



Department of Computer Science and Automation
Research in Computer and Systems Engineering

Internship Report

May 13, 2020

Documented By: Kiran Gosavi (Matrikelnummer: 60935)
Academic Supervisor: Prof. Dr.-Ing. habil. Kai-Uwe Sattler
Company Supervisor: PD Dr.-Ing. habil. Jürgen Nützel (jn@4fo.de)
Company: 4FriendsOnly.com Internet Technologies AG (www.4fo.de)
Address: Bahndamm 8, 98693, Ilmenau



4FriendsOnly.com
Internet Technologies AG

Contents

1	Acknowledgement	3
2	Introduction	4
2.1	Purpose	5
2.2	Scope	5
2.3	Future Scope	5
3	Project Architecture	6
3.1	Headless Front-end	7
4	Authentication Authorization	8
4.1	User Management with AWS	9
5	Database Schema	10
6	Content Management System : Strapi	12
7	Embedding AWS QuickSight Dashboard	14
8	Deployment Details	15
9	Summary	16
10	Declaration	16
	References	17

1 Acknowledgement

The work presented in this document wouldn't have been possible without kind and generous support from my academic as well as company supervisors. This internship work is a result of collaboration with the company "4FriendsOnly.com Internet Technologies (4FO) [1]" , and reserves all the ownership rights. The "4FriendsOnly.com Internet Technologies" is a spin-off of the Fraunhofer IDMT and was founded in the year 2000. The company has been working in the field of e-commerce, mobile solutions and cloud solutions ever since.

The company gave me the opportunity to work as an intern starting from 1st October 2019 till 1st April 2020 (6 months period). I would like to express my deepest gratitude to my company supervisor Dr.-Ing. habil. Juergen Nuetzel, who believed in my capabilities. I got an excellent opportunity to design a product from scratch, which was a great deal of experience for my professional career . I was part of the architectural design decisions team in the company. From time-to-time Dr. Juergen helped me with the architectural design decisions and gave his expert insights in this field. I got an opportunity to learn new technologies like Node.js, AWS, Strapi, MongoDB, web service and GraphQL. Designing a new product from scratch improves analytical skills as well as decision making ability. In addition to that, it was a great team-collaboration experience for me.

I would also like to express my gratitude to my academic supervisor Prof. Dr.-Ing. habil. Kai-Uwe Sattler. Prof. Sattler gave me the opportunity to complete my internship with the 4fo company. Prof. Sattler is an expert in the field of Database and Information Systems. He is currently working as a coordinator of a priority program "Scalable data management for future Hardware (spp 2037)" [2].

Last but not the least, my thanks and appreciation goes to my company colleagues, who helped me with their expertise in this field.

2 Introduction

Amazon Web Services (AWS) is a dominant player in the cloud solution market. The 4FO company provides Infrastructure-as-a-Service (IaaS) and Software-as-a-Service (SaaS) solutions to their customers using AWS cloud solutions. AWS provides a wide range of web services in the field of computing (EC2), storage(S3), security, machine learning, analytics, databases etc. Currently there are around 212 AWS web services in the market [3] . Learning AWS cloud solution technology is a valuable tech skill now-a-days.

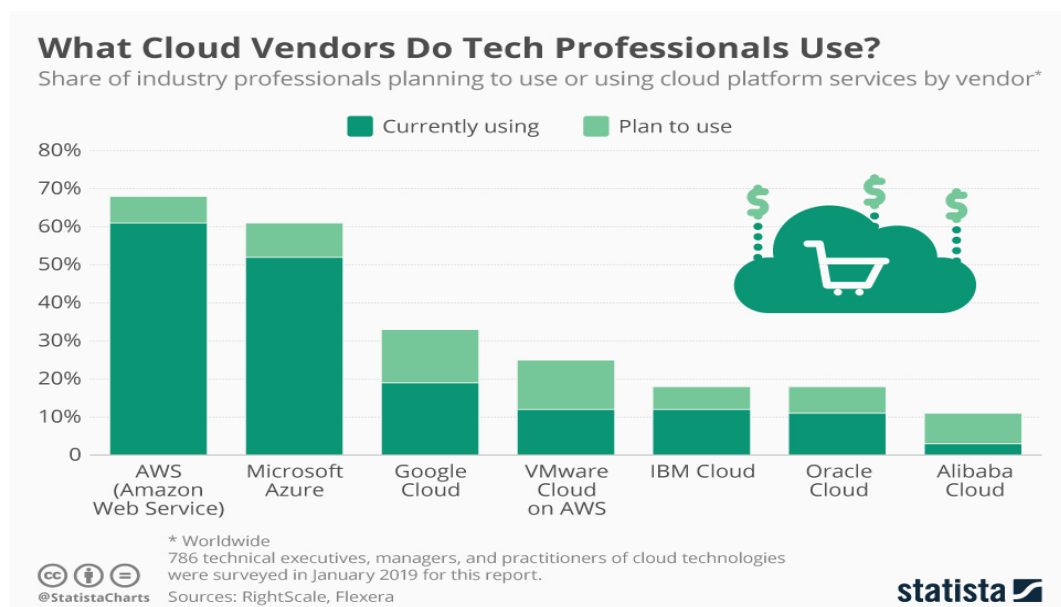


Figure 1: Amazon Web Services(AWS) in the cloud solution market[4].

As a cloud solution provider to their customers, the 4FO needs a product based on AWS cloud, which will be a centralized web service for all needs of their customers . Providing a centralized dashboard for all operations and services was the inspiration behind their new product ‘Xperiance.Cloud’. I am very glad that I got an opportunity to work on ‘Xperiance.Cloud’ project from scratch. The dashboard will be providing different services like purchasing new service(eg. Repricing, Recommendation etc.) , exploring available services provided by 4FO, billing and other user management tasks. This project is using various new technologies such as Strapi Content Management System with Node.js, MongoDB, Nosql database and Angular for the frontend. For deployment, the Xperiance.Cloud project uses AWS cloud solutions. In addition to this, it uses AWS Quicksight web service to show their customers interactive graphs based on their requirements.

2.1 Purpose

This document depicts complete information about the 'Xperiance.Cloud' project. In addition to that, it includes all the architectural design decisions explaining different technologies used e.g. Strapi as a CMS, MongoDB as a database, AWS etc. It also serves the purpose of knowledge transfer within the development team of 'Xperiance.Cloud' project. Many important aspects of the project will be covered in the area of security, storage, computing, deployment authentication and authorization etc. in depth.

2.2 Scope

The complete project of 'Xperiance.Cloud' will take more than 6 months to finish, hence this document covers version 1 of the project plan. It includes the functionality of authentication and authorization, user management, deployment on AWS, database schema design and implementation . In addition to that, version 1 also includes embedding quicksight dashboard to Xperiance.Cloud console.

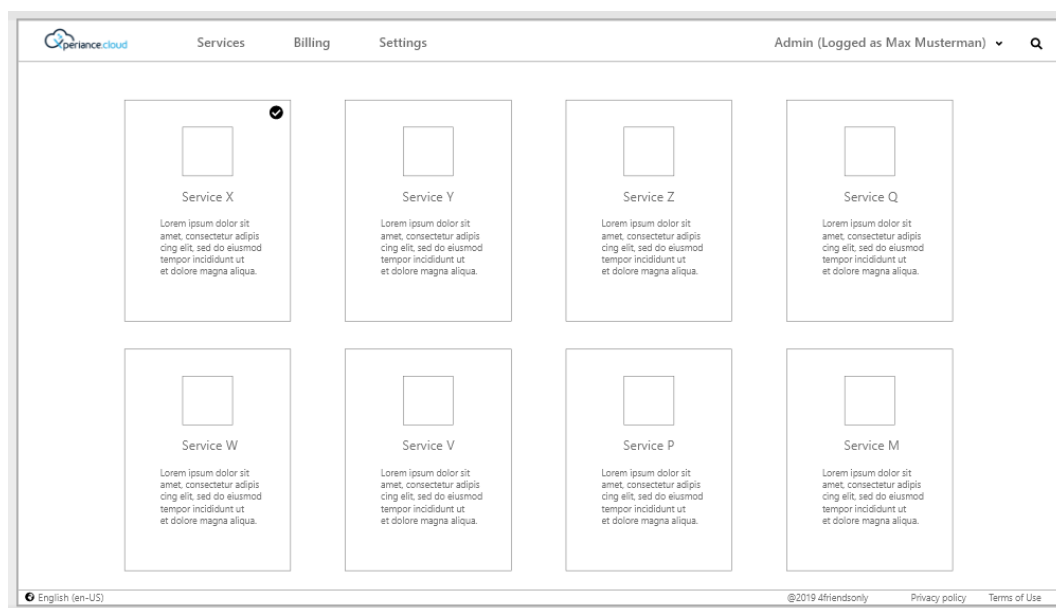


Figure 2: Snapshot of version 1 'Xperiance.Cloud' UI design.

2.3 Future Scope

The 'Xperiance.Cloud' has many functionalities in the backlog for version 2 implementation. Few of them are complete automation of the service installation, billing after the customer has placed an order without manual intervention, performance monitoring of each service using graphs to show usage information in terms of cost, memory, CPU etc.

3 Project Architecture

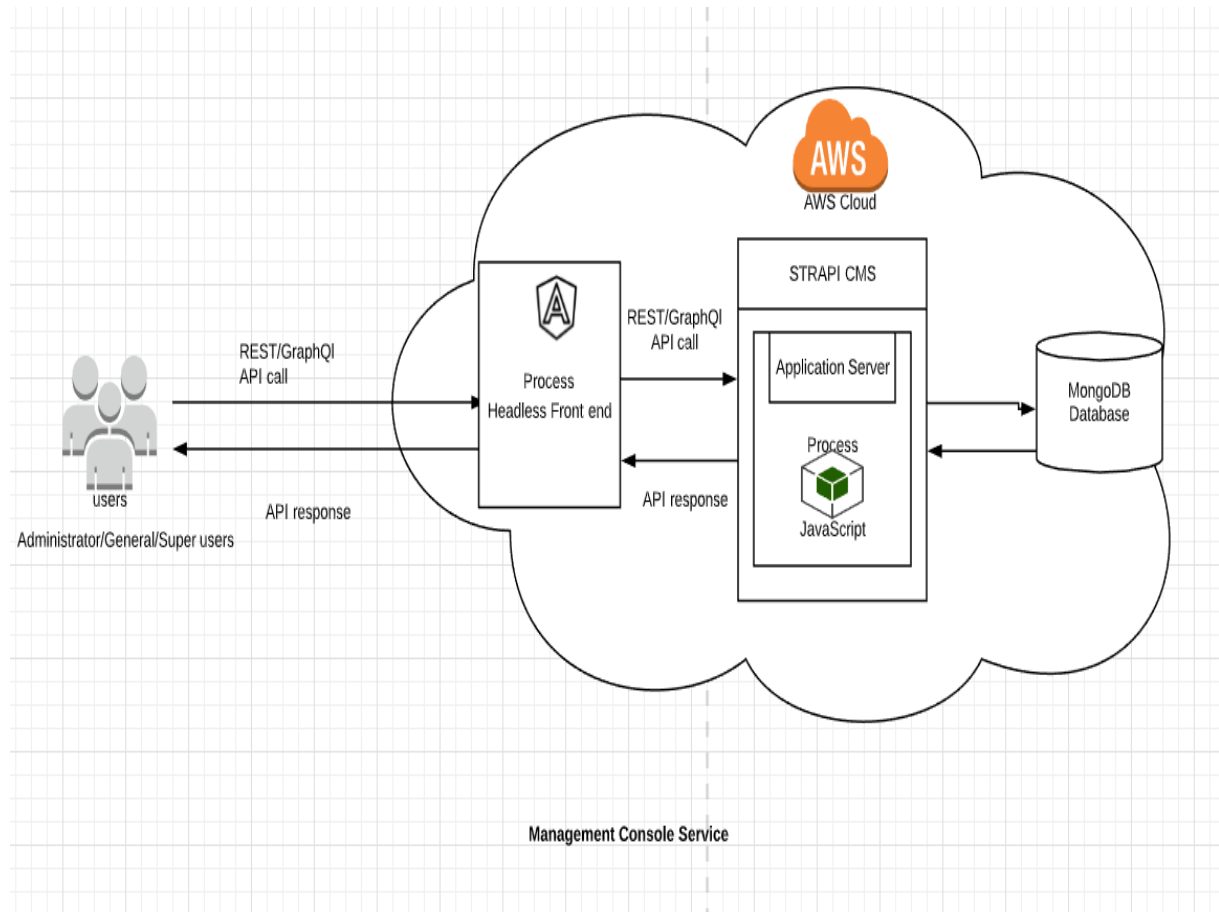


Figure 3: Block diagram of 'Xperiance.Cloud' architecture .

The Xperiance.Cloud console will serve as a central dashboard to manage, purchase or explore different services provided by 4FO Customers (eg. limmert etc) will be able to purchase and explore different services provided by 4FO. Billing of a particular customer will be done using this service. Overall functionalities can be listed below:

- Centralize dashboard for all available services
- User login management to this dashboard
- Billing service based on list of purchased services

3.1 Headless Front-end

The headless front-end is a website which is hosted in an AWS S3 bucket and is available via the URL Xperiance.Cloud. This front-end is called headless, because it is separated from its headless back-end via a defined RESTful or GraphQL API. The front-end will be written in Angular.

The web-based front-end has the following user types (roles):

- The **admin(Administrator)** is a representative of 4FO, which is the owner of the console. It is a kind of a super user. Several logins may have the admin role.
- A **customer** is a company which has access to at least one of the services. A customer has several users. The number of logins is limited by the plan a customer has chosen.
- **Anonymous** users are not logged in. Those users see only a subset of the information. E.g. marketing and pricing information.

4FO's customers have their own customers who use the service. We call them service customers. Currently the console will take care of those customers. The API of the console server will support both REST as well as GraphQL. The console server will do the following tasks with a service:

- **Setup:** For a new customer a new instance of a service will be created.
- **Configuration:** Admin or customer provides data which enables a service to run in an appropriate way.
- **Monitoring:** For billing purposes, the console monitors and visualizes certain transactions of the service.

4 Authentication Authorization

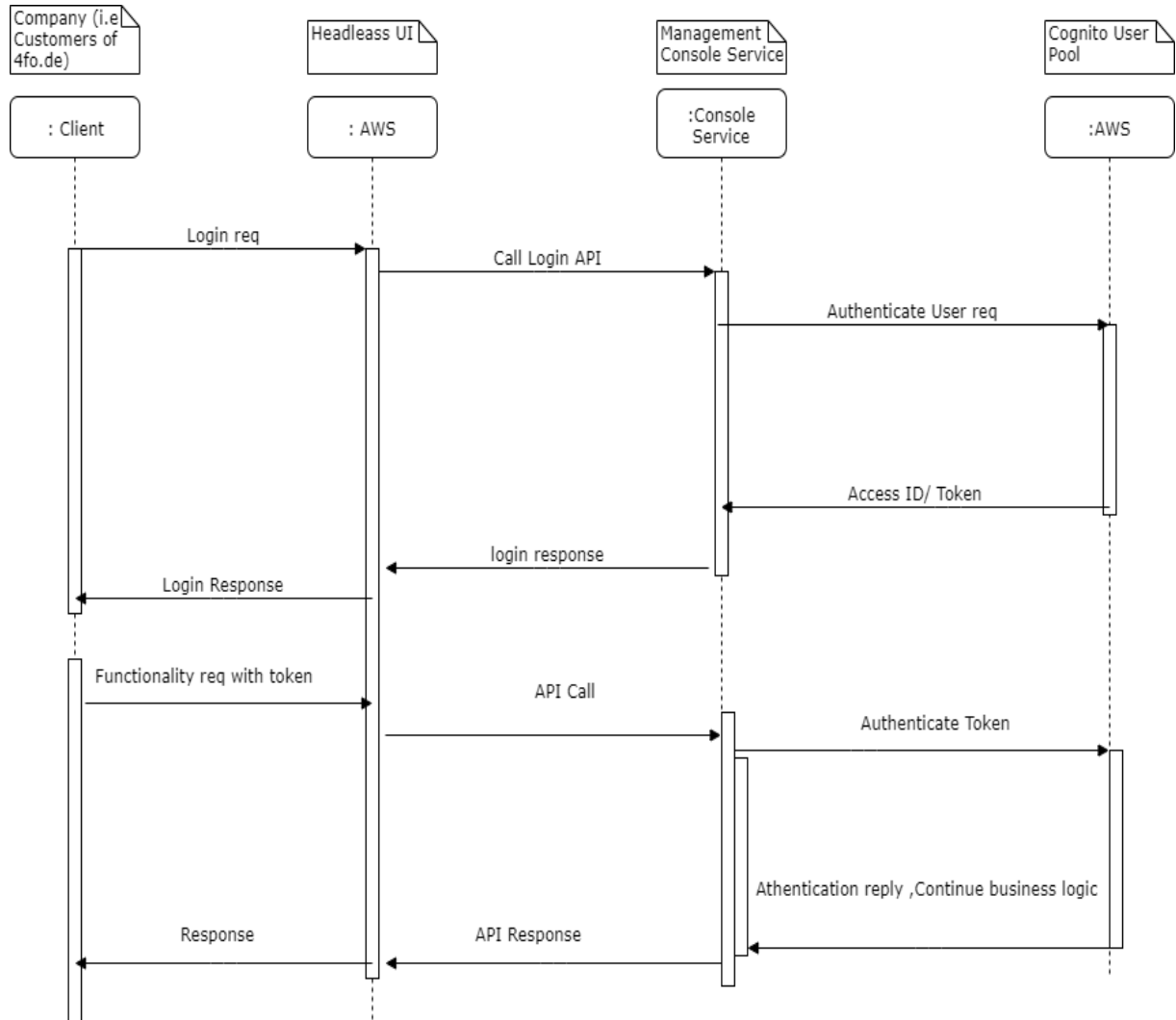


Figure 4: Authentication and authorization sequence diagram.

The management console will use AWS Cognito service for authentication. AWS cognito provides simple authentication and authorization for web or mobile applications. User pools are directories in AWS Cognito. Users can send their credentials in a request to the user pool for authentication purposes. AWS Cognito then sends JWT (JSON Web Token) if authentication is successful. Users can use this JWT token in subsequent requests for authorization .

JWT token consists of following components: :

ID Token : The ID Token contains claims about the identity of the authenticated user such as name, email, and phone number.

Access Token : The Access Token grants access to authorized resources.

Refresh Token : The Refresh Token contains the information necessary to obtain a new ID or access token.

4.1 User Management with AWS

The AWS IAM (Identity and Access Management) web service allows you to access AWS services and resources securely. The AWS IAM provides many features such as granular permission control, multi-factor authentication and eventual consistency. The management console uses AWS Cognito for user authentication and management. A dedicated user pool will be created in AWS Cognito for Management Console users. In that user pool, the following groups will be created which will have different access rights:

- Administrator user group
- General user group
- Super user group

The AWS Cognito is a great solution for authentication and authorization requirements of web or mobile applications. Cognito provides third party login services such as Facebook, Gmail etc. Following are two main components of AWS cognito:

User pool: Manages user management related operations e.g.sign-in, sign-up.

Identity pool: Manages user access to other AWS resources like EC2 instances.

User authentication and authorization process in AWS Cognito can be listed as below:

Step 1: User first sends its credential to the user pool to get a user pool token. User pool grant users the token only if the credentials are valid.

Step 2: Users can contact Identity pool to get AWS credentials in exchange for a user pool token.

Step 3: With the AWS credentials users can get access to other AWS resources like EC2 instance or Quicksight dashboard.

5 Database Schema

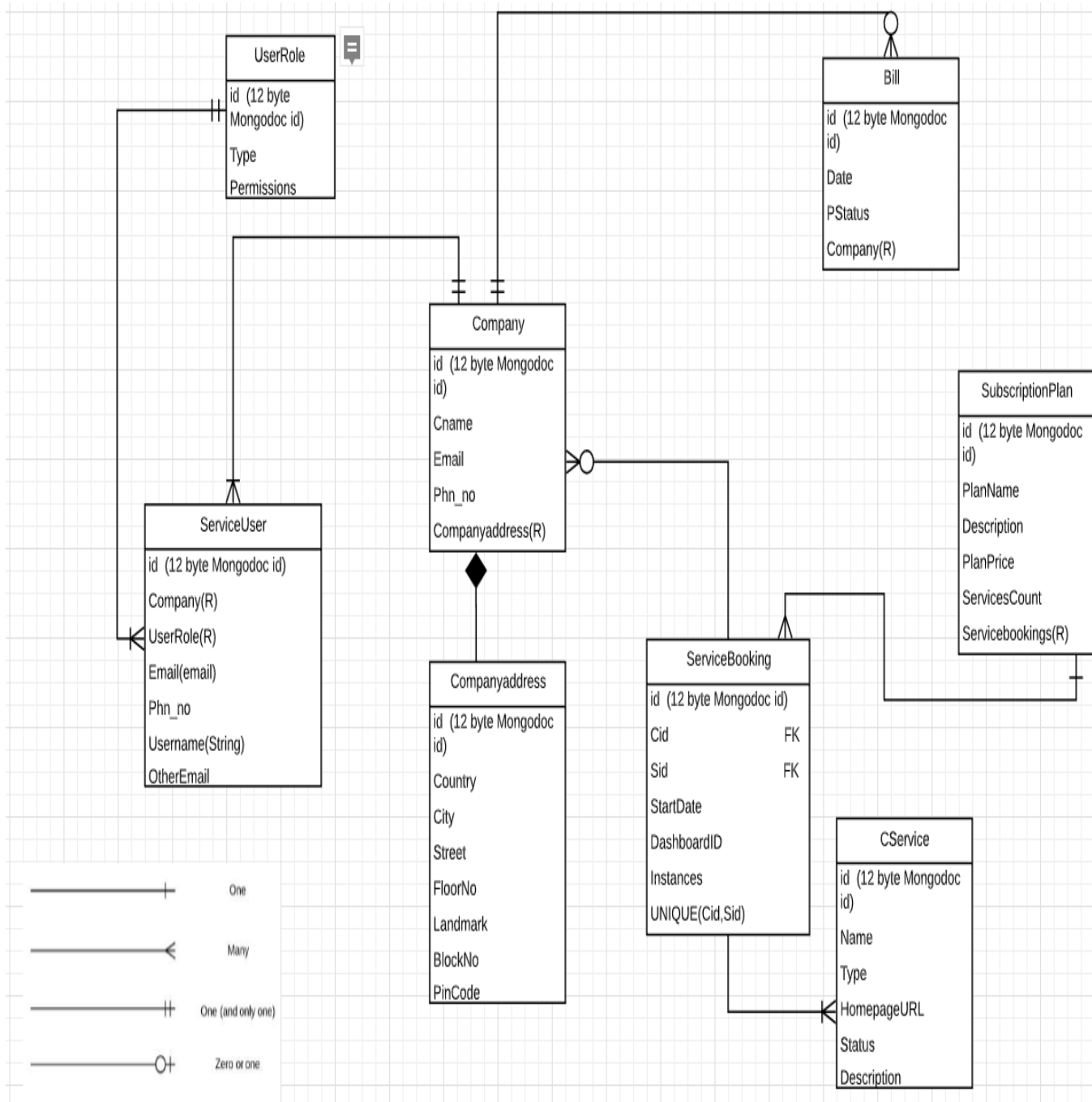


Figure 5: Database schema of 'Xperience.Cloud' project.

The Management console will use the above database schema for the backend implementation.

MongoDB : The Management Console will use MongoDB NoSQL database for backend storage. MongoDB is a popular open-source document oriented NoSQL database . It uses JSON-like documents to store data. The document model maps to the objects in our application code. MongoDB documents are similar to Javascript Object Notation (JSON), but use a variant called BSON(Binary JSON). MongoDB has several advantages like high availability, reliability, and scalability.

Why MongoDB?

- MongoDB is an open-source No-sql database.
- Data models and schema support: MongoDB supports dynamic schema which is more suitable for unstructured data. In Xperiance.cloud, we need support for dynamic schema which will support unstructured data. This feature of MongoDB allows users to build applications without defining schema first.
- Data structure: Data is stored in a table in sql databases. Whereas, unstructured data is supported by nosql databases such as image, text,audio,video,documents.
- Scaling: Traditional sql databases support vertical scaling whereas MongoDB supports horizontal scaling . i.e. a new machine is added to the cluster if an application requires more computing power.

```

local 0.000GB
> show collections
Test2
bills
companies
companyaddress
core_store
servicebooking
services
serviceusers
strapi_administrator
subscriptionplan
test
userroles
users-permissions_permission
users-permissions_role
users-permissions_user
> db.serviceusers.find().pretty()
{
  "_id" : ObjectId("5e74f9eb3ef3d80af4a70f2e"),
  "Username" : "gokirans@gmail.com",
  "OtherEmail" : "kirangosavi93@gmail.com",
  "ContactNo" : "09960797520",
  "created_at" : ISODate("2020-03-20T17:14:19.883Z"),
  "updated_at" : ISODate("2020-03-20T17:14:19.890Z"),
  "__v" : 0,
  "userrole" : ObjectId("5e74f9da3ef3d80af4a70f2d")
}
{
  "_id" : ObjectId("5eb5329998d3c53ef451d49c"),
  "Username" : "sdhjjs@hdfj.com",
  "OtherEmail" : "xysdys@gmail.com",
  "ContactNo" : "55879809-00--0",
  "created_at" : ISODate("2020-05-08T10:21:13.484Z"),
  "updated_at" : ISODate("2020-05-08T10:21:13.492Z"),
  "__v" : 0,
  "company" : ObjectId("5e7b2be7859e183a048b477b"),
  "userrole" : ObjectId("5e74f9da3ef3d80af4a70f2d")
}
  
```

Figure 6: Snapshot of MongoDB database of 'Xperiance.Cloud' project .

6 Content Management System : Strapi

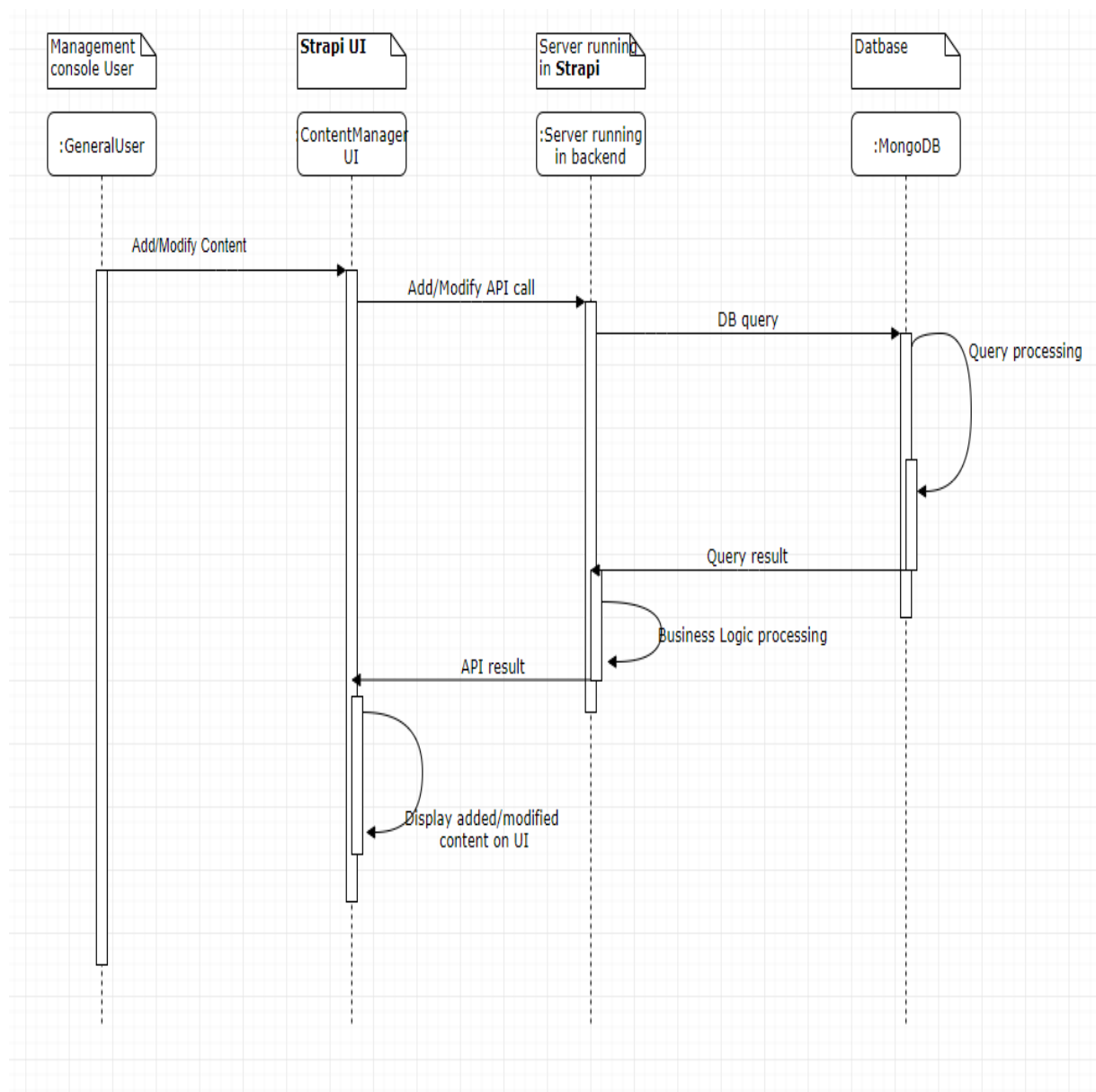


Figure 7: Sequence diagram showing internal working flow of Strapi CMS .

The Management Console System will use Strapi as Content Management Systems (CMS). Strapi is headless CMS. Content Management System allows users with minimal technical expertise to add, modify and delete content from a website. CMS has two major parts: Interactive UI and Content Delivery Application (CDA). CDA compiles and updates the content on a website which has been updated or added by the CMS user via UI.

Strapi simplifies content architecture for developers by providing database schema creation using UI. Once the content structure is defined, content editors can create, edit and delete any type of content in full autonomy from the IT department. Strapi features can be listed as follows:

- Built-in Customizable UI for content management.
- Built-in CRUD operation APIs from the Content Manager.
- Build-in authentication and authorization.

Content Type: The Strapi uses Content Type as a basic entity to represent content. Content types make up database schema. Inserting, deleting content becomes very easy with content type. Basic CRUD operations are available for every content type. It is also possible to add a custom API for any content type. By default, the access will be private. However, you can manage the accessibility of content by changing roles and permissions.

Roles Permission: The Strapi provides built-in authorization and authentication features. However, you can modify and manage the accessibility of any content type by changing roles and permissions. Different roles and permissions can be assigned to every content type. The management console will use AWS Cognito for authentication and authorization, hence every content type will have public accessibility.

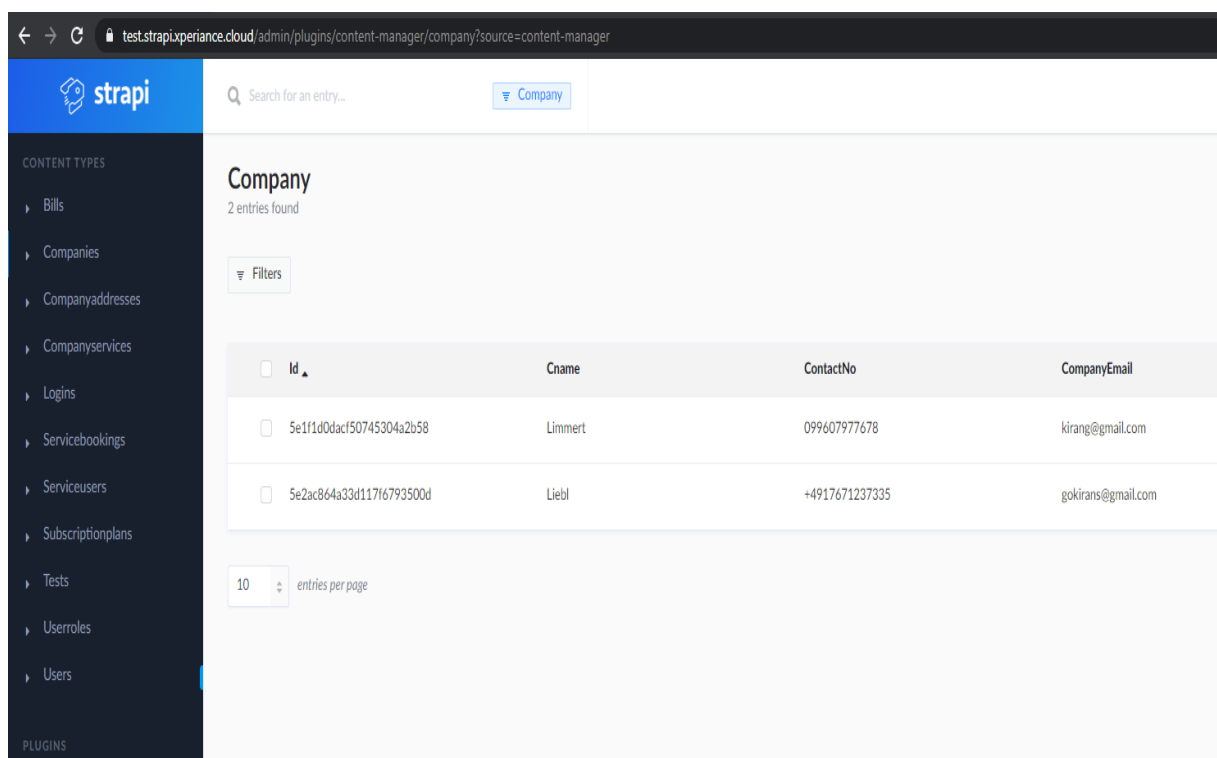


Figure 8: Snapshot of Strapi UI

7 Embedding AWS QuickSight Dashboard

The AWS quicksight dashboard is AWS's new serverless architecture which allows users to embed dashboards in an application. AWS allows you to scale your insights based on your growing user-base by enabling a pay-per-session pricing model. AWS enables dashboard viewing to the authenticated users of AWS IAM federation (or SAML, OpenID Connect). To simplify the embedding of the quick sight dashboard into your application, AWS provides quick sight JavaScript SDK. In Xperience.cloud, we have implemented a backend API to fetch quick sight dashboard URLs. This API will be consumed by the front-end UI, to display interactive dashboard graphs.

In Xperience.Cloud we have used the AWS Cognito federation. The 'getDashboard-URL' API assumes a role to get the access key, secret key and session token parameters. With these parameters AWS Identity and Access Management allow users access to the interactive quicksight dashboard .

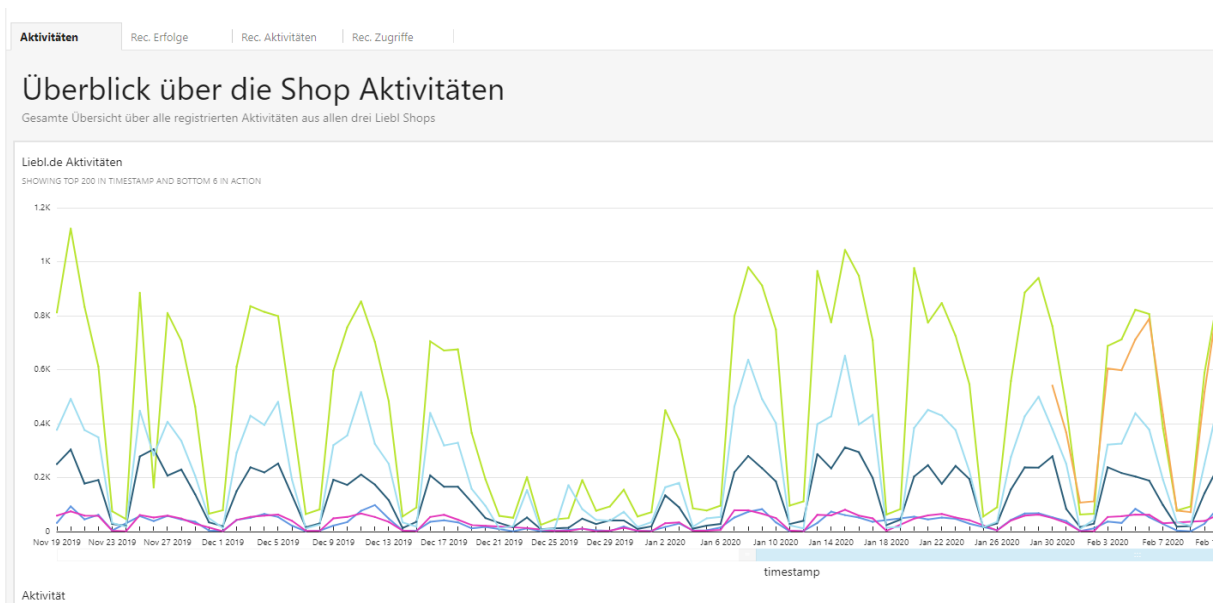


Figure 9: AWS Quicksight Embedded dashboard in 'Xperience.Cloud' project .

The process of quicksight dashboard embedding requires the following steps:

- Step 1:** Whitelist your application domain in quicksight dashboard
- Step 2:** Set permissions in your AWS account for embedded viewers
- Step 3:** Authenticate user on your application server to get embedded dashboard URL
- Step 4:** On UI page use AWS quicksight SDK, to embed the dashboard

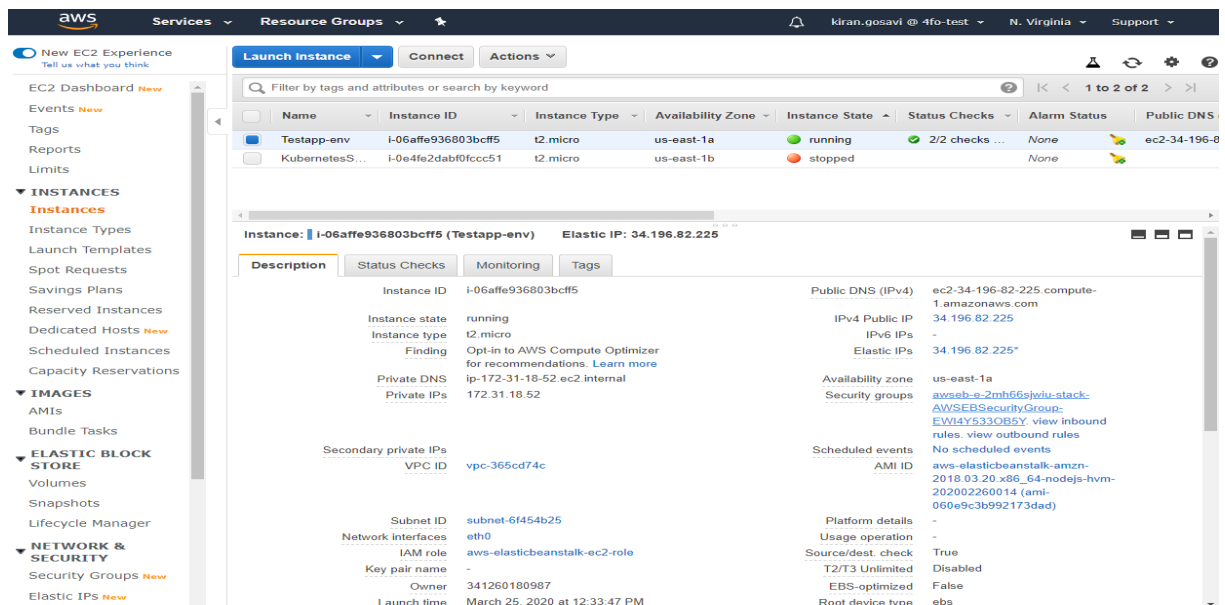
The assumed role API returns 3 parameters: access key, secret key and session token. You need to update your AWS config with this new parameter and call getDashboardEmbedUrl This AWS API will return a signed dashboard URL, which you can embed in your application.

8 Deployment Details

The Xperience.Cloud console is deployed on AWS cloud services. The Xperience.Cloud uses EC2 (Elastic Compute Cloud) AWS service for frontend and backend deployment. Users can start an EC2 instance when computation is required and can stop/pause an instance afterwards. They will be charged only for the time the instance is active, thus making it very cost efficient compared to traditional physical server deployment. EC2 instance is highly secure and uses public key cryptography. It uses a 2048-bit SSH-2 RSA public and private keys to login. Steps to create EC2 instance are as below:

- Select pre-configured machine template with desired AMI (Amazon Machine Image).
- Choose instance type with desired capacity and computing power.
- Optional step: you can configure the EC2 instance additional security setting by choosing IAM role and network access.
- Launch an instance.

After the launch of an EC2 instance, you can deploy your strapi application on that instance. For monitoring this node.js server, we have used the pm2 open-source monitoring tool.



The screenshot displays the AWS Management Console interface. On the left, a navigation menu includes sections like 'INSTANCES', 'IMAGES', 'ELASTIC BLOCK STORE', and 'NETWORK & SECURITY'. The main content area shows a table of EC2 instances. One instance, 'Testapp-env' (ID: i-06affe936803bcff5), is highlighted. Below the table, the 'Description' tab is active, showing detailed configuration for the instance. The instance is in a 'running' state, using the 't2.micro' instance type in the 'us-east-1a' availability zone. It has a public IP of 34.196.82.225 and is associated with the 'aws-elasticbeanstalk-ec2-role' IAM role. The console also shows details for the VPC, subnets, and network interfaces.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
Testapp-env	i-06affe936803bcff5	t2.micro	us-east-1a	running	2/2 checks ...	None	ec2-34-196-82-225.compute-1.amazonaws.com
KubernetesS...	i-0e4fe2dabf0fcc51	t2.micro	us-east-1b	stopped		None	

Instance: i-06affe936803bcff5 (Testapp-env)		Elastic IP: 34.196.82.225
Description	Status Checks	Monitoring
Instance ID	i-06affe936803bcff5	Public DNS (IPv4)
Instance state	running	ec2-34-196-82-225.compute-1.amazonaws.com
Instance type	t2.micro	IPv4 Public IP
Finding	Opt-in to AWS Compute Optimizer for recommendations. Learn more	34.196.82.225
Private DNS	ip-172-31-18-52.ec2.internal	IPv6 IPs
Private IPs	172.31.18.52	-
Secondary private IPs		Elastic IPs
VPC ID	vpc-365cd74c	34.196.82.225*
Subnet ID	subnet-6f454b25	Availability zone
Network interfaces	eth0	us-east-1a
IAM role	aws-elasticbeanstalk-ec2-role	Security groups
Key pair name	-	aws-elasticbeanstalk-ec2-role
Owner	341260180987	view inbound rules, view outbound rules
Launch time	March 25, 2020 at 12:33:47 PM	No scheduled events
		Scheduled events
		AMI ID
		aws-elasticbeanstalk-ec2-role
		2018.03.20.x86_64-nodesjs-hvm-202002260014 (ami-060e9c3b992173dad)
		Platform details
		-
		Usage operation
		-
		Source/dest. check
		True
		T2/T3 Unlimited
		Disabled
		EBS-optimized
		False
		Root device type
		ebs

Figure 10: Snapshot AWS EC2 instance status with other details.

9 Summary

During my tenure as an intern, I got an opportunity to learn and implement various new technologies like AWS Quicksight, Cognito, Node.js, Strapi and MongoDB. I was able to successfully design and implement an architecture of a new product from scratch.

In the 4FO, we followed an agile development process along with weekly scrum meetings and Atlassian task board, which was very helpful for tracking my own progress. We did face many challenges along the way of 'Xperiance.Cloud' version 1 development phase. However, we resolved them as a result of excellent, collaborative team work. The experience certainly did improve my problem solving and analytical skills. At the end of my tenure, we successfully delivered version 1 of a 'Xperiance.Cloud' project.

10 Declaration

I hereby declare that, all passages quoted in this document from other people's work have been specifically acknowledged by clear cross-references to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

References

- [1] *4FriendsOnly.com Internet Technologies 4FO*
<https://www.4fo.de/en/>
- [2] *Scalable data management for future Hardware (spp 2037)*
<https://www.tu-ilmenau.de/de/dbis/staff/kai-uwe-sattler/>
- [3] *AWS services count*
https://en.wikipedia.org/wiki/Amazon_Web_Services

Image References

- [4] *Amazon Web Services (AWS) in cloud solution market*
<https://www.statista.com/chart/19134/cloud-vendor-use/>