Group 05

muly
Software Architecture Document

Version <1.3>

Revision History

Date	Version	Description	Author
08/12/2022	<1.1>	Draw class and write report	**muly**
09/12/2022	<1.2>	Draw class and write report	**muly**
23/12/2022	<1.3>	Reconstruct folder tree and update DB schema	

Table of Contents

Software Architecture Document

1. Introduction

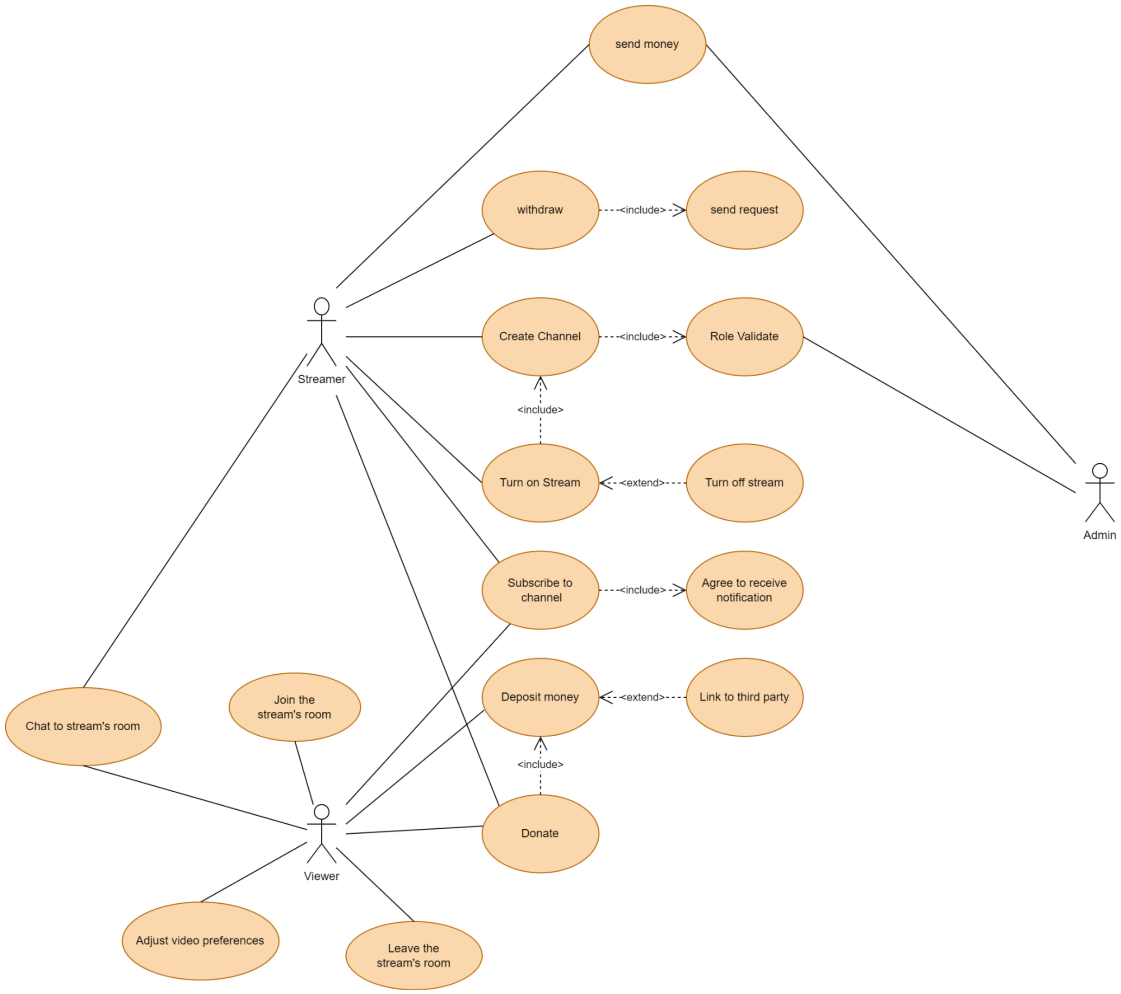
This Software Architecture Document provides an overview of the architecture of the **muly** website.

The document is intended to provide a comprehensive architectural overview of the system with different architectural views from different aspects of the system. Moreover, it conveys the significant decisions which have been made on the project.

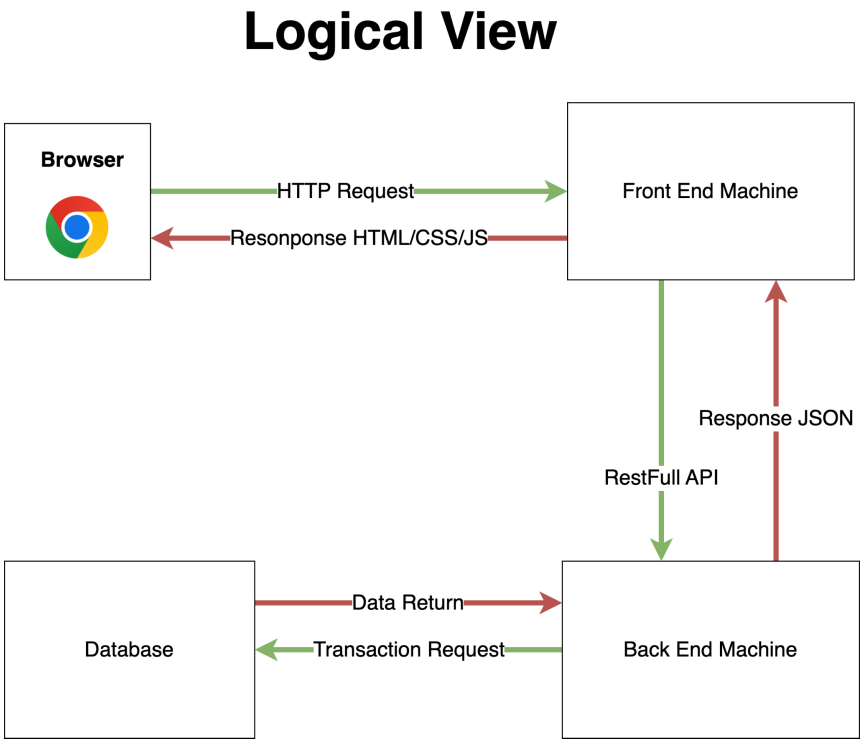
2. Architectural Goals and Constraints

- Programming Language: Go (Back-end), JavaScript(Front-end).
- The application should run all current modern browsers.
- The application should support a minimum of 50 users at the same time watching one streaming session.
- Delay of streaming should not be higher than 5 minutes.

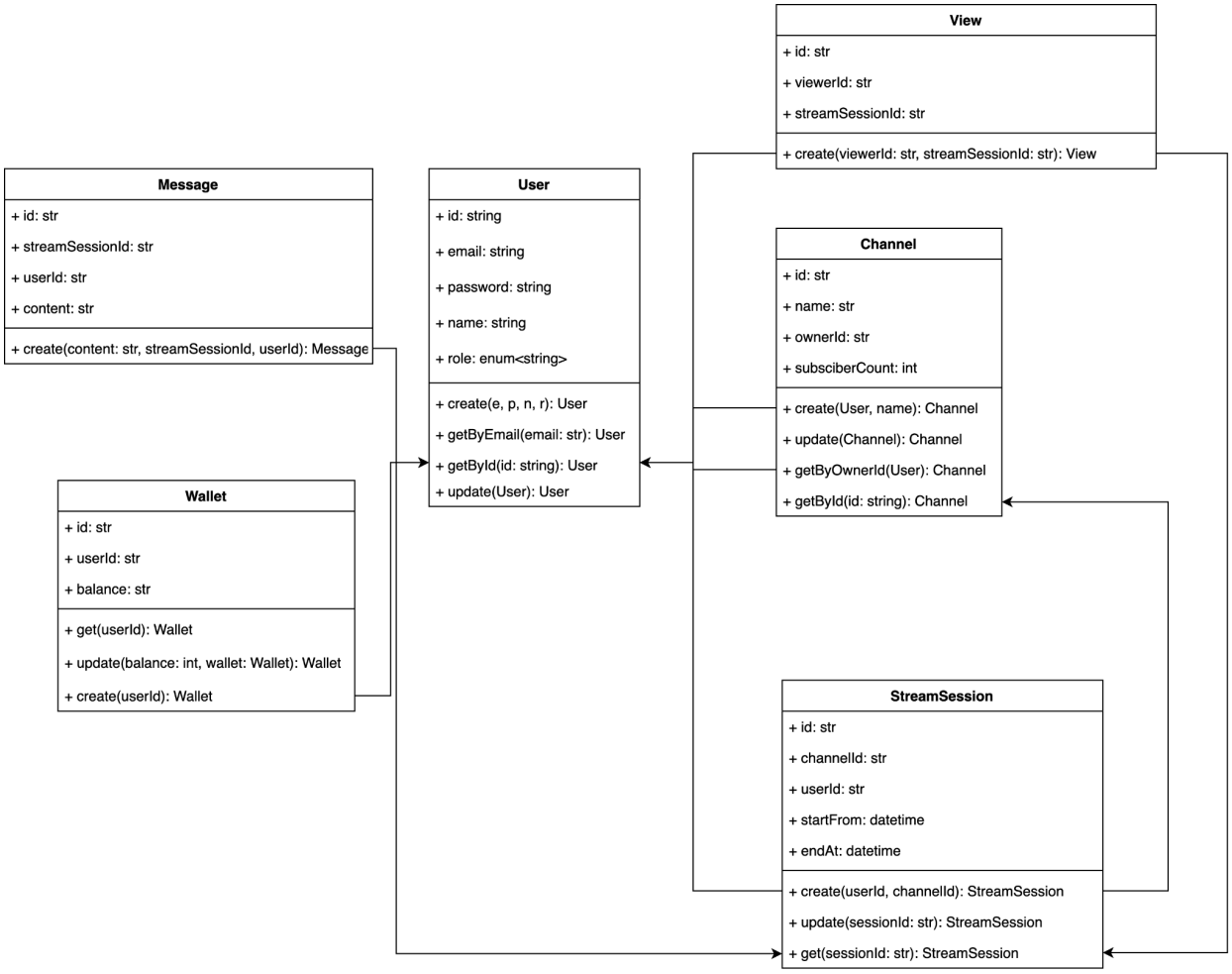
3. Use-Case Model



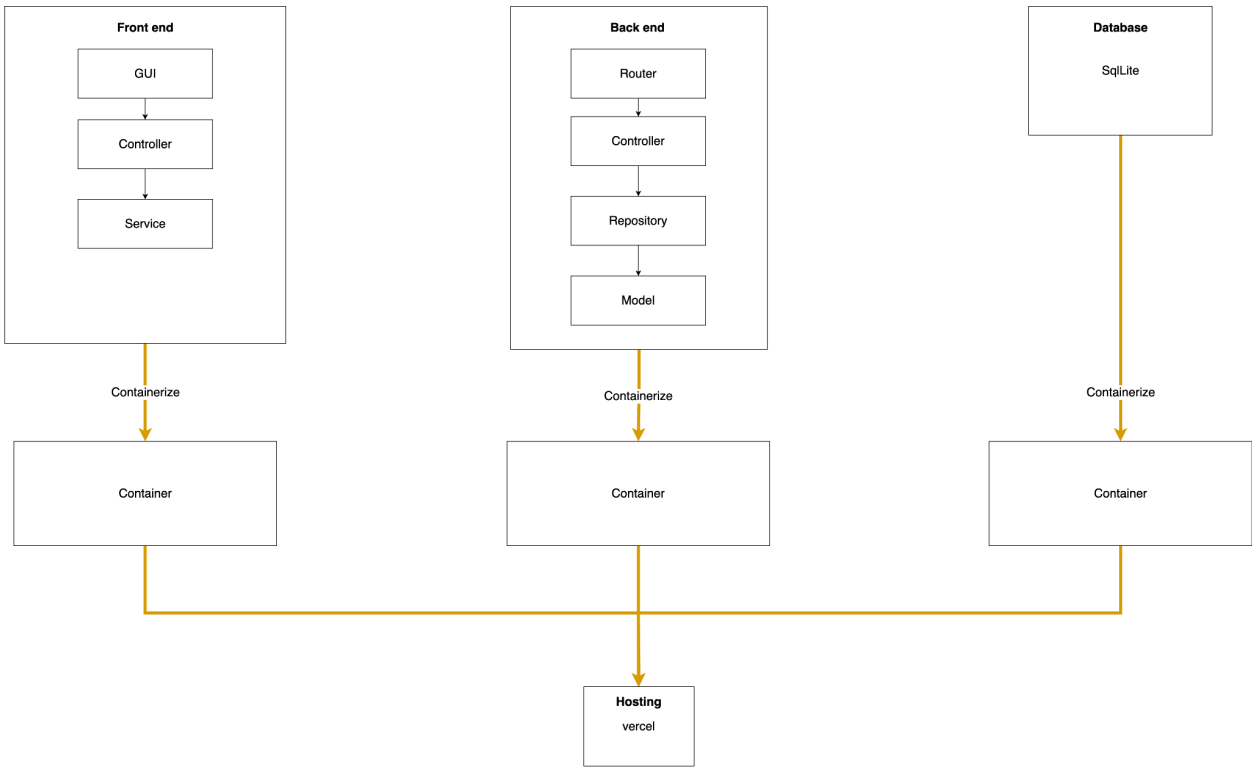
4. Logical View



4.1 Component:



5. Deployment View



The figure above shows the entire deployment view of **muly**:

- We containerize three sources front-end, back-end, and database into the container by docker.
- Deploy three sources into Vercel service with port 8080 for front-end and 3000 for back-end.
- Client sends a request to port 8080 to get HTML/CSS/JS and the front-end code will call port 3000 to execute.server-side stuff.

Node:

- **Frontend**: We deploy frontend in port 8080 to send CSS/HTML/JS to the browser client whenever they request, frontend will call to port 3000 of backend to request logic processing and data interacting
- **Backend**: Backend node will be deployed in port 3000 of EC2 service in AWS web services. Backend will call the SQLite database to get and update data
- **Database**: We deploy databases in the same EC2 instance at port default 5432

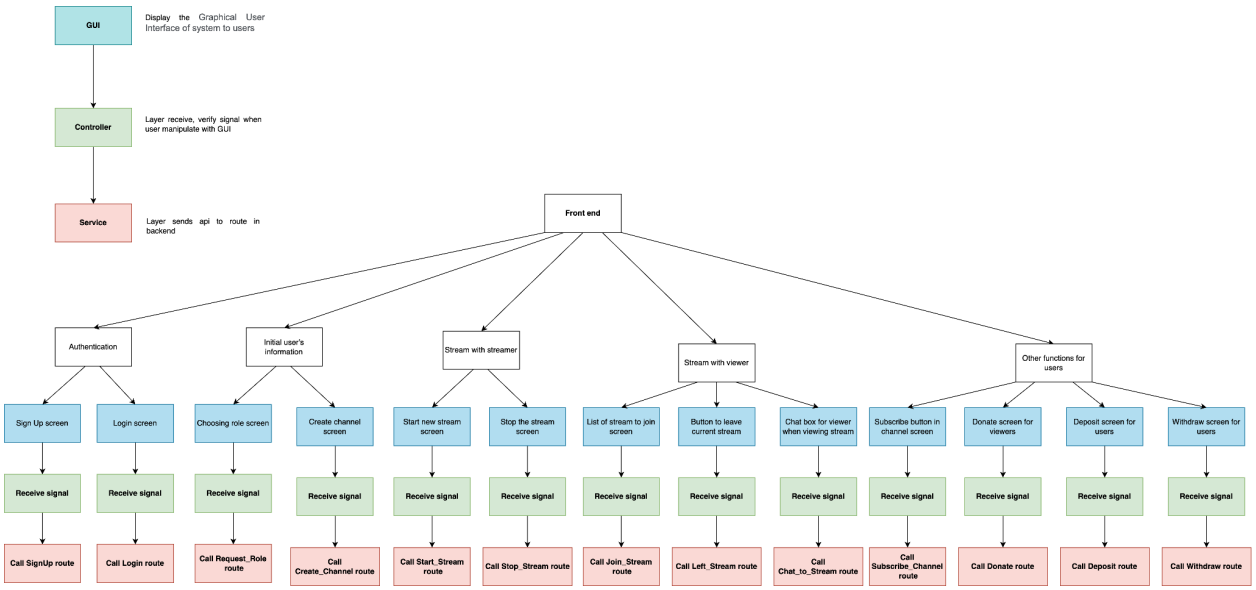
6. Implementation View

```
├── cmd
│   └── api
│       └── main.go
├── data
│   ├── config.yml
│   └── database.db
├── docs
```

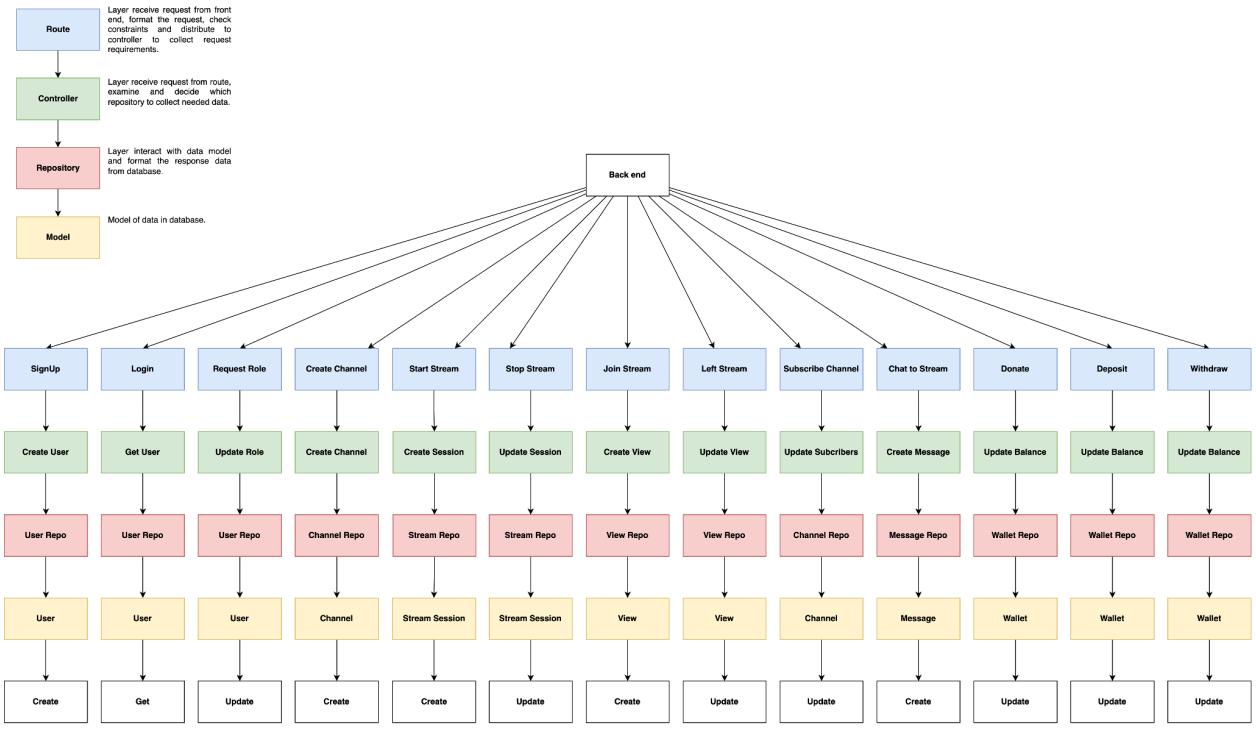
```
│   ├── docs.go
│   ├── swagger.json
│   └── swagger.yaml
└── internal
    └── api
        ├── controllers
        │   ├── channel-controller.go
        │   ├── message-controller.go
        │   ├── stream-session-controller.go
        │   ├── users-controller.go
        │   ├── view-controller.go
        │   └── wallet-controller.go
        ├── middlewares
        │   ├── auth.go
        │   ├── cors.go
        │   └── handlers.go
        ├── repositories
        │   ├── channel-repository.go
        │   ├── message-repository.go
        │   ├── stream-session-repository.go
        │   ├── user-repository.go
        │   ├── view-repository.go
        │   └── wallet-repository.go
        └── router
            └── router.go
```

7. Implementation View

7.1 Front end:


GUI
Display the Graphical User Interface of system to users
Controller
Layer receive, verify signal when user manipulate with GUI
Service
Layer sends api to route in backend
Front end
Authentication
Initial user's information
Stream with streamer
Stream with viewer
Other functions for users
Sign Up screen
Login screen
Choosing role screen
Create channel screen
Start new stream screen
Stop the stream screen
List of stream to join screen
Button to leave current stream
Chat box for viewer when viewing stream
Subscribe button in channel screen
Donate screen for viewers
Deposit screen for users
Withdraw screen for users
Receive signal
Call SignUp route
Call Login route
Call Request_Role route
Call Create_Channel route
Call Start_Stream route
Call Stop_Stream route
Call Join_Stream route
Call Left_Stream route
Call Chat_to_Stream route
Call Subscribe_Channel route
Call Donate route
Call Deposit route
Call Withdraw route

7.2 Back end:


Route
Layer receive request from front end, format the request, check constraints and distribute to controller to collect request requirements.
Controller
Layer receive request from route, examine and decide which repository to collect needed data.
Repository
Layer interact with data model and format the response data from database.
Model
Model of data in database.
Back end
SignUp
Login
Request Role
Create Channel
Start Stream
Stop Stream
Join Stream
Left Stream
Subscribe Channel
Chat to Stream
Donate
Deposit
Withdraw
Create User
Get User
Update Role
Create Channel
Create Session
Update Session
Create View
Update View
Update Subcribers
Create Message
Update Balance
Update Balance
Update Balance
User Repo
Channel Repo
Stream Repo
View Repo
Message Repo
Wallet Repo
User
Channel
Stream Session
View
Message
Wallet
Create
Get
Update