Influncer App

Documentation

middlwares:

-reachs

-addPostAuth

-validatePost

-adminAuth

-userPostsAuth

-userAuth

-validateSignup

-validateSignin

-reachs:

reaches module uses request-ip third party module to get the client's Ip V4 and store it in data base with its timstamp then it go to the next layer .

-addPostAuth :

addPostAuth is a middleware that checks if the client is authorized to post or not and requires the request to contains the Bearer token in headers.authorization and if the client has no token or hasn't premium account it returns with status 401 and if the client was admin it direct him to the next middleware directly and adding userInfo and created_by to the request body and if the client has premium account it checks if he Allowed to post or not by using isAllowedToPost local module which takes the id of the user and checks if the user has less than 2 posts this month or not and returns boolean based on this and if the user exceeded the number of posts this month it returns with status 403.

-validatePost:

validatePost is a validation middleware that validate on the requested body with help of Joi third party module and it checks that the title of the post has minimum of 3 characters and maximum of 20 and the desc (description) has minimum of 3 characters and maximum of 150 with help of Post.validation local module which contains the joi schema and if Post.validation returned undefined error it direct the request to the next layer but if the error wasn't undefined it returned with status 400 and with the error as json.

-adminAuth:

It checks if the user is admin or not and it requires the request to contains the Bearer token in headers.authorization and if the client has no token or isn't the admin it returns with status 401 and if the client was admin it direct him to the next middleware directly and adding userInfo.

-userPostsAuth:

it checks the Authentication of the user and it requires the request to contains the Bearer token in headers.authorization and adding userInfo to the request body and if the client has no token or not valid token it makes the req.query.page equals to one always to prevent unregistered from getting all posts and returns with status of 500 if internal error happened in the server.

-userAuth:

it checks the Authentication of the user and it requires the request to contains the Bearer token in headers.authorization and adding user to the request body and if the client has no token or not valid token it returns with status 401 and returns with status of 500 if internal error happened in the server.

-validateSignup:

validateSignup is a validation middleware that validate on the requested body with help of Joi third party module and it checks that the first_name has minimum of 2 characters and maximum of 30 and last_name has minimum of 2 characters and maximum of 30 and password matches the Regex:('^01[0125][0-9]{8}$') and checks the email is valid or not and age is between 18 and 85 and the phone matches the Regex:('^01[0125][0-9]{8}$') and image can be null if user don't choose any photos and plane must be basic or premium with the help of user.validation.model local module which contains the joi schema and if SignupValidation returned undefined error it direct the request to the next layer but if the error wasn't undefined it returned with status 400 and with the error as json.

-validateSigninn:

validateSignin is a validation middleware that validate on the requested body with help of Joi third party module and it checks that the password matches the Regex:('^01[0125][0-9]{8}$') and checks the email is valid or not with the help of user.validation.model local module which contains the joi schema and if SigninValidation returned undefined error it direct the request to the next layer but if the error wasn't undefined it returned with status 400 and with the error as json.

End Points:

- baseUrl/api/users/signup
- baseUrl/api/users/signIn
- baseUrl/api/users/UpgradeToPrem
- baseUrl/api/users/deleteByID
- baseUrl/api/users/getAllUsers
- baseUrl/api/users/getUserPosts/:id

- baseUrl/api/users/searchName
- baseUrl/api/users/searchAge

- baseUrl/api/posts/addPost
- baseUrl/api/posts/updatePost
- baseUrl/api/posts/PinPost
- baseUrl/api/posts/unPinPost
- baseUrl/api/posts/AddComment
- baseUrl/api/posts/deletePost
- baseUrl/api/posts/getAllPosts
- baseUrl/api/posts/getPinnedPosts
- baseUrl/api/posts/statics
- baseUrl/api/posts/getpostById/:id
- baseUrl/api/posts/likeApost
- baseUrl/api/posts/deleteLike

-baseUrl/api/users/signup:

-middleware Stack:

Reachs =>	ValidateSignup=>	signUp

requires the request on the format: req.body = { first_name, last_name, email, password, age, phone, plane, image } and it checks that the email isn't existed in the database then it insert the data to database if it will be unique in the database then it generate token for user with his id returned from database and send it to user.

-Response status:

Status	meaning
400	If validation error from joi schema
406	If the used email is already existed
500	Internal server error
201	User was created

-baseUrl/api/users/signin:

-middleware Stack:

Reachs =>	ValidateSignin=>	signUp

requires the request on the format: req.body = {email, password } and it checks that the email is existed in the database then it try to hash the sent password and compare it with on returned from the database and if it matched it generates the token for user and send it.

-Response status:

Status	meaning
400	If validation error from joi schema
401	If the used email or password is invalid
500	Internal server error
202	The data was valid

-baseUrl/api/users/upgradeToPrem

-middleware Stack:

	AdminAuth=>	upgradeToPrem

requires the request on the format: req.body = {id }&
req.headers.authorization = (Bearer token) and it checks that the email is existed in the database then it try to update isPremium to true then returned with message updated successfully.

-Response status:

Status	meaning
404	If the id doesn’t match to any record in the database
201	If the operation was done successfully
500	Internal server error
401	The token wasn’t belongs to admin

-baseUrl/api/users/deleteByID

-middleware Stack:

AdminAuth=>	upgradeToPrem

requires the request on the format: req.body = {id }&
req.headers.authorization = (Bearer token) and it checks that the email is existed in the database then it try to delete the record then returned with message updated successfully.

-Response status:

Status	meaning
404	If the id doesn’t match to any record in the database
200	If the operation was done successfully
500	Internal server error
401	The token wasn’t belongs to admin

- baseUrl/api/users/getAllUsers:

-middleware Stack:

AdminAuth=>	getAllUsers

requires the request on the format: req.headers.authorization = (Bearer token) and returned all users data except password.

-Response status:

Status	meaning
404	If there is no users in the database
200	If the operation was done successfully
500	Internal server error
401	The token wasn’t belongs to admin

- baseUrl/api/users/getUserPosts/:id :

-middleware Stack:

reachs =>	UserAuth=>	getUserPosts

requires the request on the format: req.query.page = (page) If not send it will be page = 1& req.headers.authorization = (Bearer token) and returned all posts belongs to the user and user which contains user information.

-Response status:

Status	meaning
404	If there is no posts belongs to this user in database
200	If the operation was done successfully
206	There is no posts for this user but user is existed so it returns with user information only
500	Internal server error
401	The token wasn't belongs to any user

- baseUrl/api/users/searchName:

-middleware Stack:

reachs =>	AdminAuth=>	searchName

requires the request on the format: req.query.name = (name)& req.headers.authorization = (Bearer token) and returned all records matches the name (using regEx)

-Response status:

Status	meaning
404	If there is no users with this name in database
200	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to any admin

<u>- baseUrl/api/users/searchAge:</u>

-middleware Stack:

reachs =>	AdminAuth=>	searchAge

requires the request on the format: req.query.name = {x,y}& req.headers.authorization = (Bearer token) and returned all users with age between x age and y age.

-Response status:

Status	meaning
404	If there is no users with this age in database
200	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to admin

- baseUrl/api/posts/addPost:

-middleware Stack:

reachs =>	AddPostAuth=>	ValidatePost=>	addPost

requires the request on the format: req.body= {title, desc}& req.headers.authorization = (Bearer token) then it checks if there is post with the same information or not then it add post information to the database.

-Response status:

Status	meaning
403	If the user was premium and exceeded the posts limits in the month
201	If the operation was done successfully

500	Internal server error
401	The token wasn’t belongs to user
406	If the user already have a post with same information

- baseUrl/api/posts/updatePost:

-middleware Stack:

reachs =>	AdminAuth=>	ValidatePost=>	updatePost

requires the request on the format: req.body= {id,title, desc}& req.headers.authorization = (Bearer token) then it checks if the post belongs to the user with the request or not then it updated it.

-Response status:

Status	meaning
403	If the user was premium and exceeded the posts limits in the month
201	If the operation was done successfully
500	Internal server error
401	The token wasn’t belongs to user or the post doesn’t belong to the user

- baseUrl/api/posts/PinPost:

-middleware Stack:

AdminAuth=>	PinPost

requires the request on the format: req.body= {id}&
req.headers.authorization = (Bearer token) then it checks if the post existed or not then it update isPinned = true then send a message: post was pinned successfully.

-Response status:

Status	meaning
404	If there is no post with id sent
201	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to admin

- baseUrl/api/posts/unPinPost:

-middleware Stack:

AdminAuth=>	unPinPost

requires the request on the format: req.body= {id}&
req.headers.authorization = (Bearer token) then it checks if the post existed or not then it update isPinned = false then send a message: post was pinned successfully.

-Response status:

Status	meaning
404	If there is no post with id sent
201	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to admin

- baseUrl/api/posts/AddComment:

-middleware Stack:

UserAuth=>	AddComment

requires the request on the format: req.body= {comment,postId}& req.headers.authorization = (Bearer token) then it checks if the post existed or not then it update comments by pushing it into the array of comments.

-Response status:

Status	meaning
404	If there is no post with id sent
201	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to user

- baseUrl/api/posts/deletePost:

-middleware Stack:

AdminAuth=>	deletePost

requires the request on the format: req.body= {id}& req.headers.authorization = (Bearer token) then it checks if the post existed or not thin it delete the post.

-Response status:

Status	meaning
404	If there is no post with id sent
200	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to admin

- baseUrl/api/posts/getAllPosts:

-middleware Stack:

reachs =>	UserPostsAuth=>	getAllPosts

requires the request on the format: req.body= {id}&req.query.page = (page) If not send it will be page = 1& req.headers.authorization = (Bearer token) then it checks if any post existed or not thin it sends the data in the response.

-Response status:

Status	meaning
404	If there is no post
201	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to user

- baseUrl/api/posts/getPinnedPosts:

-middleware Stack:

reachs =>	UserAuth=>	getPinnedPosts

requires the request on the format: req.headers.authorization = (Bearer token) then it checks if any pinned posts existed or not thin it sends the data in the response.

-Response status:

Status	meaning
404	If there is no pinned posts
201	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to user

- baseUrl/api/posts/statics:

-middleware Stack:

reachs =>	AdminAuth=>	statics

requires the request on the format: req.headers.authorization = (Bearer token) then it sends the statics which is noOfUsers (number of users), noOfPosts (number of posts) mostInteraction (post has the most number of likes)By sorting the records by the number of likes then send the first item in the collection to the user, noOfAllReaches (number of all requests sent to server except on the end points belongs to only the admin) noOfTodayReaches (number of today requests sent to server except on the end points belongs to only the admin), noOfTodayPosts ,noOfYesterdayPosts.

-Response status:

Status	meaning
200	If the operation was done successfully
500	Internal server error
401	The token wasn't belongs to user

- baseUrl/api/posts/getpostById/:id :

-middleware Stack:

reachs =>	UserAuth=>	getpostById/:id

requires the request on the format: req.headers.authorization = (Bearer token) then it checks if any pinned posts existed or not thin it sends the data in the response.

-Response status:

Status	meaning
404	If there is no post with sent id
200	If the operation was done successfully
500	Internal server error
401	The token wasn’t belongs to user

<u>- baseUrl/api/posts/likeApost</u>

-middleware Stack:

reachs =>	UserAuth=>	likeApost

requires the request on the format: req.headers.authorization = (Bearer token) & req.body = {post} which is the post id then it checks post is existed or not then checks if the user has liked this post or not by search in likes list with user ID and if user liked it before it do nothing and if not it add id of the user to likes list then send a message to the user with success message.

-Response status:

Status	meaning
404	If there is no post with sent id
201	If the operation was done successfully
500	Internal server error

401	The token wasn’t belongs to user

- baseUrl/api/posts/deleteLike

-middleware Stack:

reachs =>	UserAuth=>	likeApost

requires the request on the format: req.headers.authorization = (Bearer token) & req.body = {postId} which is the post id then it checks post is existed or not then delete id from likes list then send a message to the user with success message.

-Response status:

Status	meaning
404	If there is no post with sent id
200	If the operation was done successfully
500	Internal server error
401	The token wasn’t belongs to user

BRD

Business Requirements Document

1. Summary Statement

This project is designed for an influencer wants to have his own system to add his posts on it and manage the system as he wants.

2. Project Objectives

- Application enables users to be close to the influncer.
- Users can access the platform anytime and anywhere and get an excellent Actor1& Actor2 experience.
- Allow users to interact with their favorite posts
- Allowing Admin to get Statics about the system.
- Allowing Admin to control the system by deleting or upgrading users.

The business requirement will be represented in the following deliverables:

- website for (Actor1 & Actor2 & Superadmin).

3. Project scope

This project is designed for an influencer wants to post anything on his mind. The influencer wants to offer the visitor of his website 2 types of plans (Basic, Premium). Basic plan users can only leave comments onthe posts posted by the influencer. Premium plan users, however, can have some privileges such as the ability to post twice a month on the influencer platform. Not only that, but they can send an email to the influencer for private discussion. Posts can be pinned for quick access and should be presented in a viewable section on the home page for all users.

4. Functional requirements

4.1 Actor2 Features:

4.1.1 Sign up

- **Description**:
  - After entering the registration data, the user will be able to access the full functionality of the application.

4.1.2 Login

- **Description**:
  - For logging into the system, as Actor1 or Actor2 the user must enter the registered email& password. Then the user should be able to access the account.

4.1.3 Add post

- **Description**:
  - Given that an Actor2 has an account, so he can add posts twice every month an admin can post as he wants.

4.1.4 react

- **Description**:

The Actor1 & Actor2 & admin will be able to like any post

4.1.5 control post

- **Description**:

admin can remove or pin or update posts.

4.1.6 control users

- **Description**:

admin can remove or upgrade or update users.

SRS

Author:Mohamed
Shehata Alnawagy

Table of Contents

1.2.5 Actor2/ Change Language

3 Introduction

3.1 Main Scope

- Displaying all the available posts to users in the main window helping Actor1s to see and react with posts easily.

3.2 Actors

1 Actor1

- Actor1 can register on the app or on the website to see all the posts in the website.
- The Actor1 can’t add posts.

2 Actor2

- Actor2 can register on the app or on the website to see all the posts in the website.
  The Actor2 can add posts twice amonth.

3 Super Admin

- The super admin will login just like other actors but he can access dashboard for the system which have statics about the system.
- Once the super admin login into the system he will have all control to add any number of posts , delete and update posts, delete users and upgrade Actor1 to Actor2 .

3.3 Deliverables

1. Website for all Actors.
2. Dashboard for super admin.

6 Sprint Content

6.1 Sprint (1)

1.1 Actor1

- 1.1.1 Signup.
- 1.1.2 Login.
- 1.1.3 view posts
- 1.1.4 react with posts (by making like or a comment)

1.2 Actor2

- 1.2.1 Signup.
- 1.2.2 Login.
- 1.1.3 view posts
- 1.1.4 react with posts (by making like or a comment)
- 1.1.5 add posts (with limit

- **User Story:**

As an Actor1, I want to register into the system so that I can see all posts.

User Journey:

- The Actor1 opens the app.
- the Actor1& Actor2 selects signUp (for the first time).
- Login screen**[email, Password].**
- Actor1& Actor2 choose to register as a premium or as a basic.
- The Actor1& Actor2 enters his personal info.
- The Actor1& Actor2 pressed the “Submit” button.
- The Actor1& Actor2 will move to the home screen which contains posts.
- A congratulation You can now see all posts and react with it.
- The Actor2 can add a post from the home screen.
- The Actor1& Actor2 can view any profile by clicking on avatar of the creator of the post.
- The Actor1& Actor2 can view comments of the post by clicking on title of the post.

■ Validation

Field Name	Type	Mandatory	Default Value	Specifications
User Registration Screen				
First name	Text	Y	Placeholder	English letters, spaces, apostrophe (‘). min. (3), max. (100) characters. Arabic letters, spaces, apostrophe (‘). min. (3), max. (100) characters
last name	Text	Y	Placeholder	Length should be (14) digits.
password	Text	Y	Placeholder	Should follow the password complexity “Minimum of 8 characters and matches the regEx:'^[a-zA-Z0-9]{3,30}$'
email	Text	Y	Placeholder	Should be in an email format (Including at least one @ and one dot.)
age	Numerical	Y	Placeholder	Should be less than 85 and more than 18
phone	Text	Y	Empty	Should matches the regEx: '^01[0125][0-9]{8}$'
profile picture	Image	N	Empty	Should be an URL for photo
plane	Text	Y	basic	Should be premium or basic