post
- Response:
  - Successful post update: 200 post updated successfully
  - Failed post update: 500 error or 404 post not found

Delete Post

- Endpoint: `/delete/:id`
- Method: `DELETE`
- Description: Allows a user to delete an existing post.
- Request Params:
  - `id` (string): ID of the post to be deleted
- Response:
  - Successful post deletion: 200 post deleted successfully
  - Failed post deletion: 500 error

Get Single Post

- Endpoint: `/post/:id`
- Method: `GET`
- Description: Retrieves details of a single post.
- Request Params:
  - `id` (string): ID of the post to be retrieved
- Response:
  - Successful retrieval: 200 post
  - Failed retrieval: 404 error

Get All Posts

- Endpoint: `/posts`
- Method: `GET`
- Description: Retrieves details of all posts.
- Response:
    - Successful retrieval: 200 posts
    - Failed retrieval: 500 error

Comments

Create Comment

- Endpoint: `/comment/new`
- Method: `POST`
- Description: Allows a user to create a new comment on a post.
- Request Body:
    - `postId` (string): ID of the post to comment on
    - `content` (string): Content of the comment
- Response:
    - Successful comment creation: 200 Comment Created Successfully
    - Failed comment creation: 500 error

Get Comments for a Post

- Endpoint: `/comments/:id`
- Method: `GET`
- Description: Retrieves all comments for a specific post.
- Request Params:
    - `id` (string): ID of the post to retrieve comments for
- Response:
    - Successful retrieval: 200 comments
    - Failed retrieval: 500 error

Delete Comment

- Endpoint: `/comment/delete/:id`
- Method: `DELETE`
- Description: Allows a user to delete a specific comment.
- Request Params:
    - `id` (string): ID of the comment to be deleted
- Response:
    - Successful comment deletion: 200 Comment Deleted Successfully
    - Failed comment deletion: 500 error

Pipeline Screenshot

SPE_FINAL - Stage View

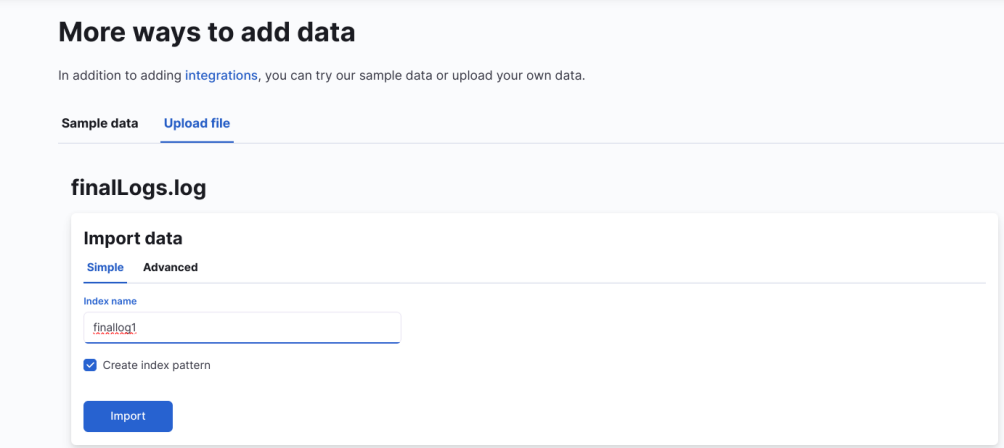
	Declarative: Checkout SCM	Stage 1: Git Clone	Stage 2: Stopping old containers	Stage 3: Testing	Stage 4: Build Client Docker Image	Stage 5: Build Server Docker Image	Stage 6: Push docker image to hub	Stage 7: Clean docker images	Stage 8: Ansible Deployment
Average stage times: (Average full run time: ~1min 51s)	1s	805ms	6s	16s	20s	7s	45s	731ms	7s
#45 Dec 15 12:40 — 1 commit	1s	760ms	12s	37s	1min 10s	9s	1min 33s	684ms	7s
#44 Dec 15 11:48 — 1 commit	1s	729ms	12s	7s	3s	6s	28s	632ms	6s
#43 Dec 15 11:35 — No Changes	1s	791ms	1s	12s	5s	7s	34s	650ms	8s
#42 Dec 14 10:47 — 1 commit	1s	940ms	397ms	8s	3s	8s	24s	959ms	7s

Using ELK Stack :

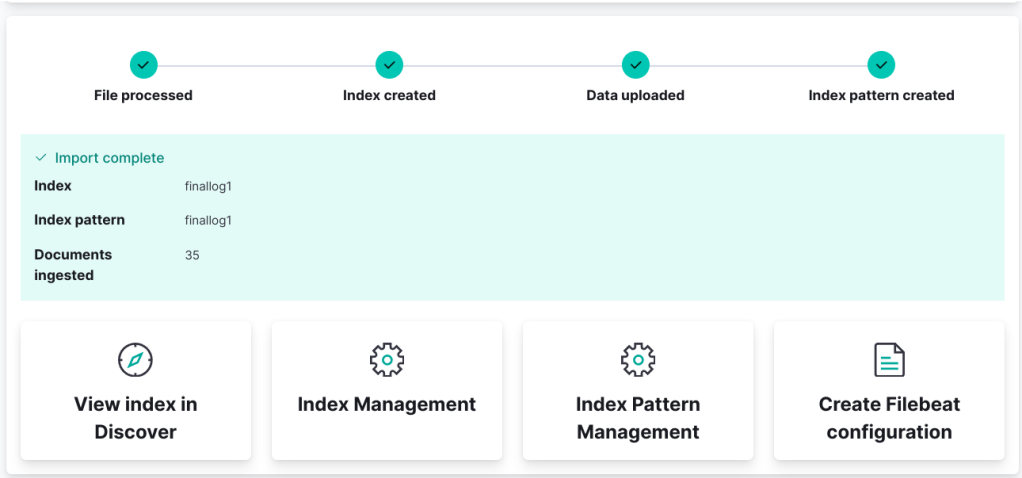
We go to http://localhost:5601/ to enter the Kibana dashboard. In the dashboard, we click on Upload a file and choose the file that we want to upload, in this case the log file from the container we copied out: Once the file shows up, we can see the file show up along with some details like the number of lines of code. More importantly, we can see that there are few settings that are already pre applied by Kibana like identifying the GROK pattern and the time pattern.

Once we import the log file, Kibana will then ask you to create an index name. The option index pattern will also be checked. This is for using multiple log files across

multiple nodes, which our project doesn't concern about so we will create a dummy index called finallog1 and leave index pattern checked and click on import.

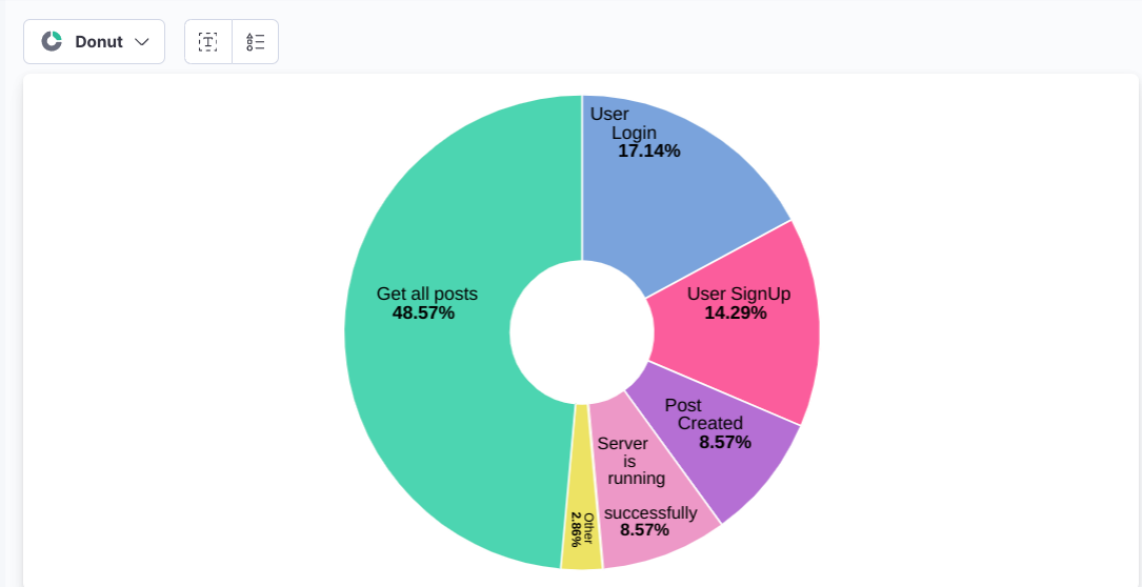


If we scroll down, there will be an option called View index in Discover which will allow you to see our log file and analyse it.



From here, it is a matter of filtering the logs according to various parameters. We can filter based on the time stamp or on the fields that it identified. After we have filtered down the logs, we can generate visualizations that match your requirements and analysis .

Here are some of the screenshots of the visualizations .


Donut
User Login 17.14%
User SignUp 14.29%
Post Created 8.57%
Server is running successfully 8.57%
Other 2.86%
Get all posts 48.57%


Bar vertical stacked
Count of records
22
20
18
16
14
12
10
8
6
4
2
0
10:36:46
10:47:22
05:20:32
05:21:11
05:21:29
Other
Top values of timestamp


35
30
25
20
15
10
5
0
Count of records
info
Top values of level