SPE Major Project Report

BloggedIt

Under Guidance of Prof. B. Thangaraju
Teaching Assistant : Chirag

GitHub Links
Front End Repo : Link
Back End Repo : Link

Submitted by
Budideti Sashank Reddy IMT2020542
Rithvik Ramasani IMT2020543

Introduction

Project Overview :

"Blogged it" is a website where people can write and read blogs. If you have something to share, you can create your own blog easily. And if you want to explore different ideas or stories, there are plenty of blogs available for you to read.

Things we can do:

Create and Read Blogs: You can write your own blogs with a title, your thoughts, and some keywords along with a blog image. Others can read what you write, and you can read blogs from different people too.

Talk About Blogs: If you like someone's blog, you can tell them by leaving a like or a comment. It's a way to chat or share your thoughts about what you read.

Find What You Want: There's a search bar to help you find specific blogs. You can search by title, author, or topic.

Overall, "Blogged it" is all about sharing and reading interesting stories and ideas.

Features of the project :

1. Blog Creation & Reading: Users can create their own blogs with titles, messages, and tags. Others can read these blogs easily.

2. User Access: Anyone without a login can read blogs available on the website. Logged-in users can create their own blogs.

3. Editing & Deleting Blogs: Blog creators have the ability to edit or delete their own blogs.

4. Likes & Comments: Users who are logged in can like blogs and leave comments to interact with the content and its creators.

5. Structured Blog Content: Each blog contains a title, message (content), blog image, and tags, allowing users to organise and find specific content easily.

6. Search Functionality: Users can search for blogs based on titles and tags, making it convenient to find relevant content.

DevOps

1. DevOps integrates engineers from both operations and development teams across all stages of a service's lifecycle, spanning from design to development and ongoing production support.

2. DevOps merges software development (Dev) with information technology operations (Ops) to accelerate the system development cycle. It aims to frequently deliver features, fixes, and updates in close alignment with business goals.

3. DevOps holds significance by enabling organisations to produce software quicker, more reliably, with superior quality, and at reduced costs. Traditionally, the separation between software development and operations led to fragmented processes, communication gaps, and sluggish software delivery, often failing to meet user needs.

4. DevOps tackles these issues by uniting development and operations teams, encouraging collaboration throughout the software development lifecycle. Leveraging automation, continuous integration, and delivery, it streamlines development, diminishes manual errors, and speeds up feedback loops.

5. Collaboration between development and operations teams forms the core of DevOps, aiming to create an automated, efficient software delivery process. The main objective is to establish a continuous delivery pipeline that ensures high-quality software is delivered rapidly and efficiently, minimising errors and enhancing communication and collaboration among various stakeholders. This is achieved through tool usage, automation,

continuous integration, delivery, and monitoring to guarantee swift, reliable, and secure software delivery.

Furthermore, DevOps fosters a culture of teamwork, communication, and responsibility among all involved teams, including developers, testers, operations, and stakeholders. This ensures a collective effort towards swift, reliable, and high-quality software delivery, addressing evolving business requirements.

Technical Stack and DevOps tools

1. Frontend : React js
2. Backend : Express, NodeJS
3. Database : MongoDB
4. Version Control System : Github
5. CI/CD pipeline : Jenkins
6. Testing : Jest, Supertest
7. Containerisation : Docker
8. Deployment : Ansible
9. Continuous Monitoring : ELK

Source Control System and GitHub

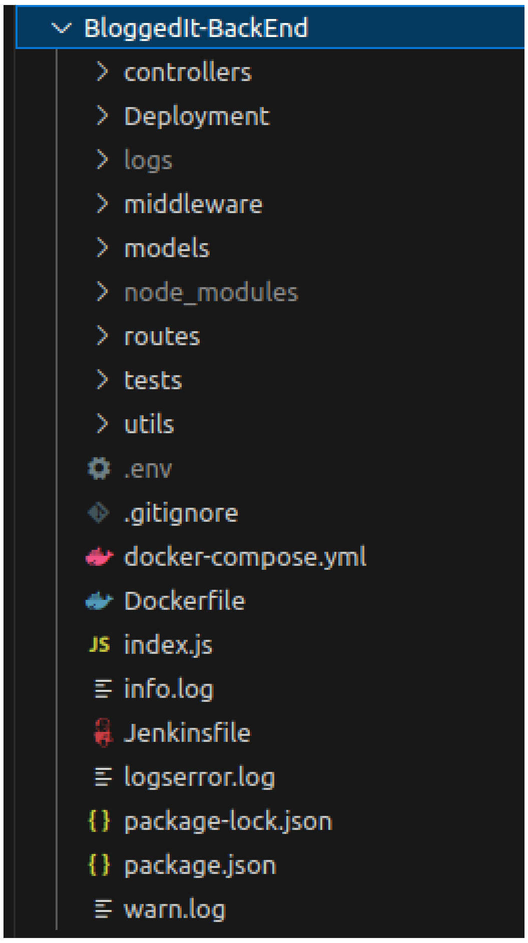
Source Control Management(SCM) is a system that helps track changes made to code or files in a project. It maintains a history of modifications, allowing multiple people to work on the same project simultaneously without interfering with each other's work. SCM helps in organising, controlling, and coordinating different versions of files or code.

GitHub, on the other hand, is a popular web-based platform that provides hosting for SCM using Git. Git is a specific type of SCM that GitHub uses.

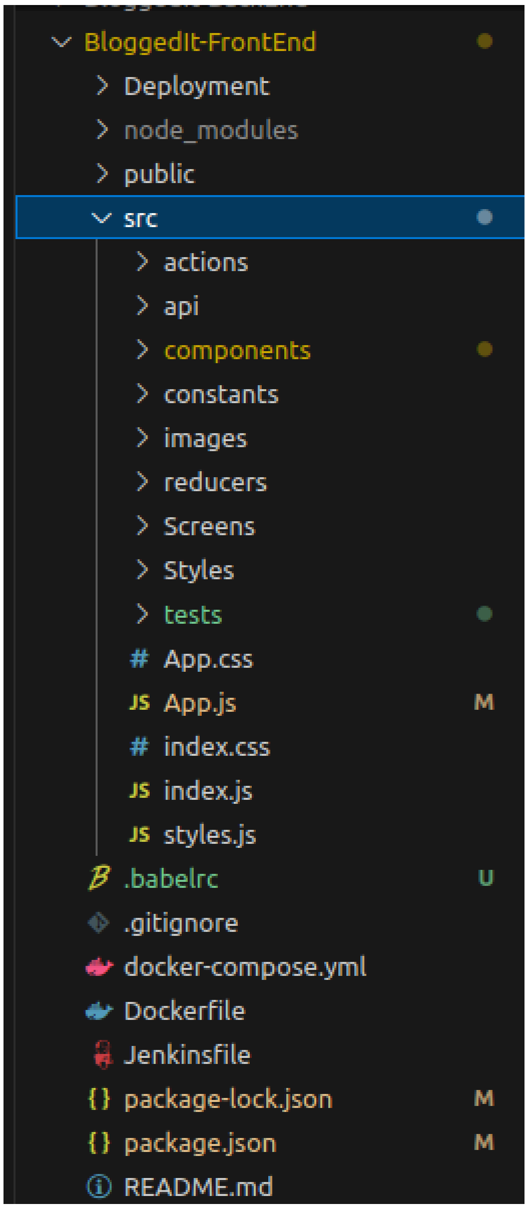
GitHub allows developers to store their Git repositories online and offers a variety of features to collaborate on projects. It includes functionalities like version control, issue tracking, project management, and collaboration tools. It's widely used in the software development community for open-source and private projects.

GitHub has become a central platform for many software projects due to its ease of use, collaboration features, and its ability to integrate with other development tools and services.

Directory Structure



```
BloggedIt-BackEnd
  controllers
  Deployment
  logs
  middleware
  models
  node_modules
  routes
  tests
  utils
  .env
  .gitignore
  docker-compose.yml
  Dockerfile
  index.js
  info.log
  Jenkinsfile
  logserror.log
  package-lock.json
  package.json
  warn.log
```



```
BloggedIt-FrontEnd
  Deployment
  node_modules
  public
  src
    actions
    api
    components
    constants
    images
    reducers
    Screens
    Styles
    tests
    App.css
    App.js                M
    index.css
    index.js
    styles.js
  .babelrc                U
  .gitignore
  docker-compose.yml
  Dockerfile
  Jenkinsfile
  package-lock.json       M
  package.json            M
  README.md
```

Server Side Architecture

Backend Overview:

1. **Routes:**
   - The backend defines several routes to handle different functionalities of the "Blogged it" application. These routes are responsible for handling incoming HTTP requests and directing them to the appropriate controllers.

2. **Controllers:**
   - Controllers contain the logic for handling the business operations related to each route. They interact with the database, validate input, and send appropriate responses. There are controllers for blogs and users.

3. **Middlewares:**
   - Middleware functions are executed before reaching the route's handler function. In this case, there is an authentication middleware (auth.js) that verifies the user's identity using JSON Web Tokens (JWT). This ensures that certain routes are only accessible to authenticated users.
4. **Blog Routes:**
   - /blogs (GET): Retrieve a paginated list of blogs.
   - /blogs/{id} (GET): Retrieve a single blog by ID.
   - /blogs/search (GET): Retrieve blogs based on a search query.
   - /blogs (POST): Create a new blog.
   - /blogs/{id} (PATCH): Update a blog by ID.
   - /blogs/{id} (DELETE): Delete a blog by ID.
   - /blogs/{id}/likeBlog (PATCH): Like a blog by ID.
   - /blogs/{id}/commentBlog (POST): Comment on a blog by ID.

5. **User Routes:**
   - /user/signin (POST): Sign in a user.
   - /user/signup (POST): Sign up a user.

6. **Authorization:**

- Authorization is handled using the auth middleware, ensuring that only authenticated users can perform certain actions, such as creating, updating, or deleting blogs.

Database Overview

1. **Database Connection:**
   - The backend connects to MongoDB Atlas using a connection URL, establishing a link between the application and the database.

2. **User Schema:**
   - Defines the structure of the User collection in the database.
   - Fields: name, email, password, and id.
   - The id field is likely an identifier for the user.

3. **Blog Schema:**
   - Defines the structure of the BlogMessage collection in the database.
   - Fields: title, message, name, creator, tags, selectedFile, likes, comments, and createdAt.
   - likes and comments are arrays to store user IDs who liked the blog and comments on the blog.
   - createdAt is automatically set to the current date when a blog is created.

4. **Models:**
   - The defined schemas are used to create models (User and BlogMessage) that interact with the MongoDB database. These models are then exported for use in the controllers.

Overall Flow:

1. **User Interaction:**
   - Users interact with the frontend, triggering HTTP requests based on their actions, such as creating a blog, liking a blog, or searching for blogs.

2. **Routes:**
   - The backend routes receive these HTTP requests and direct them to the appropriate controllers.

3. **Controllers:**
   - Controllers handle the business logic, which includes interacting with the MongoDB database using the defined models.

4. **Database Operations:**
   - The controllers perform database operations, such as retrieving, creating, updating, or deleting data, based on the user's request.

5. **Response:**
   - Appropriate responses are sent back to the frontend based on the success or failure of the database operations.

The overall architecture ensures a clear separation of concerns and follows the MERN stack (MongoDB, Express.js, React, Node.js) for full-stack development.

Client Side Architecture

Front End Overview:

1. **Actions:**
   - The frontend actions are responsible for triggering API requests to the backend and dispatching data to the reducers based on the results.

2. **API Functions:**
   - API functions are responsible for making HTTP requests to the backend. They use Axios to communicate with the defined backend API endpoints.

3. **Reducers:**
   - Reducers specify how the application's state should change in response to actions. They manage the state related to authentication and blog data.

4. **Components:**
   - Components include various reusable elements that are to be displayed on our web application.

Frontend Flow:

Signin/Signup Flow:

1. *Redux Action (signin/signup):*
   - The user initiates a sign-in or sign-up process by interacting with the UI components.
   - Redux actions (signin and signup) are dispatched, containing the user input data and a history object for navigation.

2. *API Request (api.signIn/api.signUp):*
   - The Redux actions make asynchronous API requests to the backend using axios.
   - Axios interceptors are used to attach the authentication token (if available) to the headers for authorization.

3. *Backend Processing:*
   - The backend processes the sign-in or sign-up request.
   - JWT authentication is performed on the backend, and if successful, the backend sends back relevant user data along with a new JWT token.

4. *Redux Action (AUTH):*
   - The Redux action updates the state by dispatching the AUTH action, which includes the user data obtained from the backend.

Blog Interaction Flow:

1. *Redux Action (getBlogs, getBlog, etc):*
   - User interacts with blog-related UI components triggering actions such as fetching all blogs, fetching a specific blog, creating a blog, updating a blog, etc.

2. *API Request (api.fetchBlogs, api.createBlog, etc.):*
   - Corresponding Redux actions make API requests to the backend to perform operations on blogs.

3. *Backend Processing:*
   - The backend processes the requests for fetching, creating, updating, or deleting blogs based on the specific API endpoint.

4. *Redux Action (FETCH_ALL, CREATE, UPDATE, etc.):*
   - The Redux actions (FETCH_ALL, CREATE, UPDATE, etc.) update the state with the new data obtained from the backend.

Reducer Flow:
1. *Redux Reducer (authReducer, blogsReducer):*
   - The authReducer handles authentication-related actions, updating the authentication state.
   - The blogsReducer manages blog-related actions, updating the blog-related state.
2. *State Update:*
   - The Redux reducers update the state based on the dispatched actions.

In summary, the flow involves user interaction triggering Redux actions, which in turn make API requests to the backend. The backend processes these requests, and upon success, the Redux state is updated, triggering UI rerendering. The state is managed by corresponding reducers, and the overall application state is maintained by the combined reducers.

Building and Packaging :

Developing an application includes the compilation and bundling of its source code into a package ready for deployment. This package typically contains all essential dependencies, configuration files, and assets necessary for the application's operation. The process of building an application often entails utilising a build tool or employing a CI/CD pipeline to automate these steps. Packaging the application involves crafting a distribution-ready package post its construction. This package might take the form of a container image, a ZIP archive, or other applicable formats. Packaging usually involves the use of a packaging tool or integrating it into a CI/CD pipeline for automated execution.

To build and package a React application we use npm. Node Package Manager, is a tool that comes with Node.js. It helps in managing and installing packages or libraries for JavaScript and Node.js projects. These packages could contain reusable code, frameworks, or tools that developers use to build applications.

npm allows developers to easily share and reuse code, making development more efficient.

Building and packaging a React website using npm involves following steps :

1. Initialization: Use npm init to set up your project and create a package.json file.

2. Installing React: Add React and necessary dependencies using commands like npm install react, react-dom, webpack, and babel.

3. npm Scripts: Define scripts in package.json for tasks like starting a development server (npm start) or building the project (npm run build)

4. Running Commands: Use npm to execute scripts for development and build processes.

5. Building and Packaging: Running npm run build compiles and optimises the React app for deployment, creating optimised files ready for hosting.

CI/CD Pipeline

Here for this project we created two different github repositories and we created two pipelines separately for the front end and back end.In both the pipelines we created the same stages just there will be few url changes.

The jenkins pipeline contains following stages :

Environment Configuration:

- registry: Holds the Docker registry/repository path where the Docker image will be stored.
- dockerImage: Variable to store the created Docker image.
- PORT: Port number for the application (set to 5000).

- CONNECTION_URL: Retrieves a stored credential named 'CONNECTION_URL', presumably holding a secure connection URL (Here mongoDB URL).

Stage 1: Git Clone

- Objective: Clones the code from a specified GitHub repository.
- Steps: Uses the Git plugin to clone the 'main' branch of the repository specified in the URL.

Stage 2: Test

- Objective: Executes testing for the project.
- Steps:
  Install project dependencies with npm install.
  Run tests using npm test.

Stage 3: Building image

- Objective: Builds a Docker image from the fetched code.
- Steps:
  Utilises the Docker plugin to build an image named after the registry path specified earlier.

Stage 4: Deploy Image

- Objective: Pushes the built Docker image to the Docker registry.
- Steps:
  Uses Docker plugin functionality to push the previously built image to a registry, assuming credentials are configured in Jenkins (referenced as 'DockerHubCred').

Stage 5: Ansible Deploying the Docker Image

- Objective: Deploys the Docker image using Ansible.

Steps:

- Initiates Ansible to execute the deployment.
- Configurations include:
- credentialsId: Refers to Jenkins credentials for accessing the deployment target.

- disableHostKeyChecking: Turns off host key checking for SSH connections.
- installation: Specifies the installed Ansible version.
- inventory: Points to the location of the Ansible inventory file.
- playbook: Specifies the playbook YAML file to execute.
- extraVars: Provides additional variables required by the Ansible playbook (such as connection URL and port).

Testing

Backend Tests:

1. *Backend Server Setup:*
   - We are using the supertest library to make HTTP requests to your backend server.
   - We import our Express app and a function (closeServer) from the index file.

2. *Login Tests:*
   - We have a test suite for testing wrong login passwords.
   - It sends a POST request to the /user/signin endpoint with a wrong password and expects a 400 status code and an 'Invalid credentials' message.

3. *User Registration Tests:*
   - There is a test suite for checking if a user already exists during registration.
   - It sends a POST request to the /user/signup endpoint with existing user details and expects a 400 status code and a 'User already exists' message.

4. *Password Mismatch Test:*
   - Another test suite checks if the passwords match during user registration.
   - It sends a POST request to the /user/signup endpoint with mismatched passwords and expects a 400 status code with a "Passwords don't match" message.

5. Blogs Endpoint Tests:
   - There are tests for checking the GET requests to retrieve blogs.
   - One test checks if the /blogs endpoint returns a 200 status code.

- Another test checks if the /blogs/:id endpoint returns a 200 status code.

Frontend Tests:

1. *React Component Rendering:*
   - We are using Jest and @testing-library/react for testing React components.
   - The test checks if the rendered component matches the snapshot.

2. *Mocking Dependencies:*
   - Jest is used to mock dependencies like react-redux and react-router-dom.
   - There is a mock for Redux useSelector and useDispatch.
   - Mocks are also provided for useHistory and useLocation from react-router-dom.
   - The test checks if the BlogSearch, Form and Blog component renders correctly and matches the snapshot.

Docker Containerisation

Docker Files :

We have created two separate GitHub repositories for the frontend and backend, each equipped with distinct Dockerfiles and Jenkins pipelines, resulting in the creation of two independent containers. The Dockerfile for the frontend and backend employs Node.js 18, establishes the working directory at /app, install dependencies, exposes port 3000, and initialises the application using "npm start" for frontend and exposes port 5000, executes tests with "npm test," and launches the backend server with "node index.js" for backend. Both Dockerfiles copy package.json and package-lock.json to the container and then replicate the entire current directory into the container at /app. The frontend container is tailored for a web application, exposing port 3000, while the backend container is configured for a Node.js server, exposing port 5000, running tests, and initiating the server with "node index.js."

Docker-compose files :

The docker-compose files defines a service named 'backend' using the image, exposing port 5000 and setting environment variables for connection URL and port for the backend and frontend file configures a 'frontend' service with the image, exposing port.

Continuous Monitoring (ELK)

Continuous Monitoring involves the ongoing observation and assessment of an application, system, or network to detect and address issues or vulnerabilities in real-time. It's a proactive approach to ensure the health, performance, security, and availability of software systems.

ELK, on the other hand, refers to a set of tools—Elasticsearch, Logstash, and Kibana—used together for log management and analytics:

- Elasticsearch: A distributed, RESTful search and analytics engine used for storing and indexing data. It allows for fast searching, analysing, and visualising of data in real-time.

- Logstash: A data processing pipeline that ingests, transforms, and sends data to Elasticsearch. It can collect logs from various sources, parse them, and send them to Elasticsearch for storage and analysis.

- Kibana: A data visualisation and exploration tool used to visualise data stored in Elasticsearch. It offers a user-friendly interface to create visualisations, dashboards, and perform real-time analysis.

ELK is commonly used for log aggregation, where logs from multiple sources are collected, indexed, and analysed centrally. Continuous Monitoring often involves using tools like ELK to monitor system logs, application logs, and various metrics in real-time. These tools enable teams to track system performance, detect anomalies, troubleshoot issues, and gain insights into system behaviour.

Logging :

For the logging, we are using the Winston logging library to set up logging for our backend application, and we have configured multiple transports for different log levels. The logs are being stored in separate files based on the log level. Below is an explanation of how your logging is set up:

1. *Imports:*
   - We are using the `createLogger` function from Winston along with `transports` and `format` modules.

2. *Logger Configuration:*
   - The `createLogger` function is used to create a logger instance.
   - The logger has multiple transports, each responsible for logging messages of a specific severity level.

3. *File Transports:*
   - For each log level (`info`, `error`, `warn`), there is a separate file transport configured.
   - Each file transport specifies the filename where logs of that level should be stored.
   - The `format` module is used to combine the timestamp and JSON formatting for each log entry.

4. *Combined Transport:*
   - There is an additional transport named `backend_combined.log` that doesn't specify a log level. This transport is likely meant to capture logs of all levels.
   - It also includes a timestamp and JSON formatting.

5. *Exporting the Logger:*
   - Finally, the configured logger is exported so that it can be used throughout your application.

API Documentation

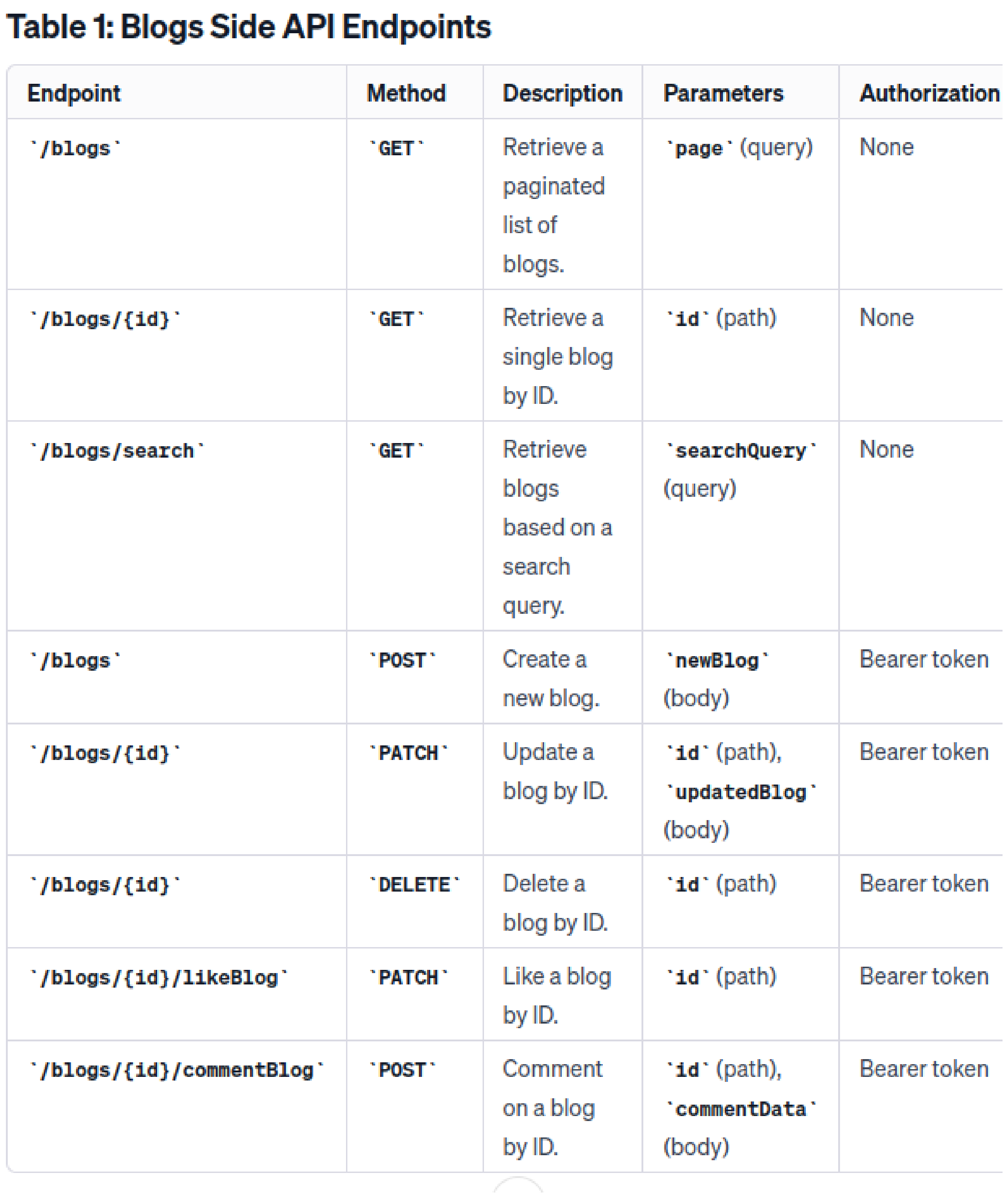


Table 1: Blogs Side API Endpoints

Endpoint	Method	Description	Parameters	Authorization
`` `/blogs` ``	`` `GET` ``	Retrieve a paginated list of blogs.	`` `page` `` (query)	None
`` `/blogs/{id}` ``	`` `GET` ``	Retrieve a single blog by ID.	`` `id` `` (path)	None
`` `/blogs/search` ``	`` `GET` ``	Retrieve blogs based on a search query.	`` `searchQuery` `` (query)	None
`` `/blogs` ``	`` `POST` ``	Create a new blog.	`` `newBlog` `` (body)	Bearer token
`` `/blogs/{id}` ``	`` `PATCH` ``	Update a blog by ID.	`` `id` `` (path), `` `updatedBlog` `` (body)	Bearer token
`` `/blogs/{id}` ``	`` `DELETE` ``	Delete a blog by ID.	`` `id` `` (path)	Bearer token
`` `/blogs/{id}/likeBlog` ``	`` `PATCH` ``	Like a blog by ID.	`` `id` `` (path)	Bearer token
`` `/blogs/{id}/commentBlog` ``	`` `POST` ``	Comment on a blog by ID.	`` `id` `` (path), `` `commentData` `` (body)	Bearer token

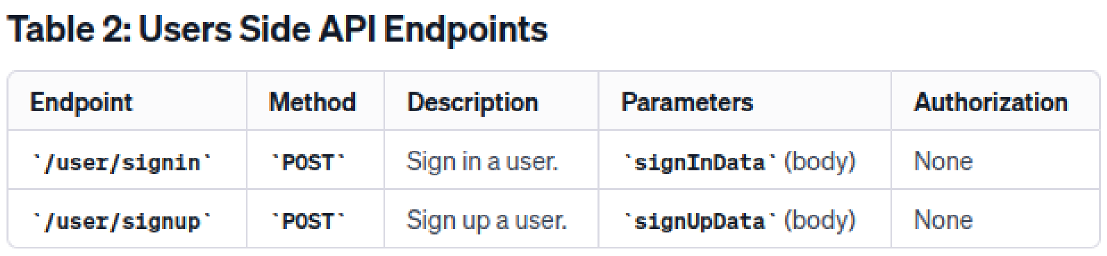


Table 2: Users Side API Endpoints

Endpoint	Method	Description	Parameters	Authorization
`` `/user/signin` ``	`` `POST` ``	Sign in a user.	`` `signInData` `` (body)	None
`` `/user/signup` ``	`` `POST` ``	Sign up a user.	`` `signUpData` `` (body)	None

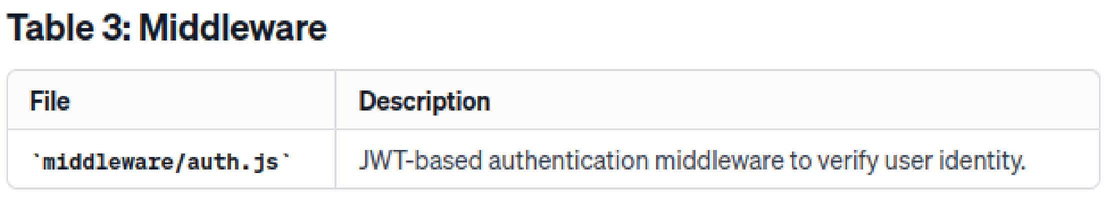


Table 3: Middleware

File	Description
`` `middleware/auth.js` ``	JWT-based authentication middleware to verify user identity.

Results

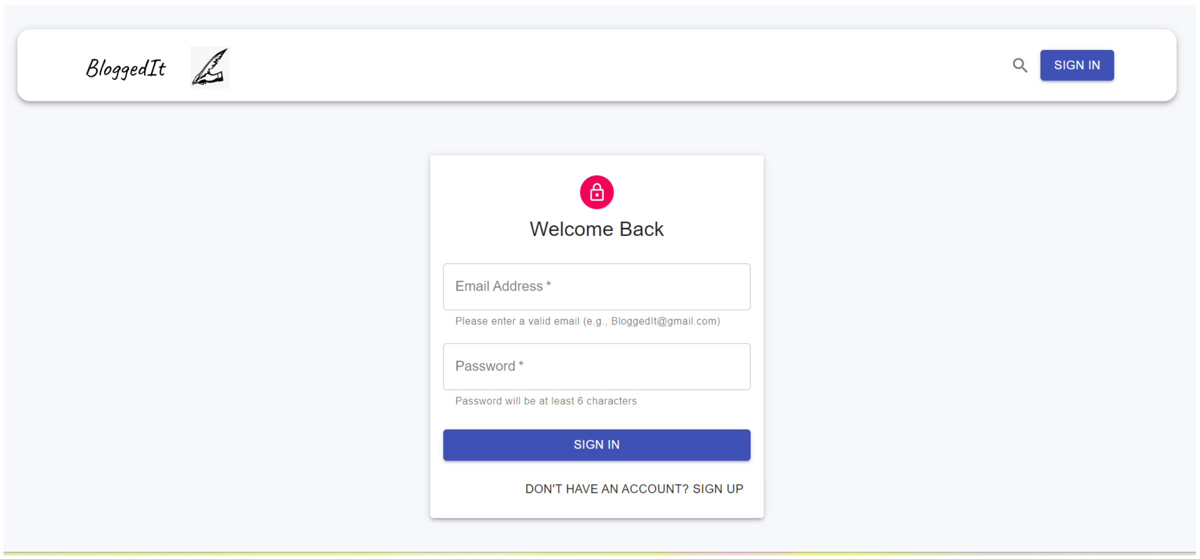

BloggedIt
SIGN IN
Welcome Back
Email Address *
Please enter a valid email (e.g., BloggedIt@gmail.com)
Password *
Password will be at least 6 characters
SIGN IN
DON'T HAVE AN ACCOUNT? SIGN UP

Fig 1: Sign In Page

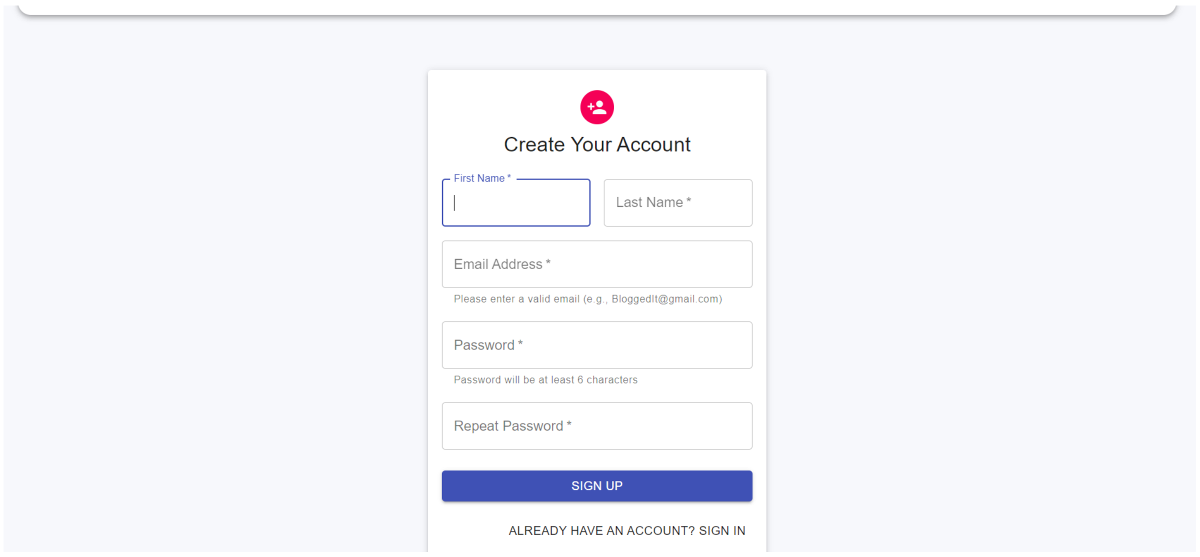
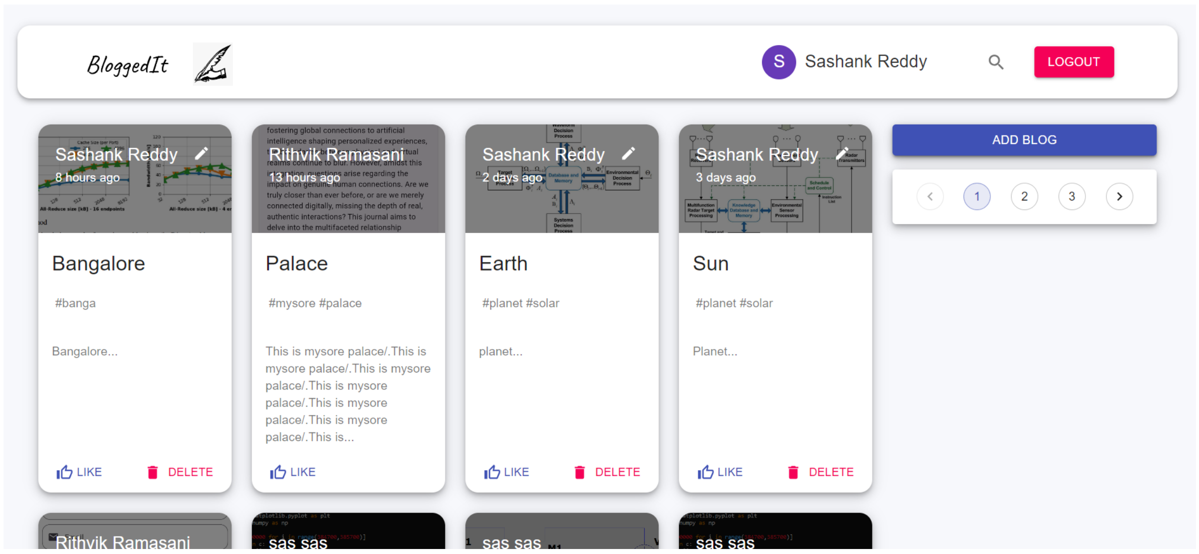

Create Your Account
First Name *
Last Name *
Email Address *
Please enter a valid email (e.g., BloggedIt@gmail.com)
Password *
Password will be at least 6 characters
Repeat Password *
SIGN UP
ALREADY HAVE AN ACCOUNT? SIGN IN

Fig 2: Sign Up Page



BloggedIt

S Sashank Reddy

LOGOUT

ADD BLOG

1 2 3

Sashank Reddy
8 hours ago
Bangalore
#banga
Bangalore...
LIKE DELETE

Rithvik Ramasani
13 hours ago
Palace
#mysore #palace
This is mysore palace/.This is mysore palace/.This is mysore palace/.This is mysore palace/.This is mysore palace/.This is...
LIKE

Sashank Reddy
2 days ago
Earth
#planet #solar
planet...
LIKE DELETE

Sashank Reddy
3 days ago
Sun
#planet #solar
Planet...
LIKE DELETE

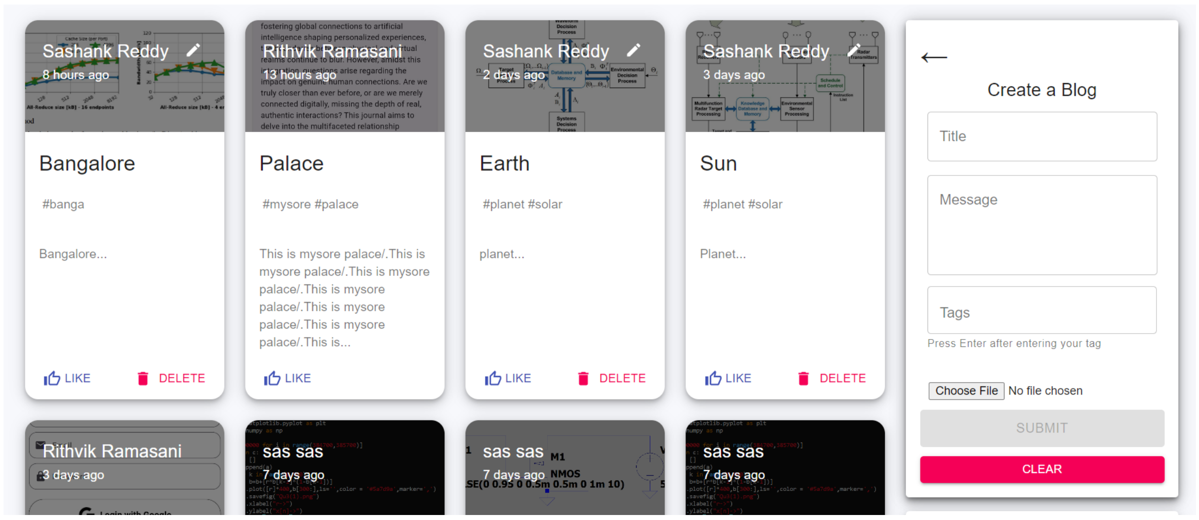
Rithvik Ramasani

sas sas

sas sas

sas sas

Fig 3 : Home Page



Sashank Reddy
8 hours ago
Bangalore
#banga
Bangalore...
LIKE DELETE

Rithvik Ramasani
13 hours ago
Palace
#mysore #palace
This is mysore palace/.This is mysore palace/.This is mysore palace/.This is mysore palace/.This is mysore palace/.This is...
LIKE

Sashank Reddy
2 days ago
Earth
#planet #solar
planet...
LIKE DELETE

Sashank Reddy
3 days ago
Sun
#planet #solar
Planet...
LIKE DELETE

Rithvik Ramasani
3 days ago

sas sas
7 days ago

sas sas
7 days ago

sas sas
7 days ago

Create a Blog

Title

Message

Tags

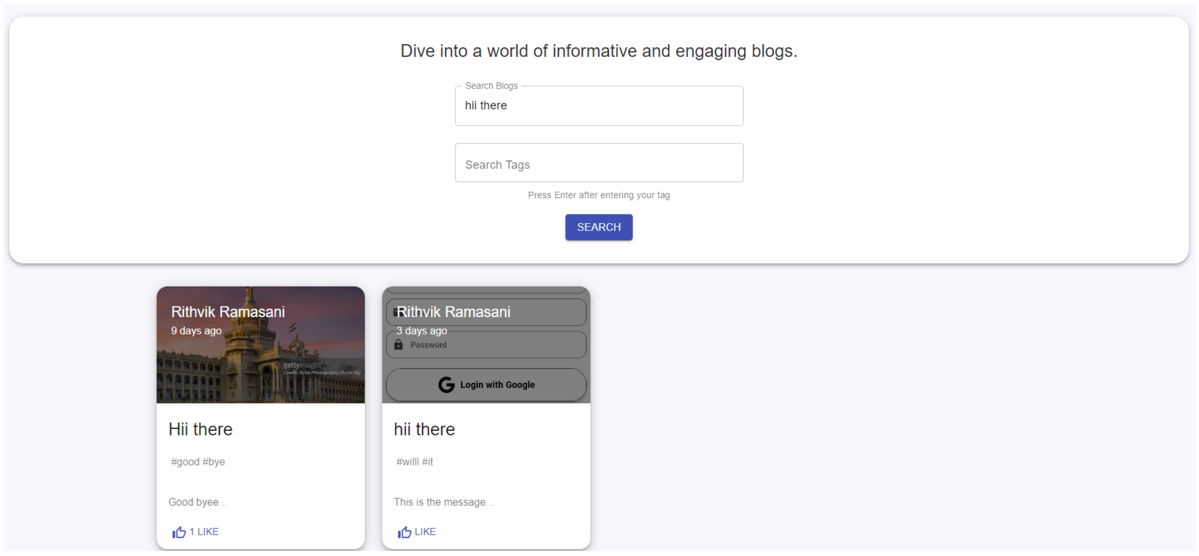
Press Enter after entering your tag

Choose File No file chosen

SUBMIT

CLEAR

Fig 4: Box for creating a Blog



Dive into a world of informative and engaging blogs.

Search Blogs

hii there

Search Tags

Press Enter after entering your tag

SEARCH

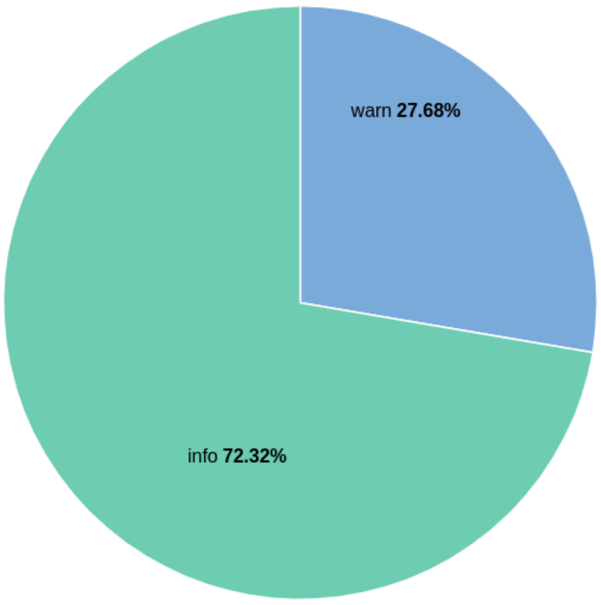
Rithvik Ramasani
9 days ago
Hii there
#good #bye
Good byee...
1 LIKE

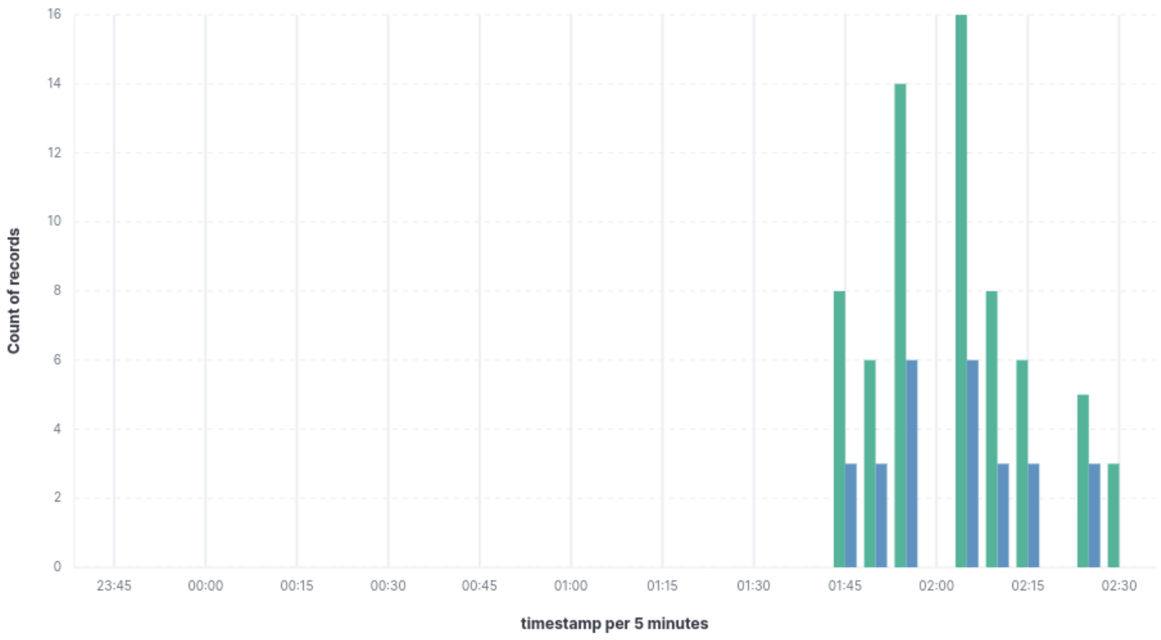
Rithvik Ramasani
3 days ago
hii there
#willl #it
This is the message...
LIKE

Fig 5 : Page for searching

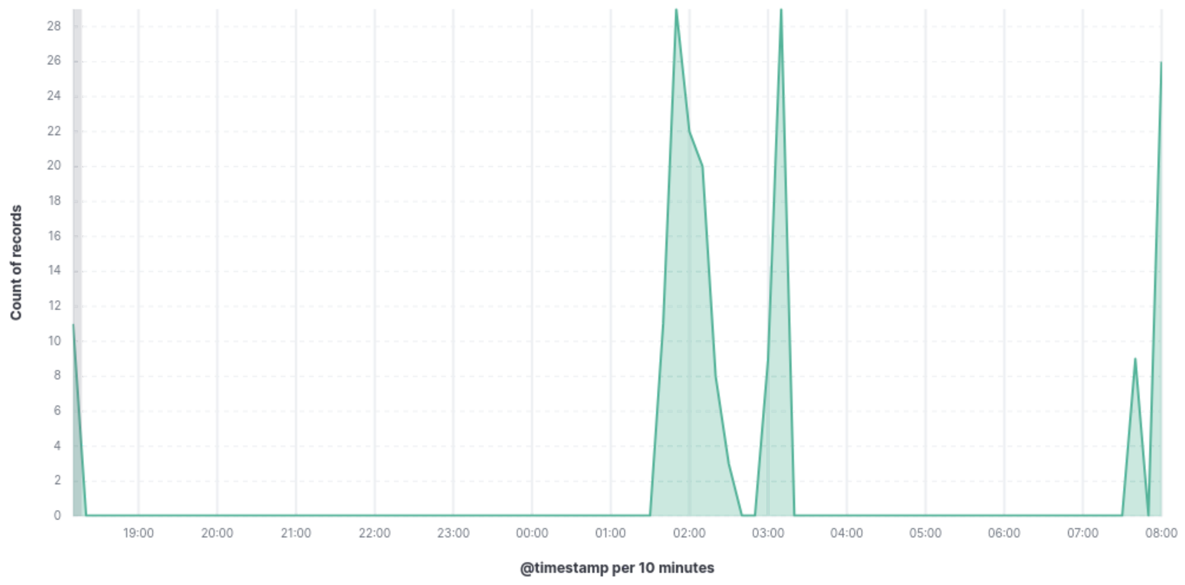
Kibana Visualisations

Combined Logs

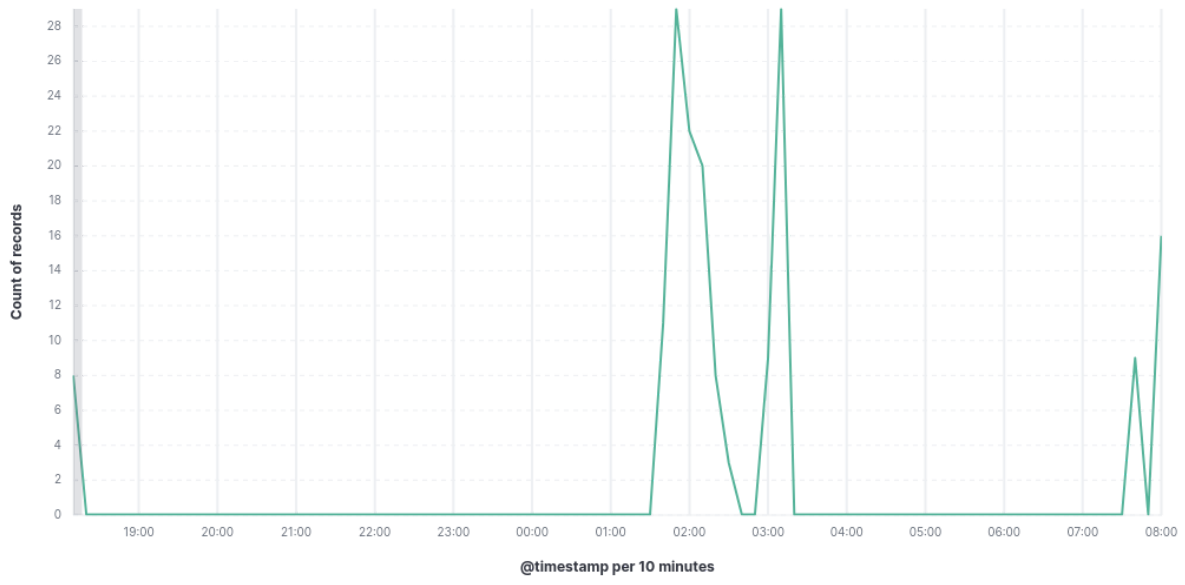




Info Level



Warn Level


Count of records
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
@timestamp per 10 minutes