Authentication Related APIs

- ## Register

Description: Create a new account
Method: POST
URL: “/register”
Pass in:
{
    Username,
    Email, // should be unique in DB
    Password, // minimum 8 characters
}
Return:
If email and password meet requirements, send an activation link to the user’s email address. Upon clicking that link, the user’s account will be created successfully, the user will be directed to the login page instantly.

If any error occurs (e.g. invalid email address, invalid password, server error…), return corresponding status code and error message in JSON format:
{
    errorMessage: “xxxx error”
}
This message will be directly used for prompts on the front end.

- ## Forgot password

Description: Reset password with email associated with the user’s account
Method: POST
URL: “/forgotpassword”
Pass in:
{
    Email
}
Return:
Send an email with a link for resetting password to user (if the email exists in database). Upon the user clicks the link, he will be landed on a page where he can enter a new password. Here we should figure out how to design the URL directing the user to this page, so that the server knows it is this user changing his password. Also we need to care about the link’s expiration time.

We should never confirm or deny the existence of an account with a given email or username. So just return status code 200 (if nothing wrong happens on the server) and a message “If there is an account with that email, an email has been sent to it.”

- Log in with email and password done

  Description: User logs in
  Method: POST
  URL: “/login”
  Pass in:
  {
  Email,
  Password
  }
  Return:
  If email and password match, return status code 200 and a JWT token encoding the user’s id. This token will be stored in the browser's local storage, and be included in the request header during other actions.

  If any error occurs (account does not exist, email and password don’t match...), follow error handling procedure.

- Log in with Google
- Log in with Facebook

User’s info Related APIs

- Get user’s profile done

  Description: Each user will have a profile stored in the User table. When an account is newly created, the user’s avatar is set as a default image. Other fields of the profile will be set as empty strings.
  This API is used for getting a user’s profile with his user id (encoded in JWT)
  Method: GET
  URL: “/profile”
  Pass in:
  JWT in the request’s header (“x-auth-token”)
  Return:
  This user’s profile (avatar url, bio, facebook, twitter, instagram).

  If any error occurs (account does not exist, email and password don’t match...), follow error handling procedure.

- ## Get other user's profile

  Description: Each user will have a profile stored in the User table. When an account is newly created, the user's avatar is set as a default image. Other fields of the profile will be set as empty strings.
  This API is used for getting a user's profile with his user id (encoded in URL)
  Method: GET
  URL: "/profile:{userID}" // feel free to rename this
  Pass in:
  userID
  Return:
  This user's profile (avatar url, bio, facebook, twitter, instagram).

  If any error occurs (account does not exist, email and password don't match...), follow error handling procedure.

- ## Edit user's profile (avatar) done

  Description: Replace a user's avatar with an image he uploaded. This image will be cropped and resized to 96X96 on the front end.
  Method: POST
  URL: ~~"/editavatar"~~ "/profile/avatar"
  Pass in:
  JWT in the request's header ("x-auth-token")
  image in a FormData object, see How to upload files to the server using JavaScript
  Return:
  status code 200 + URL to the new avatar if success

  If any error occurs, follow error handling procedure.

- ## Edit user's profile (other fields) done

  Description: Change user's [bio, facebook, twitter, instagram].
  Method: ~~POST~~ PATCH
  URL: ~~"/editprofile"~~ "/profile"
  Pass in:
  JWT in the request's header ("x-auth-token")
  JSON
  {
  bio: string,
  facebook: string:
  twitter: string:
  instagram: string
  }

Return:
status code 200 + the updated profile in JSON if success
If any error occurs, follow error handling procedure.

Events Related APIs

- ## Create a new event

Description: Create a new event (only when a user is logged in).
Method: POST
URL: “/events”
Pass in:
JWT in the request’s header (“x-auth-token”)
JSON
{
    title: string,
    location: string,
    date: string, yyyy-mm-dd,
    startTime: string 24-hour format (should still provide full format, e.g. `2016-09-22T23:21:56.027Z`)
    endTime: string 24-hour format (should still provide full format, e.g. `2016-09-22T23:21:56.027Z`)
    hostGroup: string,
    eventType: string,
    whoCanCome: string,
    foodType: string,
    foodAmount: string,
    otherInfo: string
}
Return:
If success, return code 200 + JSON
{
    message: “Successfully created a new event.”
}
If the user is not logged in (i.e. no JWT token in request head), return error message “Please log in...:”
If any other error occurs, follow error handling procedure.

- ## Delete an existing event

Description: Delete an existing event (Only if it’s created by the user logged in).
Method: DELETE
URL: “/events:{eventID}”
Pass in:
JWT in the request’s header (“x-auth-token”)
eventID is encoded in URL
Return:
If success, return code 200 + JSON

{
message: “Successfully deleted the event.”
}
If JWT and event’s creatorID do not match, return “not authenticated” status code and corresponding message.
If any other error occurs, follow error handling procedure.

- ## Watch an event

Description: Add a “watch” to an event. Each “watch” is identified by eventID and watcherID (which is a userID).
Method: POST
URL: “/events/watch/:{eventID}”
Pass in:
JWT in the request’s header (“x-auth-token”)
eventID is encoded in URL
Return:
If success, return code 200
If the user has already watched this event, return a “failed” code
If any other error occurs, follow error handling procedure.

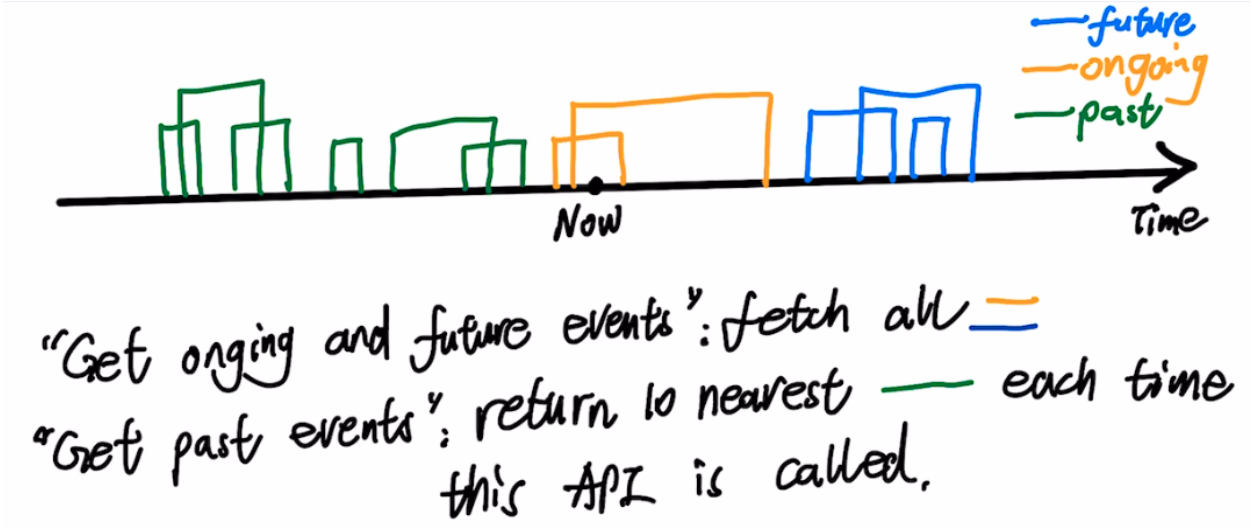
- ## Unwatch an event

Description: Remove a user’s “watch” from an event. Each “watch” is identified by eventID and watcherID (which is a userID).
Method: DELETE
URL: “/events/watch/:{eventID}”
Pass in:
JWT in the request’s header (“x-auth-token”)
eventID is encoded in URL
Return:
If success, return code 200
If the user has not watched this event, return a “failed” code
If any other error occurs, follow error handling procedure.

- ## Comment an event

Description: Add a comment to an event.
Method: POST
URL: “/events/comment/:{eventID}”
Pass in:
JWT in the request’s header (“x-auth-token”)
eventID is encoded in URL
JSON

{
content: String,
time: a time string generated when the user hits “Post” button, format: “YYYY--MM--DD HH:MM”. Or, this could be generated by DB.
}
Return:
If success, return code 200
If any error occurs, follow error handling procedure.



- Get ongoing and future events

Description: As depicted above
Method: GET
URL: “/events”
Pass in: Nothing
Return:
If success, return code 200, return events along with corresponding comments
JSON
{
events: [
{
creatorUsername: string,
creatorAvatarURL: string,
eventID: string,
title: string,
location: string,
date: string, yyyy-mm-dd,
startTime: string 24-hour format,
endTime: string 24-hour format,

```
                hostGroup: string,
                eventType: string,
                whoCanCome: string,
                foodType: string,
                foodAmount: string,
                otherInfo: string,
                numWatches: int,
                comments: [
                        {
                        commentID: string,
                        posterUsername: string,
                        posterAvatarURL: string
                        content: string,
                        postTime: string
                        },
                        ...
                    ]
                },
                ...
        ]
}
```

If any error occurs, follow error handling procedure.

- Get past events

  Description: As depicted above
  Method: GET
  URL: “/pastevents:fetched={num}”
  Pass in: number of past events already fetched (num)
  Return: Same as above

Offers Related APIs

- ## Create a new offer

Description: Create a new offer (only when a user is logged in).
Method: POST
URL: “/offers”
Pass in:
JWT in the request’s header (“x-auth-token”)
JSON
{
    description: string,
    location: string,
    date: string, yyyy-mm-dd,
    startTime: string 24-hour format,
    endTime: string 24-hour format,
    otherInfo: string

}
Return:
If success, return code 200 + JSON
{
    message: “Successfully created a new offer.”
}
If the user is not logged in (i.e. no JWT token in request head), return error message “Please log in...:”
If any other error occurs, follow error handling procedure.

- ## Delete an existing offer

Description: Delete an existing offer (Only if it’s created by the user logged in).
Method: DELETE
URL: “/deleteoffer:{offerID}”
Pass in:
JWT in the request’s header (“x-auth-token”)
offerID is encoded in URL
Return:
If success, return code 200 + JSON
{
    message: “Successfully deleted the offer.”
}
If JWT and offer’s creatorID do not match, return “not authenticated” status code and corresponding message.
If any other error occurs, follow error handling procedure.

- ## Comment an offer

Description: Add a comment to an offer.
Method: POST
URL: “/offers/comment/:{offerID}”
Pass in:
JWT in the request’s header (“x-auth-token”)
offerID is encoded in URL
JSON
{
content: String,
time: a time string generated when the user hits “Post” button, format: “YYYY--MM--DD HH:MM”. Or, this could be generated by DB.
}
Return:
If success, return code 200
If any error occurs, follow error handling procedure.

- ## Get ongoing and future offers

Description: Only fetch ongoing and future offers
Method: GET
URL: “/offers”
Pass in: Nothing
Return:
If success, return code 200, return offers along with corresponding comments
JSON
{
offers: [
{
creatorUsername: string,
creatorAvatarURL: string,
offerID: string,
description: string,
location: string,
date: string, yyyy-mm-dd,
startTime: string 24-hour format,
endTime: string 24-hour format,
otherInfo: string,
comments: [
{
commentID: string,
posterUsername: string,
posterAvatarURL: string

```
                                content: string,
                                postTime: string
                                },
                                ...
                            ]
                    },
                    ...
            ]
}
```

If any error occurs, follow error handling procedure.

Networking Related APIs

- ## Create a new post (for networking)

  Description: Create a new post (only when a user is logged in).
  Method: POST
  URL: “/posts”
  Pass in:
  JWT in the request’s header (“x-auth-token”)
  JSON
  {
  content: String
  }
  Return:
  If success, return code 200 + JSON
  {
  message: “Successfully created a new post.”
  }
  If any other error occurs, follow error handling procedure.

- ## Deleting an existing post

  Description: Delete an existing post (Only if it’s created by the user logged in).
  Method: DELETE
  URL: “/deletepost:{postID}”
  Pass in:
  JWT in the request’s header (“x-auth-token”)
  postID is encoded in URL
  Return:
  If success, return code 200 + JSON
  {
  message: “Successfully deleted the post.”
  }
  If JWT and post’s creatorID do not match, return “not authenticated” status code and corresponding message.
  If any other error occurs, follow error handling procedure.

- ## Comment a post

  Description: Add a comment to a post.
  Method: POST
  URL: “/posts/comment/:{postID}”
  Pass in:
  JWT in the request’s header (“x-auth-token”)
  postID is encoded in URL

JSON
{
content: String,
}
Return:
If success, return code 200
If any error occurs, follow error handling procedure.

- Get all posts

Description: Fetch 10 most recent posts
Method: GET
URL: “/posts:fetched={num}”
Pass in: number of most recent posts already fetched (num)
Return:
If success, return code 200, return posts (creator username, avatar URL, content), along with corresponding comments

- Search posts

Description: Filter all posts in database with a query string
Method: GET
URL: “/posts:querystring={querystring}”
Pass in: a query string
Return:
Return posts if querystring is a substring of their contents / creator usernames.

- Get posts by ID

Description: Get posts (with comments) created by a user identified by ID
Method: GET
URL: “/posts:userid={ID}”
Return:
If success, return code 200 + JSON
{
message: “Successfully created a new feedback.”
}
If success, return code 200, return posts (creator username, avatar URL, content), along with corresponding comments

MISC

Super User

Create a super user when initializing database
This super user has UserID = 0, email for logging in = brownbytes@brownbytes.bb, password = csci1320
The super user can delete any event / offer / post, and fetch all feedbacks

- ## Get all feedbacks

Description: get all feedbacks, only callable when super user is logged in
Method: GET
URL: "/feedbacks:"
Pass in: super user's JWT token in request header
Return:
If successful, return code 200, return all feedbacks (creator name, email, content).

- ## Create a new feedback

Description: Create new feedback (don't require a user logged in).
Method: POST
URL: "/feedback"
Pass in:
JSON
{
    posterName: String,
    email: String,
    feedback: String
}
Return:
If success, return code 200 + JSON
{
    message: "Successfully created a new feedback."
}
If any other error occurs, follow error handling procedure.