Google Summer of Code

PROPOSAL

LITMUS CHAOS

Rewrite litmus portal authentication server and add third-party integration of Gmail and Github.

Mentors: Karthik Satchitanand , Raj Babu Das

By

Ujjwal Agarwal

CLOUD NATIVE
COMPUTING FOUNDATION

Litmus

Index of Content

Personal Details

Name: Ujjwal Agarwal

Email Address: agarwalujjwal012@gmail.com

Github username: specter25

Other contact methods:

Linkedin : https://www.linkedin.com/in/agarwalujjwal012/
Kubernetes Slack : Ujjwal Agarwal
Skype Id: https://join.skype.com/invite/nb3kujV912Kg

College: Thapar Institute of Engineering and Technology , Patiala , India

Current Programme : Bachelors of Technology in Computer Science And Engineering.

Graduation Year : 2023

Country of residence: India

Timezone: Asia/Kolkata(+0530)

Primary Language: English

1. Project Synopsis

LitmusChaos is a Kubernetes native chaos engineering framework that helps SREs & developers find weaknesses in their deployments, with the chaos intent being defined via custom resources.Currently, authentication-server of the litmus-portal has a lot of unnecessary code complexity and uses some outdated packages like **mgo** as mentioned in this issue . Also, it has been observed that authentication-server takes more memory than graphql-server .

The **Primary goals** of this project would be to :

- Move the authentication server into a separate repository .
- Rewrite/Refactor the authentication server to make it simple, light-weight and modular .
- Use the official mongodb-driver rather than using mgo which is an outdated package .

The **Secondary goals** of this project would be to :

- Add Google Oauth2.0 integration .
- Add Github Oauth2.0 integration .
- Add reset existing password route .

2. Project Description

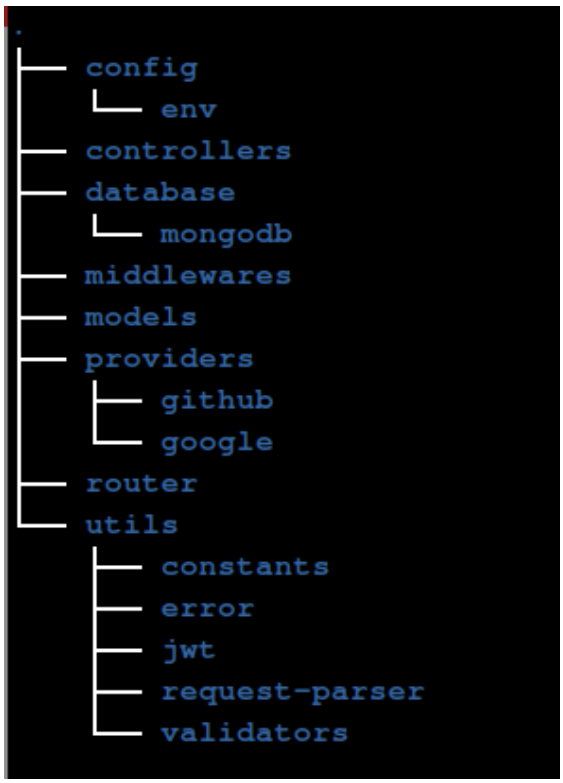
This project would primarily comprised of the following enhancements that are stated below -

2.1. Directory Structure

The following modules will be used in the project :

- **config :** This module will handle the environment configuration files
- **database :** This module will contain the functions pertaining to database connections and database querying .
- **models:** This module will define all the mongodb model schemas .

- **controllers:** This module will define the route handlers that will handle various requests .
- **router :** This module will contain the route setup
- **middlewares** : This module defines the middlewares that will be used with the routers , such as authentication middleware .
- **providers:** This module defines the google and github oauth providers and their respective utility functions .
- **Utils:** This module will have various utility subpackages
    - Error-handler : This package is the error handler for the entire rest API
    - Request-parser : Generic package to parse request params and query strings .
    - Validators : The package contains the custom validator functions for the struct fields in the mongodb bson schemas.
    - Jwt : The package contains utility functions for generating and validating jwt tokens .
    - Constants : The package contains all the constants used in the entire server .



2.2. Database Setup

The project will use the official mongodb driver for connecting and querying the database .

2.2.1. Database Connection

The server will be connected to the database as soon as the environment is loaded . The connection will be made using the sync.Once utility so that all the goroutines use the same database instance and the server connected to the database just once .
Parameters required to connect to the database are :

path : The url of the database to connect to
database : The name of the database

```
db:       You, 2 weeks ago • controller setup pending , serve
  path: "mongodb://localhost:27017"
  database: "mongo-person-production"
```

2.2.2. Database Models

The models module will contain the schema's for the models to be stored in the database , by default the following 2 models will be created :

User model → This model represents a individual user

```
// Person represents the details of single person
//
// swagger:model
You, seconds ago | 1 author (You)
type Person struct {
    Id       primitive.ObjectID          `json:"_id,omitempty" bson:"_id,omitempty"`
    Name     string                      `json:"name"  bson:"name"`
    Age      int                         `json:"age" bson:"age"`
    Gender   string                      `json:"gender" bson:"gender"`
    Email    string                      `json:"email" bson:"email"`
    Password string                      `json:"-" bson:"-"`
    Oauth    map[string]OauthCredentials `json:"oauth" bson:"oauth"`
}

You, 6 hours ago | 1 author (You)
type OauthCredentials struct {
    ProviderId    int  `json:"-" bson:"-"`
    Authenticated bool `json:"Authenticated" bson:"Authenticated"`
}
```

FilterOptions model → This model is a general model to help fetch data from mongodb using the data filters and other options .

```
type FilterOptions struct {
    Filter      bson.D
    FindOptions *options.FindOptions
}
```

2.2.3. Mongo CRUD Service

The database module in the project will have predefined functions and a mongosCrudService struct which will have generalized functions to query the database . This struct will be used in server route controllers to **remove code redundancy while querying the database .** This struct will implement the following generalized functions:

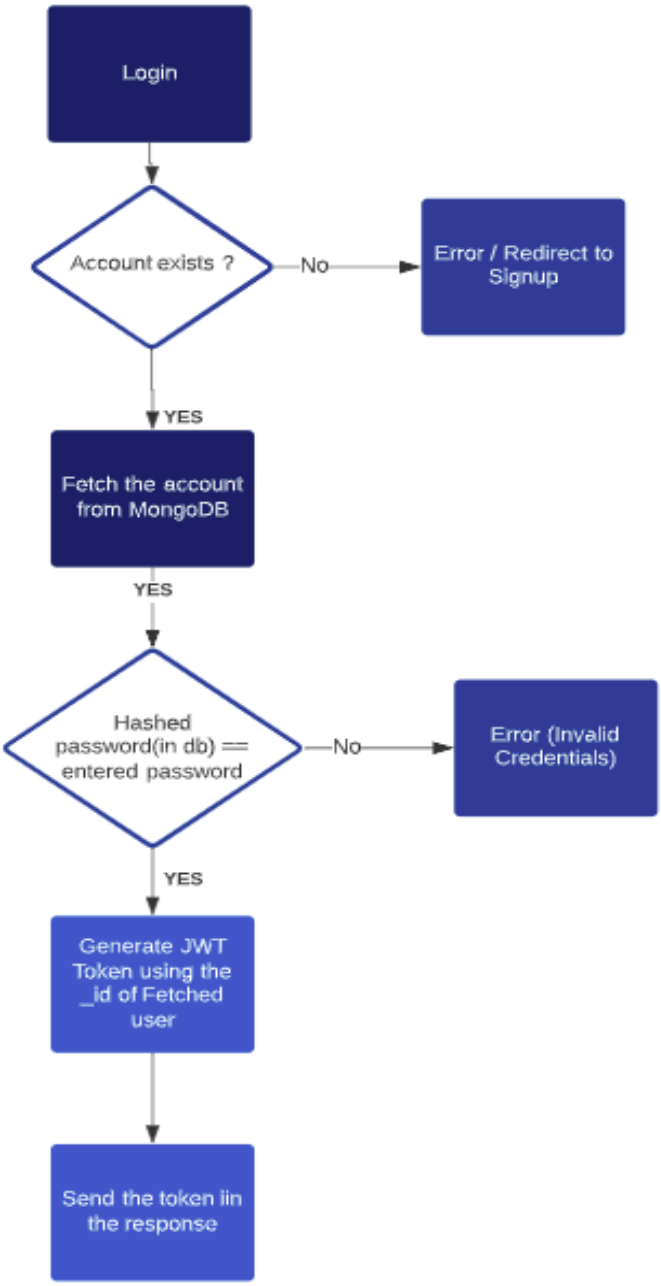
- Get all results
- Get one
- Insert one
- Find one and update
- Find one and delete

2.3. Manual Authentication workflow

The manual auth setup involves the usage of various packages including jwt for token generation and validation . The Jwt package will use the **SigningMethodHS256** signing method to generate and validate the token . The manual auth will use the bcrypt for hashing the user passwords to ensure discrepancy . Each time the password is changed the bcrypt algorithm will run and hash the password . Similarly when the user types the password during the login workflow then the bcrypt package will match the hashed password in the database and the password entered by the user . The flowing workflows will be a part of manual authentication :

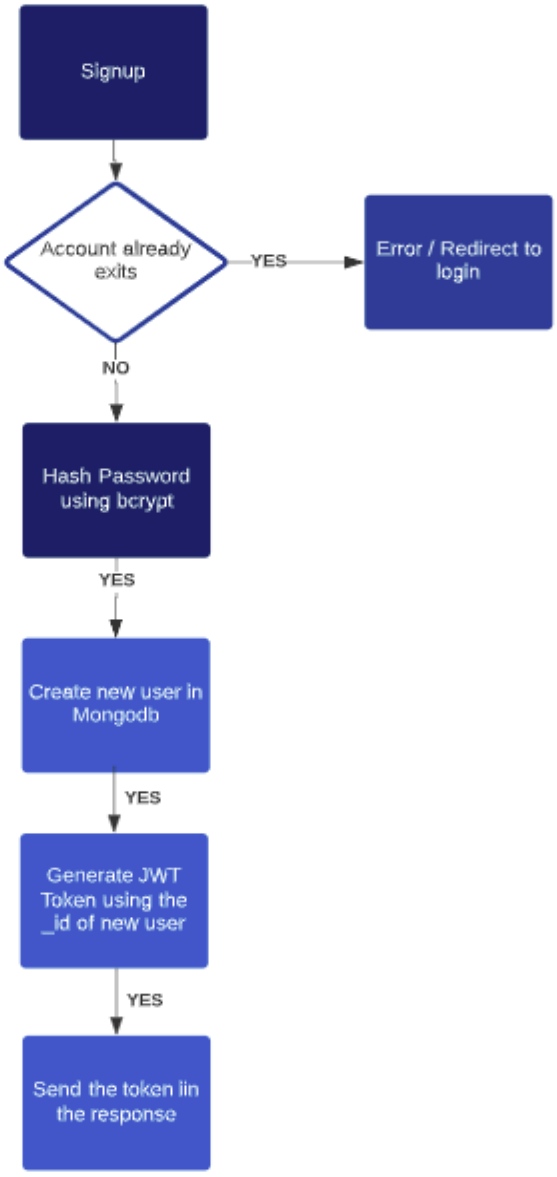
2.3.1. Login

The login workflow is as follows :



2.3.2. Signup

The signup workflow is a follows :



2.4. Google OAuth2.0 workflow

The providers package will contain the google subpackage that will define the necessary variables and functions needed for google oauth .

The following steps are involved in the configuration of the google oauth workflow:

1. The google oauth client is setup using the client ID and client Secret , generated using the google project we created on the google developer console
2. A gin session with a new cookie store is configured .
3. A random state token is generated and stored in the gin-session .
4. When the user clicks the login button and selects the account of his preference .
5. The auth middleware is fired which performs the following steps .
    - Retrieves the state and code from the query string
    - Checks if the retrieved state is equal to the state token stored in the gin-session
    - Generates a token using the code and sends it to the google client which in exchange gives the credentials of the logged in user
    - We check the user's email in mongodb , if the user exists the login workflow is implemented else the signup workflow is implemented
    - We set the user data fetched from mongodb as a variable on the gin context so that it can be retrieved to check the authentication status afterwards

2.5. Github OAuth2.0 workflow

The providers package will contain the github subpackage that will define the necessary variables and functions needed for google oauth .

The following steps are involved in the configuration of the google oauth workflow:

1. The github oauth client is set up using the client ID and client Secret , generated using the oauth project we created on Github console.
2. A gin session with a new cookie store is configured .
3. A random state token is generated and stored in the gin-session .
4. When the user clicks the login button and selects the account of his preference .
5. The auth middleware is fired which performs the following steps .
    - Retrieves the state and code from the query string

- Checks if the retrieved state is equal to the state token stored in the gin-session
- Generates a token using the code and sends it to the github client which in exchange gives the credentials of the logged in user
- We check the user's email in mongodb , if the user exists the login workflow is implemented, else the signup workflow is implemented .
- We set the user data as a variable on the gin context so that it can be retrieved to check the authentication status afterwards .

2.6. List of external packages that will be used

Gin → Gin is a web framework written in Go (Golang). It features a martini-like API with performance that is up to 40 times faster thanks to httprouter.
Mongo → Package mongo provides a MongoDB Driver API for Go.
JWT → This library supports the parsing and verification as well as the generation and signing of JWTs.
Gin-sessions → Gin middleware for session management with multi-backend support (currently cookie, Redis).
Go-oauth → oauth2 package contains a client implementation for OAuth 2.0 spec.
Crypto → This package is used to hash the password using bcrypt , before it is stored in the database .

2.7. Devops setup

2.7.1. Containerization

The project will contain docker files
dev.Dockerfile → This file will contain the development level containerization specs .
Dockerfile → file that will contain production level containerization specs .
The docker file will use the `golang:alpine` base image . The main reason to use an Alpine image is to make your resulting image as

small as possible. The base image will be smaller than 5MB. The go base image (adding go to the base alpine image) is currently 99.96MB. That's still very small. As the project's primary aim is to reduce the space used by the authentication server , using alpine image is the best solution .

The docker compose file will be basically used just for the dev environment , it will contain two services , the mongodb database and the go lang rest api .
We can also use a **mongodb Atlas instance** in the development phase to make testing easier .

2.7.2. CI/CD pipeline

The project will use the Github Actions for setting up the CI/CD pipeline . The setting up requires a **.github/workflows** directory .
The directory will have the following two workflows :

Build.yml
The file will work on xenial as the base system . The file will include a list of jobs that the pipeline has to perform .
Testing stage : These jobs will run all the tests including the pre-commit tests .
Build stage : These jobs will build the docker image .

Push.yml
The file will work on xenial as the base system . The file will include a list of jobs that the pipeline has to perform .
Push Stage : This stage will build the docker images and push them to docker hub .
Deploy stage : This stage will include jobs pertaining to deployment of the images from docker hub . The deployment platform will depend upon the organization .

Reason to prefer github actions for CI/CD pipeline are :

- One interface for both your source code repositories and your CI/CD pipelines
- Only 1 bill to pay that includes both your SCM plan and your CI/CD service (if you pay for either now that is)

- Build Matrix functionality
- Catalog of available Actions you can utilize without reinventing the wheel
- Includes macOS builds as part of free tier (macOS builds require 10 credits per build minute however compared to 1 credit per minute for linux)
- Free tier credits are renew/reset each month

We can also use Travis CI for setting up the CI/CD pipeline . Will finalize the platform during the community bonding phase while discussing the project with the mentors .

2.8. Use efficient Coding practices

- Use Sync pools : One of the highlights of Go 1.3 release was sync Pool. It is a component under the sync package to create a self-managed temporary retrieval object pool.We want to keep the garbage collector overhead as little as possible. Frequent allocation and recycling of memory will cause a heavy burden to the garbage collector. sync.Pool can cache objects that are not used temporarily and use them directly (without reallocation) when they are needed next time. This can potentially reduce the garbage collector workload and improve the performance.

- Use efficient Go coding practices : To write Go, it's important to understand its properties and idioms. It's also important to know the established conventions for programming in Go, such as naming, formatting, program construction, and so on, so that programs you write will be easy for other Go programmers to understand. The go official docs provides some guidelines in this blog to write structurally effective go programs .

- Swagger Documentation : The API documentation is an important part of the development phase . Evolution of your API's functionality is inevitable, but the headache of maintaining API docs doesn't have to be. Swagger tools take the hard work out of generating and

maintaining your API docs, ensuring your documentation stays up-to-date as your API evolves.

- Version Control good practices :The contributors to repositories should know that a well-crafted Git commit message is the best way to communicate context about a change to fellow developers (and indeed to their future selves). A difference will tell you what changed, but only the commit message can properly tell you why.Fortunately, there are well-established conventions as to what makes an idiomatic Git commit message. Indeed, many of them are assumed in the way certain Git commands function. This blog clearly mentions the semantics that need to be followed when writing good commit messages

3. Schedule of Derivables

May 17 - June 7 Community Bonding Period	• Discuss with the mentor the exact procedure and workflow. • I have mentioned the path and way for the solutions in the proposal but they are flexible and can change as per the common view with the mentor. • Along with this, I will learn the necessary skills which will help me to make the solutions easier and accurate.
June 8th - June 14th	• Create the Directory Structure • Set Up a new Gin server • Configure the dev and production environments • Connect the server with the database and implement the mongoService interface
June 16th - June 22rd	Complete Setup of Manual Auth workflow which includes : • Login route • Signup route • User model setup • Setup of error handler • Setup of Request parser • Jwt utility package
June 24th - 30th June	Setup manual auth auxiliary routes and auth middlewares which includes :

	● update user route ● forget password route ● delete user route ● auth middlewares ● validators
July 2 - 17 July (12th July - 16th July Round one Evaluations)	● Swagger documentation of the manual auth api ● Docker setup of the api , docker compose setup ● If time allows setup of CI/CD pipeline using Travis CI or Github Actions .
July 18th - 25th July	● Setup Google oauth workflow
July 26th - August 2nd	● Setup github oauth workflow . ● Sync the oauth and manual auth modules
August 3rd - August 15th	● Swagger docs completion ● Docker setup of google oauth and github oauth
August 16 - 23, 2021 (Final Round of Evaluation)	● API Testing at the development and production environments .

Apart from this, I will be maintaining a 3-part blog, writing about my work and the entire workflow and evaluations.

4. Availability Schedule

During the entire GSoC period, I am available to work for **35-40 hours per week**. I don't have any pre-planned vacations or plans during the break and will be available full time.

Due to **COVID19 shutdown** in my country, our college schedule is likely to be shifted ahead by a **minimum of 2 weeks**, thus will be consuming my summer break's first few days. But that **won't be an issue** to work schedule. I will make sure that compensation of that time is done during the following weekends of the weeks indulged in my college timings (if required).

Also, I have **kept a buffer** of few days between the tasks in my timeline, which will give me time to complete pending/delayed tasks (if any)

5. Future Developments

For me, the open-source contribution is not just restricted to Google Summer of Code. I would love to keep working on the developments and enhancements. Here are some of the ideas that I would love to implement after the entire GSoC timeline -

5.1. Using Redis to Store JWT Metadata

Yes we can login a user a generate a JWT, but there is a lot wrong with the above implementation:

1. The JWT can only be invalidated when it expires. A major limitation to this is: a user can login, then decide to logout immediately, but the user's JWT remains valid until the expiration time is reached.
2. The JWT might be hijacked and used by a hacker without the user doing anything about it until the token expires.
3. The user will need to re-login after the token expires, thereby leading to a poor user experience.

We can address the problems stated above in two ways:

- Using a persistence storage layer to store JWT metadata. This will enable us to invalidate a JWT the very second the user logs out, thereby improving security.
- Using the concept of a refresh token to generate a new access token, in the event that the access token expired, thereby improving the user experience.

One of the solutions we proffered above is saving a JWT metadata in a persistence layer. This can be done in any persistence layer of choice, but Redis is highly recommended. Since the JWTs we generate have expiry time, Redis has a feature that automatically deletes data whose expiration time has reached. Redis can also handle a lot of writes and can scale horizontally.

Since Redis is a key-value storage, its keys need to be unique, to achieve this, we will use uuid as the key and use the user id as the value.

5.2. Pre-commit hooks setup

The pre-commit hook is run first, before you even type in a commit message. It's used to inspect the snapshot that's about to be committed, to see if you've forgotten something, to make sure tests run, or to examine whatever you need to inspect in the code. Exiting non-zero from this hook aborts the commit, although you can bypass it with git commit --no-verify. You can do things like check for code style (run lint or something equivalent), check for trailing whitespace (the default hook does exactly this), or check for appropriate documentation on new methods. It's a useful tool to have in the development environment . We can run a series of tests to run such as :

- Whether the document is formatted correctly according to the specifications laid down by the specifications .
- Is there any unused dependency in the code which hasn't been removed due to incompetent local setup of the developer
- There are no redundant new lines introduced in the code
- The document follows the coding and semantic conventions laid down by the effective-go article .

Pre-commit hooks are generally used for structural and semantic checks in the code . Once the code is pushed the CI/CD file will include a check to test whether all the pre-commit tests have passed or not .

6. Contributions to Litmus Chaos

Organization	Pull Request /Issue code	Description
Litmus-Chaos	#2632(MERGED)	Add goimports CI check and format existing imports .
Litmus-Chaos	#2639 (MERGED)	Fix failing CI tests , remove go imports check from push.yml
Litmus-Chaos	#2649 (ISSUE)	Add golang-ci lint check in build pipeline and remove backend and frontend checks from push pipeline
Litmus-Chaos	#2640	Involved in the conversations of this pull request

Litmus-Chaos	#2661 (OPEN) To be tested after release on 15th April	Added golangci-lint to the pipeline and fixed all the lint errors in the code .

7. Contributions to Open Source Community

Organization	Pull Request /Issue code	Description
SPDX	#60 (MERGED)	Fixed the issue #49
SPDX	#64 (OPEN)	Fixed the issue #26
Casbin	#9 (ISSUE)	Created an issue for documentation fix .
Casbin	#10 (MERGED)	Fixed the issue #9

8. Proof of Concept

Here is an overview of the code structure in this git repository of mine - https://github.com/specter25/go-auth-server
I have implemented the :

- Manual Authentication
- Google Oauth2.0

I need a little more time and I will complete the swagger documentation and the devops setup in this repository as well .
This is a private repository , I have given it access to Raj Babu Das sir who is the project mentor .

9. Academic Experience

I am a 2nd year engineering undergraduate(Bachelor of Technology in Computer Science and Engineering) at Thapar Institute of Engineering and Technology, Patiala. I have always found computer science as a very intriguing subject and

always felt the urge to learn new stuff in this field . How are websites built ? What are web servers ? Why does a server crash ? How to prevent a server from crashing ? were some of the questions that drove me to learning more and more in this field and I eventually took it as my major subject in my B.Tech. I program a lot, I try my level best to stay focussed on my coursework as a result i have successfully maintained a **C.G.P.A. of 10.0** and have **over 2000 contributions on Github over the last year with a longest standing streak of 154 days .**

A link to my complete academic resume is here

10. Open Source Projects

10.1. Slack-bots (Project Started by Developers Student club Thapar)(Contributed in this project as a project Co-lead)

These are general purpose bots that scrape articles from medium, dev. to, telegram channels and then render them in various slack, discord channels and also in google home, alexa . The telegram is powered by NLP for summarizing the content and also filtering it. The bots will run automatically as set by the administrator.

11. Personal Projects

11.1. Azure Devops- Terraform- Kubernetes CI/CD Pipeline (project Head , MERN stack Devops)

I implemented a CI/CD pipeline on a MERN stack application . Used docker for containerisation . Then wrote Kubernetes config files and created different pods and then finally set up a CI/CD pipeline using Azure devops . I used Terraform to implement infrastructure as code and automated the creation of AWS S3 buckets , EC2 instances and I am users .

11.2. Devconnector

A MERN stack application , which is the professional social media handle of a local group of users. It has the functionality of authentication , blogs , posts , likes , comments etc . It has the functionality of image upload . It's currently hosted on heroku.

11.3. Go-lang Blockchain

A custom blockchain implementation using go-lang . A low-level computing language . I optimised the process of signing the unsigned transactions by building a custom Merkle tree which reduced the mining time of the blockchain by 3 sec. Presently I am trying to implement the GHOST protocol of Ethereum but am currently unsuccessful. I even plan to implement a Proof of Authority consensus algorithm in the future .

11.4. Organ-donation blockchain

To build an ecosystem where the donors , recipients and others can interact with each other directly on secure lines to make the entire procedure more fluid and secure . This project is a Decentralized application based on the ethereum blockchain .

12. Work/Internship Experience

Software Engineering Intern at GrowthGear Solutions LLP (28 December 2020 – 28 March 2021)

- Completed the RTC real time video streaming and video calling module . Both client side and server side . (using Web RTC connections)
- Completed real time messaging (public and private chats , polls and Q&A (using Web sockets) module and integrated it with video calling and streaming module .
- Made the analytics and role based access server side .

13. Why Me

I really value this opportunity and getting it really means a lot to me . I think that I can guarantee that I will never be careless and lethargic throughout the course of three months and put in 100% efforts towards it's completion . I think this project is apt for me. It involves a lot of constant learning about new stacks and technology. I am consistent with my efforts and never leave any task in between, no matter how hard or new it is for me. I make sure that I keep researching the issue at hand and then work to make it happen. I think this skill of mine proves to be very helpful in order to achieve feature addition and bug solving for this particular project. This project will help me to equip essential real-world software skills as well as soft skills which I believe, will help me a lot in the longer run.

14. Why Cloud Native Computing Foundation ?

Cloud Native Computing Foundation(CNCF) is one of the most prominent advocates of open-source development. I really appreciate CNCF's commitment to the community and the work it has been doing to empower the people worldwide to shape the modern Internet - an Internet that is open and accessible to all. I truly align with CNCF's belief of promoting acts of human collaboration across an open platform that contributes to individual growth and the collective future. So, for me, there is a special connection with CNCF. Also, after contributing to **Litmus Chaos**, I genuinely liked the coding standards, especially the backend and testing part. I want to work in such an environment. I feel that it will be a good learning experience for me.