# Create back-end project

Initialize: npm init -y

In the initial setting below, “main”: descript the entry point to the project, changed it to “app.js” file (alternative main.js or index.js).

Text

Description automatically generated with medium confidence

## Install typescript

npm i typescript

Potential bug:

A screenshot of a computer

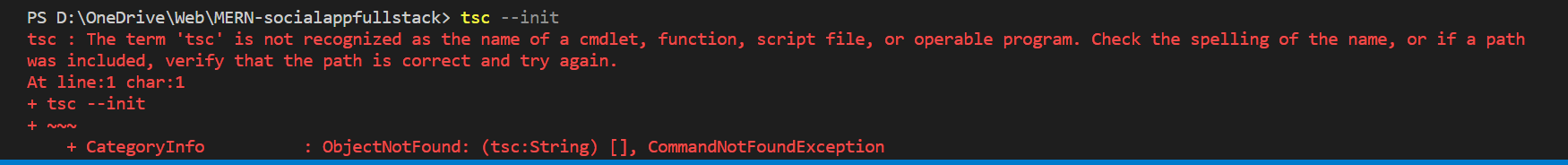
Description automatically generated

Fix: try turn off and on the wifi connection of your pc. <https://github.com/nodejs/node/issues/41056>

## Create typescript configuration file

tsc –init

Potential bug:



cause and fixed: typescript is not installed globally,

do npm install typescript@latest -g

<https://bobbyhadz.com/blog/typescript-tsc-is-not-recognized-as-internal-or-external>

## TSConfig setting

## Folder structure set up

src root folder

src\features to store all the features code

src\shared shared code

src\shared\global

src\shared\services all mongo db related services

src\shared\services\db code to connect to the mongo db

src\shared\sockets all thing related to socket io

src\shared\workers for redis message queue

src\app.ts entry point for this project, file name must be the same as "main": "app.js" in package.json file

src\config.ts configuration for this project

src\routes.ts define all the routes

src\setupDatabase.ts set up the mongo db

src\setupServer.ts set up the server

## Set up Server class

### Install express

npm i express

add following import:

import {Application, json, urlencoded, Response, Request, NextFunction } from 'express';

## Set up standard middleware

install

npm I cors helmet hpp cookie-session compression express-async-errors http-status-codes

helmet : a security library

store minimum data into cookie

need express-async-errors to cache error coming from async method

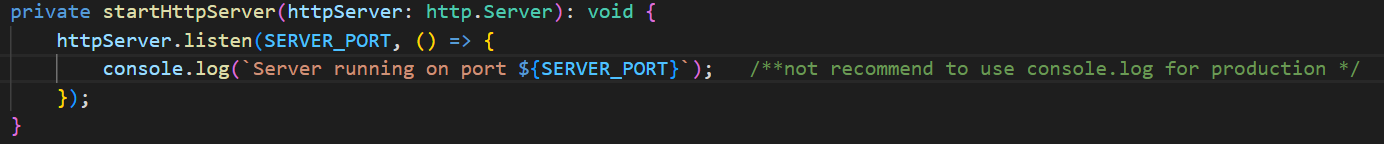
Text

Description automatically generated

## Set up http server

set server\_port constant to (any number after 1000 to 64000). except

Note: in typescript, use ` ${variable}` if want to include variable



set startServer to async, any async method will return Promise

Text

Description automatically generated

update app.ts



### install nodemon

npm i -g nodemon

add script to package.json, note: if we specify “dev”. later on if we call npm run dev, it will run the script at the end

A screenshot of a computer

Description automatically generated with medium confidence

### install ts-node

so that we can decode ts

npm i ts-node -D

### install tsconfig-paths

so all folder can be relative and still read by the typescript

<https://www.npmjs.com/package/tsconfig-paths>

npm install --save-dev tsconfig-paths

Text

Description automatically generated

## set up database

install mongoose:

npm i mongoose

Text

Description automatically generated

## set up environment config file

install dotenv

npm i dotenv

create a .env file in the root folder,

save your configurations constant

Text

Description automatically generated

## set up socket io

<https://www.npmjs.com/package/@socket.io/redis-adapter>

npm i @socket.io/redis-adapter redis socket.io

note: run command with @ in CMD, running in powershell will give error

<https://stackoverflow.com/questions/46107955/open-cmd-in-the-visual-studio-code-terminal>

add following imports to setupServer.ts

import { Server } from 'socket.io';

import { createClient } from 'redis';

import { createAdapter } from '@socket.io/redis-adapter';

add this constant to .env file : REDIS\_HOST = 'redis://localhost:6379'

update the method:

A screenshot of a computer

Description automatically generated

update startServer()

Text

Description automatically generated

## Set up route.ts file

A screenshot of a computer

Description automatically generated with medium confidence

update setupServer.ts

Text

Description automatically generated

## Set up Global error handler

create a file name error-handler under folder globals\helpers

define some custom error class as below  
Text

Description automatically generated

## logger set up

install bunyan

npm install bunyan

<https://www.npmjs.com/package/bunyan>

Graphical user interface, text

Description automatically generated

import Logger from ‘bunyan’;

add following const at the start of each ts file

/\*\* indicate the log is coming from the database \*/

const log: Logger = config.createLogger('setupDatabase');

then call the

log.error() or log.info to log the

## ESLint set up

install ESLint, editorconfig, prettier for vs code

create a file name .editorconfig under root folder

copy content from the website (<https://editorconfig.org/>) to editorconfig file.

npm i -D eslint eslint-config-prettier prettier @typescript-eslint/eslint-plugin @typescript-eslint/parser

create a file name .prettierrc.json, copy following config into the file

{

  "trailingComma": "none",

  "tabWidth": 2,

  "semi": true,

  "singleQuote": true,

  "bracketSpacing": true,

  "printWidth": 140

}

create a file name .eslintrc.json, copy the following config into the file

{

  "root": true,

  "parser": "@typescript-eslint/parser",

  "plugins": [

    "@typescript-eslint"

  ],

  "extends": [

    "eslint:recommended",

    "plugin:@typescript-eslint/recommended",

    "prettier"

  ],

  "parserOptions":  {

    "ecmaVersion":  2020,  // Allows for the parsing of modern ECMAScript features

    "sourceType":  "module"  // Allows for the use of imports

  },

  "rules": {

    "semi": [2, "always"],

    "space-before-function-paren": [0, {"anonymous": "always", "named": "always"}],

    "camelcase": 0,

    "no-return-assign": 0,

    "quotes": ["error", "single"],

    "@typescript-eslint/no-non-null-assertion": "off",

    "@typescript-eslint/no-namespace": "off",

    "@typescript-eslint/explicit-module-boundary-types": "off"

  }

}

add script to package.json file

## change absolute import

npm i ttypescript typescript-transform-paths

add following to tsconfig.json file

    "plugins": [

      { "transform": "typescript-transform-paths" },

      { "transform": "typescript-transform-paths", "afterDeclarations": true },

    ],                                                  /\* Declaration to use transform-path\*/

    "paths": {

      "@global/\*": ["src/features/shared/globals/\*"],

      "@service/\*": ["src/features/shared/services/\*"],

      "@socket/\*": ["src/features/shared/sockets/\*"],

      "@worker/\*": ["src/features/shared/workers/\*"],

      "@root/\*": ["src/\*"],   /\*make sure root directory is the last \*/

    }

# Note on DataFlow

The data will go to both Redis and Queue (to be stored in MongoDB).

Redis is like a memory cache (which respond faster) – hence good for storing data frequently used and expected immediate response.

A picture containing diagram

Description automatically generated

## Redis command and data type used:

* Strings
* Lists
  + LPUSH – prepends (add from beginning) one or multiple values to a list
  + LRANGE – gets a range of elements from a list
  + LINDEX – Gets an element from a list by its index
  + LLEN – gets the length of a list
  + LREM – removes elements from a list
  + LSET – sets the value of an element in a list by its index
  + RPUSH – appends (add from end) one or multiple values to a list
* Sets
* Hashes (field(key) : value)
  + HSET – sets the string value of a hash field
  + HGET – gets the value of a hash field stored at the specified key
  + HGETALL – gets all the fields and values stored at the specified key
  + HINCRBY – increments the number stored at the field in the hash stored at key by increment
  + HMGET – returns the values associated with the specified fields in the hash stored at key
* sorted sets
  + ZADD – adds one or more members to a sorted set
  + ZCARD – gets the number of members in a sorted set
  + ZCOUNT – returns the number of elements in the sorted set with a score between min and max
  + ZRANGE – returns a range of members in a sorted set
  + ZREM – removes one or more members from a sorted set

## Cloudinary set up

1. Create Cloudinary account and sign in
2. Create env variable for Cloud Name, API Key and API Secret
3. install cloudinary : npm i cloudinary
4. Create the function in config.ts and add it into loadConfig() function in app.ts
5. /\*\*
6. \* Set up cloudinary
7. \*/
8. public cloudinaryConfig(): void {
9. cloudinary.v2.config({
10. cloud\_name: this.CLOUD\_NAME,
11. api\_key: this.CLOUD\_API\_KEY,
12. api\_secret: this.CLOUD\_API\_SECRET
13. });
14. }

5. Create a cloudinary-upload.ts under global/helpers folder and use the following code:

Note: ? indicate the argument is optional,

{} for an object,

() for the call back

import cloudinary, { UploadApiResponse, UploadApiErrorResponse }from 'cloudinary';

export function uploads(

  file: string,

  public\_id?: string,

  overwrite?: boolean,

  invalidate?: boolean

): Promise<UploadApiResponse | UploadApiErrorResponse | undefined> {

  return new Promise((resolve) =>{

    cloudinary.v2.uploader.upload(

      file,

      {

        public\_id,

        overwrite,

        invalidate

      },

      (error: UploadApiErrorResponse | undefined, result: UploadApiResponse | undefined) => {

        if (error) resolve(error);

        resolve(result);

      }

    );

  });

}

# Authentication Features

## Validation scheme set up

install joi module

npm i joi

Create the signup, signin and password scheme

## Joi validator decorator

To perform the validation.

create a folder decorators under globals, create a file joi-validation.decorators.ts

place the following code:

import { JoinRequestValidationError } from '@global/helpers/error-handler';

import { Request } from 'express';

import { ObjectSchema } from 'joi';

type IJoiDecorator = (target:any, key: string, descriptor: PropertyDescriptor) => void;

export function joiValidation(schema: ObjectSchema): IJoiDecorator {

  return (\_tager: any, \_key: string, descriptor: PropertyDescriptor) => {

    const originalMethod = descriptor.value;

    descriptor.value = async function (...args: any[]){

      const req: Request = args[0];

      //Either use validateAsync (need call inside try catch) or validate

      const { error } = await Promise.resolve(schema.validate(req.body));

      if (error?.details) {

        throw new JoinRequestValidationError(error.details[0].message);

      }

      return originalMethod.apply(this, args);

    };

    return descriptor;

  };

}

## Auth interface and model schema:

require bcryptjs module for hashing

## Signup user method

install jsonwebtoken (used to store the token in web session) and lodash

npm i jsonwebtoken lodash

Helper methods

Text

Description automatically generated

Create auth.service.ts

Text

Description automatically generated

create signup.ts

Text

Description automatically generated

## Set up auth routes

Create authRoutes.ts

Text

Description automatically generated

update route.ts

/\*\*Define all the routes \*/

import { authRoutes } from '@auth/routes/authRoutes';

import { Application } from 'express';

const BASE\_PATH = '/api/v1';

export default (app: Application) => {

  const routes = () => {

    app.use(BASE\_PATH, authRoutes.routes());

  };

  routes();

};

Build test case for the auth routes:

Install REST-API on VSCODE,

Create a new file under new folder endpoints,

Text

Description automatically generated

run the app in dev environment, and click the send request

## Set up Redis Connection

Create new file base.cache.ts with following content under shared/services/redis folder

import { createClient } from 'redis';

import Logger from 'bunyan';

import { config } from '@root/config';

export type RedisClient = ReturnType<typeof createClient>;

export abstract class BaseCache {

  client: RedisClient;

  log: Logger;

  /\*\*

   \* This cacheName can tell where the error is coming from

   \* @param cacheName

   \*/

  constructor(cacheName: string) {

    this.client = createClient({ url: config.REDIS\_HOST });

    this.log = config.createLogger(cacheName);

    this.cacheError();

  }

  private cacheError(): void {

    this.client.on('error', (error: unknown) => {

      this.log.error(error);

    });

  }

}

Create another file redis.connection.ts in the same folder

import Logger from 'bunyan';

import { config } from '@root/config';

import { BaseCache } from '@service/redis/base.cache';

const log: Logger = config.createLogger('redisConnection');

class RedisConnection extends BaseCache {

  constructor() {

    super('redisConnection');

  }

  async connect(): Promise<void> {

    try {

      await this.client.connect();

      const res = await this.client.ping();

      console.log(res);

    } catch (error) {

      log.error(error);

    }

  }

}

export const redisConnection: RedisConnection = new RedisConnection();

Add the line to setupDatabase.ts

Text

Description automatically generated

## Save user to redis cache

1. Create new file user.cache.ts under shared/services/redis folder

import { ServerError } from './../../globals/helpers/error-handler';

import { IUserDocument } from '@user/interfaces/user.interface';

import { BaseCache } from '@service/redis/base.cache';

import { config } from '@root/config';

import Logger from 'bunyan';

const log: Logger = config.createLogger('userCache');

export class userCache extends BaseCache {

  constructor() {

    super('userCache');

  }

  public async saveUserToCache(key: string, userId: string, createdUser: IUserDocument): Promise<void> {

    const createdAt = new Date();

    /\*\*

     \* Destructure the properties from the createdUser

     \*/

    const {

      \_id,

      uId,

      username,

      email,

      avatarColor,

      blocked,

      blockedBy,

      postsCount,

      profilePicture,

      followersCount,

      followingCount,

      notifications,

      work,

      location,

      school,

      quote,

      bgImageId,

      bgImageVersion,

      social

    } = createdUser;

    const firstList: string[] = [

      '\_id', `${\_id}`,

      'uId', `${uId}`,

      'username', `${username}`,

      'email', `${email}`,

      'avatarColor', `${avatarColor}`,

      'createdAt', `${createdAt}`,

      'postsCount', `${postsCount}`,

    ];

    const secondList: string[] = [

      'blocked', JSON.stringify(blocked),

      'blockedBy', JSON.stringify(blockedBy),

      'profilePicture', JSON.stringify(profilePicture),

      'followersCount', JSON.stringify(followersCount),

      'followingCount', JSON.stringify(followingCount),

      'notifications', JSON.stringify(notifications),

      'social', `${social}`

    ];

    const thirdList: string[] = [

      'work', `${work}`,

      'location', `${location}`,

      'school', `${school}`,

      'quote', `${quote}`,

      'bgImageVersion', `${bgImageVersion}`,

      'bgImageId', `${bgImageId}`,

    ];

    const dataToSave: string[] = [...firstList, ...secondList, ...thirdList];

    /\*\*

     \* save the data as sortedset

     \*/

    try {

      if(!this.client.isOpen){

        await this.client.connect();

      }

      await this.client.ZADD('user', { score: parseInt(userId, 10), value:`${key}` });

      await this.client.HSET(`users: ${key}`, dataToSave);

    } catch (error) {

      log.error(error);

      throw new ServerError('Server error, try again');

    }

  }

}

1. Add a private method to signup.ts
2. /\*\*
3. \* Return the complete data we want to save in the cache
4. \* @param data
5. \* @param userObjectId
6. \* @returns
7. \*/
8. private userData(data: IAuthDocument, userObjectId: ObjectId): IUserDocument {
9. const { \_id, username, email, uId, password, avatarColor } = data;
10. return {
11. \_id: userObjectId,
12. authId: \_id,
13. uId,
14. username: Helpers.firstLetterUppercase(username),
15. email,
16. password,
17. avatarColor,
18. profilePicture: '',
19. blocked: [],
20. blockedBy: [],
21. work: '',
22. location: '',
23. school: '',
24. quote: '',
25. bgImageVersion: '',
26. bgImageId: '',
27. followersCount: 0,
28. followingCount: 0,
29. postsCount: 0,
30. notifications: {
31. messages: true,
32. reactions: true,
33. comments: true,
34. follows: true
35. },
36. social: {
37. facebook: '',
38. instagram: '',
39. twitter: '',
40. youtube: ''
41. }
42. } as unknown as IUserDocument;
43. };
44. add lines to signup.create() method
45. /\*\*
46. \* Add to redis cache
47. \*/
48. const userDataForCache: IUserDocument = SignUp.prototype.userData(authData, userObjectId);
49. userDataForCache.profilePicture = `https://res/cloudinary.com/dlf1ttson/image/upload/v${result.version}/${userObjectId}`;
50. await userCache.saveUserToCache(`${userObjectId}`, uId, userDataForCache);

## Install Redis Commander

npm i -g redis-commander

open a CMD terminal and type redis-commander,

Should see following messages

Text

Description automatically generated

Should see something like this after going in

Graphical user interface, text, application, email

Description automatically generated

## Create base message queue and worker

install bull:

npm i bull

install GUI to monitor the message queue:

npm i @bull-board/express @bull-board/ui

Step1: Create base.queue.ts under shared/services/queues with following content

import Queue, { Job } from 'bull';

import Logger from 'bunyan';

import { createBullBoard } from '@bull-board/api';

import { BullAdapter } from '@bull-board/api/bullAdapter';

import { ExpressAdapter } from '@bull-board/express';

import { config } from '@root/config';

import { IAuthJob } from '@auth/interfaces/auth.interface';

type IBaseJobData =

  | IAuthJob;

let bullAdapters: BullAdapter[] = [];

export let serverAdapter: ExpressAdapter;

export abstract class  BaseQueue {

  queue: Queue.Queue;

  log: Logger;

  constructor(queueName: string) {

    this.queue = new Queue(queueName, `${config.REDIS\_HOST}`);

    bullAdapters.push(new BullAdapter(this.queue));

    /\*\* Remove duplicates from the bullAdapters \*/

    bullAdapters = [...new Set(bullAdapters)];

    serverAdapter = new ExpressAdapter();

    serverAdapter.setBasePath('/queues');

    createBullBoard({

      queues: bullAdapters,

      serverAdapter

    });

    this.log = config.createLogger(`${queueName}Queue`);

    this.queue.on('completed', (job: Job) => {

      job.remove();

    });

    this.queue.on('global:completed', (jobId: string) => {

      this.log.info(`Job ${jobId} completed`);

    });

    this.queue.on('global:stalled', (jobId: string) => {

      this.log.info(`Job ${jobId} is stalled`);

    });

  }

  /\*\*

   \* add job to the queue

   \* @param name job name

   \* @param data data associated with the job

   \*/

  protected addJob(name: string, data: IBaseJobData): void {

    this.queue.add(name, data, { attempts: 3, backoff:  { type: 'fixed', delay: 5}});

  }

  /\*\*

   \* Process the data

   \* @param name

   \* @param concurrency

   \* @param callback

   \*/

  protected processJob(name: string, concurrency: number, callback: Queue.ProcessCallbackFunction<void>): void {

    this.queue.process(name, concurrency, callback);

  }

}

Step2: create auth.queue.ts in the same folder

import { IAuthJob } from '@auth/interfaces/auth.interface';

import { BaseQueue } from '@service/queues/base.queue';

class AuthQueue extends BaseQueue {

  constructor() {

    super('auth');

  }

  public addAuthUserJob(name: string, data: IAuthJob): void {

    this.addJob(name, data);

  }

}

export const authQueue: AuthQueue = new AuthQueue();

Step3: add route to routes.ts

A screenshot of a computer

Description automatically generated with medium confidence

Step 4: Update signup method

/\*\*

     \* Add to database

     \*/

    omit(userDataForCache, ['uId', 'username', 'email', 'avatarColor', 'password']);

    authQueue.addAuthUserJob('addAuthUserToDB', { value: userDataForCache });

omit some properties

import “omit” from lodash;

Should see following dashboard after running server

Graphical user interface, text, application, email

Description automatically generated

## Auth queue and worker:

create auth.worker.ts under shared/workers folder

import { DoneCallback, Job } from 'bull';

import Logger from 'bunyan';

import { config } from '@root/config';

import { authService } from '@service/db/auth.service';

const log: Logger = config.createLogger('authWorker');

class AuthWorker {

  async addAuthUserToDB(job: Job, done: DoneCallback): Promise<void> {

    try {

      const { value } = job.data;

      //add method to send data to database

      await authService.createAuthUser(value);

      job.progress(100);

      done(null, job.data);

    } catch (error) {

      log.error(error);

    }

  }

}

export const authWorker: AuthWorker = new AuthWorker();

add following line to the constructor of auth.queue

Graphical user interface, text

Description automatically generated

Similarly, create the user.worker.ts

import { DoneCallback, Job } from 'bull';

import Logger from 'bunyan';

import { config } from '@root/config';

import { userService } from '@service/db/user.service';

const log: Logger = config.createLogger('userWorker');

class UserWorker {

  async addUserToDB(job: Job, done: DoneCallback): Promise<void> {

    try {

      const { value } = job.data;

      //add method to send data to database

      await userService.addUserData(value);

      job.progress(100);

      done(null, job.data);

    } catch (error) {

      log.error(error);

    }

  }

}

export const userWorker: UserWorker = new UserWorker();

and user.queue.ts

import { BaseQueue } from '@service/queues/base.queue';

import { userWorker } from '@worker/user.worker';

class UserQueue extends BaseQueue {

  constructor() {

    super('user');

    this.processJob('addUserToDB', 5, userWorker.addUserToDB);

  }

  public addUserJob(name: string, data: any): void {

    this.addJob(name, data);

  }

}

export const userQueue: UserQueue = new UserQueue();

and user.service.ts

import { IUserDocument } from '@user/interfaces/user.interface';

import { UserModel } from '@user/models/user.schema';

class UserService {

  public async addUserData(data: IUserDocument): Promise<void> {

    await UserModel.create(data);

  }

}

export const userService: UserService = new UserService();

Update signup.ts under features/auth/controllers folder

Timeline

Description automatically generated with medium confidence

## Add JSON web token to session and signin

create signin.ts under features/auth/controllers folder with the following content

import JWT from 'jsonwebtoken';

import { Request, Response} from 'express';

import { config } from '@root/config';

import { joiValidation } from '@global/decorators/joi-validation.decorators';

import HTTP\_STATUS from 'http-status-codes';

import { authService } from '@service/db/auth.service';

import { BadRequestError } from '@global/helpers/error-handler';

import { loginSchema } from '@auth/schemes/signin';

import { IAuthDocument } from '@auth/interfaces/auth.interface';

import { IUserDocument } from '@user/interfaces/user.interface';

import { userService } from '@service/db/user.service';

export class SignIn {

  @joiValidation(loginSchema)

  public async read(req: Request, res: Response): Promise<void> {

    const { username, password } = req.body;

    const existingUser: IAuthDocument = await authService.getAuthUserByUsername(username);

    if(!existingUser) {

      throw new BadRequestError('Invalid credentials');

    }

    const passwordsMatch: boolean = await existingUser.comparePassword(password);

    if(!passwordsMatch){

      throw new BadRequestError('Invalid credentials');

    }

    const user: IUserDocument = await userService.getUserByAuthId(`${existingUser.\_id}`);

    const userJwt: string = JWT.sign(

      {

        userId: user.\_id,

        uId: existingUser.uId,

        email: existingUser.email,

        username: existingUser.username,

        avatarColor: existingUser.avatarColor

      },

      config.JWT\_TOKEN!

    );

    req.session = { jwt: userJwt };

    const userDocument: IUserDocument = {

        ...user,

        authId: existingUser!.\_id,

        username: existingUser!.username,

        email: existingUser!.email,

        avatarColor: existingUser!.avatarColor,

        uId: existingUser!.uId,

        createdAt: existingUser!.createdAt,

    } as IUserDocument;

    res.status(HTTP\_STATUS.OK).json({ message: 'User login successfully', user: userDocument, token:userJwt});

  }

}

Update auth/route

Graphical user interface, text, application

Description automatically generated

Update user service class

Text

Description automatically generated

## User logout controller

Create signout.ts under folder features/auth/controllers with the following content:

import HTTP\_STATUS from 'http-status-codes';

import { Request, Response } from 'express';

export class SignOut {

  public async update(req: Request, res: Response): Promise<void> {

    /\*\*

     \* reset the cookie's session to null for sign out

     \*/

    req.session = null;

    res.status(HTTP\_STATUS.OK).json({ message: 'Logout successful', user: {}, token: ''});

  }

}

add the route

Text

Description automatically generated

Text

Description automatically generated

## Get user from Mongodb

add following lines to user.service.ts

  public async getUserById(userId: string): Promise<IUserDocument> {

    const users: IUserDocument[] = await UserModel.aggregate([

      { $match: { \_id: new mongoose.Types.ObjectId(userId )}},

      // Lookup return an array

      { $lookup: { from: 'Auth', localField: 'authId', foreignField: '\_id', as: 'authId'}},

      // unwind return it as object

      { $unwind: '$authId'},

      //populate it back

      { $project: this.aggregateProject()}

    ]);

    return users[0];

  }

  public async getUserByAuthId(authId: string): Promise<IUserDocument> {

    const users: IUserDocument[] = await UserModel.aggregate([

      { $match: { authId: new mongoose.Types.ObjectId( authId )}},

      { $lookup: { from: 'Auth', localField: 'authId', foreignField: '\_id', as: 'authId'}},

      { $unwind: '$authId'},

      { $project: this.aggregateProject()}

    ]);

    return users[0];

  }

  /\*\*

   \* value 0 means it will be excluded

   \* value 1 means it will be included and returned

   \* @returns

   \*/

  private aggregateProject() {

    return {

      \_id: 1,

      username: '$authId.username',

      uId: '$authId.uId',

      email: '$authId.email',

      avatarColor: '$authId.avatarColor',

      createdAt: '$authId.createdAt',

      postsCount: 1,

      work: 1,

      school: 1,

      quote: 1,

      location: 1,

      blocked: 1,

      blockedBy: 1,

      followersCount: 1,

      followingCount: 1,

      notifications: 1,

      social: 1,

      bgImageVersion: 1,

      bgImageId: 1,

      profilePicture: 1

    };

  }

}

## Add current user controller

create file current-user.ts under features/auth/controllers folder with the following content:

import HTTP\_STATUS from 'http-status-codes';

import { userService } from '@service/db/user.service';

import { Request, Response } from 'express';

import { UserCache} from '@service/redis/user.cache';

import { IUserDocument } from '@user/interfaces/user.interface';

const userCache: UserCache = new UserCache();

export class CurrentUser {

  public async read(req: Request, res: Response): Promise<void> {

    let isUser = false;

    let token = null;

    let user = null;

    const cachedUser: IUserDocument = await userCache.getUserFromCache(`${req.currentUser!.userId}`) as IUserDocument;

    //if cached user existed, use it. else get the userby id

    const existingUser: IUserDocument = cachedUser ? cachedUser : await userService.getUserById(`${req.currentUser!.userId}`);

    if (Object.keys(existingUser).length) {

      isUser = true;

      token = req.session?.jwt;

      user = existingUser;

    }

    res.status(HTTP\_STATUS.OK).json({token, isUser, user});

  }

}

add a new route for features\auth\routes

import { CurrentUser } from '@auth/controllers/current-user';

import express, { Router } from 'express';

class CurrentUserRoutes {

  private router: Router;

  constructor(){

    this.router = express.Router();

  }

  public routes(): Router {

    this.router.get('/currentuser', CurrentUser.prototype.read);

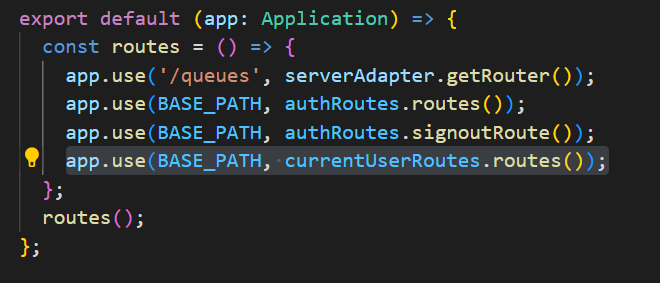
    return this.router;

  }

};

export const currentUserRoutes: CurrentUserRoutes = new CurrentUserRoutes();

Update route file



## Add authentication middleware

create auth-middleware.ts under globals/helpers folder with the following content

import { Request, Response, NextFunction } from 'express';

import JWT from 'jsonwebtoken';

import { config } from '@root/config';

import { NotAuthorizedError } from '@global/helpers/error-handler';

import { AuthPayload } from '@auth/interfaces/auth.interface';

export class AuthMiddleware {

  /\*\*

   \* verify user exist

   \*

   \*/

  public verifyUser(req: Request, \_res: Response, next: NextFunction): void {

    if(!req.session?.jwt){

      throw new NotAuthorizedError('Token is not available, please login again.');

    }

    try {

      const payload: AuthPayload = JWT.verify(req.session?.jwt, config.JWT\_TOKEN!) as AuthPayload;

      req.currentUser = payload;

    } catch (error) {

      throw new NotAuthorizedError('Token is invalid, please login again.');

    }

    next();

  }

  /\*\*

   \* check if user have authentication

   \*/

  public checkAuthentication(req: Request, \_res: Response, next: NextFunction): void {

    if(!req.currentUser){

      throw new NotAuthorizedError('Authentication is required to access this route');

    }

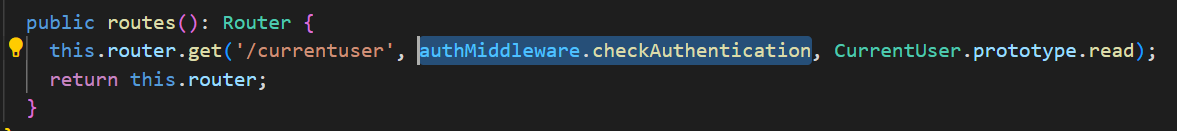
    next();

  }

}

export const authMiddleware: AuthMiddleware = new AuthMiddleware();

update the routes to include call to the function



Text

Description automatically generated

# Email Feature

## Set up mail transport class

install nodemailer (for local environment) and @sendgrid/email(for production) and ejs (template engine)

npm i nodemailer @sendgrid/mail ejs

set up new env configs:

SENDER\_EMAIL=

SENDER\_EMAIL\_PASSWORD=

SENDGRID\_API\_KEY=

SENDGRID\_SENDER=

Create a mail.transport.ts under shared/services/emails folder

import nodemailer from 'nodemailer';

import Mail from 'nodemailer/lib/mailer';

import Logger from 'bunyan';

import sendGridMail from '@sendgrid/mail';

import { config } from '@root/config';

import { BadRequestError } from '@global/helpers/error-handler';

interface IMailOptions {

  from: string;

  to: string;

  subject: string;

  html: string;

}

const log: Logger = config.createLogger('mailOptions');

sendGridMail.setApiKey(config.SENDGRID\_API\_KEY!);

class MailTransport {

  public async sendEmail(receiverEmail: string, subject: string, body: string): Promise<void> {

    if (config.NODE\_ENV === 'test' || config.NODE\_ENV === 'development') {

      this.developmentEmailSender(receiverEmail, subject, body);

    } else {

      this.productionEmailSender(receiverEmail, subject, body);

    }

  }

  /\*\*

   \* Send development email

   \* @param receiverEmail

   \* @param subject

   \* @param body

   \*/

  private async developmentEmailSender(receiverEmail: string, subject: string, body: string): Promise<void> {

    // create reusable transporter object using the default SMTP transport

    const transporter:Mail = nodemailer.createTransport({

      host: 'smtp.ethereal.email',

      port: 587,

      secure: false, // true for 465, false for other ports

      auth: {

        user: config.SENDER\_EMAIL!,

        pass: config.SENDER\_EMAIL\_PASSWORD!,

      },

    });

    const mailOptions: IMailOptions = {

      from: `Chat App <${config.SENDER\_EMAIL!}>`,

      to: receiverEmail,

      subject,

      html: body

    };

    try {

      await transporter.sendMail(mailOptions);

      log.info('Development email sent successfully');

    } catch (error) {

      log.error('Error sending email', error);

      throw new BadRequestError('Error sending email');

    }

  }

  private async productionEmailSender(receiverEmail: string, subject: string, body: string): Promise<void> {

    const mailOptions: IMailOptions = {

      from: `Chat App <${config.SENDER\_EMAIL!}>`,

      to: receiverEmail,

      subject,

      html: body

    };

    try {

      await sendGridMail.send(mailOptions);

      log.info('Production email sent successfully');

    } catch (error) {

      log.error('Error sending email', error);

      throw new BadRequestError('Error sending email');

    }

  }

}

export const mailTransport: MailTransport = new MailTransport();

Create new ethereal email account and copy the credentials to the .env file to test it

<https://ethereal.email/create>

## Set up Email queue and worker

Email Queue: to add the data type IEmailJob to the base queue

import { IEmailJob } from './../../../features/user/interfaces/user.interface';

import { BaseQueue } from '@service/queues/base.queue';

import { emailWorker } from '@worker/email.worker';

class EmailQueue extends BaseQueue {

  constructor() {

    super('emails');

    this.processJob('forgotPasswordEmail', 5, emailWorker.addNotificationEmail);

  }

  public addEmailJob(name: string, data: IEmailJob): void {

    this.addJob(name, data);

  }

}

export const emailQueue: EmailQueue = new EmailQueue();

Email worker

import { DoneCallback, Job } from 'bull';

import Logger from 'bunyan';

import { config } from '@root/config';

import { mailTransport } from '@service/emails/mail.transport';

const log: Logger = config.createLogger('emailWorker');

class EmailWorker {

  async addNotificationEmail(job: Job, done: DoneCallback): Promise<void> {

    try {

      const { template, receiverEmail, subject } = job.data;

      //send email

      await mailTransport.sendEmail(receiverEmail, subject, template);

      job.progress(100);

      done(null, job.data);

    } catch (error) {

      log.error(error);

    }

  }

}

export const emailWorker: EmailWorker = new EmailWorker();

## Set up password reset templates:

Create folders and the file. the .ejs contains the email template

Text

Description automatically generated

example code in the .ts file, note the arguments to be passed on depends on the template:

import fs from 'fs';

import ejs from 'ejs';

class ForgotPasswordTemplate {

  public passwordResetTemplate(username: string, resetLink: string): string {

    return ejs.render(fs.readFileSync(\_\_dirname + '/forgot-password-template.ejs', 'utf8'), {

      username,

      resetLink,

      image\_url: 'https://w7.pngwing.com/pngs/120/102/png-transparent-padlock-logo-computer-icons-padlock-technic-logo-password-lock.png'

    });

  }

}

export const forgotPasswordTemplate: ForgotPasswordTemplate = new ForgotPasswordTemplate();

additional: install moment for ip

npm i moment ip

## Set Up Forgot password controller

### Common structure

routes create the required HTTP POST/GET request and call the required method in the controller

import { config } from '@root/config';

import HTTP\_STATUS from 'http-status-codes';

import { authService } from '@service/db/auth.service';

import { BadRequestError } from '@global/helpers/error-handler';

import { IAuthDocument } from '@auth/interfaces/auth.interface';

import { joiValidation } from '@global/decorators/joi-validation.decorators';

import { emailSchema } from '@auth/schemes/password';

import crypto from 'crypto';

import { emailQueue } from '@service/queues/email.queue';

/\*\*

 \* Controller class for password reset

 \*/

export class Password {

  //validate the scheme

  @joiValidation(emailSchema)

  public async create(req: Request, res: Response): Promise<void> {

    const { email } = req.body;

    //Get user from authService, if no user, throw error

    const existingUser: IAuthDocument = await authService.getAuthUserByEmail(email);

    if (!existingUser) {

      throw new BadRequestError('Invalid credentials');

    }

    //generate random character as new password

    const randomBytes: Buffer = await Promise.resolve(crypto.randomBytes(20));

    const randomCharacters: string = randomBytes.toString('hex');

    //call the authService updatePasswordToken method, set expiring time of the password

    await authService.updatePasswordToken(`${existingUser.\_id!}`, randomCharacters, Date.now() \* 60 \* 60 \* 1000);

    //Add reset link, email template, and put these into the emailQueue job

    const resetLink = `${config.CLIENT\_URL}/reset-password?token=${randomCharacters}`;

    const template: string = forgotPasswordTemplate.passwordResetTemplate(existingUser.username, resetLink);

    emailQueue.addEmailJob('forgotPasswordEmail', { template, receiverEmail: email, subject: 'Reset your password'});

    //Response to the user

    res.status(HTTP\_STATUS.OK).json({ message: 'Password reset email sent.' });

  }

}

Add following method to auth.service()

  /\*\*

   \* Update password token

   \* @param authId

   \* @param token

   \* @param tokenExpiration

   \*/

  public async updatePasswordToken(authId: string, token: string, tokenExpiration: number): Promise<void> {

    //updateOne is the MongoDB method to update the password

    await AuthModel.updateOne({ \_id: authId }, {

      passwordResetToken: token,

      passwordResetExpires: tokenExpiration

    });

  }

# Unit Test Set Up

install jest <https://jestjs.io/docs/getting-started>

npm install --save-dev jest

npm i @types/jest

npm i -D ts-jest

Key structure.

1. Mostly concentrated on testing method in controllers
2. Use jest.mock to mock the function called by the tested method
3. Simulate the arguments and responds
4. Use jest.spyOn to mock the function.

# POST feature

Define interfaces

Define the model schema to be used in the mongo db database

Define joi validation schemes

Set up the controller for each function . Example below

import HTTP\_STATUS from 'http-status-codes';

import { Request, Response } from 'express';

import { joiValidation } from '@global/decorators/joi-validation.decorators';

import { postSchema } from '@post/schemes/post.schemes';

import { ObjectId } from 'mongodb';

import { IPostDocument } from '@post/interfaces/post.interface';

export class Create {

  @joiValidation(postSchema)

  public async post(req: Request, res: Response): Promise<void> {

    //destructure the propety from req.body

    const { post, bgColor, privacy, gifUrl, profilePicture, feelings } = req.body;

    const postObjectId: ObjectId = new ObjectId();

    const createdPost: IPostDocument = {

      \_id: postObjectId,

      userId: req.currentUser!.userId,

      username: req.currentUser!.username,

      email: req.currentUser!.email,

      avatarColor: req.currentUser!.avatarColor,

      profilePicture,

      post,

      bgColor,

      feelings,

      privacy,

      gifUrl,

      commentsCount: 0,

      imgVersion: '',

      imgId: '',

      createdAt: new Date(),

      reactions: { like: 0, love: 0, happy: 0, sad: 0, wow: 0, angry: 0}

    } as IPostDocument;

    res.status(HTTP\_STATUS.CREATED).json({ message: 'Post created successfully' });

  }

}

## Set up socket io

create file post.ts under shared/sockets folder with the following content

import { Server, Socket } from 'socket.io';

// Use this to emit the event inside the controller outside the socketioposthandler class

let socketIOPostObject: Server;

/\*\*

 \* Use socket IO post handler so user can get faster response when they post any thing

 \*/

export class SocketIOPostHandler {

  private io: Server;

  constructor(io: Server) {

    this.io = io;

    socketIOPostObject = io;

  }

  public listen(): void {

    this.io.on('connection', (socket: Socket) => {

      //TODO change later

      console.log('Post socketio handler');

    });

  }

}

Update the method in set up server

A screenshot of a computer

Description automatically generated

## Set up save to redis cache

import { ServerError } from '@global/helpers/error-handler';

import { BaseCache } from '@service/redis/base.cache';

import { config } from '@root/config';

import Logger from 'bunyan';

import { ISavePostToCache } from '@post/interfaces/post.interface';

const log: Logger = config.createLogger('postCache');

/\*\*

 \* Export the class and create instance inside controller

 \* due to potential connection issue

 \*/

export class PostCache extends BaseCache {

  constructor() {

    super('postCache');

  }

  public async savePostToCache(data: ISavePostToCache): Promise<void> {

    const { key, currentUserId, uId, createdPost } = data;

    const {

      \_id,

      userId,

      username,

      email,

      avatarColor,

      profilePicture,

      post,

      bgColor,

      feelings,

      privacy,

      gifUrl,

      commentsCount,

      imgVersion,

      imgId,

      reactions,

      createdAt

    } = createdPost;

    const firstList: string[] = [

      '\_id',

      `${\_id}`,

      'userId',

      `${userId}`,

      'username',

      `${username}`,

      'email',

      `${email}`,

      'avatarColor',

      `${avatarColor}`,

      'profilePicture',

      `${profilePicture}`,

      'post',

      `${post}`,

      'bgColor',

      `${bgColor}`,

      'feelings',

      `${feelings}`,

      'privacy',

      `${privacy}`,

      'gifUrl',

      `${gifUrl}`

    ];

    const secondList: string[] = [

      'commentsCount',

      `${commentsCount}`,

      'reactions',

      JSON.stringify(reactions),

      'imgVersion',

      `${imgVersion}`,

      'imgId',

      `${imgId}`,

      'createdAt',

      `${createdAt}`

    ];

    const dataToSave: string[] = [...firstList, ...secondList];

    try {

      if (!this.client.isOpen) {

        await this.client.connect();

      }

      //Get the post count

      const postCount: string[] = await this.client.HMGET(`users:${currentUserId}`, 'postsCount');

      log.info(`post count: ${postCount[0]}`);

      //Use multi() to create multiple redis command and execute all the methods

      const multi: ReturnType<typeof this.client.multi> = this.client.multi();

      await this.client.ZADD('post', { score: parseInt(uId, 10), value: `${key}` });

      multi.HSET(`posts:${key}`, dataToSave);

      //increment the count

      const count: number = parseInt(postCount[0], 10) + 1;

      //Update the count

      multi.HSET(`users:${currentUserId}`, ['postsCount', count]);

      multi.exec();

    } catch (error) {

      log.error(error);

      throw new ServerError('Server error. Try again.');

    }

  }

}

## Set up post queue and worker

create queue,

create worker,

create services for the worker.

# Generate user script

install faker

npm i @faker-js/faker

create a seed.ts file under src folder

install canvas for avatar

npm i canvas

iEjqzhVdgZt7qfkz