Image Sharing Cloud Application

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By
Ritik Bhardwaj (181262)

UNDER THE SUPERVISION OF
Dr. Rajni Mohana

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

DECLARATION

I hereby declare that this project has been done by me under the supervision of Dr. Rajni Mohana**,** **Affiliation,** Jaypee University of Information Technology. I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:
Dr. Rajni Mohana
Associate Professor
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology

Submitted by:
Ritik Bhardwaj(181262)
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the project report titled **“Image Sharing Cloud Application”** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by Ritik Bhardwaj (181262) during the period from Jan 2022 to Apr 2022 under the supervision of **Dr. Rajni Mohana,** Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Ritik Bhardwaj (181262)
The above statement made is correct to the best of my knowledge.

Dr. Rajni Mohana
Associate Professor
Computer Science & Engineering
Jaypee University of Information Technology, Waknaghat

ACKNOWLEDGEMENT

Foremost, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I am grateful and wish my profound indebtedness to Supervisor **Dr. Rajni Mohana**, Associate Professor. Department of CSE Jaypee University of Information Technology, Waknaghat for blessing me with her guidance and utter support throughout the project. Her vast knowledge & keen interest in the field of "**Cloud Computing**" has really guided us to understand each and every topic. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to Dr. Rajni Mohana, Department of CSE, for her kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Ritik Bhardwaj (181262)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Freeb is an image sharing application and social network platform wherein people who love photography or are an admirer of good pictures can share their images as well as view them for business or pleasure.

It allows users to upload photos, like other's images as well as share their opinion of them.
This application is specifically built to provide exposure to new artists or students or any other person who is unable to find the correct platform to showcase their talent and therefore is built in such a way that it can be experienced by anyone and everyone free of cost.

In today's world where social media is everything this application is perfect for users to enhance their reach and increase their popularity amongst youth or adults.
Not only will the user be able to build huge engagement but will also be able to get good marketing opportunities which will provide the perfect exposure needed by anyone for further job prospects.

CHAPTER 1 : INTRODUCTION

1.1 Introduction

Freeb is an image sharing cloud application. Users can create their accounts and upload images. The user data is stored in a MySql database. The images are stored in AWS S3 object storage for scalability and fast access. The user session is stored safely in AWS DynamoDB.

1.2 Objectives

To create a fast, scalable and secure image sharing cloud application by using industrial best practices.

1.3 Motivation

To apply industrial best practices and create a fast, scalable and secure cloud application to share images.

1.4 Libraries/Frameworks Used

React.js	-	A fast front end library.
Node.js	-	JavaScript Runtime environment.
Express.js	-	A fast and low overhead backend framework.
AWS SDK for Node.js	-	AWS SDK for integrating AWS services with node.js backends.

AWS provides a host of services such as compute instances, databases; sql and no sql. Analytics and a clean dashboard to manage user access and assign roles for better management of resources.

1.5 Technical Requirements

- **AWS EC2** compute instance for running microservices.
- **AWS S3** object storage for storing images
- **AWS DynamoDB** storage for storing user session data.
- **Nginx** server to act as a reverse proxy and also host static react app.
- **Docker** process on the machine to run node microservices on different linux containers for isolation.
- **Mysql** database also running in a docker container to store the user data

1.5.1 Hardware Configuration

Table 1 : Hardware Configuration

Processor	Apple M1 chip, 8-core CPU
RAM	8 GB
Hard Disk	256 GB SSD
Monitor	13”
Mouse	
Keyboard	

1.5.2 Software Configuration

Table 2 : Software Configuration

Operating System	Windows 10, Mac OS
Language	Javascript
Runtime environment	Node.js
Package Manager	Npm

CHAPTER 2 : LITERATURE SURVEY

1) **The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information**

   S. Sivakorn, I. Polakis and A. D. Keromytis, "The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information," *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 724-742, doi: 10.1109/SP.2016.49.

2) **Everything You Ever Wanted to Know About Session Management in Node.js**

   But then there is always the risk of sensitive information leakage and can even end up in the wrong hands.. The majority of users aren't necessarily bad actors, but there might be some applicants looking for ways to exploit an application. Therefore, most security concerns should be addressed as soon as possible by the developer and maintainer of the application and the developer of whatever session management library one is using.

CHAPTER 3 : SDLC

3.1 Feasibility Study

We have evaluated the feasibility of the system into the
following categories:

- Technical feasibility
- Operational feasibility
- Economic feasibility
- Schedule feasibility

3.1.1 Technical Feasibility

The evaluation of technical feasibility is a tricky but an important part because, at this point in time there isn't any detailed design of the system, making it difficult to assess issues like performance, costs, deployment etc. It is very important to make sure that the project is feasible from a technical standpoint. The project must be practical and possible with the technologies we have.

3.1.2 Operational Feasibility

Proposed project is useful as long as it can be turned into information systems that will meet the requirements of the operation. Once the project is deployed, it must work as planned. It should be easy to operate by a team of technical professionals.

3.1.3 Economic Feasibility

It attempts to weigh the costs of developing and implementing a new system, against the benefits that would increase over a period of time from having the new system in place. Since the project would be deployed on the cloud on AWS. We know that AWS charges for the type of service and how the service is in use. Therefore it is very crucial that the operational cost is not off the charts and manageable.

3.1.4 Schedule Feasibility

A project is basically of no use if it takes too long to be completed before it becomes useful. Typically, this is about estimating how long the system will take to develop, and whether or not it can be completed in a given period of time using some methods such as payback period. Schedule feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the deadlines given for the project reasonable? Some projects are initiated with specific deadlines. It is necessary to figure out whether the deadlines are mandatory or desirable. A minor deviation can be encountered in the original schedule decided at the commencement of the project. The application development is possible in terms of schedule.

3.2 Requirement Definition

After the extensive analysis of the issues faced within the system, we are familiarized with the needs and requirements of the current system. The requirement that the system needs is categorized into the functional and non-functional requirements. These requirements are listed below:

3.2.1 Functional Requirements

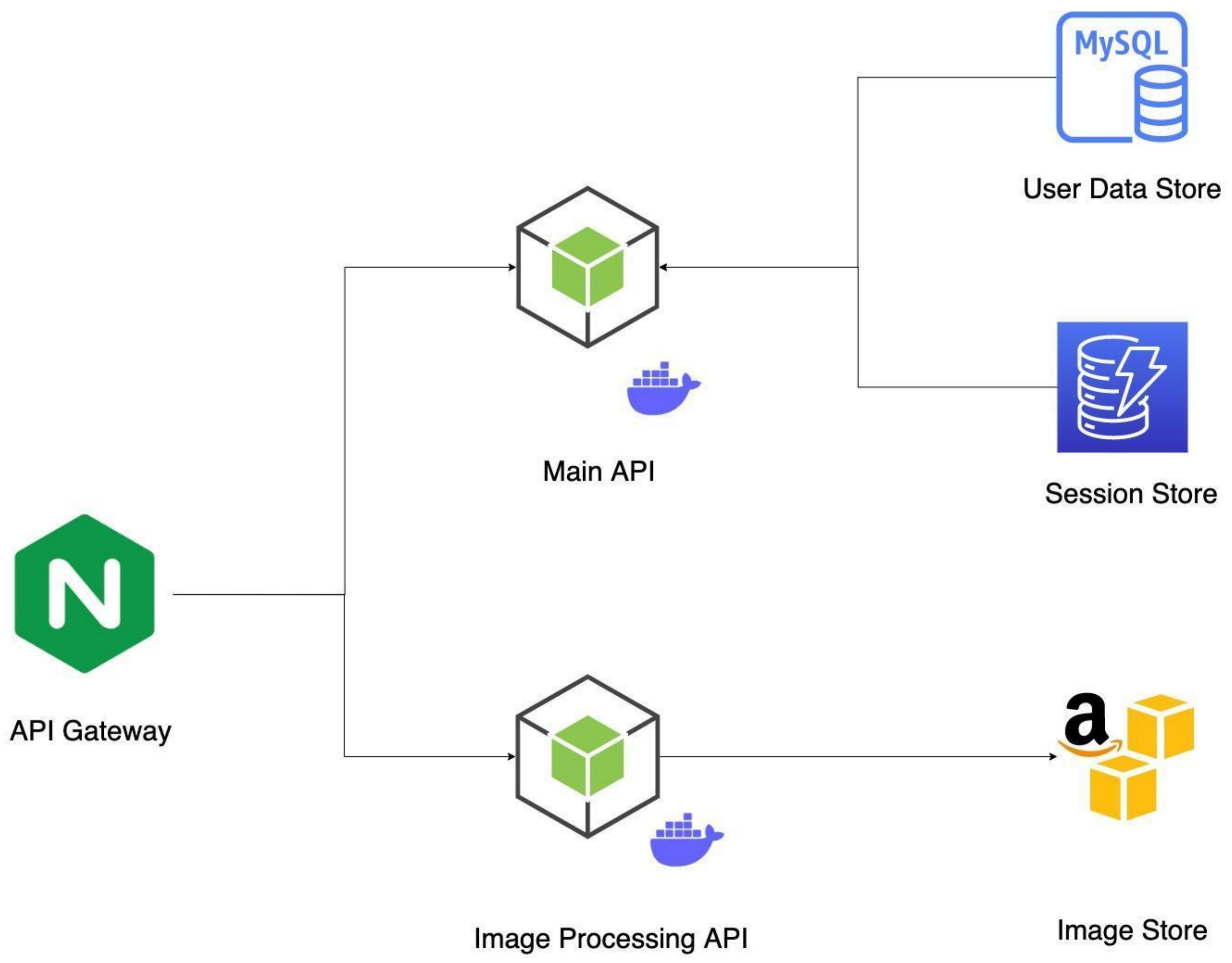
The system should be able to :

- Store user data and retrieve it on demand
- The storage of images should be seamless and fast.
- The session storage must be implemented without any security concerns

3.2.2 Non-Functional Requirements

The constraints that define how the system should do what it is supposed to do.

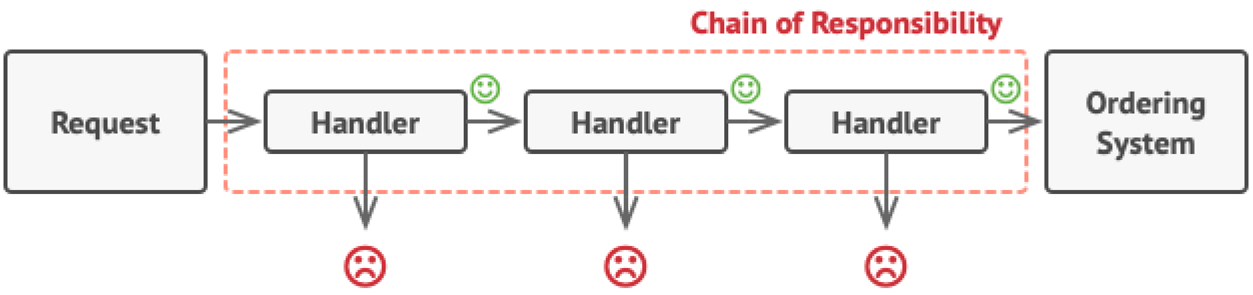
3.3 System Design Diagram



3.3 Backend design patterns

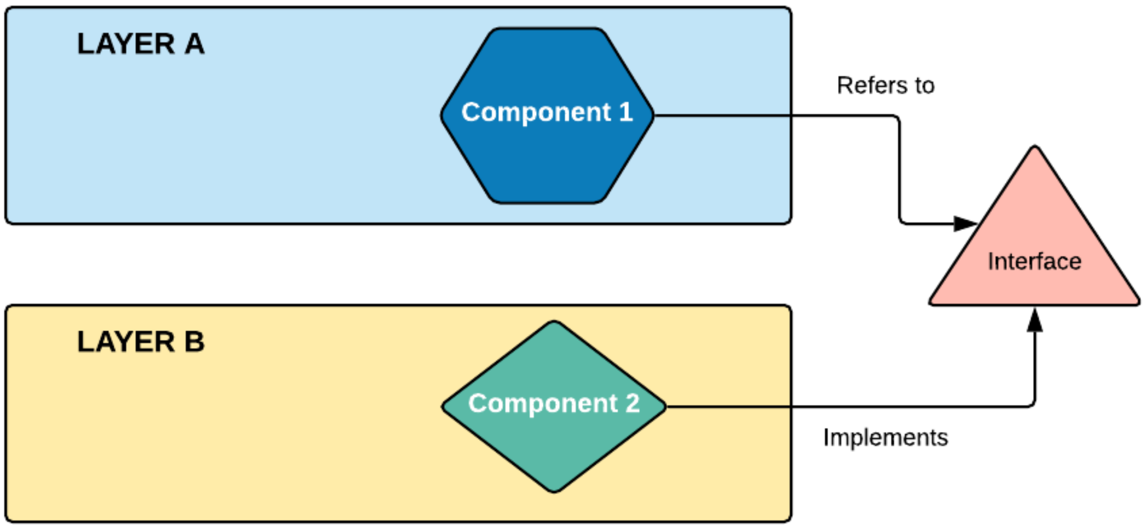
1) Chain of responsibility

The chain of responsibility is a behavioral design pattern that allows you to pass the request along a chain of handlers. Upon receiving a request, each handler decides either to process it or to pass to the next handler.



2) Dependency Injection

It is a programming design pattern that makes a class independent of its dependencies. It achieves that by decoupling the object creation from its usage. This helps to achieve the SOLID's dependency inversion principle.



CHAPTER 4 : IMPLEMENTATION

4.1 Identification of features

The cloud application features:

- Creation of user account ie Signup
- Deletion of user account ie Remove account
- User login with persistence ie creation of session
- User logout; deletion of session data
- Users can post images. Images stored in S3 storage
- Users can delete images

4.2 React.js - Front end library for performance

1. **Component Based** - Every piece of UI inside react is known as a component. You can compose components to create a complex UI. The behavior of the components is predictable and modifiable by using state and props. The state object stores the current state of the object and can be changed on the fly on any condition.

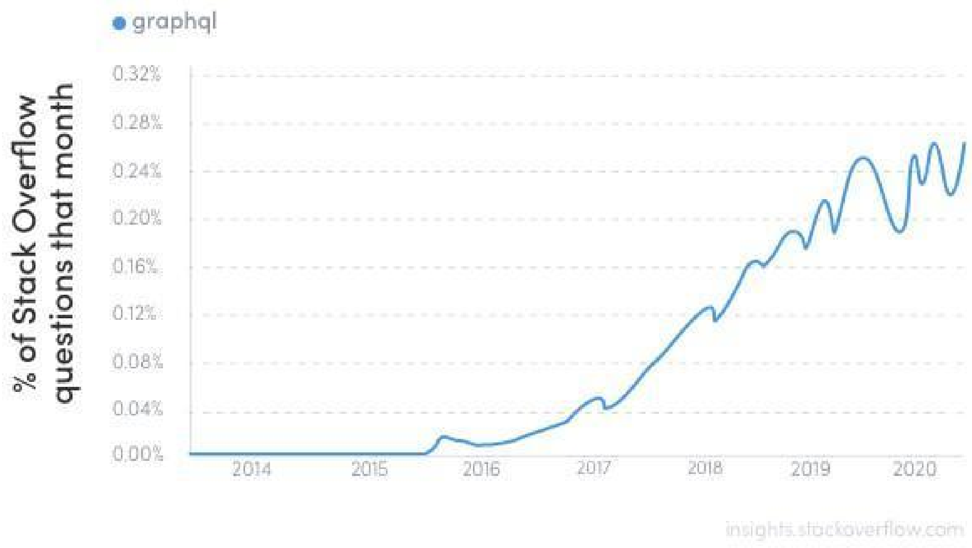
2. **State and Props** - State object allows us to set the current state of the component. Stuff can be dynamically rendered based on the state variables. In order to compose the object and control it on the fly we use props. Props allow us to pass data to components and compose the UI in a top down fashion.

3. **Virtual DOM** - Virtual DOM is a peculiar feature of react.js and it makes UI updates much faster. The virtual DOM is an alternate representation of the actual DOM in the memory and keeps track of changes in the UI. It compares those changes with the original DOM and updates only those things that are changed. This avoids updating the whole DOM which takes time. This increases the speed of UI changes.

4.3 Node.js - Back-end technology for performance and security

Node.js is an asynchronous event-driven Javascript runtime environment built on top of Google Chrome's V8 engine. Node.js is designed to build scalable network applications. Node.js uses an event loop and runs on a single thread unlike other server softwares that use concurrency models in which OS threads are employed.

No function in Node.js directly performs I/O, so the process never blocks except when the I/O is performed using synchronous methods of Node.js standard library.



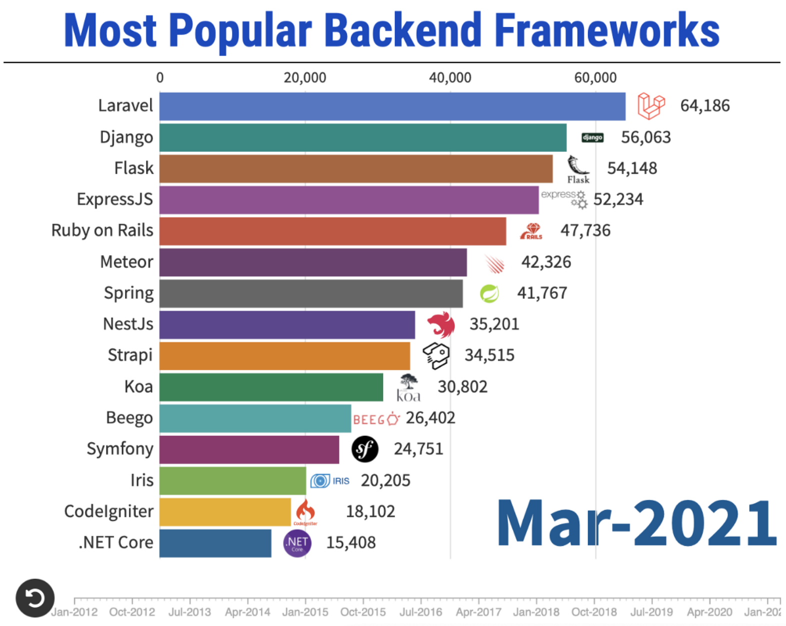
Node.js has become very popular over the last 4 year making it the best choice for startups. It comes with a very comprehensive documentation. It’s very easy to get started with and many people who have been working with legacy technology are making a shift to Node.js.

4.4 Express.js - Back-end framework for Node.js

Express

Express.js is a free and open-source web application framework for Node.js. Features of express.js are:

1. Faster server side development.
2. Middleware
3. Routing
4. Templating
5. Debugging



Express.js is one of the most popular backend frameworks in the world

4.4.1 Security considerations with Express.js

Some security best practices to use with express.js

1. **Secure the connection and data** - We can make use of Helmet - a Node.js package. It is a collection of 13 middleware functions to set up appropriate response headers for security.

2. **Protect your cookies** - The best practice here is to not use the in-built storage and use some sort of session stored in a database because it's very easy to get the cookie from the browser and tamper with it. Any malicious script run on the browser can access the cookies and tamper with. Also cookies can only be used to store little information because it comes with an overhead

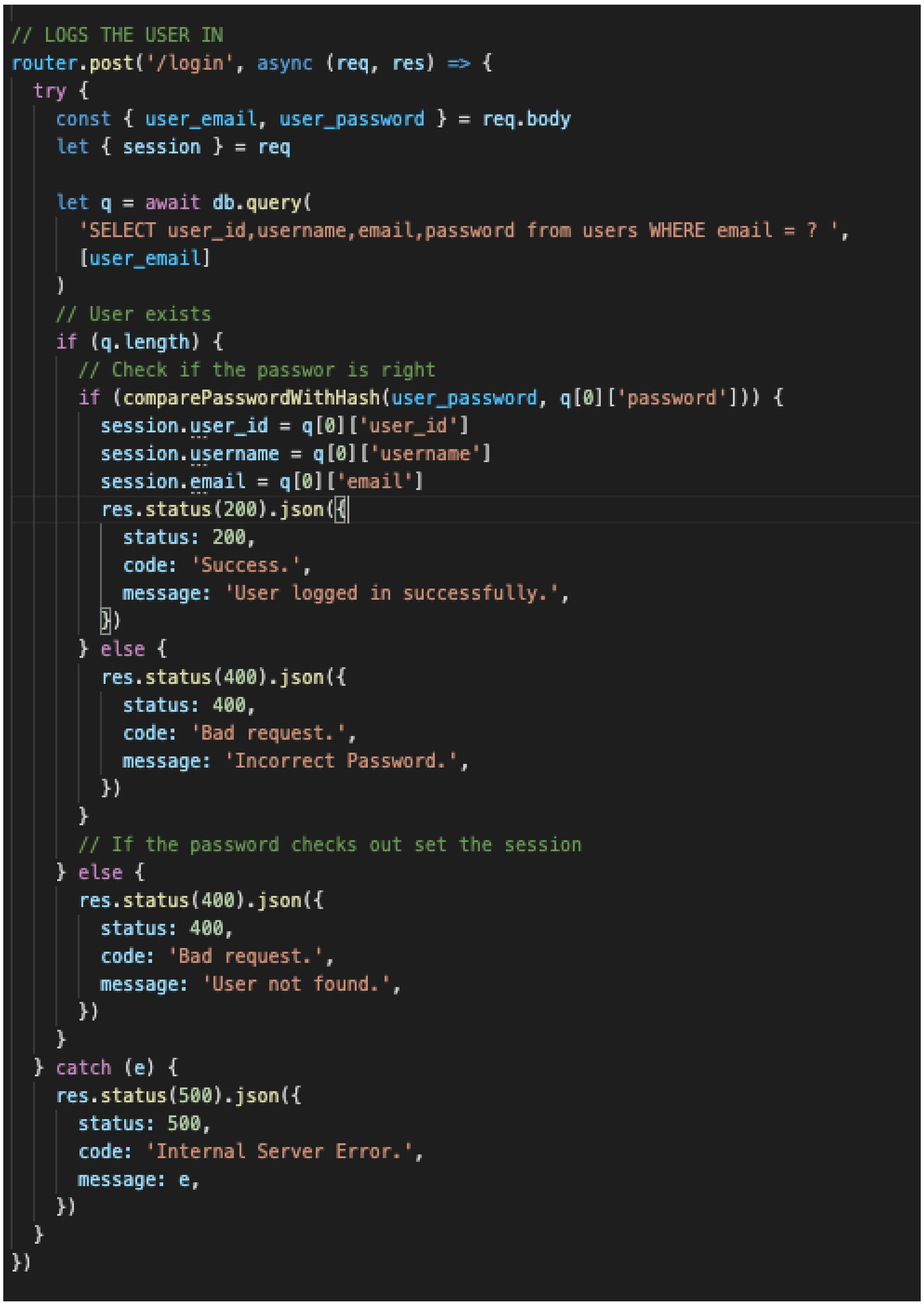
3. **Secure your dependencies** - NPM is a very versatile package manager but it's always a good habit to make sure that your project dependencies are free from any vulnerabilities. We can use NPM in built features to check for it. Use 'npm audit' to check the severity of vulnerability of each and every package and its dependencies.

4. **Validate the input of the users** - It is very important to validate the input of the users. In compatibility between the provided input type and the expected input type can cause errors at the back-end and cause it to crash. It can be avoided by placing the input procession logic inside a try-catch block. But it's a very crucial step to validate and possibly sanitize user input before calling other services such as the database.

4.5 Code snippets for back-end modules

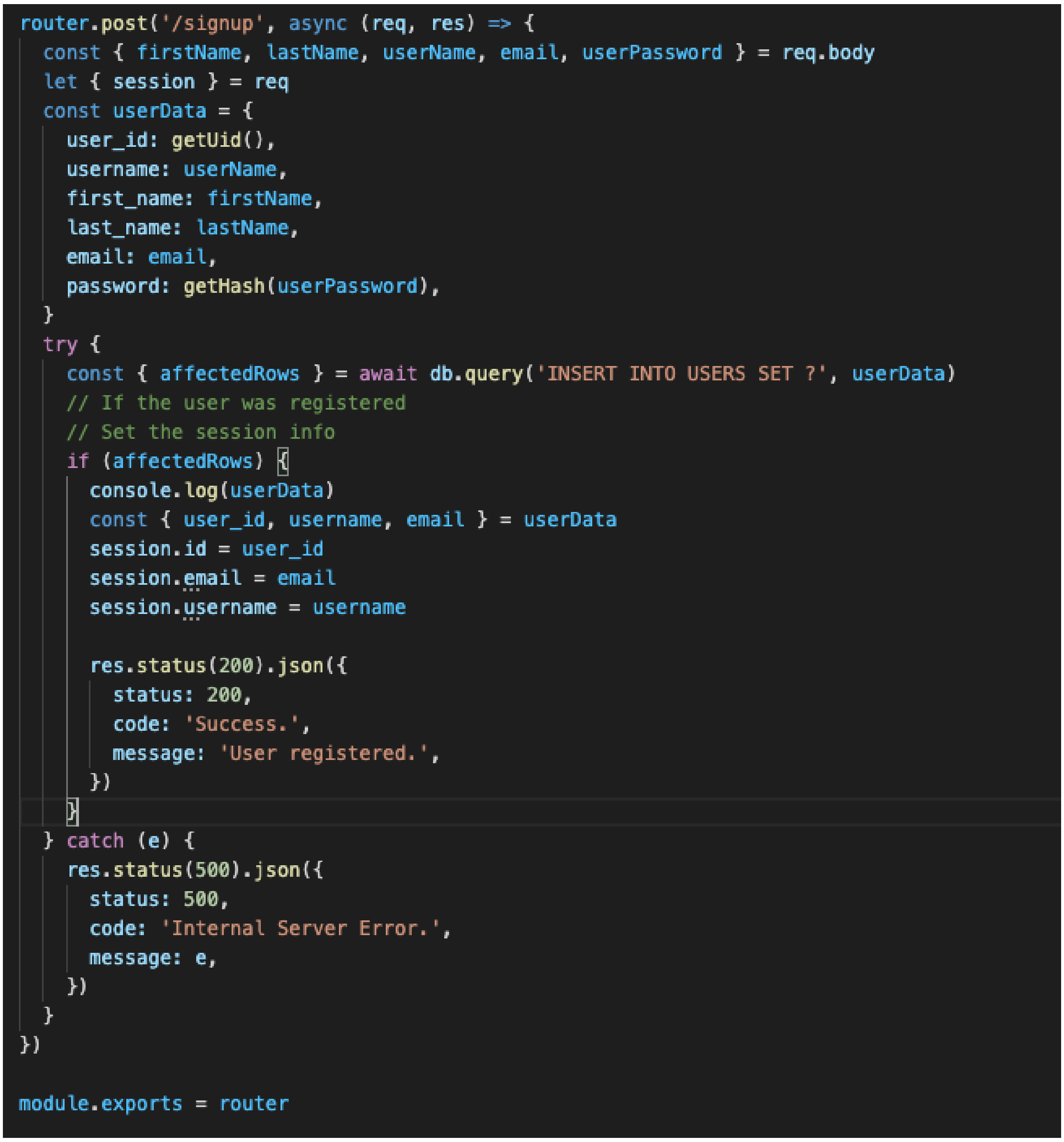
The backend code is divided into multiple files for the sake of modularity. Routes are clubbed into categories and kept in separate files. Express provides an interface to separate routes and make them a part of the middleware chain. **Express Router** is used to achieve that.

1. **LoginRoutes.js** - Handle the user login - POST : http://localhost:8000/api/login



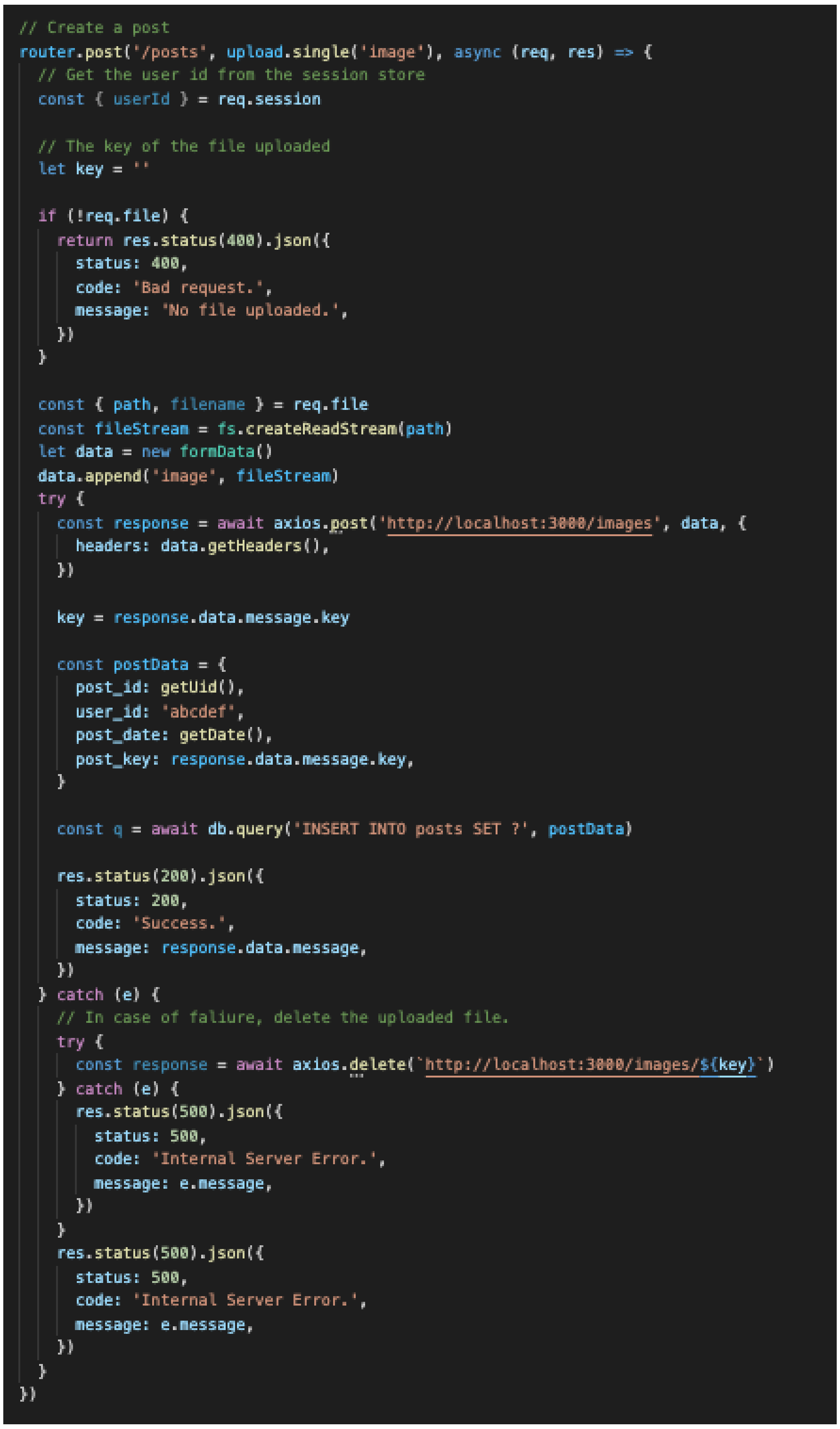
```
// LOGS THE USER IN
router.post('/login', async (req, res) => {
  try {
    const { user_email, user_password } = req.body
    let { session } = req

    let q = await db.query(
      'SELECT user_id,username,email,password from users WHERE email = ? ',
      [user_email]
    )
    // User exists
    if (q.length) {
      // Check if the passwor is right
      if (comparePasswordWithHash(user_password, q[0]['password'])) {
        session.user_id = q[0]['user_id']
        session.username = q[0]['username']
        session.email = q[0]['email']
        res.status(200).json({
          status: 200,
          code: 'Success.',
          message: 'User logged in successfully.',
        })
      } else {
        res.status(400).json({
          status: 400,
          code: 'Bad request.',
          message: 'Incorrect Password.',
        })
      }
      // If the password checks out set the session
    } else {
      res.status(400).json({
        status: 400,
        code: 'Bad request.',
        message: 'User not found.',
      })
    }
  } catch (e) {
    res.status(500).json({
      status: 500,
      code: 'Internal Server Error.',
      message: e,
    })
  }
})
```

2. **SignupRoutes.js** - Handle the user signup - POST : http://localhost:8000/api/signup



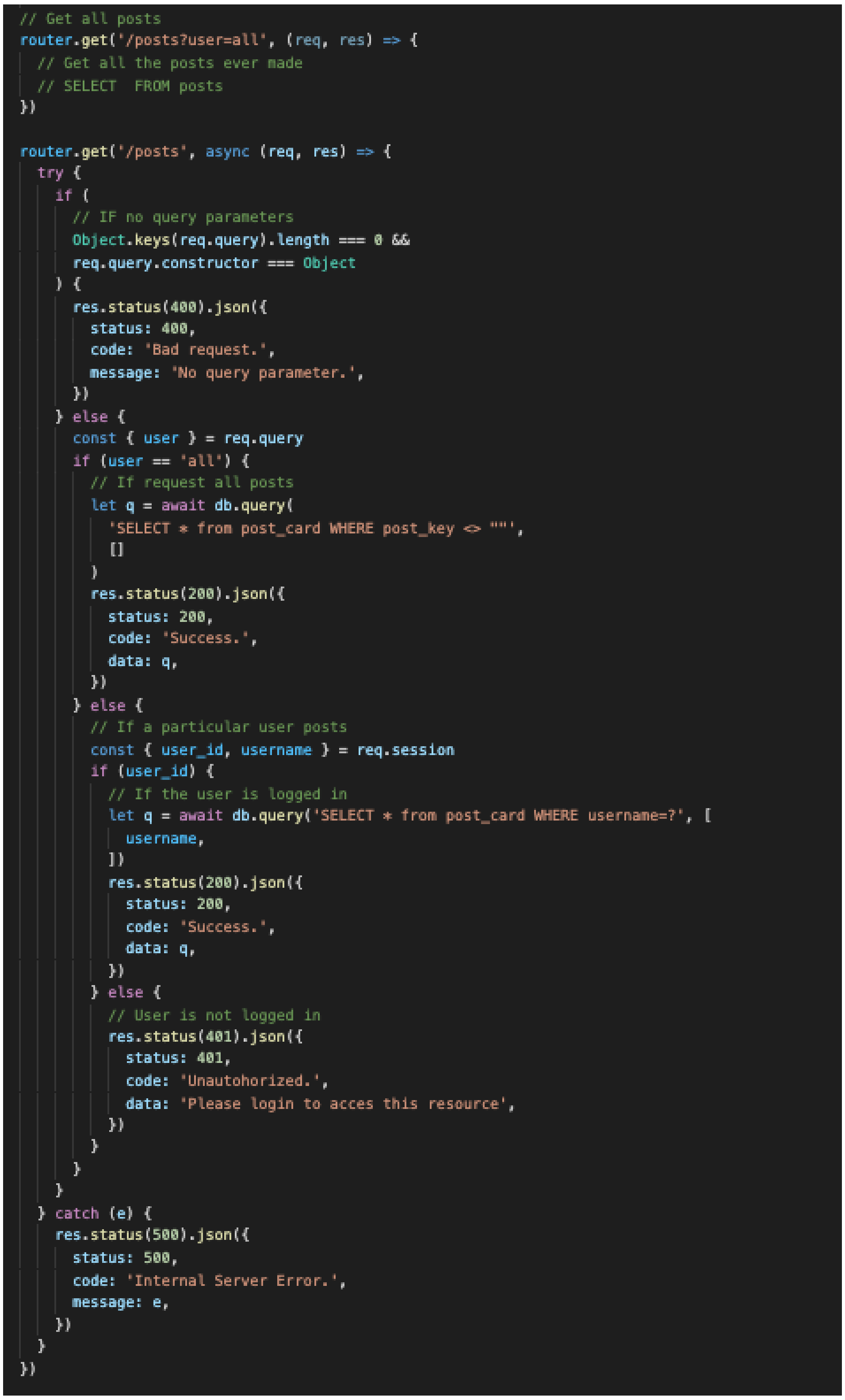
```
router.post('/signup', async (req, res) => {
  const { firstName, lastName, userName, email, userPassword } = req.body
  let { session } = req
  const userData = {
    user_id: getUid(),
    username: userName,
    first_name: firstName,
    last_name: lastName,
    email: email,
    password: getHash(userPassword),
  }
  try {
    const { affectedRows } = await db.query('INSERT INTO USERS SET ?', userData)
    // If the user was registered
    // Set the session info
    if (affectedRows) {
      console.log(userData)
      const { user_id, username, email } = userData
      session.id = user_id
      session.email = email
      session.username = username

      res.status(200).json({
        status: 200,
        code: 'Success.',
        message: 'User registered.',
      })
    }
  } catch (e) {
    res.status(500).json({
      status: 500,
      code: 'Internal Server Error.',
      message: e,
    })
  }
})

module.exports = router
```

3. **PostRoutes.js** - Create posts - POST : http://localhost:8000/api/posts



```
// Create a post
router.post('/posts', upload.single('image'), async (req, res) => {
  // Get the user id from the session store
  const { userId } = req.session

  // The key of the file uploaded
  let key = ''

  if (!req.file) {
    return res.status(400).json({
      status: 400,
      code: 'Bad request.',
      message: 'No file uploaded.',
    })
  }

  const { path, filename } = req.file
  const fileStream = fs.createReadStream(path)
  let data = new formData()
  data.append('image', fileStream)
  try {
    const response = await axios.post('http://localhost:3000/images', data, {
      headers: data.getHeaders(),
    })

    key = response.data.message.key

    const postData = {
      post_id: getUid(),
      user_id: 'abcdef',
      post_date: getDate(),
      post_key: response.data.message.key,
    }

    const q = await db.query('INSERT INTO posts SET ?', postData)

    res.status(200).json({
      status: 200,
      code: 'Success.',
      message: response.data.message,
    })
  } catch (e) {
    // In case of faliure, delete the uploaded file.
    try {
      const response = await axios.delete(`http://localhost:3000/images/${key}`)
    } catch (e) {
      res.status(500).json({
        status: 500,
        code: 'Internal Server Error.',
        message: e.message,
      })
    }
    res.status(500).json({
      status: 500,
      code: 'Internal Server Error.',
      message: e.message,
    })
  }
})
```

4. **PostRoutes.js** - Get posts - GET : http://localhost:8000/api/posts?user=username

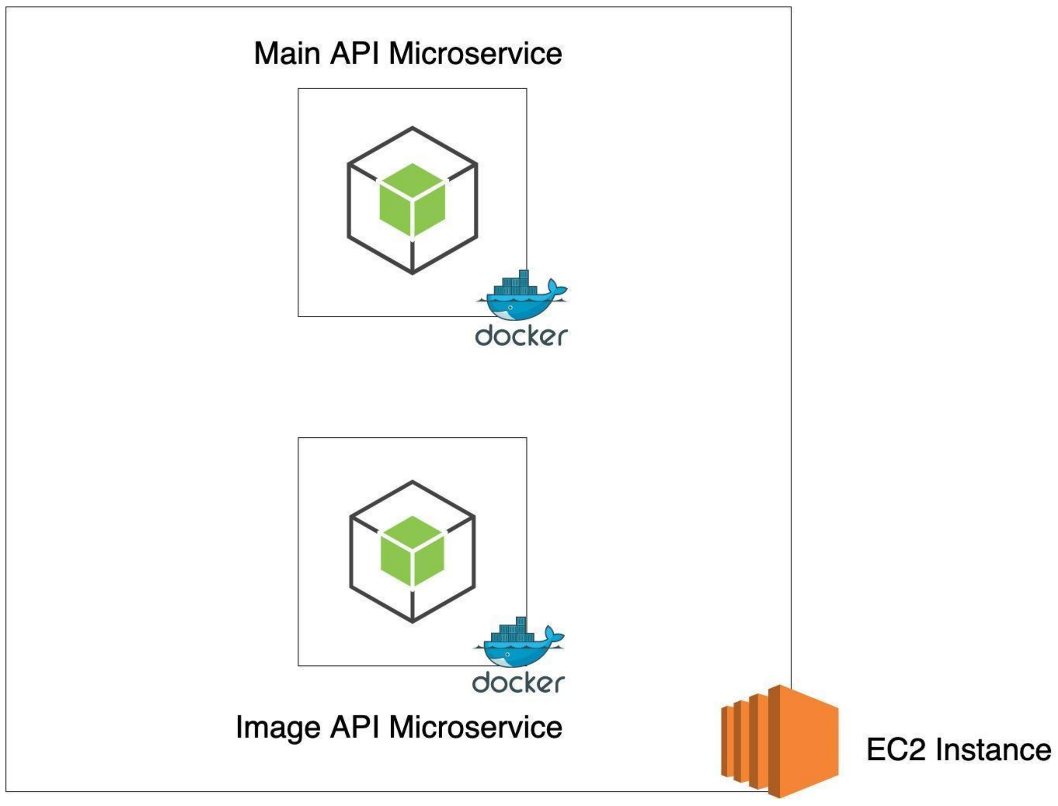


```
// Get all posts
router.get('/posts?user=all', (req, res) => {
  // Get all the posts ever made
  // SELECT  FROM posts
})

router.get('/posts', async (req, res) => {
  try {
    if (
      // IF no query parameters
      Object.keys(req.query).length === 0 &&
      req.query.constructor === Object
    ) {
      res.status(400).json({
        status: 400,
        code: 'Bad request.',
        message: 'No query parameter.',
      })
    } else {
      const { user } = req.query
      if (user == 'all') {
        // If request all posts
        let q = await db.query(
          'SELECT * from post_card WHERE post_key <> ""',
          []
        )
        res.status(200).json({
          status: 200,
          code: 'Success.',
          data: q,
        })
      } else {
        // If a particular user posts
        const { user_id, username } = req.session
        if (user_id) {
          // If the user is logged in
          let q = await db.query('SELECT * from post_card WHERE username=?', [
            username,
          ])
          res.status(200).json({
            status: 200,
            code: 'Success.',
            data: q,
          })
        } else {
          // User is not logged in
          res.status(401).json({
            status: 401,
            code: 'Unautohorized.',
            data: 'Please login to acces this resource',
          })
        }
      }
    }
  } catch (e) {
    res.status(500).json({
      status: 500,
      code: 'Internal Server Error.',
      message: e,
    })
  }
})
```

4.6 Docker - Run API’s in isolation in containers

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host.

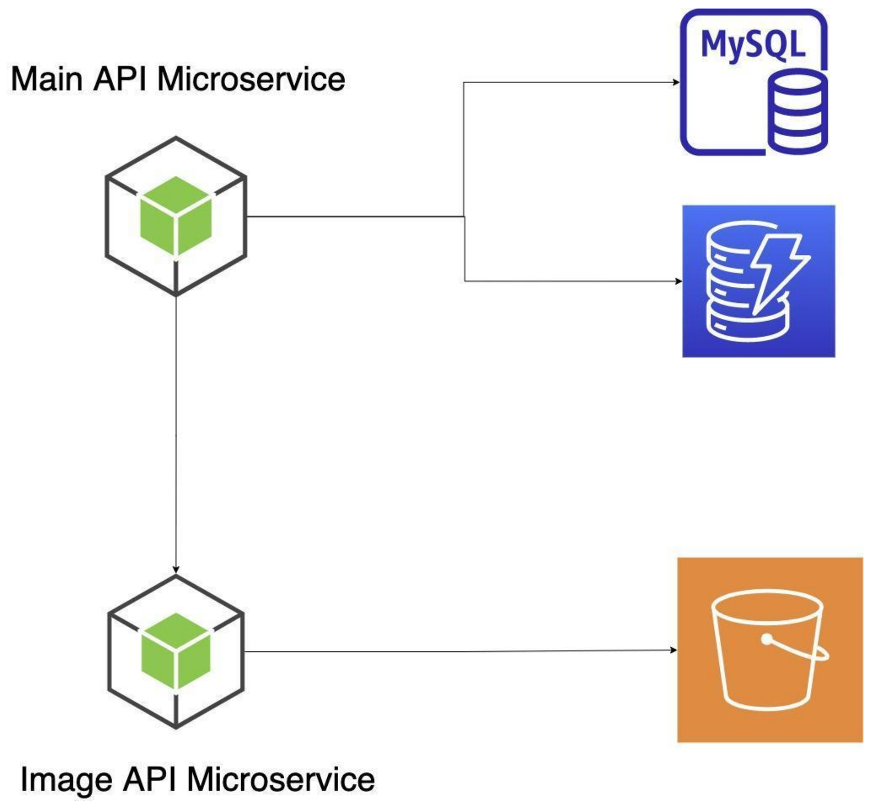
Docker in out project



4.7 Microservices architecture

Microservice architecture is a cluster of self contained, isolated services connected to each other. Each microservice in the network is responsible for exactly one task.

- Microservices are small, independent, and loosely coupled. A single small team of developers can write and maintain a service.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Supports polyglot programming. For example, services don't need to share the same technology stack, libraries, or frameworks.



4.7.1 Benefits of microservice architecture

1. **Microservices are individually scalable** - Since microservices are separate from each other we can scale them individually without affecting the characteristics of other services.

2. **Microservice reduces downtime by fault isolation** - Since the services are isolated from each other either physically or at the software level, in case once service goes down it does not affect the fidelity of the other services and it can be attended to separately.

3. **Microservice make development easier** - A team of people can work separately on a very specific service without messing with the code of the other services which may require some other set of professionals. Thus microservice makes it very easy to work on the code base as compared to a monolith application.

4. **Each microservice can be deployed independently** - Since each microservice is a standalone API in itself. It can be deployed separately from others. Each API is a modular piece of code.

4.7.2 Microservices in our project

Our Project involves two microservices

1)	**Main API**	-	Handling user related functions
2)	**Image API**	-	Storing and fetching images from the S3 storage

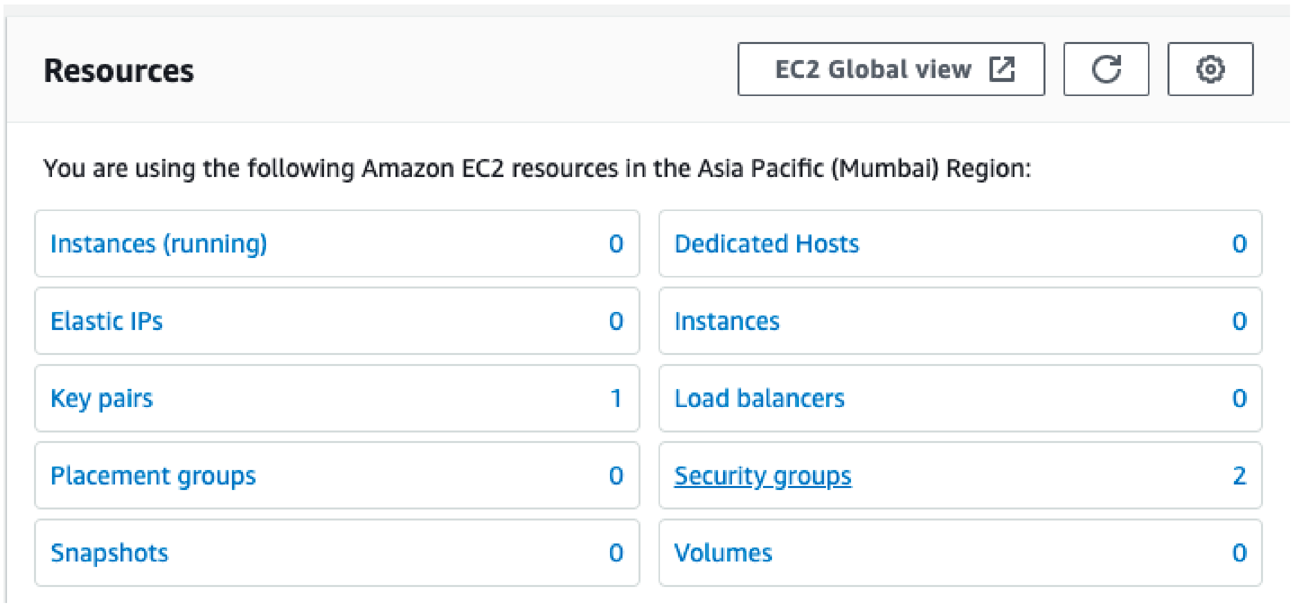
Main API Microservice

Image API Microservice

4.8 AWS EC2 - Linux compute instance on the cloud

Amazon EC2 is a web service that provides secure and resizable computing capacity on the cloud. It is used to make scalable web applications. It has a very simple interface to manage the instance with ease. It also includes automatic scaling features as the project grows and also provides analytics.

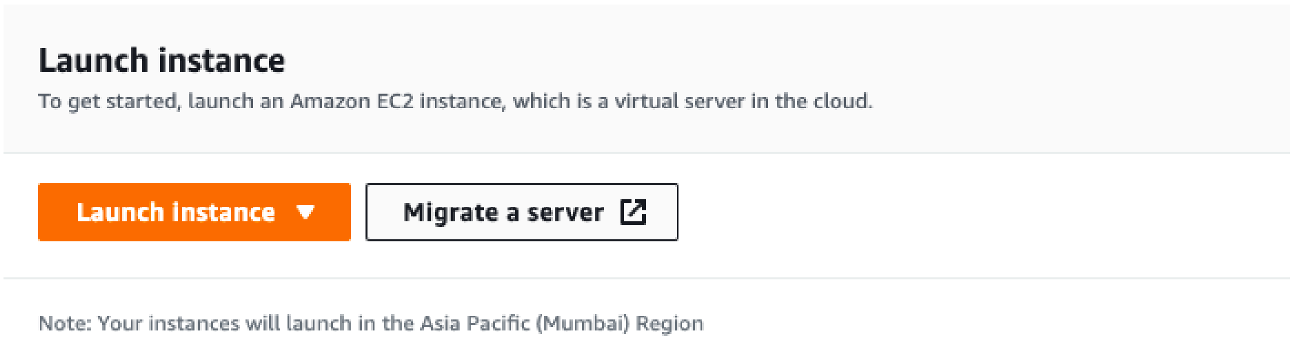
EC2 Dashboard



You are using the following Amazon EC2 resources in the Asia Pacific (Mumbai) Region:

Instances (running)	0	Dedicated Hosts	0
Elastic IPs	0	Instances	0
Key pairs	1	Load balancers	0
Placement groups	0	Security groups	2
Snapshots	0	Volumes	0

Launching an EC2 instance

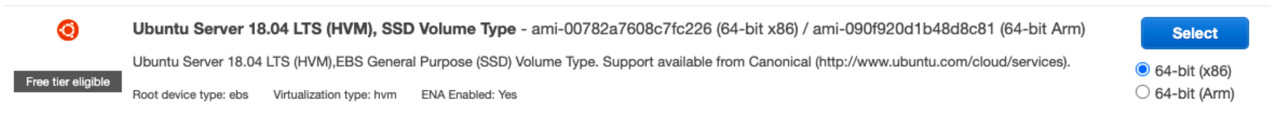


Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Note: Your instances will launch in the Asia Pacific (Mumbai) Region

Launch the instance with the OS of choice - Ubuntu



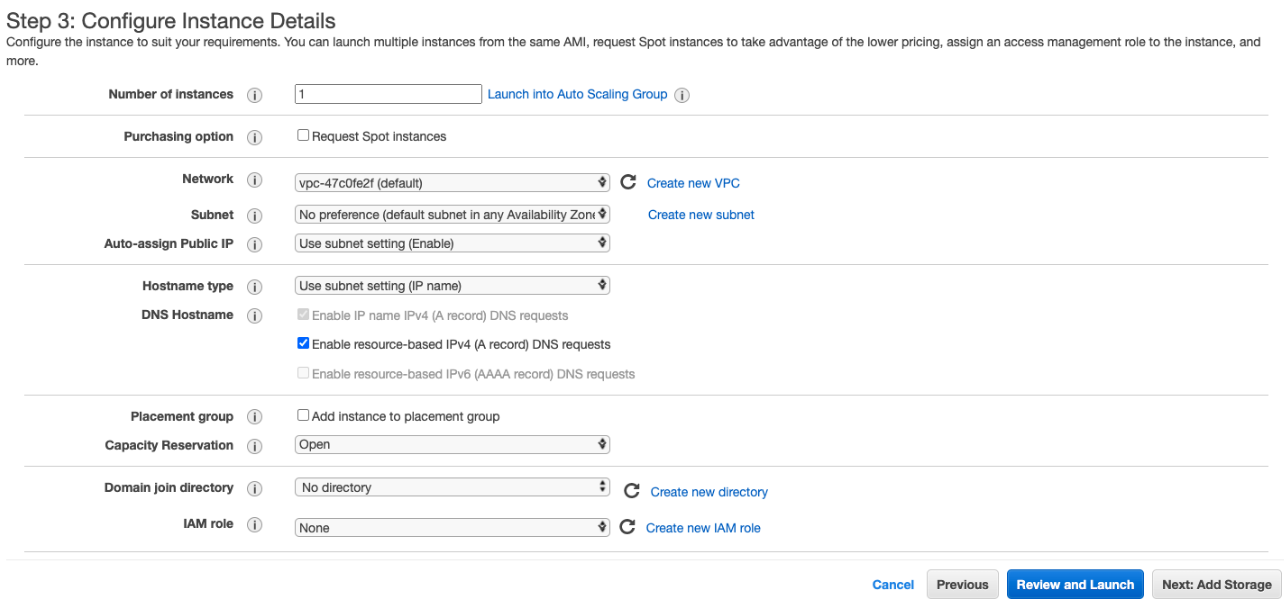
Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-00782a7608c7fc226 (64-bit x86) / ami-090f920d1b48d8c81 (64-bit Arm)

Ubuntu Server 18.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root device type: ebs   Virtualization type: hvm   ENA Enabled: Yes

[✓ 64-bit (x86)] 64-bit (Arm)

Configure the instance



Step 3: Configure Instance Details

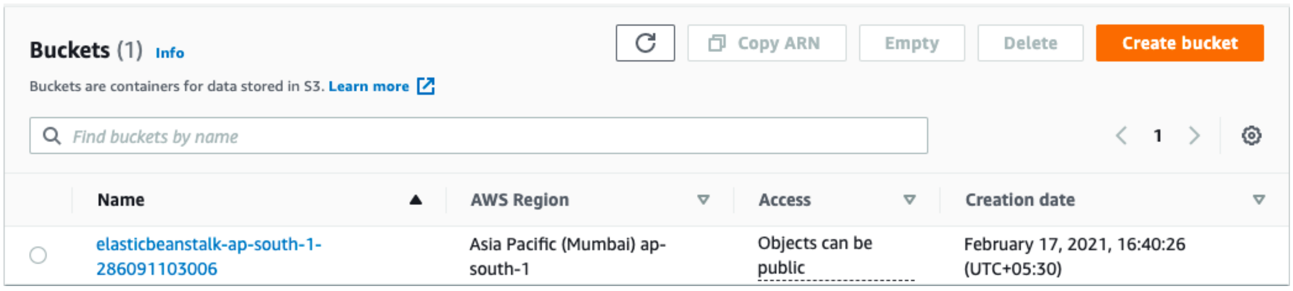
Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1 — Launch into Auto Scaling Group
Purchasing option	☐ Request Spot instances
Network	vpc-47c0fe2f (default) — Create new VPC
Subnet	No preference (default subnet in any Availability Zone) — Create new subnet
Auto-assign Public IP	Use subnet setting (Enable)
Hostname type	Use subnet setting (IP name)
DNS Hostname	☑ Enable IP name IPv4 (A record) DNS requests
	☑ Enable resource-based IPv4 (A record) DNS requests
	☐ Enable resource-based IPv6 (AAAA record) DNS requests
Placement group	☐ Add instance to placement group
Capacity Reservation	Open
Domain join directory	No directory — Create new directory
IAM role	None — Create new IAM role

4.9 AWS S3 - object storage

Amazon S3 is an object storage offering competitive data storage offering scalability, data availability, security and speed. All kinds of applications can use S3 as its storage. It offers cost effective storage and easy to use features. It is based on a proprietary storage technology developed by Amazon.

AWS S3 Dashboard

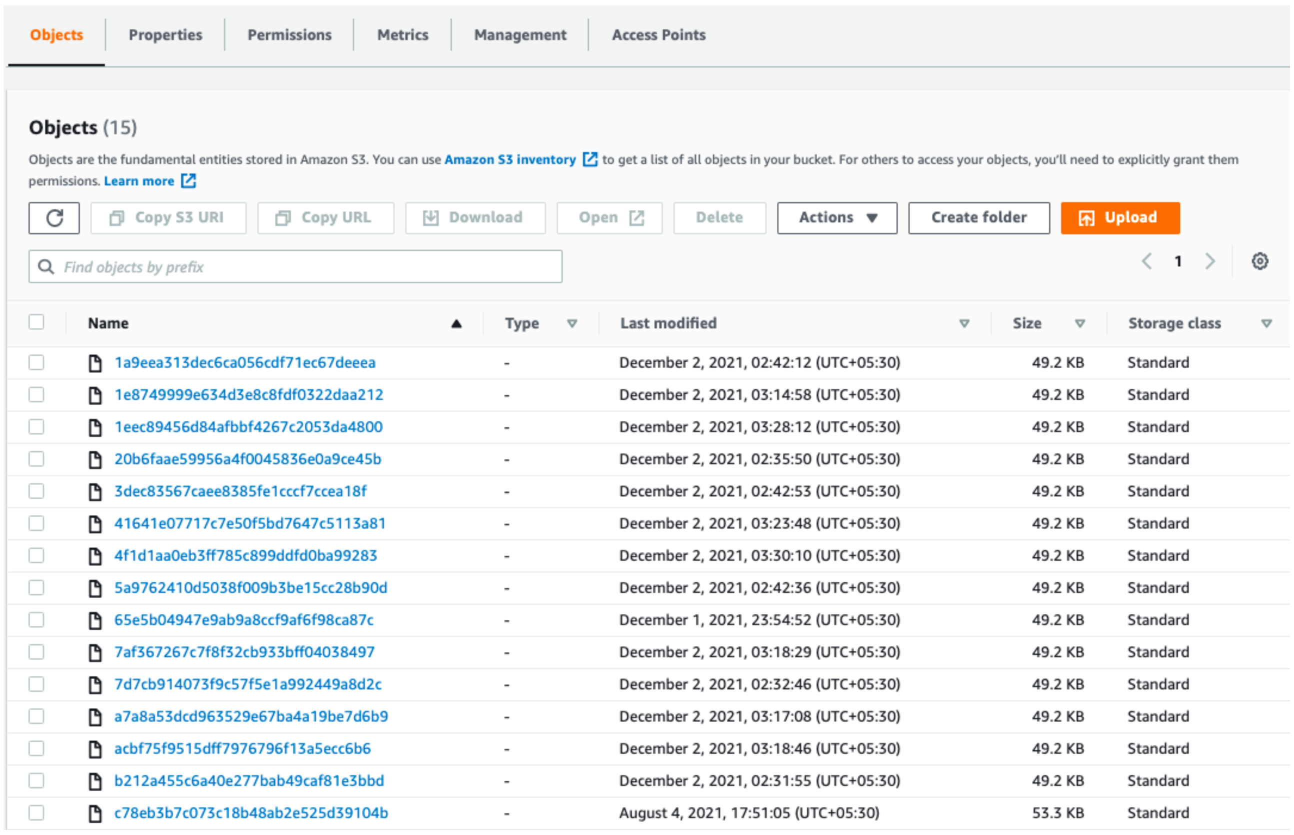


Buckets (1) Info

Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
elasticbeanstalk-ap-south-1-286091103006	Asia Pacific (Mumbai) ap-south-1	Objects can be public	February 17, 2021, 16:40:26 (UTC+05:30)

Images stored in the database



Objects (15)

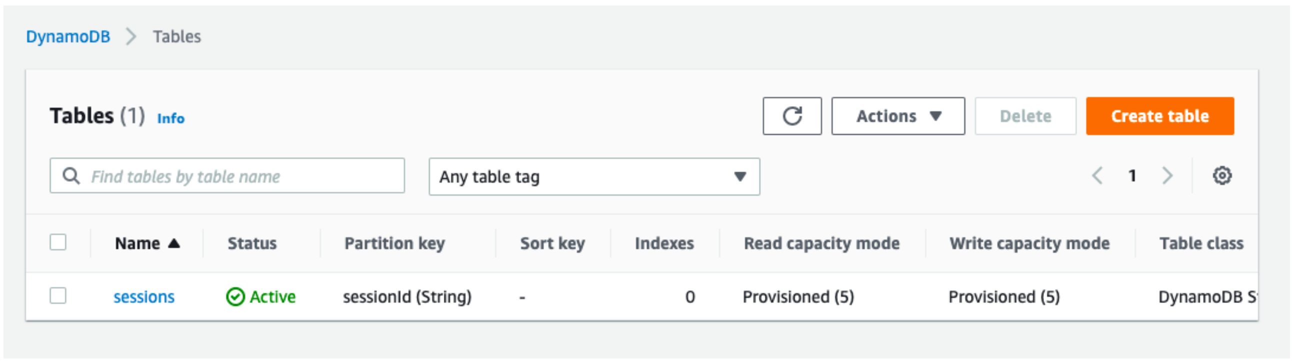
Objects are the fundamental entities stored in Amazon S3. You can use **Amazon S3 inventory** to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. **Learn more**

Name	Type	Last modified	Size	Storage class
1a9eea313dec6ca056cdf71ec67deeea	-	December 2, 2021, 02:42:12 (UTC+05:30)	49.2 KB	Standard
1e8749999e634d3e8c8fdf0322daa212	-	December 2, 2021, 03:14:58 (UTC+05:30)	49.2 KB	Standard
1eec89456d84afbbf4267c2053da4800	-	December 2, 2021, 03:28:12 (UTC+05:30)	49.2 KB	Standard
20b6faae59956a4f0045836e0a9ce45b	-	December 2, 2021, 02:35:50 (UTC+05:30)	49.2 KB	Standard
3dec83567caee8385fe1cccf7ccea18f	-	December 2, 2021, 02:42:53 (UTC+05:30)	49.2 KB	Standard
41641e07717c7e50f5bd7647c5113a81	-	December 2, 2021, 03:23:48 (UTC+05:30)	49.2 KB	Standard
4f1d1aa0eb3ff785c899ddfd0ba99283	-	December 2, 2021, 03:30:10 (UTC+05:30)	49.2 KB	Standard
5a9762410d5038f009b3be15cc28b90d	-	December 2, 2021, 02:42:36 (UTC+05:30)	49.2 KB	Standard
65e5b04947e9ab9a8ccf9af6f98ca87c	-	December 1, 2021, 23:54:52 (UTC+05:30)	49.2 KB	Standard
7af367267c7f8f32cb933bff04038497	-	December 2, 2021, 03:18:29 (UTC+05:30)	49.2 KB	Standard
7d7cb914073f9c57f5e1a992449a8d2c	-	December 2, 2021, 02:32:46 (UTC+05:30)	49.2 KB	Standard
a7a8a53dcd963529e67ba4a19be7d6b9	-	December 2, 2021, 03:17:08 (UTC+05:30)	49.2 KB	Standard
acbf75f9515dff7976796f13a5ecc6b6	-	December 2, 2021, 03:18:46 (UTC+05:30)	49.2 KB	Standard
b212a455c6a40e277bab49caf81e3bbd	-	December 2, 2021, 02:31:55 (UTC+05:30)	49.2 KB	Standard
c78eb3b7c073c18b48ab2e525d39104b	-	August 4, 2021, 17:51:05 (UTC+05:30)	53.3 KB	Standard

4.10 AWS DynamoDB - Key-value pair storage for session data

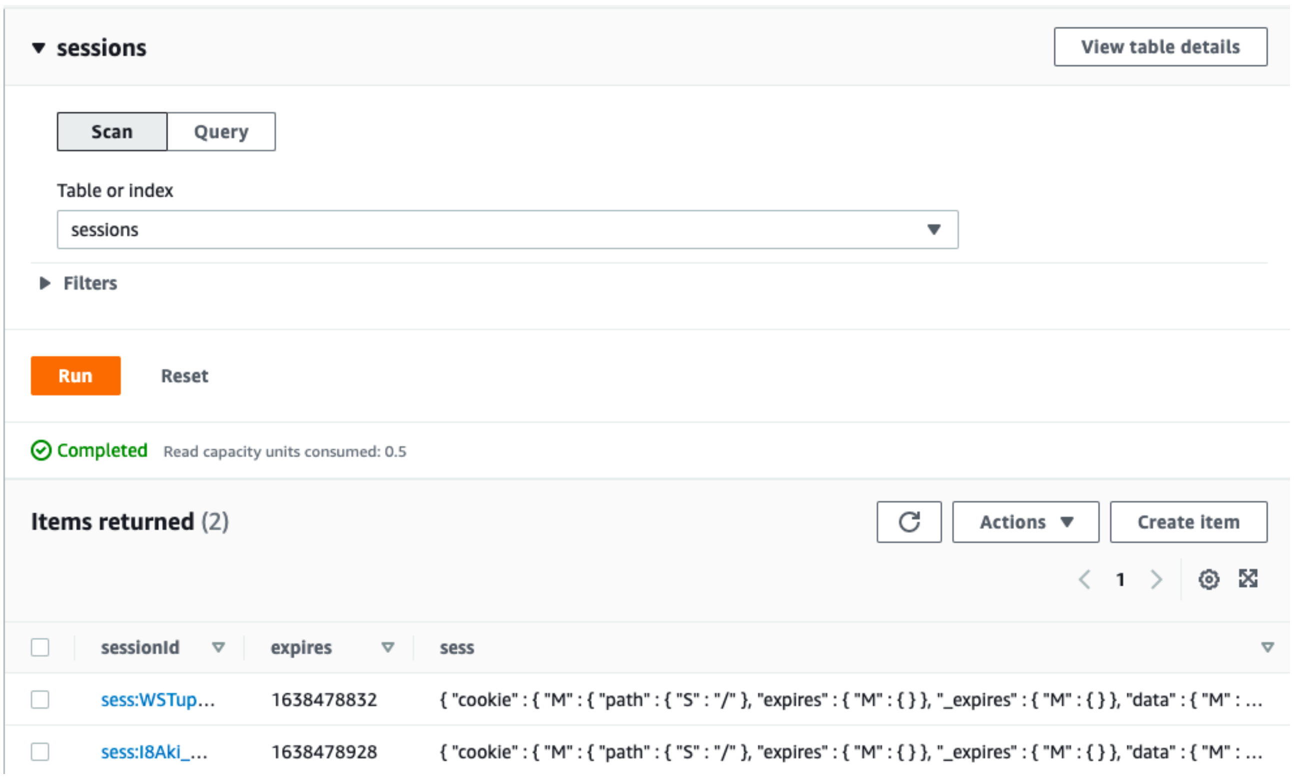
Amazon DynamoDB is a serverless, No-Sql database. It is designed to run high-performance applications at any scale. It offers built-in security with continuous backup and automatic multi region replication.

DynamoDB - Dashboard



	Name ▲	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Table class
☐	sessions	✓ Active	sessionId (String)	-	0	Provisioned (5)	Provisioned (5)	DynamoDB S

DynamoDB - Session Data



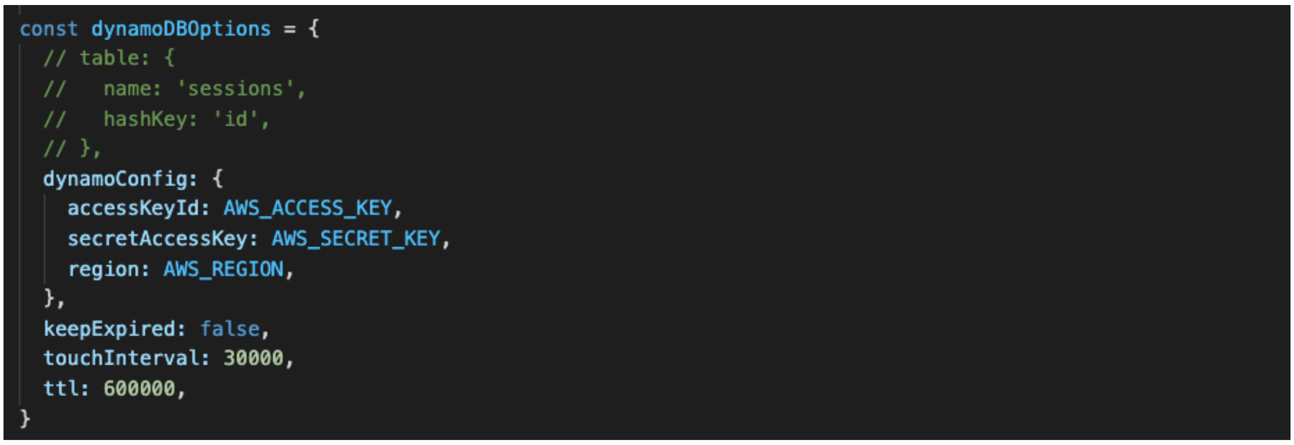
Completed Read capacity units consumed: 0.5

Items returned (2)

	sessionId	expires	sess
☐	sess:WSTup…	1638478832	{ "cookie" : { "M" : { "path" : { "S" : "/" }, "expires" : { "M" : { } }, "_expires" : { "M" : { } }, "data" : { "M" : …
☐	sess:I8Aki_…	1638478928	{ "cookie" : { "M" : { "path" : { "S" : "/" }, "expires" : { "M" : { } }, "_expires" : { "M" : { } }, "data" : { "M" : …

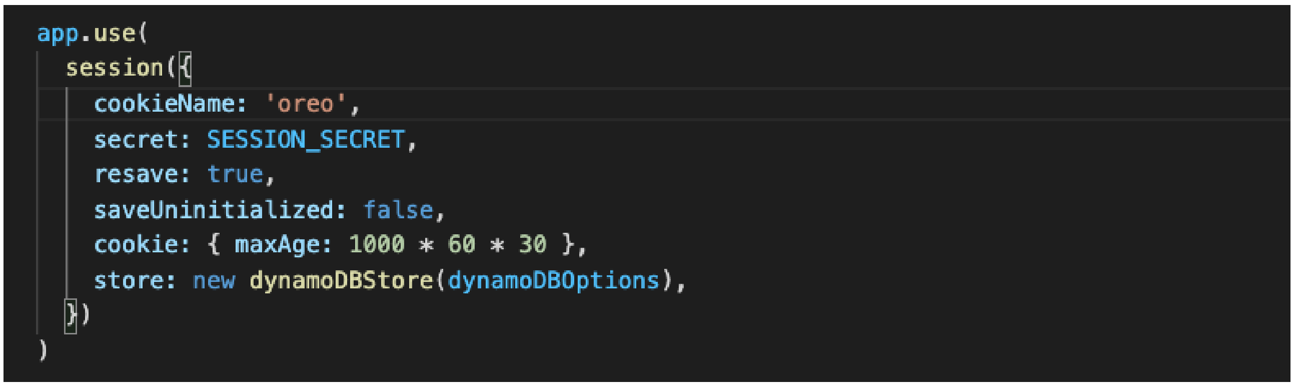
4.11 Integration of Node.js back-end with DynamoDB

Set up options



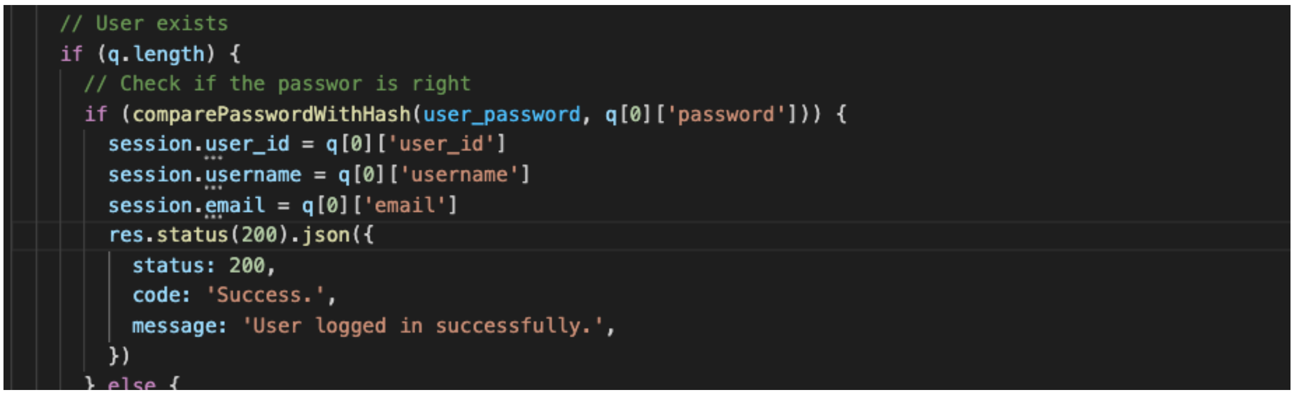
```
const dynamoDBOptions = {
  // table: {
  //   name: 'sessions',
  //   hashKey: 'id',
  // },
  dynamoConfig: {
    accessKeyId: AWS_ACCESS_KEY,
    secretAccessKey: AWS_SECRET_KEY,
    region: AWS_REGION,
  },
  keepExpired: false,
  touchInterval: 30000,
  ttl: 600000,
}
```

Use dynamoDB as a store for session information



```
app.use(
  session({
    cookieName: 'oreo',
    secret: SESSION_SECRET,
    resave: true,
    saveUninitialized: false,
    cookie: { maxAge: 1000 * 60 * 30 },
    store: new dynamoDBStore(dynamoDBOptions),
  })
)
```

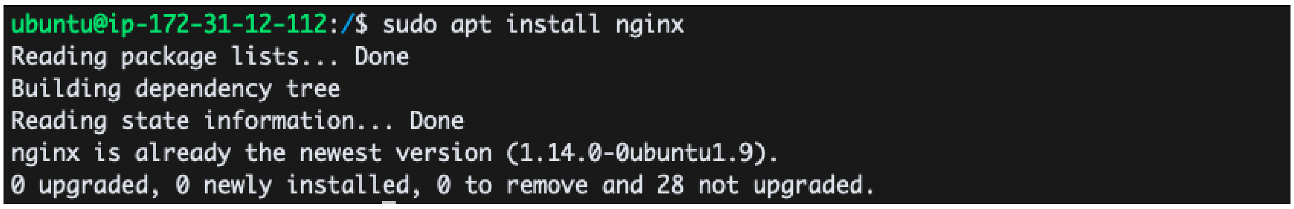
Set data in the session when the user logs in



```
  // User exists
  if (q.length) {
    // Check if the passwor is right
    if (comparePasswordWithHash(user_password, q[0]['password'])) {
      session.user_id = q[0]['user_id']
      session.username = q[0]['username']
      session.email = q[0]['email']
      res.status(200).json({
        status: 200,
        code: 'Success.',
        message: 'User logged in successfully.',
      })
    } else {
```

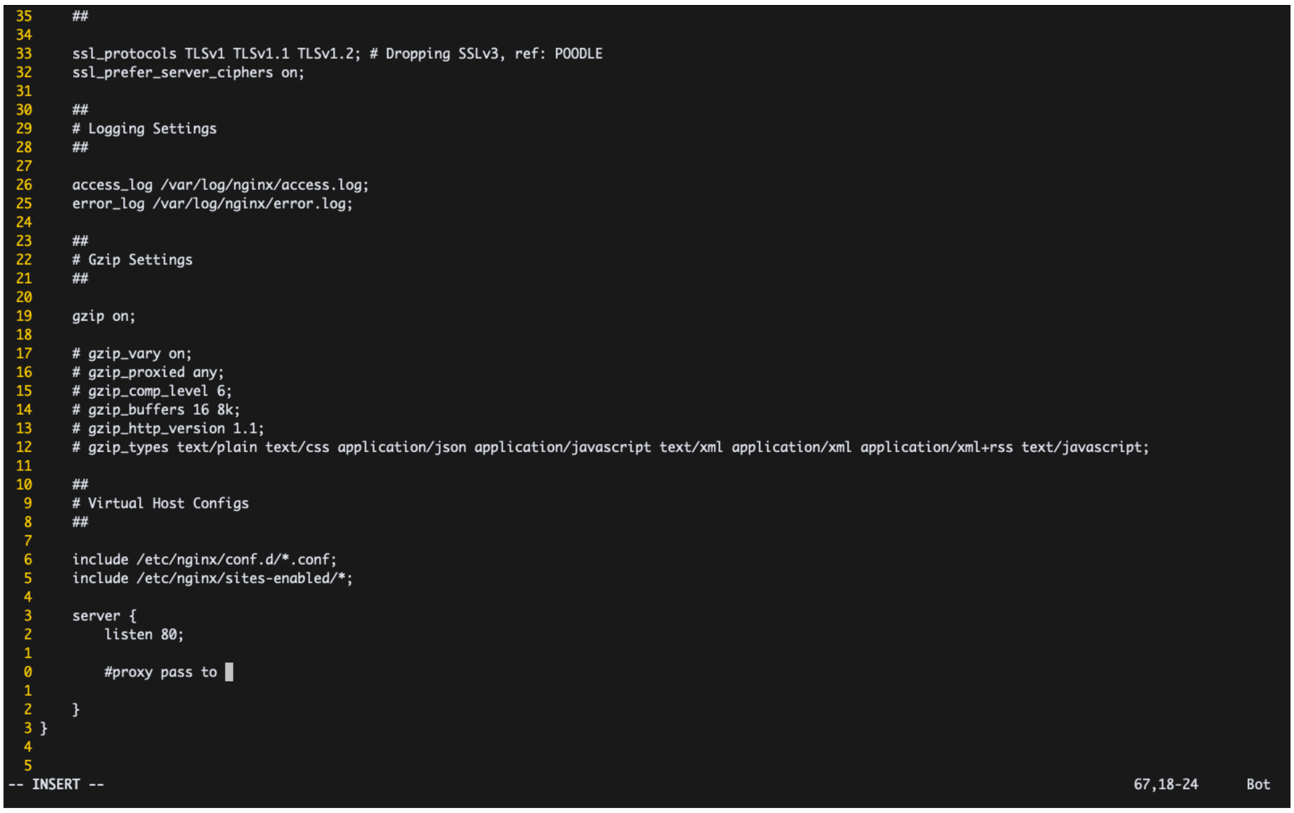
4.12 Nginx setup in ubuntu

Install Nginx

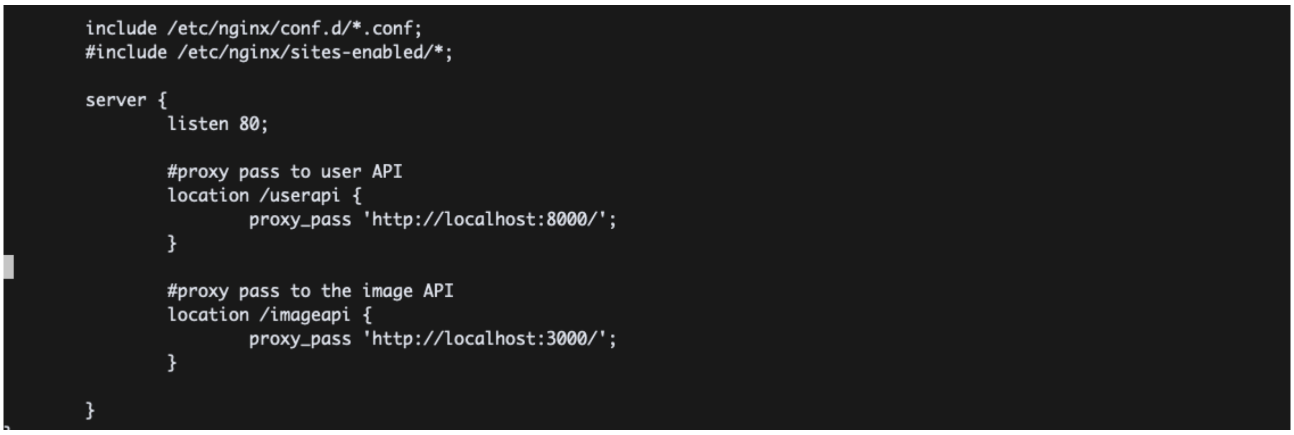


```
ubuntu@ip-172-31-12-112:/$ sudo apt install nginx
Reading package lists... Done
Building dependency tree
Reading state information... Done
nginx is already the newest version (1.14.0-0ubuntu1.9).
0 upgraded, 0 newly installed, 0 to remove and 28 not upgraded.
```

Setup Nginx as a reverse proxy

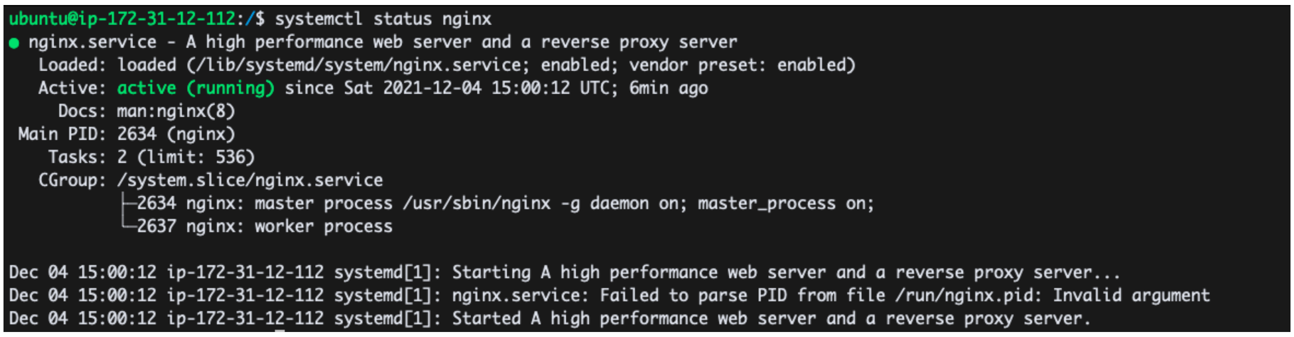


```
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # Gzip Settings
        ##

        gzip on;

        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript;

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;

        server {
            listen 80;

            #proxy pass to

        }
}


-- INSERT --                                                                    67,18-24      Bot
```



```
        include /etc/nginx/conf.d/*.conf;
        #include /etc/nginx/sites-enabled/*;

        server {
                listen 80;

                #proxy pass to user API
                location /userapi {
                        proxy_pass 'http://localhost:8000/';
                }

                #proxy pass to the image API
                location /imageapi {
                        proxy_pass 'http://localhost:3000/';
                }

        }
```

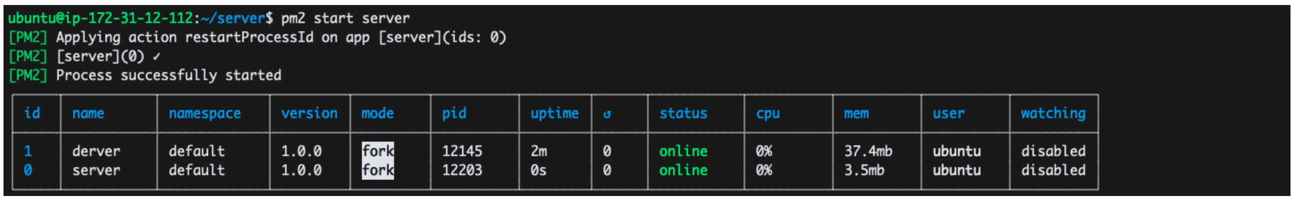
Start Nginx server



```
ubuntu@ip-172-31-12-112:/$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-12-04 15:00:12 UTC; 6min ago
     Docs: man:nginx(8)
 Main PID: 2634 (nginx)
    Tasks: 2 (limit: 536)
   CGroup: /system.slice/nginx.service
           ├─2634 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
           └─2637 nginx: worker process

Dec 04 15:00:12 ip-172-31-12-112 systemd[1]: Starting A high performance web server and a reverse proxy server...
Dec 04 15:00:12 ip-172-31-12-112 systemd[1]: nginx.service: Failed to parse PID file /run/nginx.pid: Invalid argument
Dec 04 15:00:12 ip-172-31-12-112 systemd[1]: Started A high performance web server and a reverse proxy server.
```

4.12 PM2 Process manager setup

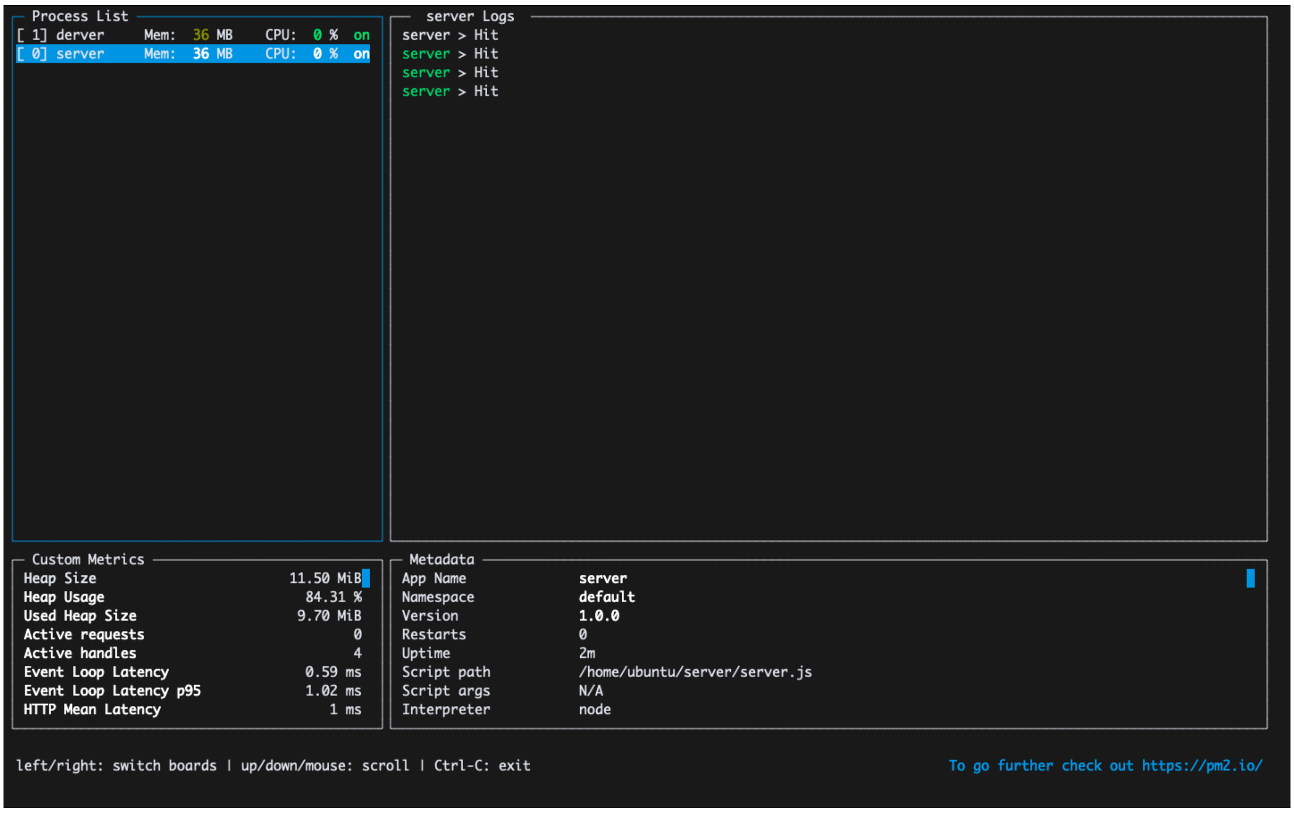
Start PM2 process manager



```
ubuntu@ip-172-31-12-112:~/server$ pm2 start server
[PM2] Applying action restartProcessId on app [server](ids: 0)
[PM2] [server](0) ✓
[PM2] Process successfully started
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
1	derver	default	1.0.0	fork	12145	2m	0	online	0%	37.4mb	ubuntu	disabled
0	server	default	1.0.0	fork	12203	0s	0	online	0%	3.5mb	ubuntu	disabled

PM2 dashboard



```
┌ Process List ─────────────────────────────┐ ┌ server Logs ──────────────────────────────────────────────
[ 1] derver      Mem:  36 MB    CPU:  0 %  on   server > Hit
[ 0] server      Mem:  36 MB    CPU:  0 %  on   server > Hit
                                                server > Hit
                                                server > Hit

┌ Custom Metrics ───────────────────────────┐ ┌ Metadata ─────────────────────────────────────────────────
Heap Size                          11.50 MiB    App Name              server
Heap Usage                           84.31 %    Namespace             default
Used Heap Size                      9.70 MiB    Version               1.0.0
Active requests                            0    Restarts              0
Active handles                             4    Uptime                2m
Event Loop Latency                   0.59 ms    Script path           /home/ubuntu/server/server.js
Event Loop Latency p95               1.02 ms    Script args           N/A
HTTP Mean Latency                       1 ms    Interpreter           node

left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit                To go further check out https://pm2.io/
```

CHAPTER 5 : CONCLUSION

5.1 Results Achieved

An image sharing application successfully deployed onto the cloud on AWS.

5.2 Applications

This application can be used by people to share pictures with other people.

5.3 Limitations

This app lacks a lot of features that some of the other large scale apps have such as the ability to follow other people and like and comment and stuff like that.

5.4 Future Work / Scope

1. Ability to like and comment on posts
2. Follow other people and see their posts in your feed
3. Implement messaging facility in the app

REFERENCES

1. **https://docs.aws.amazon.com/**
2. **https://docs.docker.com/reference/**
3. **https://expressjs.com/en/4x/api.html**
4. **https://arpadt.com/articles/persisting-espress-session-in-dynamodb**