Vectre: Sprint 4 Documentation

Table of Contents

Login Flow

1. **(Frontend)** User presses “Connect with Metamask” button @ `*FRONTEND_BASE_URL/login*`
    a. Sends wallet address from Metamask in request body to `*BACKEND_BASE_URL/users/login/nonce*`
2. **(Backend)** `POST */users/login/nonce*` returns user.nonce from User specified by `*walletAddress*`
3. **(Frontend)** Received nonce from backend. Prompts user to sign message containing nonce with Metamask
    a. Sends wallet address signed message from Metamask in request to `*BACKEND_BASE_URL/users/login*`
4. **(Backend)** POST */users/login* verifies signed message with Metamask
    a. Generates JWT (JSON Web Token) for User specified by `*walletAddress*`
    b. Updates *user.nonce* with a (new) randomly generated nonce
    c. Returns JWT token in an HttpOnly cookie (token expires after 7 days)
5. **(Frontend)** Stores JWT cookie
    a. Cookie is sent with all requests to authorize User access to restricted endpoints

Neo4j Queries

User

```
DROP CONSTRAINT user_properties_wallet_address IF EXISTS;
DROP CONSTRAINT user_properties_wallet_address_unique IF EXISTS;

CREATE CONSTRAINT user_properties_walletAddress IF NOT EXISTS
FOR (user:User)
REQUIRE user.walletAddress IS NOT NULL;
CREATE CONSTRAINT user_properties_walletAddress_unique IF NOT EXISTS
FOR (user:User)
REQUIRE user.walletAddress IS UNIQUE;
CREATE CONSTRAINT user_properties_username_unique IF NOT EXISTS
FOR (user:User)
REQUIRE user.username IS UNIQUE;

CREATE CONSTRAINT user_properties_username IF NOT EXISTS
FOR (user:User)
REQUIRE user.username IS NOT NULL;
CREATE CONSTRAINT user_properties_username_unique IF NOT EXISTS
FOR (user:User)
REQUIRE user.username IS UNIQUE;

CREATE CONSTRAINT user_properties_name IF NOT EXISTS
FOR (user:User)
REQUIRE user.name IS NOT NULL;

CREATE CONSTRAINT user_properties_nonce IF NOT EXISTS
FOR (user:User)
REQUIRE user.nonce IS NOT NULL;
```

```
MATCH (user:User)
OPTIONAL MATCH (user)<-[f:FOLLOWS]-(follower: User)
WITH user, count(f) AS followerCount
SET user.initialWeeklyFollowers = followerCount;
```

Post

```
CREATE CONSTRAINT post_properties_postID IF NOT EXISTS
FOR (post:Post)
REQUIRE post.postID IS NOT NULL;

CREATE CONSTRAINT post_properties_postID_unique IF NOT EXISTS
FOR (post:Post)
REQUIRE post.postID IS UNIQUE;

CREATE CONSTRAINT post_properties_timestamp IF NOT EXISTS
FOR (post:Post)
REQUIRE post.timestamp IS NOT NULL;

CREATE CONSTRAINT post_properties_text IF NOT EXISTS
FOR (post:Post)
REQUIRE post.text IS NOT NULL;

CREATE CONSTRAINT post_properties_edited IF NOT EXISTS
FOR (post:Post)
REQUIRE post.edited IS NOT NULL;
```

Notification

```
CREATE CONSTRAINT notification_properties_notificationID IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.notificationID IS NOT NULL;

CREATE CONSTRAINT notification_properties_notificationID IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.notificationID IS UNIQUE;

CREATE CONSTRAINT notification_properties_toUser IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.toUser IS NOT NULL;

CREATE CONSTRAINT notification_properties_fromUser IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.fromUser IS NOT NULL;

CREATE CONSTRAINT notification_properties_action IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.action IS NOT NULL;

CREATE CONSTRAINT notification_properties_read IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.read IS NOT NULL;

CREATE CONSTRAINT notification_properties_timestamp IF NOT EXISTS
FOR (notif:Notification)
REQUIRE notif.timestamp IS NOT NULL;
```

Community

```
CREATE CONSTRAINT community_properties_communityID IF NOT EXISTS
FOR (community: Community)
REQUIRE community.communityID IS NOT NULL;

CREATE CONSTRAINT community_properties_communityID_unique IF NOT EXISTS
FOR (community: Community)
REQUIRE community.communityID IS UNIQUE;

CREATE CONSTRAINT community_properties_name IF NOT EXISTS
FOR (community: Community)
REQUIRE community.name IS NOT NULL;
```

```
MATCH (c:Community) SET c.initialWeeklyMemberCount=c.memberCount RETURN c;
MATCH (u:User)-[owns:OWNS]->(c:Community) DELETE owns;
```

Node Properties

Notification

Property	Constraints
notificationID	NOT NULL, UNIQUE
toUser	NOT NULL
fromUser	NOT NULL
action	NOT NULL
postID	
read	NOT NULL
timestamp	NOT NULL

Post

Property	Constraints
postID	NOT NULL, UNIQUE
author	NOT NULL
text	NOT NULL
imageURL	NOT NULL
edited	NOT NULL
timestamp	NOT NULL
parent	
repostPostID	

User

Property	Constraints
walletAddress	NOT NULL, UNIQUE
username	NOT NULL, UNIQUE
name	NOT NULL
nonce	NOT NULL
bio	
profilePic	
banner	
dashboard	

Community

Property	Constraints
communityID	NOT NULL, UNIQUE
name	NOT NULL
bio	
profilePic	
banner	

Endpoints

Users

GET /users	
Description	Returns all User nodes
Authentication/ Authorization	None
Request Body	None

Response Body	`{`
	`  success: true,`
	`  users: [user1, user2, ...]`
	`}`

GET /users/{walletAddress}	
Description	Returns User node with specified walletAddress
Authentication/ Authorization	None
Request Body	None

Response Body	`{`
	`  success: true,`
	`  user: queried_user`
	`}`

GET /users/search/{searchVal}	
Description	Returns users with name/username/walletAddress fields containing the case insensitive searchVal (walletAddress is only searched if searchVal starts with “0x”)
Authentication/ Authorization	None
Request Body	None

Response Body	`{`
	`  success: true,`
	`  users: [user1, user2, ...]`
	`}`

<table>
<tr><th colspan="2">GET /users/{walletAddress}/posts</th></tr>
<tr><td><b>Description</b></td><td>Retrieves all posts made by user with specific wallet address</td></tr>
<tr><td><b>Authentication/ Authorization</b></td><td>None</td></tr>
<tr><td><b>Request Body</b></td><td>None</td></tr>
<tr><td><b>Response Body</b></td><td><pre>
{
  success: true,
  posts: [post1, post2, ...]
}
// Post object:
{
  author: wallet address of poster
  text: text contents of post
  imageURL: image contents of post, can be empty string
  edited: boolean, true iff post was updated
  timestamp: UTC timestamp
}

{
  success: false,
  message: specific error message regarding issue
}
</pre></td></tr>
</table>

<table>
<tr><th colspan="2">GET /users/{walletAddress}/communities</th></tr>
<tr><td><b>Description</b></td><td>Returns the communities that ther user is a member of.</td></tr>
<tr><td><b>Authentication/ Authorization</b></td><td>None</td></tr>
<tr><td><b>Request Body</b></td><td>None</td></tr>
<tr><td><b>Response Body</b></td><td><pre>
{
  success: true,
  communties: [community1, communit2, ...]
}

{
  success: false,
  message: “User does not exist.”
</pre></td></tr>
</table>

	}

POST /users/register	
Description	Creates User from request body data
Authentication/ Authorization	None

Request Body	{
	  walletAddress: "0x93asdf89uy930304ad",
	  username: "example_username",
	  name: "Example User",
	  bio: ""
	}

Response Body	{
	  success: true,
	  user: new_user
	}

POST /users/login/nonce	
Description	Returns nonce from User specified in request body
Authentication/ Authorization	None

Request Body	{
	  walletAddress: "0x93asdf89uy930304ad"
	}

Response Body	{
	  success: true,
	  nonce: "938483"
	}

POST /users/login	
Description	Validates signed nonce with Metamask and returns specified User’s JWT in a cookie
Authentication/ Authorization	None

Request Body	`{`
	`  walletAddress: "0x93asdf89uy930304ad",`
	`  signedNonce: "34809a8dfajdkf"`
	`}`

Response Body	`{`
	`  success: true,`
	`  authorizationToken: "yJuYW1lIjoiSm9lIENvZGVyI"`
	`}`

Cookies	- “token”: “yJuYW1lIjoiSm9lIENvZGVyI” - HttpOnly - Expires in 7 days

GET /users/login/currentUser	
Description	Returns User node logged in with cookie JWT
Authentication/ Authorization	Logged in
Request Body	None

Response Body	`{`
	`  success: true,`
	`  user: loggedInUser`
	`}`

PUT /users/{walletAddress}/update	
Description	Updates profile properties of User node with specified walletAddress
Authentication/ Authorization	Logged in & walletAddress must match

Request Body	`{`
	`  username: "updatedUsername",`
	`  name: "Updated User",`
	`  bio: "Cheese pizza is my favourite",`
	`  profilePic: *base64 image data encoding, // *OPTIONAL`
	`  banner: *base64 image data encoding, // *OPTIONAL`
	`}`

Response Body	`{`
	`  success: true,`
	`  Message: "Edit success."`
	`}`

DELETE /users/{walletAddress}/delete	
Description	Delete User node with specified walletAddress
Authentication/ Authorization	Logged in & walletAddress must match
Request Body	None

Response Body	`{`
	`  success: true,`

	message: "Deleted User" }

GET /users/{walletAddress}/nft	
Description	Returns the list of NFTs a user owns within the given wallet, and validates the NFT avatar.
Authentication/ Authorization	None
Request Body	None
Response Body	{ "success": true, "nft": [{ "tokenID": 77809613, "name": "Ryder Ripps Bored Ape Yacht Club", "imageURL": "https://lh3.googleusercontent.com/uCKMr5LZBAfr49-dFuMrWc903x-u4NxBpPywwLjLS9DLMWZDfmjPJW6v", "contractAddress": "0x15545614507f46d954ab1f9c472e26506a99c5f8" }, { "tokenID": 77809597, "name": "Ryder Ripps Bored Ape Yacht Club", "imageURL": "https://lh3.googleusercontent.com/HHMNNmiLf_8HCmaVXT1cfX2uS1ATeAg8sATVqvEw4mwa", "contractAddress": "0x15545614507f46d954ab1f9c472e26506a99c5f8" }, { "tokenID": 77809576, "name": "Ryder Ripps Bored Ape Yacht Club", "imageURL": "https://lh3.googleusercontent.com/RTvkOt5Yykay8LuCzy4Ep9UsTaOotYr5lBvpu_oEGoe", "contractAddress": "0x15545614507f46d954ab1f9c472e26506a99c5f8" },],

	"message": "Successfully retrieved NFTs for user with wallet address 0x749C89F7F6054C8CD4a982c93E3E05e996BD5C19" }

GET /users/funds	
Description	Returns the wallet funds for a specific walletAddress.
Authentication/ Authorization	Yes, authenticated. User must be logged in.
Request Body	None

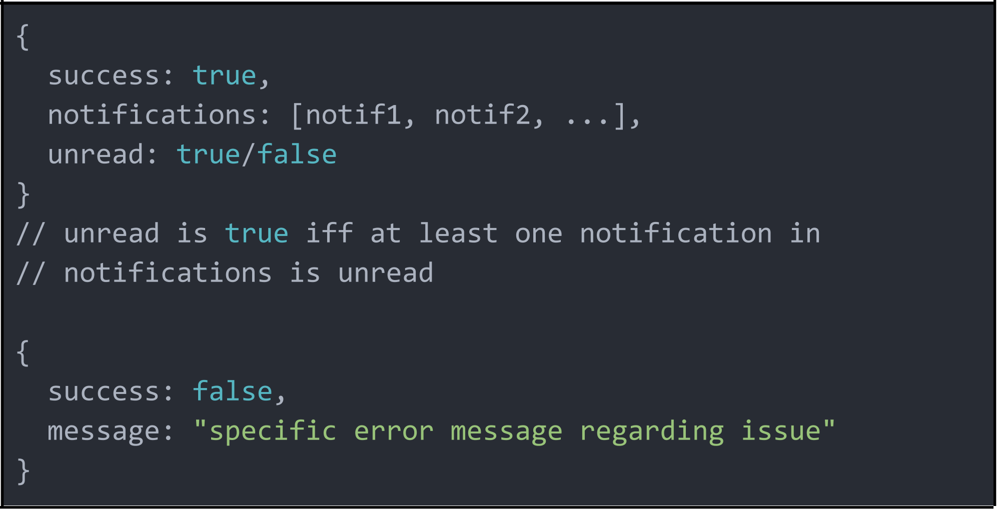
Response Body	Successful:
	{
	"success": true,
	"funds": 1.111133923,
	“message”: “Successfully retrieved user's wallet fund”
	}
	
	Failed:
	{
	"success": false,
	"error": “Wallet address does not exist”,
	“message”: “Failed to retrieve wallet funds.”
	}

POST /users/{walletAddress}/updateDashboard	
Description	Updates dashboard field of User node with specified walletAddress
Authentication/ Authorization	Logged in
Request Body	{ walletAddress: “0x93asdf89uy930304ad”, dashboard: “ [{ "image_url": "https://uploads-ssl.webflow.com/a19_imagesloaded.jpg", "collection_name": "Doodles", "asset_contract": "0x23e700f9b556651f5c9bead14bd5c63200178b13" "token_id": "3690", }, {

	"image_url": "https://uploads-ssl.webflow.com/a19_imagesloaded.jpg", "collection_name": "Doodles", "asset_contract": "0x23e700f9b556651f5c9bead14bd5c63200178b13" "token_id": "3692", }, ….] }

Response Body	{
	"success": true,
	"user": {
	"id": "3",
	"walletAddress":
	"0xD6Fdc7C527844c62e85a76c03e2F2142c82AeDBf",
	"username": "horsekingu",
	"name": "horseking",
	"bio": "coolest kid ever",
	"dashboard": "help update my dashboard ples"
	}
	}

GET /users/{walletAddress}/notifications	
Description	Retrieves all notifications addressed to user with specific wallet address
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	
	{
	  success: true,
	  notifications: [notif1, notif2, ...],
	  unread: true/false
	}
	// unread is true iff at least one notification in
	// notifications is unread
	
	{
	  success: false,
	  message: "specific error message regarding issue"
	}

GET /users/{walletAddress}/followers

Description	Retrieves all users that follow user with specified wallet address
Authentication/ Authorization	None
Request Body	None

Response Body	`{`
	`  success: true,`
	`  followers: [follower1, follower2, ...]`
	`}`
	
	`{`
	`  success: false,`
	`  message: "specific error message regarding issue"`
	`}`

GET /users/{walletAddress}/following	
Description	Retrieves all users that user with specified wallet address follows
Authentication/ Authorization	None
Request Body	None

Response Body	`{`
	`  success: true,`
	`  following: [followingUser1, followingUser2, ...]`
	`}`
	
	`{`
	`  success: false,`
	`  message: "specific error message regarding issue"`
	`}`

POST /users/{walletAddressToFollow}/follow

Description	Creates a follow relationship between logged in user and user with specified wallet address to follow
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: "Successfully followed user"`
	`}`
	
	`{`
	`  success: false,`
	`  message: "specific error message regarding issue"`
	`}`

POST /users/{walletAddressToUnfollow}/unfollow	
Description	Deletes a follow relationship between logged in user and user with specified wallet address to unfollow
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: "Successfully unfollowed user"`
	`}`
	
	`{`
	`  success: false,`
	`  message: "specific error message regarding issue"`
	`}`

Posts

<table>
<tr><th colspan="2">GET /posts/{postID}</th></tr>
<tr><td><b>Description</b></td><td>Retrieve post specified by postID</td></tr>
<tr><td><b>Authentication/ Authorization</b></td><td>None</td></tr>
<tr><td><b>Request Body</b></td><td>None</td></tr>
<tr><td><b>Response Body</b></td><td>

```
{
  success: true,
  post: postWithPostID
}

{
  success: false,
  message: "specific error message regarding issue"
}
```

</td></tr>
</table>

<table>
<tr><th colspan="2">POST /posts/create</th></tr>
<tr><td><b>Description</b></td><td>Create a new post for User with specified walletAddress</td></tr>
<tr><td><b>Authentication/ Authorization</b></td><td>Logged in</td></tr>
<tr><td><b>Request Body</b></td><td>

```
{
  text: "text contents of the post",
  imageData: *base64 image data encoding, // *OPTIONAL
  repostPostID: "optional postID of post to repost"
}
```

</td></tr>
<tr><td><b>Response Body</b></td><td>

```
{
  success: true,
  message: "Successfully created post/repost"
  postID: "new postID"
}

{
  success: false,
  message: "Failed to create post/repost"
}
```

</td></tr>
</table>

POST /posts/{postID}/update	
Description	Updates an already existing post object with the new information
Authentication/ Authorization	Logged in & walletAddress must match

Request Body	`{`
	`  text: "text contents of the post",`
	`  imageURL: "image contents of the post, can be empty",`
	`}`

Response Body	`{`
	`  success: true,`
	`  message: "Post updated successfully"`
	`}`
	
	`{`
	`  success: false,`
	`  message: error message regarding specific issue`
	`}`

POST /posts/{postID}/like	
Description	Creates a “LIKED” relationship from user to post
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: "Successfully liked post"`
	`}`
	`{`
	`  success: false,`
	`  message: "Failed to like post",`
	`  error: error`
	`}`

POST /posts/{postID}/unlike	
Description	Removes a “LIKED” relationship from user to post if it exists
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: "Successfully unliked post"`
	`}`
	`{`
	`  success: false,`
	`  message: "Failed to unlike post",`
	`  error: error`
	`}`

GET /posts/{postID}/checkLike	
Description	Checks whether current user has liked post with specified postID
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	`{`
	`  success: true,`
	`  alreadyLiked: true/false`
	`}`
	
	`{`
	`  success: false,`
	`  message: "Failed to check if post was already liked",`
	`  error: error`
	`}`

GET /posts/{postID}/likes	
Description	Gets all users that liked a post
Authentication/ Authorization	None
Request Body	None

Response Body	{
	  success: true,
	  users: [user1, user2,..]
	}
	
	{
	  success: false,
	  message: "Failed to get users",
	}

POST /posts/feed	
Description	Gets feed of posts for a user
Authentication/ Authorization	Logged in user

Request Body	{
	  start: 1, // *OPTIONAL, default value is 0
	  size: 5, // *OPTIONAL, default value is 10
	}

Response Body	{
	  success: true,
	  posts: [post1, post2, ..]
	}
	
	{
	  success: false,
	  message: "Failed to get posts",
	  error: error
	}

POST /posts/search/{searchVal}	
Description	Returns posts with text field containing the case insensitive searchVal
Authentication/ Authorization	None

Request Body	`{`
	`  start: 1, // *OPTIONAL, default value is 0`
	`  size: 5, // *OPTIONAL, default value is 10`
	`}`

Response Body	`{`
	`  success: true,`
	`  posts: [post1, post2, ..]`
	`}`
	`{`
	`  success: false,`
	`  message: "Failed to get posts",`
	`  error: error`
	`}`

DELETE /posts/{postID}	
Description	Allow author to delete their post with the matching `postID`.
Authentication/ Authorization	Current logged in user.
Request Body	`none`

Response Body	`{`
	`  success: true,`
	`  message: “Successfully deleted post.”`
	`}`
	`{`
	`  success: false,`
	`  message: “User is not the author of the post.”`
	`}`
	`{`
	`  success: false,`
	`  message: "Failed to delete post",`
	`  error: error`
	`}`

Comments (Posts)

GET /posts/{postID}/comments	
Description	Retrieve all comments associated to the post specified by postID
Authentication/ Authorization	None
Request Body	None

Response Body	{
	  success: true,
	  comments: [comment1, comment2, ...]
	}
	
	{
	  success: false,
	  message: "specific error message regarding issue"
	}

POST /posts/create/{postID}/comment	
Description	Create comment specified by the body for the post specified by postID
Authentication/ Authorization	Logged in user

Request Body	{
	  "text": "specific text content",
	  "parent": "the postID of the post to comment on"
	}

Response Body	{
	  success: true,
	  posts: [comment1, comment2, ...]
	}
	
	{
	  success: false,
	  message: "specific error message regarding issue"
	}

Notifications

POST /notifications/{notificationID}/read	
Description	Sets field \`read=true\` for notification with specified notificationID
Authentication/ Authorization	Logged in user
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: "Successfully read notification"`
	`}`
	
	`{`
	`  success: false,`
	`  message: "specific error message regarding issue"`
	`}`

Communities

GET /communities/search/{searchVal}	
Description	Returns communities with name/bio/communityID fields containing the case insensitive searchVal
Authentication/ Authorization	None
Request Body	None

Response Body	{
	  success: true,
	  communities: [community1, community2, ...]
	}

GET /communities/{communityID}	
Description	Returns a community with matching communityID
Authentication/ Authorization	None
Request Body	None

Response Body	{
	  success: true,
	  community: community
	}
	
	{
	  success: false,
	  message: “Community does not exist.”
	}
	
	{
	  success: false,
	  message: “Failed to fetch community.”,
	  error: error_message
	}

GET /communities/getAll	
Description	Returns all communities on the platform.
Authentication/ Authorization	None
Request Body	None

Response Body	{
	  success: true,
	  communities: [community1, community2, ...]
	}

GET /communities/{communityID}/members/{walletAddress}/roles	
Description	Returns the roles of User with matching \`walletAddress\` in Community with matching \`communityID\`
Authentication/ Authorization	None
Request Body	None

Response Body	{
	  success: true,
	  roles: [member, …]
	}
	
	{
	  success: false,
	  message: “User or Community does not exist.”
	}
	
	{
	  success: false,
	  message: “Failed to all roles of User in Community.”,
	  error: error_message
	}

POST /communities/create	
Description	Create a new community with the current login user as the Owner.
Authentication/ Authorization	Logged in User

Request Body	{
	  communityID: “uniqueID”,
	  name: “communityName”,
	  bio: <optional> “bio”,
	  profilePic: <optional> “image link”,
	  banner: <optional> “image link”,
	  discordLink: <optional> “link”,
	  instagramLink: <optional> “link”,
	  twitterLink: <optional> “link”,
	  websiteLink: <optional> “link”,
	  ethLink: <optional> “link”,
	}

Response Body	{
	  success: true,
	  message: “Successfully created a community.”,
	  communityID: communityID
	}
	
	{
	  success: false,
	  message: <reason why creation failed>
	}
	
	{
	  success: false,
	  message: “Failed to fetch community.”,
	  error: error_message
	}

PUT /communities/{communityID}/update	
Description	Update a community matches provided `communityID` if the logged in user has the permission to do so.
Authentication/ Authorization	Logged in User

Request Body

```
{
  communityID: “uniqueID”,
  name: “communityName”,
  bio: <optional> “bio”,
  profilePic: <optional> “image link”,
  banner: <optional> “image link”,
  discordLink: <optional> “link”,
  instagramLink: <optional> “link”,
  twitterLink: <optional> “link”,
  websiteLink: <optional> “link”,
  ethLink: <optional> “link”,
}
```

Response Body

```
{
  success: true,
  message: “Successfully updated community.”,
  communityID: communityID
}

{
  success: false,
  message: <Reason why update failed.>
}

{
  success: false,
  message: “Failed to fetch community.”,
  error: error_message
}
```

POST /communities/{communityID}/join	
Description	Add the current logged in user as a Member of community with matching \`communityID\`
Authentication/ Authorization	Logged in User
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: “Successfully joined community.”`
	`}`
	
	`{`
	`  success: false,`
	`  message: <Reason why user cannot join community.>`
	`}`
	
	`{`
	`  success: false,`
	`  message: “Failed to fetch community.”,`
	`  error: error_message`
	`}`

POST /communities/{communityID}/leave	
Description	Remove the current logged in user from community with matching \`communityID\`
Authentication/ Authorization	Logged in User
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: “Successfully leaved community.”`
	`}`
	
	`{`
	`  success: false,`
	`  message: <Reason why leave action is not performed.>`
	`}`

	{
	  success: false,
	  message: “Failed to fetch community.”,
	  error: error_message
	}

POST /communities/{communityID}/feed	
Description	Gets feed of posts for a community
Authentication/ Authorization	None

Request Body	{
	  start: 1, // *OPTIONAL, default value is 0
	  size: 5, // *OPTIONAL, default value is 10
	  sort: “timestamp” // *OPTIONAL, can be timestamp or likes
	  order: “asc” // *OPTIONAL, can be asc or desc
	}

Response Body	{
	  success: true,
	  posts: [post1, post2, ..]
	}
	
	{
	  success: false,
	  message: "Failed to get posts",
	  error: error
	}

POST /communities/{communityID}/promote/{walletAddress}	
Description	An endpoint for community moderator to promote a member with matching \`walletAddress\` to moderator status inside the community with matching \`communityID\`. If the current logged in user is not moderator of said community, then the endpoint returns success as false.
Authentication/ Authorization	Logged in User
Request Body	None

Response Body	{
	  success: true,
	  message: “Successfully leaved community.”
	}
	
	{
	  success: false,
	  message: “User does not have the permission to promote Member.”
	}
	
	{
	  success: false,
	  message: <Reason why leave action is not performed.>
	}
	
	{
	  success: false,
	  message: “Failed to fetch community.”,
	  error: error_message
	}

POST /communities/{communityID}/ban/{walletAddress}	
Description	An endpoint for community moderator to ban a member with matching \`walletAddress\` from interacting with posts inside the community with matching \`communityID\`. If the current logged in user is not moderator of said community, then the endpoint returns success as false. Banning a member also deletes the posts of that member inside the community.
Authentication/ Authorization	Logged in User
Request Body	None

Response Body	{
	  success: true,
	  message: “Successfully leaved community.”
	}
	
	{
	  success: false,
	  message: “User does not have the permission to ban Member.”
	}
	
	{
	  success: false,
	  message: <Reason why leave action is not performed.>
	}
	
	{
	  success: false,
	  message: “Failed to fetch community.”,
	  error: error_message
	}

POST /communities/{communityID}/unban/{walletAddress}	
Description	An endpoint for community moderator to unban a member with matching \`walletAddress\` from interacting with posts inside the community with matching \`communityID\`. If the current logged in user is not moderator of said community, then the endpoint returns success as false.
Authentication/ Authorization	Logged in User
Request Body	None

Response Body	{
	success: true,
	message: “Successfully leaved community.”
	}
	
	{
	success: false,
	message: “User does not have the permission to unban Member.”
	}
	
	{
	success: false,
	message: <Reason why leave action is not performed.>
	}
	
	{
	success: false,
	message: “Failed to fetch community.”,
	error: error_message
	}

POST /communities/{communityID}/delete/{postID}	
Description	An endpoint for community moderator to delete a post within their community with matching `postID`. If the current logged in user is not moderator of said community, then the endpoint returns success as false.
Authentication/ Authorization	Logged in User
Request Body	None

Response Body	`{`
	`  success: true,`
	`  message: “Successfully deleted post.”`
	`}`
	
	`{`
	`  success: false,`
	`  message: “User does not have the permission to delete post”`
	`}`
	
	`{`
	`  success: false,`
	`  message: <Reason why leave action is not performed.>`
	`}`
	
	`{`
	`  success: false,`
	`  message: “Failed to delete post.”,`
	`  error: error_message`
	`}`

GET /communities/trending	
Description	Returns a list of communities that are most popular since the last Sunday
Authentication/ Authorization	None
Request Body	None

Response Body	`{`
	`  success: true,`
	`  communities: [community1, community2, ...]`
	`}`

Frontend Components

Wrappers

AppWrapper	
Path	`/vectre/frontend/src/components/AppWrapper/AppWrapper.js`
Description	A wrapper for pages with a predefined background and navigation bar. This allows for reusability.
Props	None
Children	Anything that is reasonable and is logically within a page.

Usage/Example	`<AppWrapper>`
	`    <Post />`
	`    <UserCommentInput />`
	`    <Comments />`
	`</AppWrapper>`

ButtonLinkWrapper	
Path	`/vectre/frontend/src/components/Buttons/ButtonLinkWrapper/ButtonLinkWrapper.js`
Description	A wrapper for buttons that may require a Link tag from Chakra UI.
Props	href
Children	Any Button that would require a link.

Usage/Example	`<ButtonLinkWrapper`
	`    href={"/home"}>`
	`    <Button />`
	`</ButtonLinkWrapper>`

ContentWithCommunityButtons	
Path	`/vectre/frontend/src/components/Buttons/ButtonLinkWrapper/ButtonLinkWrapper.js`
Description	A wrapper that is used for wrapping content in home page with a list of buttons associated with the communities on the left side
Props	sideButtonsList
Children	None

Usage/Example	`<ContentWithCommunityButtons sideButtonsList={{btn1, btn2, ...}}>`
	`    //Content`
	`</ContentWithCommunityButtons>`

Buttons

IconSquareButton	
Path	`/vectre/frontend/src/components/Buttons/IconSquareButton/IconSquareButton.js`
Description	A square icon button component with customisable props.
Props	display, px, py, color, bg, icon, ...otherProps
Children	None

Usage/Example	`<IconSquareButton`
	`     px={"5px"}`
	`     py={"5px"}`
	`     bg={"white"}/>`

TextButton	
Path	`/vectre/frontend/src/components/Buttons/TextButton/TextButton.js`
Description	A text button component with customisable props.
Props	display, px, py, color, bg, icon, ...otherProps
Children	None

Usage/Example	`<TextButton`
	`     px={"5px"}`

	`py={"5px"}`
	`bg={"white"}/>`

NavBar

NavBar	
Path	`/vectre/frontend/src/components/NavBar/index.js`
Description	A NavBar component. Is referenced within AppWrapper.
Props	None
Children	None
Usage/Example	`<NavBar/>`

Posts

PostComponent	
Path	`/vectre/frontend/src/components/PostComponent/PostComponent.js`
Description	A Post component that is the entire post object (on the feed or postpage). Can also act as a comment component, so certain properties would be different depending on the value of item.parent. (if item.parent is null, the post is not a comment, else otherwise)
Props	item (json object with post details like imageURL, timestamp, postID, parent, author info…)
Children	None
Usage/Example	`<PostComponent item={item} />`

UserCommentComponent	
Path	`/vectre/frontend/src/components/UserCommentComponent/UserCommentComponent.js`
Description	A form comment component that exists under the PostComponent when the user is on a post's page (i.e. the link '/post/{postID}'). The user will use this component to submit a comment onto a post.
Props	item (json object with post details like imageURL, timestamp, postID, parent, author info…)

Children	None
Usage/Example	`<UserCommentComponent item={item} />`

Notifications

NotificationPopover	
Path	`/vectre/frontend/src/components/Notifications/NotificationPopover.js`
Description	A wrapper for the notification window and the button to open/close the window.
Props	None
Children	The Popover trigger (button), the unread markers, and the notifications along with their corresponding headers.
Usage/Example	`<NotificationPopover/>`

Notifications Component	
Path	`/vectre/frontend/src/components/Notifications/Notifications.js`
Description	The container to display notification objects inside the window of the NotificationPopover. This component also sorts the notifications into categories based on their created time.
Props	hasUnread and setHasUnread. Both are used to control the unread status of the notifications.
Children	The Notification’s and the headers.
Usage/Example	`<NotificationPopover hasUnread={hasUnread} setHasUnread={setHasUnread}/>`

Notification Component	
Path	`/vectre/frontend/src/components/Notifications/Notification.js`
Description	Represent a single notification.
Props	All attributes of a notification: \`fromuser\`, \`action\`, \`read\`, \`postID\`, \`notificationID\`.
Children	The notification message, the avatar, and the notification icon.
Usage/Example	`<NotificationPopover hasUnread={hasUnread} setHasUnread={setHasUnread}/>`

Dashboard

Dashboard Component	
Path	`/vectre/frontend/src/components/Dashboard/Dashboard.js`
Description	Represents the entire NFT dashboard found on the user profile. It is referenced within Profile.

Props	`loggedInUser,`
	`profileWalletAddress,`
	`currentDashboard,`
	loggedInUser provides the details of the current user, providing the wallet address to obtain the dashboard and for the nft call to retrieve nfts from the wallet.
	profileWalletAddress is the current profile page's user's wallet address and will allow for the modification of the dashboard if profileWalletAddress and loggedInUser are the same.
	currentDashboard is currently set dashboard by profile's wallet address.

Children	DashboardTop, DashboardMid, DashboardBot, EditDashboardModal

Usage/Example	`<Profile>`
	`  <Dashboard`
	`    loggedInUser={this.props.loggedInUser}`
	`    profileWalletAddress={this.props.profileWalletAddress}`
	`    currentDashboard={this.props.user.dashboard}/>`
	`<Profile/>`

Dashboard Top	
Path	`/vectre/frontend/src/components/Dashboard/DashboardTop/DashboardTop.js`
Description	Represents the header of the dashboard Module, containing the header which holds the Title and Icon labelled "NFT Dashboard".
Props	None.
Children	None.

Usage/Example	`<Dashboard>`
	`   <DashboardTop/>`
	`   <DashboardMid currentDashboard={currentDashboard} />`
	`   <DashboardBot onOpen={onOpen} />`
	`   <DashboardEditModal isOpen={isOpen} onClose={onClose} />`
	`</Dashboard>`

Dashboard Mid	
Path	`/vectre/frontend/src/components/Dashboard/DashboardMid/DashboardMid.js`
Description	Represents the middle of the NFT dashboard, containing the dashboard component that displays the set NFTs on the dashboards by the user.
Props	`currentDashboard,` currentDashboard is the currently set dashboard by the profile's wallet address, such that DashboardMid knows which nfts have been set by the user and can be displayed.
Children	None.

Usage/Example	`<Dashboard>`
	`    <DashboardTop/>`
	`    <DashboardMid currentDashboard={currentDashboard} />`
	`    <DashboardBot onOpen={onOpen} />`
	`    <DashboardEditModal isOpen={isOpen} onClose={onClose} />`
	`</Dashboard>`

Dashboard Bot	
Path	`/vectre/frontend/src/components/Dashboard/DashboardBot/DashboardBot.js`
Description	Represents the bottom part of the NFT dashboard. This provides the logged in user (if on their profile) to be able to edit and set nfts on the dashboard.
Props	`onOpen,` onOpen allows for the opening up of the Edit Modal to set NFT dashboard.
Children	None.

Usage/Example	`<Dashboard>`
	`    <DashboardTop/>`
	`    <DashboardMid currentDashboard={currentDashboard} />`
	`    <DashboardBot onOpen={onOpen} />`
	`    <DashboardEditModal isOpen={isOpen} onClose={onClose} />`
	`</Dashboard>`

Dashboard Edit Modal	
Path	`/vectre/frontend/src/components/Dashboard/DashboardEditModal/DashboardEditModal.js`
Description	Represents the selection tool that allows the user to select NFTs that exist on their wallet, and allows the user to select up to 3 NFTs for their dashboard.

Props	`isOpen,`
	`onClose`
	`,`
	
	isOpen allows for the checking of the Edit Modal to set NFT dashboard if it is open.
	onClose allows for the closing of the EditModal when desired.

Children	NFTImage.

Usage/Example	`<Dashboard>`
	`<DashboardTop/>`
	`<DashboardMid currentDashboard={currentDashboard} />`
	`<DashboardBot onOpen={onOpen} />`
	`<DashboardEditModal isOpen={isOpen} onClose={onClose} />`
	`</Dashboard>`

NFT Items

NFT Image	
Path	`/vectre/frontend/src/components/Dashboard/NFTItem/NFTItem.js`
Description	Represents an NFT on the NFT selector tool when setting the backend of the dashboard.

Props	`handleSelectAdd,`
	` handleSelectDelete,`
	` selectedList,`
	` setSelectedList,`
	` nftItem,`
	
	handleSelectAdd is a function that adds the toggled NFTImage into selectedList.
	handleSelectDelete is a function that removes the toggled NFTImage into selectedList.
	selectedList is a list of JSON objects containing all the selected NFT items with a maximum length of 3.
	setSelectedList allows for the setting of the state of the selectedList.
	nftItem provides a JSON object containing NFT data to be supplied to the NFTImage, including most crucially the imageURL.

Children	None.

Usage/Example	`<DashboardEditModal>`
	`  <NFTImage`
	`    handleSelectDelete={handleSelectDelete}`
	`    handleSelectAdd={handleSelectAdd}`
	`    selectedList={selectedList}`
	`    setSelectedList={setSelectedList}`
	`    nftItem={nftItem}`
	`    maxSelected={3} />`
	`</DashboardEditModal>`

NFT Avatar Modal	
Path	`/vectre/frontend/src/components/Modals/NFTAvatarModal/NFTAvatarModal.js`
Description	Represents the modal to select an NFT avatar.

Props	`isOpen,`
	`onClose,`
	`data,`
	`profilePicImageData,`
	`setProfilePicImageData,`
	`profilePicTokenID,`
	`setProfilePicTokenID,`
	`profilePicImageLink,`
	`setProfilePicImageLink`
	isOpen handles opening the modal.
	onClose handles closing the modal.
	data is the logged in user's data.
	profilePicImageData is the user's uploaded image.
	setprofilePicImageData is to set the profilePicImageData.
	profilePicTokenID is the user's tokenID selected NFT.
	setprofilePicTokenID is to set the profilePicTokenID.
	profilePicImageLink is to set the NFT's link.
	setprofilePicImageLink is to set the profilePicImageLink.

Children	None.

Usage/Example	`<NFTAvatarModal>`
	`  <NFTImage`
	`    handleSelectDelete={handleSelectDelete}`
	`    handleSelectAdd={handleSelectAdd}`
	`    selectedList={selectedList}`
	`    setSelectedList={setSelectedList}`
	`    nftItem={nftItem}`
	`    maxSelected={1} />`
	`</NFTAvatarModal>`

Search

Individual Search Result Component	
Path	`/vectre/frontend/src/components/IndividualSearchResult/IndividualSearchResult.js`
Description	Represents one single search result that pops up after you click search in the search page.
Props	`result` result should be a single search result that contains the profilePic, banner and followerCount if the result is a user, membersCount if the result is a community, username if the result is a user, name if the result is a community, walletAddress if the result is a user, communityId if the result is a community
Children	None.

Usage/Example	`<Box width={"100%"} display={"grid"} gridTemplateColumns={"1fr 1fr"} gridGap="24px">`
	`            {results.map((result, i) => {`
	`                return (`
	`                    <IndividualSearchResult key={i} result={result}/>`
	`                )`
	`            })}`
	`</Box>`

Moderators

Generic Warning Modal	
Path	`/vectre/vectre/frontend/src/components/Modals/GenericWarningModal/GenericWarningModal.js`
Description	Shows a modal depending on the button pressed. The three possible choices is when the moderator bans a user, promotes a user, or deletes a post.

Props	`type,`
	`actionBtnOnClick,`
	`isOpen,`
	`onClose`
	type is either ban info, promote info, or delete info. An onclick function is passed as a prop into this. isOpen and onClose are chakra ui's modal functions.

Children	None.
Usage/Example	`<GenericWarningModal type={type} isOpen={isOpen} onClose={onClose} actionBtnOnClick={()=>someFunction()} />`

Verified NFT Avatar	
Path	`/vectre/vectre/frontend/src/components/VerifiedNFTAvatar/VerifiedNFTAvatar.js`
Description	Shows a spinning ring if the user has an NFT.

Props	`data,`
	`type,`
	Data is the logged in user.
	type is either ban info, promote info, or delete info. An onclick function is passed as a prop into this. isOpen and onClose are chakra ui’s modal functions.

Children	None.
Usage/Example	`<VerifiedNFTAvatar type={type} data={data} ={()=>someFunction()} />`

Trending	
Path	/vectre/vectre/frontend/src/components/Trending/Trending.js
Description	Shows the trending page.

Props	const trendingUsers = useSelector(trendingUsersSelector)
	    const trendingCommunities =
	useSelector(trendingCommunitiesSelector)
	    const dispatch = useDispatch()

Children	None.
Usage/Example	<Trending <IndividualSearchResult bg={'rgba(198, 219, 255, 0.11)'} key={i} result={result} trending={true} /> <IndividualSearchResult key={i} result={result} trending={true} /> />

Search	
Path	/vectre/vectre/frontend/src/components/Search/Search.js
Description	Shows a spinning ring if the user has an NFT.

Props	const dispatch = useDispatch()
	    const searchedUsers = useSelector(searchedUsersSelector)
	    const searchedCommunities =
	useSelector(searchedCommunitiesSelector)
	    const searchedPosts = useSelector(searchedPostsSelector)
	
	    const [searchInput, setSearchInput] = useState(".*")

Children	None.

Usage/Example	</GenericButtonsPopoverWrapper>
	           <SearchResultContainer
	results={getUsersAndCommunitiesResults()} />
	
	           <GenericButtonsPopoverWrapper

Entity Card	
Path	`/vectre/vectre/frontend/src/components/EntityCard/EntityCard.js`
Description	Shows a spinning ring if the user has an NFT.

Props	`key,`
	`iconBoxSize = "68px",`
	`bg = "white",`
	`data,`
	`href,`
	`primaryText,`
	`secondaryText,`
	`tertiaryText,`
	`...otherProps`

Children	None.

Usage/Example	`<Flex flexDirection={'column'} gap={'5px'}>`
	`{communitiesList.map((community, i) =>`
	`<EntityCard`
	`key={i}`
	`iconBoxSize={'40px'}`
	`bg={"none"}`
	`primaryText={cutText(community.name, 23)}`
	`secondaryText={"< " + cutText(community.communityID, 28) + " >"}`
	`href={"/c/" + community.communityID}`
	`data={community} >`
	`<ToggleHollowButton`
	`onText={'Joined'}`
	`offText={'Join'}`
	`isOn={community.alreadyJoined}`
	`onClick={() => handleJoin(community)} />`
	`</EntityCard>`
	`)}`

Header And Filter	
Path	/vectre/vectre/frontend/src/components/HeaderAndFilter/HeaderAndFilter.js
Description	Adds a header for the given section as well as the filter button.
Props	icon, onToggle, text
Children	None.

Usage/Example	<GenericButtonsPopoverWrapper
	margin={'0 0 0 20px'}
	buttons={
	<>
	{userCommunitiesButtonsList.map((element, i) => (
	<TextButton
	key={i}
	height={'fit-content'}
	text={element.typeData.title}
	onClick={(e) => {
	element.onClick();
	e.stopPropagation();
	}}
	rightIcon={element.typeData.icon}
	/>
	))}
	</>}>
	<HeaderAndFilter text={getUsersAndCommunitiesHeader()} icon={<AiFillFilter size={'1.3rem'} />} />
	</GenericButtonsPopoverWrapper>

Create Post Component	
Path	`/vectre/vectre/frontend/src/components/HeaderAndFilter/HeaderAndFilter.js`
Description	Allows the user to create a post.
Props	`communityID`
Children	None.

Usage/Example	`<Stack`
	`{`
	`loggedInUserRoles.includes("member") ?`
	`<CreatePostComponent communityID={communityID} /> : null`
	`}`
	`{feed.map((item, i) => {`
	`return (`
	`<Box key={i}>`
	`<PostComponent item={item}`
	`fromFeed={true} />`
	`</Box>`
	`)`
	`})}`
	`</Stack>`