Technical Document

- ChatRoom.controller.js
  - Dependencies
    - ChatRoom model: a Mongoose model that represents the chat room schema.
    - express-fileupload: A middleware for handling file uploads in Express.js.
    - Express: A web application framework for Node.js.

  - Functions
    - `createChatRoom`
      - This function creates a new chat room by taking the request object (req) and response object (res) as input. The function first creates a new instance of the ChatRoom model by destructuring the request body and adding the owner field to it, which is set to the _id of the current user.
      - If the save operation is successful, the function updates the chatRooms and chatRoomCreatedCount fields of the current user and saves the changes to the database. Finally, it sends a response with the status code 201 and the newly created chat room object. If there is an error during the save operation, the function sends a response with the status code 400 and the error object.
    - `readAllChatRoom`
      - This function retrieves all chat rooms created by the current user by taking the request object (req) and response object (res) as input. The function finds all chat rooms in the database where the owner field is equal to the _id of the current user.
      - If the find operation is successful, the function sends a response with the retrieved chat rooms as an array. If there is an error during the find operation, the function sends a response with the status code 500 and the error object.
    - `getAllFollowedChatRoom`
      - This function retrieves all chat rooms followed by the current user by taking the request object (req) and response object (res) as input. The function finds all chat rooms in the database where the developer field is equal to the _id of the current user.
      - If the find operation is successful, the function sends a response with the retrieved chat rooms as an array. If there is an error during the find operation, the function sends a response with the status code 500 and the error object.
    - `readChatRoomById`
      - This function retrieves a chat room by its ID by taking the request object (req) and response object (res) as input. The function finds a chat room in the database where the _id field is equal to the id field in the request body.
      - If the find operation is successful and the chat room exists, the function sends a response with the retrieved chat room object. If the chat room does not exist, the function sends a response with the status code 404. If there is an error during the find operation, the function sends a response with the status code 500.
    - `Exports`
      - This module exports the createChatRoom, readAllChatRoom, and readChatRoomById functions to be used in other modules.

- Developer.controller.js
  - Dependencies
    - Developer module: A Mongoose Schema representing the developer model.
    - UploadDocuments module: A Mongoose Schema representing the uploaded documents model.
    - multer module: A middleware for handling multipart/form-data requests, primarily used for file uploading.
    - ClassRoom module: A Mongoose Schema representing the chat room model.
    - Document module: A Mongoose Schema representing the document model.

  - Functions
    - `Register`
      - The Register function is used to register a new developer. It creates a new instance of the Developer model, saves it to the database using await developer.save(), generates an authentication token using await developer.generateAuthToken(), sets the token in a cookie using res.cookie('access_token', token), and sends the response as an object containing the developer and a redirect URL to the home page.
    - `Login`
      - The Login function is used to log in an existing developer. It calls the findByCredentials method of the Developer model to find the developer with the given email and password. If found, it generates an authentication token using the await developer.generateAuthToken(), sets the token in a cookie using res.cookie('access_token', token), and sends the response as an object containing the developer and a redirect URL to the home page.
    - `Home Page`
      - The showHomePage function is used to render the home page of the logged-in developer. It gets the current developer using req.developer, passes it to the res.render method along with the name of the EJS view to be rendered.
    - `Profile`
      - The Profile function is used to render the profile page of the logged-in developer. It gets the current developer using req.developer, passes it to the res.render method along with the name of the EJS view to be rendered.
    - `Logout`
      - The Logout function is used to log out the current developer by removing the authentication token from the tokens array of the Developer model. It gets the current developer using req.developer, filters out the token to be removed using req.token, and saves the updated developer to the database using await req.developer.save(). Finally, it renders the login page.
    - `UpdateProfile`
      - The UpdateProfile function is used to update the profile of the current developer. It checks if the update operation is valid by checking if the keys in the req.body object are allowed. If the update operation is valid, it updates the current developer using req.developer[update] = req.body[update] and saves the updated developer to the database using await req.developer.save(). Finally, it sends the response as an object containing the developer and a redirect URL to the profile page.
    - `DeleteProfile:`
      - The DeleteProfile function is used to delete the profile of the current developer. It removes the developer from the database using await req.developer.remove() and removes the developer's ID from the following and follower objects of other developers using await Developer.updateMany(). Finally, it sends the response as the deleted

- Like.controller.js
  - Dependencies
    - Like: This module is a schema for storing information in MongoDB.
    - UploadDocuments: This module is a schema for storing document information in MongoDB.

  - Functions
    - `likeDocument`
      - It searches for an existing like in the database using the findOne method of the Like schema.
      - If there is no existing like, it creates a new like using the new keyword and the Like schema.
      - It then searches for the document to be liked using the findOne method of the UploadDocuments schema.
      - It updates the document's likeCount and likedBy fields to reflect the new like.
      - It saves the updated document and the new like to the database using the save method.
      - It updates the document's likeCount and likedBy fields to remove the existing like.
      - It removes the existing like from the database using the remove method.
      - It sends a response back to the client indicating that the like has been removed.

- Manager.controller.js
  - Dependencies
    - require('multer'): This is used for uploading files, such as documents, images, etc.
    - require('../models/manager'): This is a custom module that defines the schema for the "Manager" model in MongoDB.
    - require('../models/documents'): This is a custom module that defines the schema for the "Documents" model in MongoDB.
    - require('../models/chatRoom'): This is a custom module that defines the schema for the "ClassRoom" model in MongoDB.
    - require('path'): This is a built-in Node.js module that provides utilities for working with file and directory paths.

  - Functions
    - `Register`
      - This route is used to register a new manager. It takes the manager's details from the request body, creates a new "Manager" document in MongoDB, and returns the created document along with a JSON Web Token (JWT).
    - `Login`
      - This route is used to log in a manager. It takes the manager's email and password from the request body, checks if the email and password match with an existing "Manager" document in MongoDB, and returns the document along with a JWT.
    - `Logout`
      - This route is used to log out a manager. It takes a JWT from the request headers, removes the JWT from the "tokens" array of the corresponding "Manager" document in MongoDB, and redirects to the login page.
    - `UpdateProfile`

- This route is used to update the manager's profile. It takes the updated profile details from the request body, checks if the update operation is valid, updates the corresponding "Manager" document in MongoDB, and returns the updated document.

uploadDocument

- This route is used to upload a document to a chat room. It takes the chat room name, document name, description, and the file to upload from the request body and file, creates a new "UploadDocuments" document in MongoDB, and returns the created document.

getDocument

- This route is used to get all the documents uploaded by the manager. It fetches all the "UploadDocuments" documents in MongoDB that are uploaded by the manager and returns them.

showAllMembers

- This route is used to get all the members of a chat room. It takes the chat room name from the request body, fetches the corresponding "ClassRoom" document in MongoDB, and returns the "students" array of the document.

searchChatRoom

- This route is used to search for a chat room. It takes the chat room name from the request body, fetches the corresponding "ClassRoom" document in MongoDB, and returns the document.

loadSearch

- This route is used to load the search page for managers. It renders the "ManagerSearch" view.

showHomePage

- This route is used to load the home page for managers. It renders the "ManagerHome" view.

Profile

- This route is used to load the profile page for managers. It renders the "ManagerProfile" view.

loadHome

- This route is used to load the documents uploaded by the manager. It fetches all the "Documents" documents in MongoDB that are uploaded to the chat rooms owned by the manager and returns them.

- Reply.controller.js
  - Dependencies
    - Like: This module is a schema for storing information in MongoDB.
    - UploadDocuments: This module is a schema for storing document information in MongoDB.
    - Reply: This module is a schema for storing information in MongoDB

  - Functions
    - `createReply`
      - This function is responsible for creating a new reply to a post. It takes in the request and response objects as parameters, creates a new instance of the Reply model, increments the reply count of the corresponding UploadDocuments model, and saves the reply object to the database.
    - `deleteReply`
      - This function is responsible for creating a new reply to a post. It takes in the request and response objects as parameters, creates a new instance of the Reply model, increments the reply count of the corresponding UploadDocuments model, and saves the reply object to the database.
    - `likeReply`
      - This function is responsible for adding or removing a like on a reply. It takes in the request and response objects as parameters, finds the like object using the reply's id and the user's id, updates the like count of the corresponding Reply model, saves the like object to the database if it doesn't exist, removes the like object from the database if it already exists, and returns a success message.
    - `getReply`
      - This function is responsible for retrieving all replies for a given post. It takes in the request and response objects as parameters, finds all reply objects using the post's id, and returns the list of replies.

- Middleware, auth.js
  - Introduction
    - This technical document describes a Node.js module that provides two middleware functions for authentication: developerAuth and managerAuth. The module relies on the jsonwebtoken library to handle JSON Web Tokens (JWTs) for authentication. The module also requires access to two MongoDB models: Developer and Manager, which are expected to have an array of tokens in their respective schemas.

  - Installation
    - To use this module, you need to have Node.js and MongoDB installed. You also need to install the following dependencies:
    - npm install jsonwebtoken
    - Then, you can require the module in your Node.js application:
    - const auth = require('./path/to/auth/module');

  - Usage
    - This module exports two middleware functions: developerAuth and managerAuth. Both functions take three parameters: req, res, and next. The req parameter represents the incoming request, the res

parameter represents the outgoing response, and the next parameter is a callback function that is called when the middleware has finished its work.

- ### developerAuth
  - The developerAuth middleware function checks if the request comes from an authenticated developer. It expects a valid JWT to be present in the access_token cookie of the request. The function decodes the JWT and tries to find a matching developer in the Developer MongoDB model. If a matching developer is found, the function attaches the JWT and the developer object to the req object and calls the next function. Otherwise, it sends a 401 response with an error message.

- ### managerAuth
  - The managerAuth middleware function is similar to developerAuth, but it checks if the request comes from an authenticated manager instead. It expects a valid JWT to be present in the access_token cookie of the request. The function decodes the JWT and tries to find a matching manager in the Manager MongoDB model. If a matching manager is found, the function attaches the JWT and the manager object to the req object and calls the next function. Otherwise, it sends a 401 response with an error message.

MODELS

ChatRoom.js

- `owner`: A reference to the Manager who owns the chat room, which is represented as a MongoDB ObjectId. This field is required and has a ref property that points to the Manager model.
- `name`: The name of the chat room, which is a string. This field is required and is trimmed to remove any leading or trailing whitespace.
- `description`: A description of the chat room, which is a string. This field is required and is trimmed to remove any leading or trailing whitespace.
- `developer`: An object that stores the details of the developers who are currently in the chat room. This field is optional and has a default value of an empty object.
- `developerCount`: The number of developers who are currently in the chat room, which is an integer. This field is optional and has a default value of 0.
- `timestamps`: An option that adds createdAt and updatedAt timestamps to the chat room document.

Developer.js

The Developer schema defines the structure of the Developer model. It has the following fields:

- `name`: a string representing the name of the developer.
- `email`: a string representing the email of the developer. It is required and should be unique. It is also validated using the validator package.
- `age`: a number representing the age of the developer. It has a default value of 0 and is validated to be positive.
- `password`: a string representing the password of the developer. It is required and has a minimum length of 5. It is also validated to not contain the string "password."
- `chatRoomCount`: a number representing the count of chat rooms the developer owns. It has a default value of 0.

- `userType`: a string representing the type of user. In this case, it is "developer" and is required.
- `following`: a mixed type representing the developers that this developer follows. It has a default value of an empty object.
- `tokens`: an array of tokens representing the JSON web tokens generated for the developer.
- `avatar`: a buffer representing the image avatar of the developer.

The schema also has two virtual fields:

- `tasks`: a virtual field that references the task model and is used to populate the tasks field of the developer.
- `documentData`: a virtual field that references the DocumentData model and is used to populate the documentData field of the developer.

Documents.js

- `developerId`: The ID of the developer who uploaded the document. This field is a reference to the developer model.
- `managerId`: The ID of the manager who owns the document. This field is a reference to the manager model.
- `chatRoomId`: The ID of the chat room associated with the document. This field is a reference to the ClassRoom model.
- `name`: The name of the document. This field is required.
- `fileUpload`: The uploaded file. This field is not currently specified and can be updated as per the requirements.
- `description`: A description of the document. This field is required.
- `likeCount`: The number of likes the document has received. This field defaults to 0.
- `replyCount`: The number of replies the document has received. This field defaults to 0.
- `likedBy`: An array of user IDs who have liked the

Likes.js

- `UserId`: This property is of type mongoose. Schema.Types.ObjectId is required. It refers to the user who liked the post or replied. This property is a reference to the developer model.
- `postId`: This property is of type mongoose. Schema.Types.ObjectId refers to the post that was liked. This property is a reference to the UploadDocuments model.
- `postReplyId`: This property is of type mongoose. Schema.Types.ObjectId refers to the reply that was liked. This property is a reference to the reply model.
- `timestamps`: This option creates a createdAt and updatedAt field for the model.

Manager.js

- `name`: A String field that is required and trimmed.
- `email`: A string field that is required, unique, and trimmed. The value must be a valid email address.
- `age`: A number field with a default value of 0. The value must be a positive number.

- `password`: a string field that is required and trimmed. The value must have a minimum length of 7 characters and cannot contain the word "password" in any case.
- `userType`: A string field that is required.
- `chatRoomCreatedCount`: A number field with a default value of 0.
- `chatRoomFollowedCount`: A number field with a default value of 0.
- `chatRooms`: A mixed field with a default value of an empty object. This field stores an object containing chat room details created by the manager.
- `tokens`: An array of objects, each with a token property that is a string. This field stores auth tokens generated for the manager.
- `avatar`: A buffer field that stores binary data for an avatar image.

Reply.js

- `UserId`: represents the ID of the developer who created the reply. It is a required field and is of type mongoose.Schema.Types.ObjectId. The reference model for this field is the developer model.
- `content`: represents the content of the reply. It is of type String and has a maximum length of 1000 characters.
- `postId`: represents the ID of the post to which the reply belongs. It is a required field and is of type mongoose.Schema.Types.ObjectId. The reference model for this field is the UploadDocuments model.
- `likeCount`: represents the number of likes the reply has received. It is of type Number and has a default value of 0.
- `userName`: represents the name of the user who created the reply. It is of type string.