

Date

December 1, 2024

# Coffee Spot

## Team 3 Members

Miguel Antonio Logarta - **Lead** |

Alan Yu - **Scrum Master** | Diane Bilse - **Front End** |

Halia Tavares - **Front End** | John Bagwell - **Back End** |

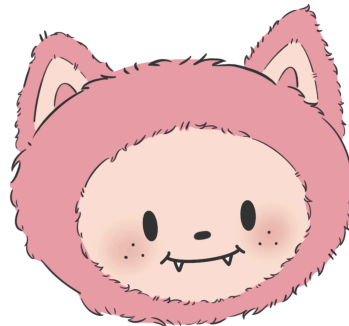
Su Tun (Emily Su) - **Git Master** | Timmy Tram - **Back End**

**Lead** |

## Milestone 4

Unit Testing,  
Integration Testing,  
Coding Styles,  
Documentation  
Generation

SW Engineering CSC648/848 - Section 01



History

Date

V0.1

12.01.2024

# 1. QA Testing

## Unit Testing: Testing Overview

<a href="#">Github Test Cases Directory</a>	<a href="#">Test Cases Directory Link</a>
---	---

Tested Features	Purpose
Admin Analytics API	This should return the count of items we have in the database.
Bookmarks API	This should make sure we can get all the bookmarks from the database so we can show it to the users.
Locations API	This gets all the locations so we can display it on the home page or search pages.
Reviews API	This should get all the reviews addressed to the locations.
Users API	This should make sure we can get the user's details and create users.

## Unit Testing: Coverage Summary

---

### Admin Analytics API

- **Functional Coverage:** 100%
- **Statement Coverage:** 100%

#### Summary:

- This API has complete coverage on both functional and statement levels.
- **Confidence Level:** We can be 100% sure this API is working as intended.

---

## Bookmarks API

- **Functional Coverage:** 100%
- **Statement Coverage:** 90%

### Summary:

- While statement coverage is slightly below full, functionality is fully covered.
- **Confidence Level:** We can be confident this API is working as intended.

---

## Locations API

- **Functional Coverage:** 71%
- **Statement Coverage:** 78%

### Summary:

- This API covers getting and creating locations.
- Test cases for **editing locations** have not yet been added.
- **Note:** Integration tests confirm that editing functionality works.

---

## Reviews API

- **Functional Coverage:** 100%
- **Statement Coverage:** 100%

### Summary:

- Full coverage ensures all functionality, including fetching reviews, is verified.

---

## Users API

- **Functional Coverage:** 75%
- **Statement Coverage:** 50%

**Summary:**

- This API covers **getting users** and **creating users**—the most frequently used functions.
- Test cases for **editing user information** are not yet written.

---

**Running the Tests:**

1. git clone  
<https://github.com/Miguel-Antonio-Logarta/csc648-01-fa24-csc648-01-fa24-team03-duplicate.git>
2. cd '.\CSC 648\csc648-01-fa24-csc648-01-fa24-team03-duplicate\application\'
3. git checkout dev | This step is only necessary if we tests aren't in main branch
4. npm install
5. npx prisma generate
6. npm run test

tramtapplicationdev22.9.016msnpm run test

> csc648-01-2024-team03-coffee-spot@0.1.0 test

> jest --silent --coverage

PASS\_\_test\_/api.bookmarks.test.ts

PASS\_\_test\_/api.locations.test.ts

PASS\_\_test\_/api.reviews.test.ts

PASS\_\_test\_/api.analytics.test.ts

PASS\_\_test\_/api.users.test.ts

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	63.79	43.85	60	63.74	
prisma	100	80	100	100	
prisma.ts	100	80	100	100	11
src/app/api/analytics	100	100	100	100	
route.ts	100	100	100	100	
src/app/api/auth/[...nextauth]	15.38	0	0	15.38	
authOptions.ts	15.38	0	0	15.38	15-54
src/app/api/bookmarks	90.9	66.66	100	90.9	
route.ts	90.9	66.66	100	90.9	28
src/app/api/locations	78.04	100	71.42	80	
route.ts	78.04	100	71.42	80	134-171
src/app/api/reviews	100	100	100	100	
route.ts	100	100	100	100	
src/app/api/users	50.79	27.77	75	50	
route.ts	50.79	27.77	75	50	82,105-109,131-206

Test Suites: 5 passed, 5 total

Tests: 18 passed, 18 total

Snapshots: 0 total

Time: 3.353 s, estimated 4 s

tramtapplicationdev22.9.05.274s






## Integration Testing

Test Case ID	1	Test Case Description	Test Login Functionality		
Created By	Timmy Tram	Reviewed by	Alan Yu	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1	username = Benas	
			2	password = 123456	
Test	Verify on entering valid username and password, the customer can login				
Step #	Step Details	Expected Results	Actual Results		PASS ✓ / FAIL ✗
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected		✓
2	Click the Login button, top right	Should show Login Page	As Expected		✓
3	Enter User Credentials	Credentials are able to be entered	As Expected		✓
4	Click the login button to submit	Should flash Welcome \${username}! and redirect to home page	As Expected		✓

Test Case ID	2	Test Case Description	Test Signup Functionality		
Created By	Timmy Tram	Reviewed by	John Bagwell	Version	1.0







Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1	username = testuser1	
			2	email = testuser1@test.com	
			3	password = testpassword	
Test	Verify that the customer can create a new account				
Step #	Step Details	Expected Results	Actual Results	PASS ✓ / FAIL ✗	
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	✓	
2	Click Sign up button the top right	Should redirect to Signup Page	As Expected	✓	
3	Enter user Credentials	Credentials are able to be entered	As Expected	✓	
4	Click the sign in button to submit	Should flash User Created Successfully and redirect to Login Page	As Expected	✓	

Test Case ID	3	Test Case Description	User can view all locations		
Created By	Timmy Tram	Reviewed by	John Bagwell	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓

S#	Prerequisites	S#	Test Data	
1	Access to internet browser	1		
Test	Verify that any user can view all locations and individual location information			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Stay in Home directory	If user scrolls they should see all locations our site has to offer.	As Expected	
3	Click on any of the view Details button	It should take you to /locationInfo/{locationId} and you should be able to see the unique information of that place as well as any comments	As Expected	

Test Case ID	4	Test Case Description	User can leave behind reviews on places		
Created By	Timmy Tram	Reviewed by	Alan Yu	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1	Review = "This is a test review for the integration test"	
2	User is logged in. (Refer to Test Case 1)		2	Rating = 5	










Test	User clicks on a location and wants to leave a review			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Click on any of the view Details button	It should take you to /locationInfo/{locationId} and you should be able to see the unique information of that place as well as any comments	As Expected	
3	Click on the Write a Review! button	It should take you to /writeReview/{locationId} and you should be able to see a section where you can write a review	As Expected	
4	Enter Test Data	Should be able to enter in data	As Expected	
5	Click submit	It should flash a "review successfully created!" and redirect you back to locationInfo/{locationId}	As Expected	








Test Case ID	5	Test Case Description	User can bookmark a location		
Created By	Timmy Tram	Reviewed by	John Bagwell	Version	0.1

Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1		
2	User is logged in. (Refer to Test Case 1)		2		
Test	User wants to bookmark a location				
Step #	Step Details	Expected Results	Actual Results	PASS ✓ / FAIL ✗	
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	✓	
2	Click on any of the view Details button	It should take you to /locationInfo/{locationId} and you should be able to see the unique information of that place as well as any comments	As Expected	✓	
3	Click on the Bookmark button	It should flash a toast saying "Bookmark Created!" and the bookmark button now says Unbookmark	As Expected	✓	

Test Case ID	6	Test Case Description	User can view their bookmarks		
Created By	Timmy Tram	Reviewed by	Alan Yu	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓






S#	Prerequisites	S#	Test Data	
1	Access to internet browser	1		
2	User is logged in. (Refer to Test Case 1)	2		
3	User already has bookmarks (Refer to Test Case 5)	3		
Test	User wants to view their bookmarked locations			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Click on My Profile in the navbar	Should redirect to User's Profile Page	As Expected	
3	Stay on profile Page	Should be able to scroll through all bookmarked locations if you have many locations bookmarked	As Expected	
4	Click on any of the bookmark location cards	Should redirect to that specific location's details page	As Expected	


Test Case ID	7	Test Case Description	User can unbookmark locations		
Created By	Timmy Tram	Reviewed by	John Bagwell	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass 






S#	Prerequisites	S#	Test Data	
1	Access to internet browser	1		
2	User is logged in. (Refer to Test Case 1)	2		
3	User already has bookmarks (Refer to Test Case 5)	3		
Test	User wants to remove bookmarked locations			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Go to any location details page that you have bookmarked either through your profile page or home page or any other methods	Location details page is shown, should see a unbookmark button	As Expected	
3	Click on unbookmark button	Should flash a message saying "Bookmark deleted successfully!" and the unbookmark button is now a Bookmark button again	As Expected	
4	Click on My Profile in the navbar	Should redirect to User's Profile Page	As Expected	
5	Stay on profile Page	Should see that bookmark location is no longer listed	As Expected	

Test Case ID	8	Test Case Description	User can Logout		
Created By	Timmy Tram	Reviewed by	John Bagwell	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1		
2	User is logged in. (Refer to Test Case 1)		2		
Test	User wants to logout				
Step #	Step Details	Expected Results	Actual Results	PASS ✓ / FAIL ✗	
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	✓	
2	Click Logout button on navbar	Logout button now shows Login and signup ensuring user is logged out	As Expected	✓	








Test Case ID	9	Test Case Description	Ensure User cannot write a review if not logged in		
Created By	Timmy Tram	Reviewed by	John Bagwell	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1		


Test	Say the user is not logged in and they click on the write a review button			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Click on any of the view Details button	It should take you to /locationInfo/{locationId} and you should be able to see the unique information of that place as well as any comments	As Expected	
3	Click on the Write a Review! button	It should take you to /writeReview/{locationId}, however, user is met with a page saying, "YOU ARE NOT AUTHORIZED TO VIEW THIS PAGE"	As Expected	

Test Case ID	10	Test Case Description	<b>Ensure User cannot bookmark a location if not logged in</b>		
Created By	Timmy Tram	Reviewed by	Alan Yu	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass 
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1		



Test	Say the user clicks on the login to bookmark button because they aren't logged in			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Click on any of the view Details button	It should take you to /locationInfo/{locationId} and you should be able to see the unique information of that place as well as any comments. Bookmark button should say "Login to Bookmark"	As Expected	
3	Click on Login to Bookmark button	Should redirect you to the Login Page	As Expected	







Test Case ID	11	Test Case Description	Admin User can view Admin Dashboard		
Created By	Timmy Tram	Reviewed by	Alan Yu	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass ✓
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1	username = "CoffeeSpotAdmin"	
2			2	password = "admin123456"	
Test	Admin logs in and wants to view how many locations, users, bookmarks, and reviews there are				

Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/</a>	Site should open	As Expected	
2	Click the Login button, top right	Should show Login Page	As Expected	
3	Enter User Credentials	Credentials are able to be entered	As Expected	
4	Click the login button to submit	Should flash Welcome \${username}! and redirect to home page	As Expected	
5	Stay in Home Page and look in navbar for Dashboard/Analytics	Should be able to see this if logged in with admin account	As Expected	
6	Click on Dashboard/Analytics	Should redirect to /admin and see the number of locations, users, reviews and bookmarks	As Expected	

Test Case ID	12	Test Case Description	<b>Admin User wants to create a location</b>		
Created By	Timmy Tram	Reviewed by	Alan Yu	Version	1.0
Tester's Name	Timmy Tram	Date Tested	12/1/2024	Test Case (Pass/Fail)	Pass 
S#	Prerequisites		S#	Test Data	
1	Access to internet browser		1	Name = "Test"	
2	User is logged into Admin User and is on admin dashboard page (Refer to Test Case 11)		2	Address = "Test"	



3		3	Category = LIBRARY	
4		4	Image = image of location.png or .jpg	
Test	Admin user gets requested to create a new location, so they go and create one			
Step #	Step Details	Expected Results	Actual Results	PASS  / FAIL 
1	Navigate to <a href="https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/admin">https://csc648-01-fa24-csc648-01-fa24-team03-duplicate-w53a.vercel.app/admin</a>	Site should open	As Expected	
2	Click Create Location button	Should show form for creating a location	As Expected	
3	Fill in with relevant information	Should be able to fill in information	As Expected	
4	Click on Upload Image	Should see the image show up in the form and get a flash saying Image uploaded successfully	As Expected	
5	Click on Create Location button	Should get a flash saying "Location Created Successfully"	As Expected	

Features Tested	PASS  / FAIL 
Login Functionality	
Signup Functionality	
User can view all Locations on Home Page	
Users can leave behind Reviews on Locations	

Users can Bookmark a Location	✓
Users can view their Bookmarks	✓
Users can Unbookmark Locations	✓
User can Logout	✓
Users must be logged in to write Reviews	✓
Users cannot make Bookmarks unless they login	✓
Admins can view Admin Dashboard	✓
Admins can create a Location	✓

- The Integration Test file is: Coffee Spot Integration Tests.xlsx
- Link to Download Integration Test Spreadsheet: [Milestone 4 Folder Link](#)

## 2. Coding Practices

### I. Coding Style

#### Coding Standards

- **File and Variable Naming Conventions:**
  - React Components must use **PascalCase** (e.g., [MyComponent.tsx](#)).
    - All React component files need to follow the PascalCase naming convention.
    - An example: SearchForm.tsx contains a SearchForm component

- An exception to the PascalCase rule would be for Next.js routes. Next.js routes should be **lowercase**. Examples of these would include page.tsx, loading.tsx, layout.tsx, etc...

- <https://nextjs.org/docs/app/api-reference/file-conventions>

- Variables should be named in **camelCase** (e.g., `userInput`).

- Example: `[count, setCount] = useState<number>(0);`

- Constants must use **ALLCAPS** with underscores for separation (e.g., `API_ENDPOINT`).

- **Exports:**

- Only one React component per file. Use a **default export** instead of a named export.

- An example:

```
const MyComponent = (props: MyComponentProps) => {  
  ...  
}  
  
export default MyComponent
```

- **Styling and Formatting:**

- Use **Tailwind classes** for styling and avoid inline styles unless necessary.

- Example: When you're using a preset color for a background, use the name instead of the hex value

- `className="bg-tea-green"` over  
`className="bg-[#D1DAAF]"`

- Keep Tailwind classes in a logical order, grouping similar utilities (e.g., layout, spacing, typography).

- **Indentation:**

- Use **2 spaces** for indentation; tabs must be replaced with 2 spaces.

- **Code Formatting:**

- Ensure consistent formatting with **Prettier**, adhering to the [.prettierrc](#) configuration. Use VSCode's "Format Document" (Ctrl+Shift+P) to automatically have **Prettier** format your code.
- Additional rules can be found in `eslinttrc.json` and `tsconfig.json`

- **React-Specific Rules:**

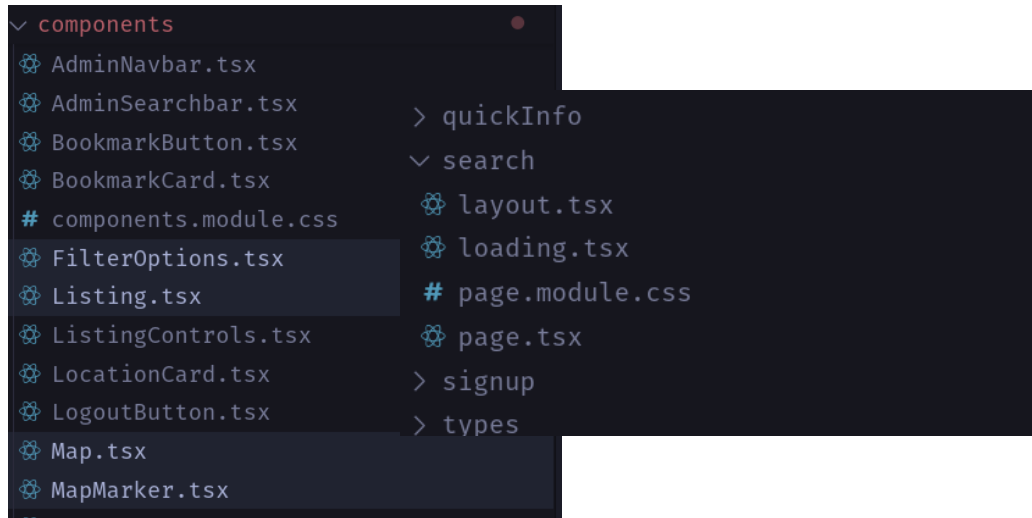
- Functional components are preferred over class components.
- Destructure props and avoid using `any` as a type (follow TypeScript conventions).
- Hooks like `useState` and `useEffect` should be declared at the top of the component.

- **Code Readability:**

- Write self-documenting code with clear, meaningful variable and function names.
- Include inline comments for non-obvious logic and JSDoc for complex functions.
- **File Structure:**
  - Prefer placing custom React components inside of the /components folder.
  - Next.js routes should only contain files relevant to the route itself.



The /search route contains components that are not relevant to the route itself. FilterOptions, Listing, Map, and MapMarker components should be moved to the /components folder



The above images show that components have been moved to /components and the search route now contains all the relevant files.

- **Testing:**

- All components and utility functions should include tests if possible, using the team's preferred testing framework Jest.

These standards promote readability, maintainability, and consistency across the codebase.

Some P1 features that use these coding standards:

1. Search Results:

<https://github.com/Miguel-Antonio-Logarta/csc648-01-fa24-csc648-01-fa24-tea-m03-duplicate/blob/dev/application/src/app/search/page.tsx>

2. Login Feature:

<https://github.com/Miguel-Antonio-Logarta/csc648-01-fa24-csc648-01-fa24-team03-duplicate/blob/dev/application/src/app/login/page.tsx>

3. Make Reviews:

<https://github.com/Miguel-Antonio-Logarta/csc648-01-fa24-csc648-01-fa24-team03-duplicate/blob/dev/application/src/app/api/reviews/createReview/%5BlocationId%5D/route.ts>

4. Admin Dashboard:

<https://github.com/Miguel-Antonio-Logarta/csc648-01-fa24-csc648-01-fa24-team03-duplicate/blob/dev/application/src/app/admin/page.tsx>

5. Searchbar:

<https://github.com/Miguel-Antonio-Logarta/csc648-01-fa24-csc648-01-fa24-team03-duplicate/blob/dev/application/src/app/components/NavSearch.tsx>

## II. (Extra Credit of 10 Pts) Documentation Generation

While we did not generate HTML documentation, we did create markdown files that documents our backend API and Frontend Hooks.

- For the Backend API Documentation, we listed all the possible endpoints and their HTTP Actions. We also provided a description of what it does and what

their expected Request or Response Body is. Additionally, we list if

Authentication is required and what level of Authorization is required to reach this endpoint.

- For the Frontend Hooks Documentation, we list all the hooks which the frontend team can use to easily interact with the backend. We provide a description of what each hook does, expected parameters, special notes if any, and an example of how you would use the hook in a Page file.
  - In other words, these hooks abstract away the logic when it comes to fetching data from the backend. This allows the frontend to focus mostly on the UI aspect of the frontend and not worry about the backend.

Documentation	Github URL
Backend API Documentation	<a href="#">Backend API Documentation Link</a>
Frontend Hooks Documentation	<a href="#">Frontend Hooks Documentation Link</a>

In addition to the documentation, the backend and hook related files have comments similar to the documentation shown to make it easier to understand what the endpoints



or hooks expect.

```
/**
 * @Auth - Required (ADMIN)
 * @Endpoint - POST /api/locations
 * @description - Creates a new location with the provided data.
 * @returns - the newly created location.
 */
export async function POST(req: NextRequest) {
  const session = await getServerSession(authOptions);

  if (!session || session.user.role !== 'ADMIN') {
    return NextResponse.json({ error: "Unauthorized." }, { status: 401 });
  }
  // You, last month • protect createReview, user update info, location...
```

```
/**
 * @Notes - Use this hook in the frontend to edit a location.
 * @param {any} formData - The data used to edit a location. You, yesterday • backend now actu
 * @param {Session} session - The session object.
 * @description - A custom hook that edits a location.
 * @returns {Object} - Returns an object containing the editLocation function and a loading state.
 */
const useEditLocation = (..) => {
  const [loading, setLoading] = useState(false);

  /**
   * @param locationId id of location to edit
   * @param formData form data to edit location
   * @param session used to check if user is admin
   */
  const editLocation = async (locationId: string, formData: any, session: Session) => {
```

## Bookmark Endpoints

### Get All Bookmarks

- **Endpoint:** `/api/bookmarks`
- **Method:** `GET`
- **Description:** Get all the bookmarks from the database
- **Response Body:**

```
[
  {
    "id": "bookmark-id",
    "user": {
      "id": "user-id",
      "username": "username",
      "email": "example@example.com"
    },
    "creationDate": "2024-10-08T18:58:32.977Z"
  },
  ...
]
```

### Create a Bookmark

- **Endpoint:** `/api/bookmarks/createBookmark/${locationId}`
- **Method:** `POST`
- **Description:** Create a bookmark when signed in.
- **Authentication:** REQUIRED
- **Authorization:** MUST BE CUSTOMER OR HIGHER ROLE
- **Request Body:**

```
{
  "userId": session.user.id
}
```

# Hooks Documentation

## Overview

This folder contains CUSTOM React hooks. These hooks serves as a way to abstract away a lot of the boilerplate when it comes to safely fetching data from the backend api.

## Hook Details

### useCreateBookmark.ts

- **Purpose:** Handles creation of a bookmark for a location assuming the user is logged in.
- **Parameters:**
  - `locationId` : The id of the given location to bookmark
  - `session` : The session object from `useSession()`
- **Special Note:** This uses `window.location.reload()`
- **Usage Example:**

```
const { createBookmark, loading } = useCreateBookmark();

const handleClick = async () => {
  if (session) {
    await createBookmark(locationId, session);
  }
}
```



### useCreateLocation.ts

- **Purpose:** Handles the creation of a location
- **Parameters:**