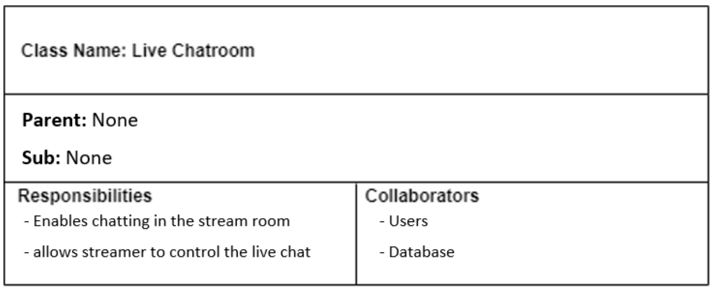
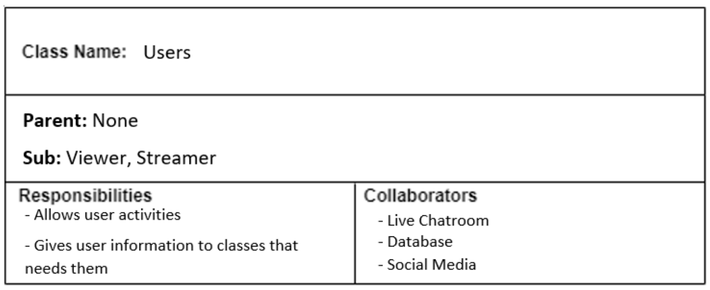
CRCs

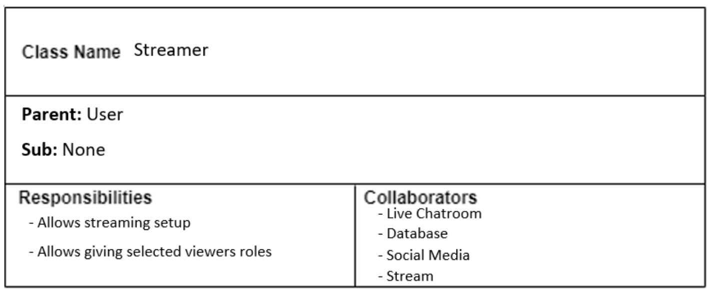
Backend:



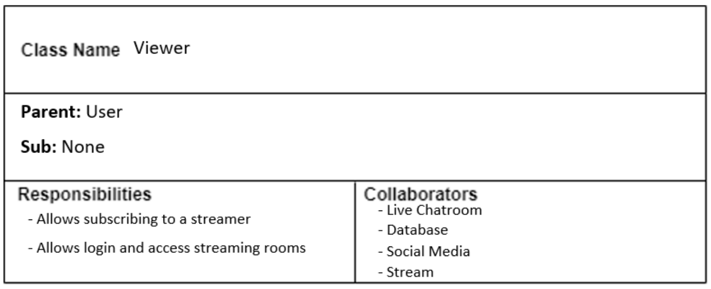
Class Name: Live Chatroom	
Parent: None **Sub:** None	
Responsibilities - Enables chatting in the stream room - allows streamer to control the live chat	Collaborators - Users - Database



Class Name: Users	
Parent: None **Sub:** Viewer, Streamer	
Responsibilities - Allows user activities - Gives user information to classes that needs them	Collaborators - Live Chatroom - Database - Social Media



Class Name Streamer	
Parent: User **Sub:** None	
Responsibilities - Allows streaming setup - Allows giving selected viewers roles	Collaborators - Live Chatroom - Database - Social Media - Stream



Class Name Viewer	
Parent: User **Sub:** None	
Responsibilities - Allows subscribing to a streamer - Allows login and access streaming rooms	Collaborators - Live Chatroom - Database - Social Media - Stream

Class Name: post.js	
Parent: Social Media **Sub:** None	
Responsibilities: - create posts - delete posts	**Collaborators:** - post schema - user schema

- update posts - like/dislike posts	

Live Streams:

app.js	
• Runs the entire server • routes different files together • runs the media server	• media server • route folder

media_server.js	none none
• Starts an RTMP server for the video streaming functionality to work.	• app.js

passport.js	
• Login and register strategies go here	• Schema

deafault.js	
• Configuration for the RTMP server • ffmpeg.exe installation location also goes here	

thumbnails.js	
• create a live thumbnail for each live stream	• default.js • helpers.js

Schema.js	
• exporting the UserSchema.js file	• UserSchema

UserSchema.js	
• Define a user schema • checking for validation of password • hashing out recorded passwords	

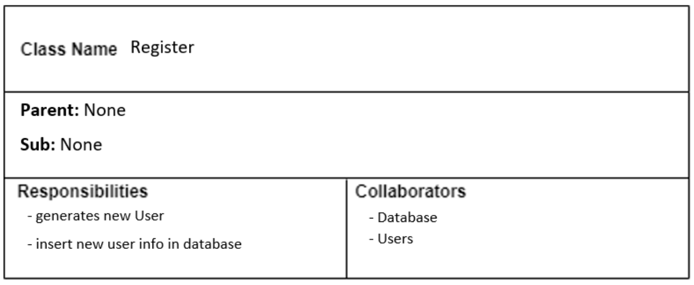
helpers.js	
• Helper class for generating stream thumbnails	• default.js

login.js	
• routing the login call	• app.js • passport.js

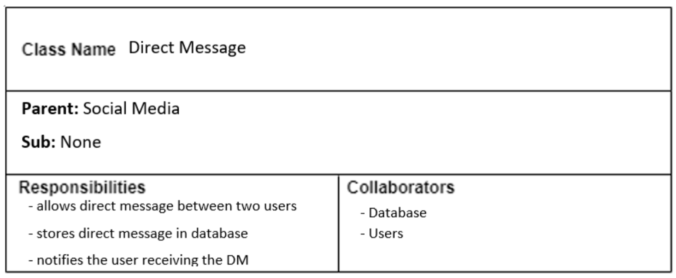
Settings.js	
• routing the stetting call • generating and assigning users a streaming key	• Schema

Class Name Database	
Parent: None **Sub:** None	
Responsibilities - stores User data - stores Social Media data - allows data access from other classes (e.g. Live Chatroom, Login)	Collaborators - Social Media - Users - Login - Register - Live Chatroom

Class Name Login	
Parent: None **Sub:** None	
Responsibilities - gives permission to certain user activities - links a client to a User	Collaborators - Database - Users - Stream



Class Name Register	
Parent: None **Sub:** None	
Responsibilities - generates new User - insert new user info in database	Collaborators - Database - Users



Class Name Direct Message	
Parent: Social Media **Sub:** None	
Responsibilities - allows direct message between two users - stores direct message in database - notifies the user receiving the DM	Collaborators - Database - Users

Frontend:

Class Name: index.js	
Parent: None **Sub:** None	
Responsibilities: - Render all frontend pages	**Collaborators:** - App.js

Class Name: App.js	
Parent: None **Sub:** None	
Responsibilities: - A hub for all pages	**Collaborators:** - Home.js - NavBar.js - Login.js - Stream.js - SignUp.js - Profile.js - EditProfile.js

Class Name: NavBar.js

Parent: None	
Sub: None	
Responsibilities:	**Collaborators:**
- create the navigation bar for the whole site	- App.js

Class Name: Home.js	
Parent: None	
Sub: None	
Responsibilities:	**Collaborators:**
- Develop the home page	- App.js

Class Name: Login.js	
Parent: None	
Sub: None	
Responsibilities:	**Collaborators:**
- create the login page	- App.js

Class Name: SignIn.js	
Parent: None	
Sub: None	
Responsibilities:	**Collaborators:**
- create sign in page	- App.js

Class Name: Stream.js	
Parent: None	
Sub: None	
Responsibilities:	**Collaborators:**
- create stream page	- App.js

Class Name: Profile.js	
Parent: None	
Sub: Progile_Post.js	

Responsibilities:	Collaborators:
- create profile page	- App.js - Profile_Post.js

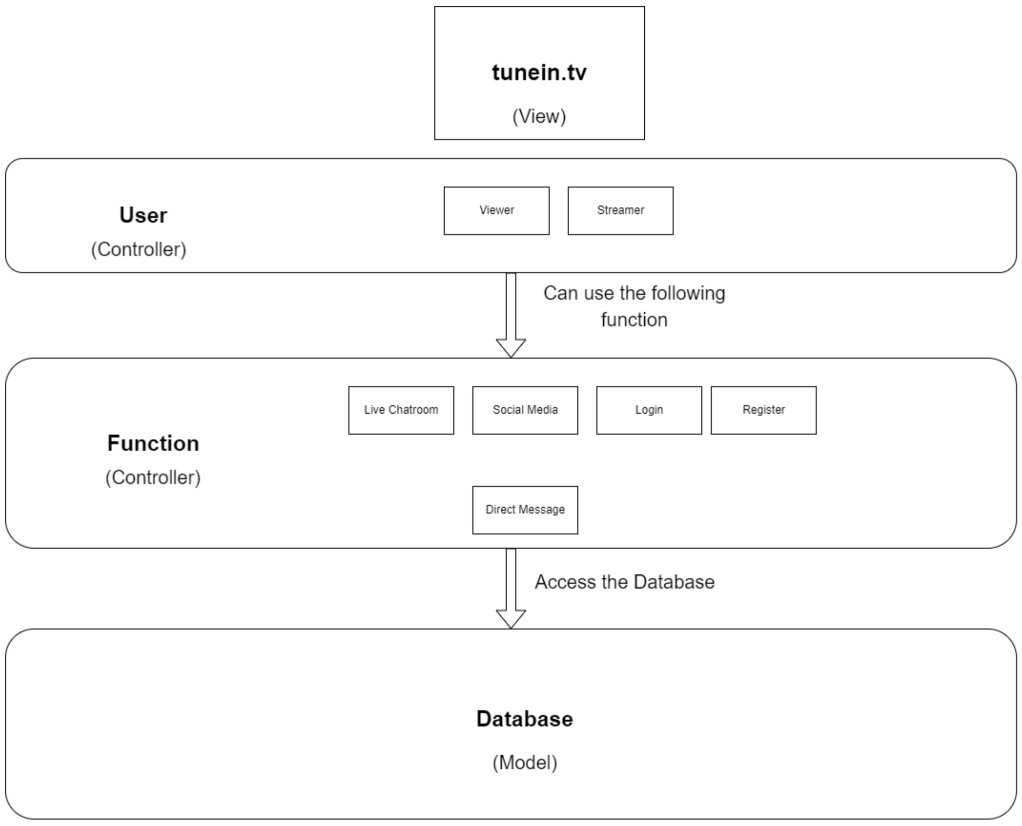
Class Name: Profile_Post.js	
Parent: Profile.js **Sub:** None	
Responsibilities: - create post	**Collaborators:** - App.js - Profile.js

Class Name: EditProfile.js	
Parent: None **Sub:** None	
Responsibilities: - create edit profile page	**Collaborators:** - App.js - Profile.js

Description of System Interaction with the Environment

The system is expected to run on a PC with a widely accepted OS like Windows, macOS and Linux. We do not expect full support for mobile devices. The machines should have an active network connection for new browsing. The client machine should have a modern web browser such as Chrome or Firefox. The server machine should have python 3.10.2 and Node.js runtime installed, and have the environment set up according to the instructions in README.md included in the project.

Architecture Diagram



User

- Chat -> Live Chatroom
- Send -> Direct Message -> User
- Post, Reply, Comment -> Social Media

Viewer

- Subscribe -> Streamer
- Watch -> Stream

Streamer

- Setup -> Stream
- Assign roles -> Viewer

Database

- Store -> Users
- Store -> Social Media

- Give user Information -> Live Chatroom
- Give user Information -> Login

Register

- info put in -> Database
- Create -> User

System Decomposition

Model: database

View: frontend website / html pages

Controller:

The system is separated into two parts, one focusing on stream and the other focusing on social media. Both parts have their data stored in the database and are connected through Users. Users are created by Register, after which the user data are stored in the database. When login the user passes authentication, after which their clients are linked to the user using browser cookie.

In the streaming part, depending on which subclass of Users, they can either set up a stream in their room or join a room that is active. They can also chat in that room's live chatroom. Streamers have many powers in their room, including assigning special roles exclusive to the room to viewers.

In the social media part, users can make posts, do activities related to posts, or send a direct message to another user.

Viewers can subscribe to Streamers, and they can follow other users to receive notifications on their social media updates.

Error Handling:

Invalid inputs when logging in result in authentication failure, in which case the client would not be able to log in and have access to user activities.

A stream error causes the streamer to disconnect from their stream room, where the stream would end. The room will not be shut down.

We expect the database to be stable without error since all database insert/query are done in the controller.