**API Documentation:**

Welcome to the documentation for TechBits. This document provides details on how to use and integrate the APIs.

**Authentication:**

Base URL: http://localhost:3000/user

**Login:**

POST <http://localhost:3000/user/login>

The Login API expects a JSON object containing 2 variables – username and password:

{

    "username": "Test123",

    "password": "test"

}

The username and password is initialized by de-structuring the request body and then the API checks if their values are undefined.

If any of the 2 fields are undefined, the API returns the following JSON error response with the status code 400:

{

    "error": {

        "status": 400,

        "message": "Username/Password cannot be blank"

    }

}

If they are not undefined, the API proceeds to validate the credentials using bcrypt by passing the credentials and executing the Mongoose static function findAndValidate which is defined in the userSchema.

userSchema.statics.findAndValidate = async function (user, pwd) {

    const db\_user = await this.findOne({ username: user })

    if (db\_user) {

        const validation\_result = await bcrypt.compare(pwd, db\_user.password);

        return validation\_result ? db\_user : false

    } else {

        return false

    }

}

If the validation is successful, the static method returns the corresponding user document from the Mongo database and uses JWT to generate a token that is valid for 5 hours. The token is generated by encrypting the index ID of the user document with TOKEN\_SECRET that was defined in the environment variables. The JWT token is then stored in the session store along with the index ID of the user and the response is sent to the client.

If the validation fails, the static method returns a Boolean false value and returns the following JSON error response with the status code 401:

{

    "error": {

        "status": 401,

        "message": "Invalid Credentials"

    }

}

**Logout:**

POST http://localhost:3000/user/logout

The Logout API uses the ‘isLoggedIn’ middleware to check if the client is authenticated or not. If the client is authenticated, the user document index and the JWT token which are stored in the session store are set to null and the following JSON message is returned with the status code 200:

{

    "status": 200,

    "message": "Logged Out"

}

If the client is not authenticated, the following JSON error response is sent back to the client with the status code: 400

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

**Sign-up:**

POST http://localhost:3000/user/sign-up

The Sign-up API expects a JSON object containing the username, password, email and a name object which contains first and last name of the user in the request body. The data should be of String type.

Example:

{

    "username": "Test234",

    "password": "test",

    "email": "test@gmail.com",

    "name": {"first": "FirstName", "last": "LastName"}

}

Once the request is received, the API performs validations before creating the new user document and authenticating the client. All the fields are set as required and if any of the fields are missing, the following JSON error responses are sent with the status code 400:

If one field/value is missing:

{

    "error": {

        "status": 400,

        "message": "[field] is required"

    }

}

If there are more than one fields/values missing:

{

    "error": {

        "status": 400,

        "message": "Invalid/Incomplete Data"

    }

}

Username is a unique index in the database which is stored in lower case. If a new request is sent with a username that is already created, the API sends the following JSON error response with the status code 400:

{

    "error": {

        "status": 400,

        "message": "This username is already taken, please enter a different one."

    }

}

If the data is valid, the API creates & saves the user document in the database. Then the JWT token is generated and the new user index and token are saved in the session store which is sent back to the client. The API returns a JSON response containing the username, email and the full name which is generated using the mongoose virtuals with the status code 201:

{

    "username": "test234",

    "email": "test@gmail.com",

    "name": "FirstName LastName"

}

**Posts:**

Base URL: http://localhost:3000/posts

**Create Post:**

POST <http://localhost:3000/posts/new>

The create posts API is a protected route, so the middleware ‘isLoggedIn’ will first check if the client/user is authenticated before passing the request to the create post controller. If the client/user is not authenticated, the following JSON error response is returned by the API with the error code 401:

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

The create posts API expects the post data - title, summary, content, tags, status and the image file in the request body. All the fields are expected to be of String type except for “image” which would be a file of types jpeg/png/jpg.

The multer middleware will upload/store the image and will update the filename in the request object which would later be used to store the path of the image in the database. If there were no images uploaded, a path for the default cover image will be stored in the database.

The fields – title, summary and author are set as required in the Posts schema. The status field is and enum of the type String which accepts two values – ‘draft’ and ‘published’. If the status field is not sent over the request, the value will be set to ‘draft’ by default.

The author object is a reference to the Users document. The Users collections is queried with the document id that is stored in the session store and is used to set the author of the post.

The post details will then be used to create a new post document that will be stored in the ‘Posts’ collection in the database and the API sends back the following JSON response object containing the new post id with the status code 201:

{

    "id": "[new\_post\_id]"

}

If any of the required fields are missing, the API sends back the following JSON error response with the status code 400:

{

    "error": {

        "status": 400,

        "message": "[field] is required"

    }

}

**View Post:**

GET <http://localhost:3000/posts/:post_id>

The view post API expects the document ID of a post to be passed in the request parameters. When a request is received, the API proceeds to query the Posts collection with the ID from the request parameters, populates the author details and returns the following JSON response object if the document is found:

Sample response:

{

    "post": {

        "\_id": "65cf2d9141c4748c0d95043b",

        "author": {

            "name": {

                "first": "Nitin",

                "last": "Pandit"

            },

            "\_id": "65c461ef1e9b311cc5269265",

            "email": "Nitinp1999@gmail.com",

            "username": "nitinp1999"

        },

        "title": "test",

        "summary": "test",

        "content": "",

        "image": "../../blog-cover-picture.png",

        "tags": [],

        "status": "published",

        "createdAt": "2024-02-16T09:40:33.172Z",

        "updatedAt": "2024-02-16T09:40:33.172Z",

        "\_\_v": 0

    }

}

If the document is not found, it returns the following JSON error response object:

{

    "error": {

        "status": 404,

        "message": "Not Found"

    }

}

**Edit Post:**

PATCH http://localhost:3000/posts/:post\_id

The edit posts API is a protected route, so the middleware ‘isLoggedIn’ will first check if the client/user is authenticated before passing the request to the next middleware/controller. If the client/user is not authenticated, the following JSON error response is returned by the API with the error code 401:

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

The edit posts API also uses the ‘validateAuthor’ middleware to prevent unauthorized users from posting and updates by comparing the user document ID that was stored in the session store with the author ID that is present in the post document (which is queried using the post ID from the request parameters) before passing the request to the edit posts controller. If the client is not the author of the post, the following JSON error message is returned with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

The edit post API expects the document ID of a post to be passed in the request parameters along with the updated post details such as the title, summary, content, status, image, and an additional object – deleteImage which is an object containing 2 keys – image (String) and toDelete (Boolean).

If the image was updated, the image object in the request body will contain the new image that was uploaded while the image key in the deleteImage object will contain the value/filename of the old image that has to be deleted and the toDelete key value will be set to true.

Example:

{

  "title": "test",

  "summary": "test",

  "content": "Test",

  "tags": [],

  "status": "published",

"image": [Object File],

 "deleteImage": '{"image":"../../uploads/a54ce90b973b9c74406b08b054c865b9","toDelete":true}'

}

If there weren’t any changes to the image, the toDelete value in the deleteImage object will be set to false and the image object will contain the current images filename as a string.

Example:

{

  "title": "test",

  "summary": "test 4",

  "content": "Test",

  "tags": ["test"],

  "status": "published",

  "image": "../../uploads/e1177be718a9f67b8b5538f84626a97c",

  "deleteImage": {"image":"../../uploads/e1177be718a9f67b8b5538f84626a97c","toDelete": false }

}

The API then proceeds to query the Post collection with the document ID that was passed in the request parameters and updates the document with the new values if a document was found, and returns the following JSON response object with the status code 200:

{

    "status": 200,

    "message": "Post Updated"

}

If the document was not found, it returns the following JSON error response with the status code 404:

{

    "error": {

        "status": 404,

        "message": "Not Found"

    }

}

**Delete Post:**

DELETE http://localhost:3000/posts/:post\_id

The delete posts API is a protected route, so the middleware ‘isLoggedIn’ will first check if the client/user is authenticated before passing the request to the next middleware/controller. If the client/user is not authenticated, the following JSON error response is returned by the API with the error code 401:

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

The delete posts API also uses the ‘validateAuthor’ middleware to prevent unauthorized users from deleting any posts by comparing the user document ID that was stored in the session store with the author ID that is present in the post document (which is queried using the post ID from the request parameters) before passing the request to the edit posts controller. If the client is not the author of the post, the following JSON error message is returned with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

After the validations, the delete posts controller queries the post collections with the document ID from the request parameters and proceeds to delete the post from the database. After the post has been deleted, the corresponding likes and comments of the post are deleted with the help of mongoose post middleware. The image of the post is also deleted from the storage and the following JSON response message is sent back with the status code 200:

{

    "status": 200,

    "message": "success"

}

**Get All Posts**

GET [http://localhost:3000/posts/all?page={page\_number}&size={page\_size}](http://localhost:3000/posts/all?page=%7bpage_number%7d&size=%7bpage_size%7d)

The 'get all posts' API, accessed through the 'GET /posts/all' endpoint, utilizes the page and size values provided in the request query to retrieve sorted and paginated results from the post documents. The API queries the database by filtering on the 'published' status of posts and sorts them in descending order based on the creation date, with the newest posts appearing first.

The API only populates the author ID, username, title, summary, image, tags, createdAt fields of each post document. Then it queries the comments collection for each post to get the total count of comments and adds it to the post object. The API sends back the following JSON response message with the status code 200:

{

    "pages": 5,

    "page": "1",

    "size": "10",

    "posts": [

        {

            "\_id": "65cf2d9141c4748c0d95043b",

            "author": {

                "\_id": "65c461ef1e9b311cc5269265",

                "username": "nitinp1999"

            },

            "title": "test",

            "summary": "test 4",

            "image": "../../uploads/ef14db98efff9303edd0564ee133d466",

            "tags": [],

            "createdAt": "2024-02-16T09:40:33.172Z",

            "\_\_v": 0,

            "commentsCount": 0

        },….]

}

‘pages’ in the response object correspond to the total number of pages, ‘page’ corresponds to the current page, ‘size’ corresponds to the page size and ‘posts’ is an Array of post documents.

**My Posts:**

GET <http://localhost:3000/posts/myposts>

The ‘my posts’ API is a protected API which uses the ‘isLoggedIn’ middleware to check if the client/user is authenticated before sending back the results. If the user/client is not authenticated, the API sends back the 401 JSON error message.

This API is similar to ‘get all posts’ API but instead of filtering on the post status, it filters on the user ID that is present in the session store for querying the paginated results. The API populates author ID, author username, title, summary, image, tags, status, createdAt, updatedAt and status of each post and also queries the comments collection for each post to the get the count and responds back with the following JSON message with the status code 200:

{

    "pages": 9,

    "page": 1,

    "size": 5,

    "posts": [

        {

            "\_id": "65cf2d9141c4748c0d95043b",

            "author": {

                "\_id": "65c461ef1e9b311cc5269265",

                "username": "nitinp1999"

            },

            "title": "test",

            "summary": "test 4",

            "image": "../../uploads/ef14db98efff9303edd0564ee133d466",

            "tags": [],

            "status": "published",

            "createdAt": "2024-02-16T09:40:33.172Z",

            "updatedAt": "2024-02-16T10:40:05.263Z",

            "\_\_v": 0,

            "commentsCount": 0

        }….]

}

If there are no posts, the API responds back with the following JSON message with the status code 200:

{

    "pages": 9,

    "page": 1,

    "size": 5,

    "posts": []

}

**Comments:**

Base URL: http://localhost:3000/comments

**Create Comment:**

POST http://localhost:3000/comments/:post\_id/new

The ‘create comment’ API is a protected API which uses the ‘isLoggedIn’ middleware to check if the client/user is authenticated before sending back the results. If the user/client is not authenticated, the API sends back the 401 JSON error message.

The API queries the user collections with the user document ID that is stored in the session store and the post collections with the post ID that is sent in the request parameters. If the user/post is not found, the API sends back the JSON 404 error message, else, it proceeds to check if the post in the ‘draft’ status or not. If the post has the status ‘draft’, the API sends back the following JSON error response with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

If the post is in ‘published’ status, a new comment document is created and saved using the user document, post document and the comment that was passed in the request body, and sends back the following JSON message with the status code 201:

{

    "status": 201,

    "comment": {

        "author": {

            "name": {

                "first": "Nitin",

                "last": "Pandit"

            },

            "\_id": "65cd9d297670de7e120fdf0f",

            "username": "test123",

            "email": "test@gmail.com"

        },

        "post": "65cc89c991a8c76fdf8645da",

        "comment": "test comment”,

        "createdAt": "2024-02-16T11:32:57.727Z",

        "updatedAt": "2024-02-16T11:32:57.727Z",

        "\_\_v": 0

    }

}

**Get Comments:**

GET http://localhost:3000/comments/:post\_id/all

The ‘get comments’ API queries the comments collection with the post document ID that is passed in the request parameters to get all the comment documents related to a particular post and populates the author username, email and name, and sends the following JOSN response object with the status code 200 if the comments were found:

{

    "status": 200,

    "comments": [

        {

            "\_id": "65cf4a9e332b2285e0927bfe",

            "author": {

                "name": {

                    "first": "Nitin",

                    "last": "Pandit"

                },

                "\_id": "65cd9d297670de7e120fdf0f",

                "username": "test123",

                "email": "test@gmail.com"

            },

            "post": "65cf2d9141c4748c0d95043b",

            "comment": "Test",

            "createdAt": "2024-02-16T11:44:30.187Z",

            "updatedAt": "2024-02-16T11:44:30.187Z",

            "\_\_v": 0

        }….

    ]

}

If the comments are not found for a post, it returns the following response with the status code 200:

{

    "status": 200,

    "comments": []

}

**Delete Comments:**

DELETE http://localhost:3000/comments/:comment\_id

The delete comments API is a protected route, so the middleware ‘isLoggedIn’ will first check if the client/user is authenticated before passing the request to the next middleware/controller. If the client/user is not authenticated, the following JSON error response is returned by the API with the error code 401:

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

The delete comments API also uses the ‘validateAuthor’ middleware to prevent unauthorized users from deleting any comments by comparing the user document ID that was stored in the session store with the author ID that is present in the comment document (which is queried using the comment ID from the request parameters) before passing the request to the delete comments controller. If the client is not the author of the comment, the following JSON error message is returned with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

If the client/author is authenticated and authorized, the API will delete comment from the comments collection using the comment document ID from the request parameters and will send back the following JSON response message with the status code 200:

{

    "status": 200,

    "message": "Comment Deleted"

}

If the comment document is not found, the API returns the following error message with the status code 404:

{

    "error": {

        "status": 404,

        "message": "Not Found"

    }

}

**Likes:**

Base URL: http://localhost:3000/likes

**Create Likes:**

POST http://localhost:3000/likes/:post\_id/new

The ‘create likes’ API is a protected API which uses the ‘isLoggedIn’ middleware to check if the client/user is authenticated before sending back the results. If the user/client is not authenticated, the API sends back the 401 JSON error message.

The ‘create likes’ API expects a Boolean Like value to be passed in the request body along with the post ID in the request parameters

The API queries the user collections with the user document ID that is stored in the session store and the post collections with the post ID that is sent in the request parameters. If the user/post is not found, the API sends back the JSON 404 error message, else, it proceeds to check if the post in the ‘draft’ status or not. If the post has the status ‘draft’, the API sends back the following JSON error response with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

If the post is in ‘published’ status, a new like document is created and saved using the user document, post document and the Boolean like that was passed in the request body, and sends back the following JSON message with the status code 201:

{

    "status": 201,

    "\_id": "65cf4fb7297d87cf016e551d"

}

Where ‘\_id’ is the new like document ID.

**Likes**

GET <http://localhost:3000/likes/:post_id>

The 'get likes’ API, accessed through the 'GET /likes/' endpoint, expects the post document ID to be passed in the request parameters. It then uses this post document ID to query the Likes collection to find all the likes that have a reference to the post document. The API iterates through the results to calculate the total number of likes and dislikes and stores them in the result object. If the user is logged in, it checks if the author ID of the like is same as the user ID stored in the session store and creates a Boolean ‘isLiked’ variable, and also saves the like document ID which could be used to update/delete at the client. The API sends back the following JSON message with the status code 200:

{

    "likeCount": 0,

    "dislikeCount": 1,

    "isLiked": false,

    "\_id": "65cf4fb7297d87cf016e551d"

}

If there are no Like documents found, the API sends the following JSON messasge with the status code 200:

{

    "status": 200,

    "message": "There are no Likes yet"

}

**Update Likes**

PATCH http://localhost:3000/likes/:like\_id

The ‘update likes’ API is a protected route, so the middleware ‘isLoggedIn’ will first check if the client/user is authenticated before passing the request to the next middleware/controller. If the client/user is not authenticated, the following JSON error response is returned by the API with the error code 401:

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

The ‘update likes’ also uses the ‘validateAuthor’ middleware to prevent unauthorized users from posting any updates by comparing the user document ID that was stored in the session store with the author ID that is present in the like document (which is queried using the like ID from the request parameters) before passing the request to the ‘update likes’controller. If the client is not the author of the Like document, the following JSON error message is returned with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

The edit post API expects the like document ID to be passed in the request parameters and updated Boolean like value in the body. It validates the like value in the request body and then uses the like document ID to find and update the data in the database and returns the following JOSN message with the status code 200:

{

    "status": 200,

    "message": "Like Updated"

}

If the data is invalid, it returns the following JSON error message with the status code 400:

{

    "error": {

        "status": 400,

        "message": "Invalid Data"

    }

}

**Delete Likes**

DELETE http://localhost:3000/likes/:like\_id

The ‘delete likes’ API is a protected route, so the middleware ‘isLoggedIn’ will first check if the client/user is authenticated before passing the request to the next middleware/controller. If the client/user is not authenticated, the following JSON error response is returned by the API with the error code 401:

{

    "error": {

        "status": 401,

        "message": "Not Authenticated"

    }

}

The ‘delete likes’ API also uses the ‘validateAuthor’ middleware to prevent unauthorized users from deleting any likes by comparing the user document ID that was stored in the session store with the author ID that is present in the likes document (which is queried using the likes ID from the request parameters) before passing the request to the ‘delete likes’ controller. If the client is not the author of the like, the following JSON error message is returned with the status code 403:

{

    "error": {

        "status": 403,

        "message": "Not Authorized"

    }

}

If the client/author is authenticated and authorized, the API will delete like from the Likes collection using the like document ID from the request parameters and will send back the following JSON response message with the status code 200:

{

    "status": 200,

    "message": "Comment Deleted"

}