Date: Page:

① Subscription model

① toggle Subscription →

- Channelid (user) → req.params
- SubscriberId → req.user._id
  - check if ~~Subscr~~ already Subscribed (from Subscription DB)
    - ↳ true → unSubscribe → Deletecone.
    - ↳ false → Subscribe → Create.
- Send response.

② getuserChannelSubscriber

- SubscriberId ~~of~~ (user) → req.params.
- Subscription.aggregate → (~~Subscriberlist~~) (channelSubscriber
  - ↳ $match: channel → new mongoose.Types.objId (Subs
  - ↳ $lookup: → from: users
    - localfield: Subscriber
    - ~~foreign~~ foreignField: _id
    - as: Subscriber
    - → Pipeline: [$Project: → username, fullname, avatar]
  - ↳ $addfield: Subscriber: $first: $ subscriber.
- Subscriberlist → channelSubscriber.map(item → item.Subscriber
- return response.

Date: Page:

① Subscription model

① toggle Subscription →

→ Channelid (user) → req.params
→ Subscriber.Id → req.user.id.
check if ~~Subscr~~ already Subscribed (from Subscription DB)
↳ true → unSubscribe → Deletone.
↳ false → Subscribe → Create.
→ Send response.

② getuserChannelSubscriber

→ SubscriberId (user) → req.parans.
→ Subscription.aggregate → (~~Subscriber list~~) (ChannelSubscriber
↳ $match: channel → new mongoose.Types.objId (Subs
↳ $lookup: → from: users
localfield: Subscriber
foreignField - id
as: Subscriber
↳ Pipeline: [$Project: → username, fullname, avatar]
↳ $addfield: Subscriber: $first: $ subscriber.

→ Subscribedlist → ChannelSubscriber.map(item → item.Subscrib

→ return response -

Date: Page:

Video model

getVideo By Id →

1. VideoId → req.Params → isValidObject ✓
2. find ~~VideoId~~ in Videos Collection By VideoId
3. find user in users collection by req.user?._id (only get watchhist)
4. increment view of Video By 1 (based on watchhistory)
   → if (not users.watchhistory.includes(VideoId)
   ↳ find by VideoId → inc. view by 1, return new ($inc)
5. Add Video to watchhistory of req.user using $addtoset operator return new
6. ~~Audit~~ Video. aggregate
   ↳ match → _id : VideoId
   ↳ Lookup → in users, localfield → owner, forfield → _id
   ↳ Pipeline → username, avatar, fullname, _id → (Project)

Update Video → (title, discription, thumbnail)

1. VideoId → req.params. → isValid.Object ✓
2. {title, dis(.} → req.body → validate ✓
3. thumbnailLocalPath → req.file?.Path
4. oldVideo - find in Video collection by VideoId → (only thumbnail object → Contains → url & PublicId)
5. Upload new video on Cloudinary
6. Update database of this video findbyId and update (videoId, thumbnail)
7. delete from cloudinary (old video.PublicId)
8. return response.

Toggle publish →

Date: Page:

① VideoId → (req.params) → isValidObjectId ✓
② find in video collection ~~and return~~ by videoId and return '_id, isPublished, owner
③ Validate owner of the video (req.user._id === video.owner)
④ update video publish status. find by VideoId and update isPublished : !video?.isPublished
⑤ return response

⑤ Delete video

① VideoId → (req.params) → isValidObjectId ✓
② fetch video details by videoId and return ~~{ _id, v}~~ { _id, owner, videofile, thumbnail }
③ Validate owner of the video (video.owner.toString !== req.user._id.toString()) → return error(Not Authorized)
④ delete video & ~~the~~ thumbnail from cloudinary (~~delete from~~
↳ (deleteFromCloudinary, deleteVideoFromCloudinary)
↳ check if return status from cloudinary is "ok"
⑤ → Delete video from Video collection using (videoId)
→ In user collection update all records (updateMany) where ~~watch~~ watch history has videoId
↳ $Pull videoId from watchhistory
→ user.updateMany ({ watchhistory : videoId }, { $Pull : { watchhistory : videoId } })
⑥ return response.

Date: Page:

① → get All Videos

② const { Page = 1, limit = 10, query = "", sortBy = "createdAt", sortType = 1, userId } = req.query

③ matchCondition
↳ $or → title: { $regex: query, $option: 'i' }
↳ description: { $regex: query, $option: 'i' }

④ if userId → matchCondition.owner = new objectId(userId)

⑤ ~~Agg~~ Video.aggregate
↳ $match: matchCondition
↳ $lookup: from: users
localfield: owner
foreignfield: _id
as: owner
↳ pipeline: [$Project → _id, fullname, avatar, username]
↳ $addfield: owner: → $first: $owner.
↳ $sort: [sortby || "createdAt"] : sortby || 1

⑥ Options for aggregatePaginate →
↳ { Page, limit: parseInt(limit), customLabels:
↳ totalDocs: "totalVideos"
↳ docs: "videos"

⑦ Video.aggregatePaginate (Video.aggregate (from step 4), options)
↳ .then → (result →
if (result?.videos?.length === 0 && userId)
↳ return (200, [], "No video found")
↳ else (return (200, result, Video fetched Successfully)
↳ try { return (200, result, "fetched Successfully") }
catch (error) { throw new ApiError (error.message) }

Date: Page:

③ HealthCheck - Controller

① Check db connection readyState. (dbStatus)
~~→ mongoose~~
→ dbStatus = mongoose.connection.readyState ? "db connec." : "db disco."

② Create healthcheck object.
↳ dbStatus, uptime: process.uptime, message: "OK", timestamp: Date.now(), hrtime: process.hrtime().

③ try :→ ~~healthcheck~~ add ServerStatus to healthcheck object
↳ returns response with Status 200
↳ if dbStatus = "db disconnected" ~~retur~~ throw an error

④ catch → catch the thrown error or any other ~~runtime~~ other. and return the res. with Status "500"

④ ~~Addcomme~~n Comment. Controller

A ④ addComment

① get videoId from Params and validate Object Id
↳ (req.Params) → isValidObjectId() ✓

② get comment ~~text~~ content from req.body and validate its not empty
↳ (content.trim() === "") ✗

③ find video by videoId from "Videocollection
↳ (!video) → Error ("video not found")

④ find user by req.user?._id to validate user exists in DB.
↳ (!user) → Error ("user not found")

⑤ Create a new comment with content, video and owner
↳ await Comment.create({ content, video: videoId, owner: req.user?._id })

⑥ ↳ (!newComment) → "Comment not added Successfully"

⑦ return Response.

(B) Update Comment

① get CommentId from Params and Validate ObjectId
↳ (req.Params) → isValidObjectId() ✓

② find Comment by and CommentId from Comments Collection
↳ (!Comment) → "Comment not found"

③ check if loged in user is the owner of the comment.
↳ (Comment?.owner?.toString() !== req.user?._id.toString())
↳ Error ("You are not authorized to update this Comment")

④ get Comment content from req.body and validate its not empty.
↳ (content.trim() === "") ✗ → "Content is req."

⑤ update the Comment with new content
↳ Comment.findByIdAndUpdate(CommentId, {content}, {new: true})

⑥ ↳ (!updateComment) → "Comment not updated"

⑦ return response

(C) Delete Comment

① get CommentId from Params and validate ObjectId
↳ (req.Params) → isValidObjectId() ✓

② find Comment by CommentId from Comments Collection
↳ (!Comment) → "Comment not found"

③ check if logged in user is the owner
↳ (Comment?.owner?.toString() !== req.user?._id.toString())
↳ Error ("Not authorized")

④ deleteComment from database
↳ Comment.findByIdAndDelete(commentId)
⑤ ↳ (!deleteComment) → "Couldn't delete Comment"
⑥ return response.

★ ⑦ Get Video Comments

① get videoId from Params
↳ (req.Params) → isValidObjectId() ✓
② get Page and Limit from query parameters.
↳ const { Page = 1, Limit = 10 } = req.query
③ CommentAggregate
↳ $match : { Video : new Mongoose.Types.ObjectId(videoId) }
↳ $lookup: { from : "users",
localField : "owner",
foreignField : "_id",
as : "owner",
↳ Pipeline: [$Project: → _id, username, avatar]
↳ $addfield : owner : → $first : $owner
↳ $Sort : CreatedAt : 1
④ options for aggregatePaginate
↳ { Page : ParseInt(Page), Limit : ParseInt(Limit), Custom label
↳ totalDocs → totalComments
↳ ~~Docs~~ docs → comments
⑤ Comment.aggregatePaginate(CommentAggregate (step 3), options)
↳ (err, result) → if (err)
↳ throw new Error
↳ return result with status 200 with Succ
↳ if (result.totalComments === 0)
↳ return with msg → "NO comment found

Date: Page:

(5) Playlist · Controller

(A) Create Playlist

1. get { name, description } from req.body
2. check if name and description is provided (and make sure their value is not empty.
3. find the user from req.user?._id to validate user
   - ↳ (!user) → user not found
4. Create CreatedPlaylist from name, description & owner: req.user?._id
   - ↳ (!createdPlaylist) → Playlis coun't be created.
5. return response.

(B) AddVideoTo Playlist

1. get { PlaylisId, VideoId } from req.Params;
2. check PlaylistId, VideoId are valid mongodb objectIds
   - ↳ (!isValidObjectId(PlaylistId) || !isValidObjectId(VideoId)),
     - ↳ Invalid playlist id or video Id.
2. check if Playlist exists in Playlists collection by PlaylisId.
3. check if video exists in videos collection by VideoId
4. check if user is authenticated by req.user?._id
5. check if current logedIn.user is owner of Playlist.
   - ↳ (user._id.toString() !== Playlist.owner.toStrings)
     - ↳ unauthorized acces.
6. check if video is already in Playlist
   - ↳ Playlist.video.includes(VideoId)
     - ↳ Video already in Playlist.

Date Page

⑦ Add video to Playlist
↳ await Playlist.findByIdAndUpdate(PlaylistId, { $Push: { video: videoId } }, {new: true})

⑧ return response.

⑨ removeVideoFromPlaylist

→ follow till Step ⑤ from AddVideoToPlaylist

⑥ check if video is in Playlist
↳ (! Playlist.video.includes(VideoId)) → Error - "video not in Playlist"

⑦ Remove video from Playlist
↳ findByIdAndUpdate(PlaylistId, { $Pull: { video: videoId } }, {new: true});

⑧ Return response.

⑩ Delete Playlist

① get {PlaylistId} from req.Params.
② check if PlaylistId is valid mongodb Id → (isValidObjectId) ✓
③ check if Playlist exists in Playlists Collection by PlaylistId
④ check if user is authenticated by req.user?._id
⑤ check if user is authorized to delete playlist.
⑥ Delete Playlist by PlaylistId.
⑦ return response

⑪ Update Playlist

① get {Playlis Id} from req.query → isValid ObjectId ✓
get {name, discriptim} → req.body.

Date: _____ Page: _____

(3) check if name & description is provided
(3) check if Playlist exists in playlists collection via (PlaylistId)
(4) check if user is authenticated via req.user?._id
(5) check if user is authorized to update Playlist
(6) update playlist
(7) return response

(F) getUserPlaylists

(1) get {userId} from req.Params.
↳ isValidObjectId → ✓
(2) check if user exists via userId
(3) Playlist.aggregate → $(playlists)
↳ $match → owner : userId
↳ $lookup → from : videos as {video}
↳ $lookup → from users as VideoOwner
↳ $project → username, avatar, fullname.
↳ $addfield → VideoOwner : {$first : '$VideoOwner'}
↳ $Project → {thumbnail : "$thumbnail.url"
title , duration , views , description,
VideoFile : "$videoFile.url",
VideoOwner : "$VideoOwner" }
↳ ~~$addfield~~ → video : "$video"
↳ $addfield → totalVideos : { $size : "$video" }
(4) if (!Playlists) → Error → "Error fetching user playlist"
(5) return response

(G) getPlaylistById

Date: Page:

① get { playlistId } from req.Params → isValidObjectId ✓

② check if Playlist exists in playlists collection via PlaylistId

③ Playlist.aggregate - (Playlists)

↳ $match → _id : new mongoose.Types.ObjectId(playlistId)

↳ $lookup → from videos as {video}

↳ $lookup → from users as {VideoOwner}

↳ $Project → username, avatar, fullname

↳ $addfield → videoOwner → $first : $videoOwner

↳ $Project → {thumbnail: "$thumbnail.url", title, duration, videofile: "$videofile", views, description, videoOwner : "$videoOwner" }

↳ $addfield → totalVideos → $size : "$video"

↳ $addfield → video : "$video"

↳ $lookup → from users as "PlaylistOwner"

↳ $Project → username, avatar, fullname.

↳ $addfield → PlaylistOwner : "$PlaylistOwner"

↳ $unset → "owner" //removes the "owner" field from Playlist document

④ Check if Playlists were found.

↳ (!Playlist) → Error fetching Playlists"

⑤ return first value from Playlists array as response.

=

⑥ tweet.controller.js

Ⓐ CreateTweet

Date: Page:

① get {content} from req.body
② check if user is authonticated by req.user?._id
③ check if content is provided
↳ (!content || content.trim() === "") → X
④ create tweet
↳ {owner : req.user?._id, content}
⑤ return response.

(B) Get User Tweets

① get userId from req.parans
② get {Page = 1, limit = 10} from req.query
③ check if userId is valid objectId
↳ isValidObjectId(userId) → ✓
④ find & authenticate user by req.user?._id
⑤ tweetAggregation Pipeline
↳ $match → owner : userId
↳ $lookup → from:users as:owner
↳ $project → {email, username, fullname, avatar}
↳ $addField → owner → {$first : "$owner"}
↳ $Sort → CreatedAt : -1,
⑥ options for aggregatePaginate
↳ {Page : ParseInt(Page), limit: ParseInt(limit),
CustomLabels : { totalDocs : "TweetCounts",
docs : "Tweets" }}
⑦ Tweet.aggregatePaginate(tweetAggregation(steps), options)
⑧ if(!) → "Some Internal error occured" → Error
⑨ return response

Date: Page:

(C) Update Tweets

1. Get {tweetId} from req.Params
2. get {Content} from req.body
3. Check if tweetId is valid objectId → isValid.ObjectId → ✓
4. Check if Provided Content is not empty (if Content is provided)
5. check if tweet exists in Collection by tweetId.
6. authenticate if user exists from req.user?._id
7. Validate if user is owner of tweet
   - tweet.owner.toString() !== req.user._id.toString()
     - "Not authorized.
8. find and update tweet.
9. return response.

(D) delete Tweet

1. Get {tweetId} from req.Params.
2. Check if {tweetId} is ValidObjectId. → ✓
3. Check if tweet exists in collection via tweetId
4. authenticate if user Exists via req.user?._id
5. Validate if users is owner of this tweet
   - tweet.owner.toString() !== req.user._id.toString()
     - "Not authorized"
6. find and delete Tweet
7. return response.

Date: Page:

like.controller

(A) Toggle VideoLike

① Get {videoId} from req.Params → isValidObjectId → ✓
② check if video exists in video collection via videoId
③ check if video already liked by user
↳ likedVideo = Like.findOne({video: videoId})
④ togglelike → (isliked)
↳ likedVideo ? Like.deleteOne(likedVideo) : Like.create({
video: videoId,
likedBy: req.user?.id
})
⑤ return response

(B) ToggleCommentLike → Similar as "ToggleVideoLike"

(C) ToggleTweetLike → Similar as "ToggleVideoLike"

(D) GetLikedVideos

① find and authenticate user via req.user?._id
② likedVideos → aggregation Pipeline
↳ $match → likedBy → req.user?._id
↳ $lookup → from: videos, as: video, Lo.Fi: video, FF: _id
↳ $lookup → from: users as: owner
↳ $Project: { username, fullname, avatar }
↳ $addfield → owner → $first: "$owner"
↳ $Project → {title, description, thumbnail, owner }
↳ $unwind: "$video"
↳ $replaceRoot: { newRoot: "$video" }

Date: _____ Page: _____

→ $replaceRoot :→ is a Stage in mongoDB's aggregation pipeline that replace the current document with a Specified embedded document. This can be useful when you want to promote a Subdocument or a field within document to the top level.

- In Simple terms-
- Before '$replaceRoot' : you have a document with a nested Structure.
- After '$replaceRoot' : you change the document structure so that a nested document becomes the main document.

⑧ dashboard Controller

(A) getChannel Videos

① get user from "req.user?._id"
② Videos → aggregation Pipeline on Video
↳ $match : {owner : ..(req.user?._id)
↳ $project : { title, discription, thumbnail, videofile,
views, duration, isPublished }
③ return response

(B) getChannel Status

① get user and validate via req.user?._id
② channelStatus → aggregation Pipeline on users
↳ $match : {_id : ...(req.user._id)
↳ $lookup : from : video, LF : _id, FF : owner, as : TotalVideos
↳ $lookup : from : likes, LF : _id, FF : Video, as : VideoLikes
↳ $lookup : from : comments, LF : _id, FF : video, as : TotalComments.
↳ $addfields : viddilkes : { $first : "$VideoLikes" }
↳ $addfields : { totalComments : { $size : "$TotalComments" } }
↳ $lookup : from : Subscriptions, LF : _id, FF : channel, as : Subscribers
↳ $lookup : from : Subscriptions, LF : _id, FF : Subscriber, as : SubscribedTo
↳ $lookup : from : tweet, LF : _id, FF : owner, as : tweet
↳ $lookup : from : likes, LF : _id, FF : tweet, as : TweetLikes.
↳ $addfield : TweetLikes : $first : "$TweetLikes"

Date: Page:

↳ $lookup: from: comments, LF: _id, FF: owner, as: comments.
↳ $lookup: from: "likes", LF: _id, FF: Comment, as: "commentlikes"
↳ $addFields: ~~from~~ "commentlikes": { $first: "$commentLikes"

↳ $Project
↳ {username, email, fullname, avatar,
TotalComments: { $sum: "$TotalVideos.TotalComments" },
TotalViews: { $sum: "$TotalVideos.Views" },
TotalVideos: { $size: "$TotalVideos" },
Subscribers: { $size: "$subscribers" },
SubscribedTo: { $size: "$SubscribedTo" }
TotalTweets: { $size: "$tweets" },
TotalLikes: {
VideoLikes: { $size: "$TotalVideos.videoLikes" },
tweetlikes: { $size: "$tweets.TweetLikes" },
commentlikes: { $size: "$comments.commentlikes" },
total: { $sum: [{ $size: "$TotalVideos.videoLikes" },
{ $size: "$tweets.TweetLikes" },
{ $size: "$comments.commentlikes" }] },
}
}

③ return response —