

# GITHUB API

Presenters:

Ashwini Bailangadi,  
Akshith Kumar Reddy Balappagari Gnaneswara,  
Libina Bovan Thomas,  
Venkata Gowtham Reddy Iska,  
Rohith Surya Podugu

---



# GitHub in a Nutshell

- GitHub is a web-based platform.
- It's used for version control and collaboration.
- Developers host, manage, and collaborate on code.
- It leverages Git, a distributed version control system.
- Enables teams to work together seamlessly.



# Automation and Integration with GitHub

- Developers often need to programmatically interact with GitHub for tasks like creating repositories, managing issues, and automating workflows.
- Essential in software development to streamline processes and improve efficiency.
- **Octokit.js** provides a convenient way for developers to programmatically access and manage GitHub resources from their JavaScript applications



## What is Octokit.js?

It is a JavaScript library developed by GitHub for interacting with the GitHub API. It provides a convenient way for developers to programmatically access and manage GitHub resources from their JavaScript applications.

### Key features

- Authentication support - Personal Access token, OAuth
- CRUD operations for repositories, issues, and more
- Pagination handling
- Rate limiting and error handling
- Event handling

### Installation

Typically using npm or yarn.



# **EXPLORING GITHUB API USING OCTOKIT LIBRARY**



# Authentication

Step 1: Generate a Personal Access Token from GitHub


Step 2: Install Octokit.js

Step 3: Use the Personal Access Token with Octokit.js

Create an Octokit instance with your personal access token and then use the `users.getAuthenticated()` method to retrieve information about the authenticated user. This method fetches details about the user whose token is used for authentication.

```
// Create a personal access token at https://github.com/settings/tokens/new?scopes=repo
const octokit = new Octokit({ auth: 'personal-access-token123' });

// Compare: https://docs.github.com/en/rest/reference/users#get-the-authenticated-user
const {
  data: { login },
} = await octokit.rest.users.getAuthenticated();
console.log("Hello, %s", login);
```



```
router.get('/:owner/:repo', async (req, res) => {
  const { owner, repo } = req.params;
  try {
    const { data: commits } = await octokit.repos.listCommits({ owner, repo });
    // Extract only the desired fields from each commit
    const filteredCommits = commits.map((commit) => ({
      sha: commit.sha,
      author: commit.commit.author.name,
      date: commit.commit.author.date,
      message: commit.commit.message,
    }));
    res.json(filteredCommits);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

## Git Commits

It records the changes made to the code, which can include additions, modifications, or deletions of files and lines of code.

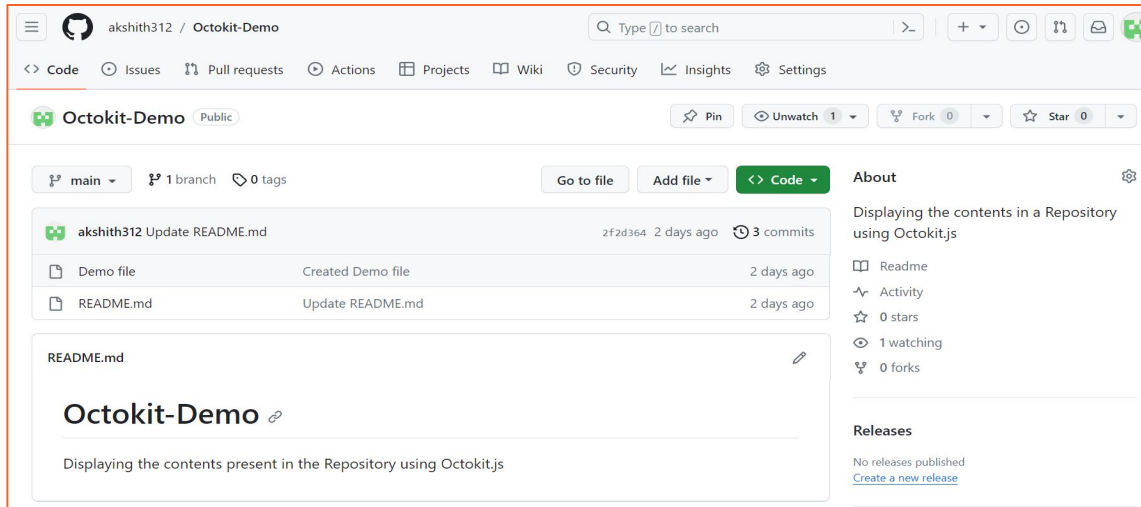
In this example, `octokit.repos.listCommits` is used to fetch a list of commits from a specific GitHub repository.



# GitHub Repositories

What is a repository?

- A repository is the most basic element of GitHub. It's a place where you can store your code, your files, and each file's revision history.
- Repositories can have multiple collaborators and can be either public or private.







# Performing operations on a repository with GitHub API

- We can create, update, and retrieve information of a repository using the GitHub API.
- Using GitHub API to perform such operations will help us in scenarios where we might need to automate the creation, updation and deletion of repositories which can help developers streamline this process and save time.
- It also helps us in reporting the metrics related to the repository. Analyzing this data might give one the idea of project progress, code quality and collaboration.
- We use Octokit library to perform operations on a repository through the GitHub API.
- Octokit.repos helps us to perform the operations through some functions.

```
octokit.rest.repos.delete({  
  owner,  
  repo,  
});
```

# Create a Repository

- We define a route which takes a POST request.
- In this request we pass the repository name that we would like to create as a part of the URL.
- We use `octokit.repos.createForAuthenticatedUser` method to create a repository for the associated user.
- We also have other parameters that we can include to the method like `description`, `private` (boolean value) which will be linked to the repository.

```
(property) createForAuthenticatedUser: (params?: (RequestParameters & Omit<{  
  name: string;  
  description?: string | undefined;  
  homepage?: string | undefined;  
  private?: boolean | undefined;  
  has_issues?: boolean | undefined;  
  ... 17 more ...;  
  is_template?: boolean | undefined;  
}, "baseUrl" | ... 1 more ... | "mediaType">)) | undefined) => Promise<...>
```



# Get a Repository

- We define a route which takes a GET request.
- In this request we pass the repository name that we would like to fetch as a part of the URL.
- We use `octokit.repos.getContent` method to get the files related to the repository.

```
(property) getContent: (params?: (RequestParameters & Omit<ToOctokitParameters<{  
  parameters: {  
    query?: {  
      ref?: string | undefined;  
    } | undefined;  
    path: {  
      owner: string;  
      repo: string;  
      path: string;  
    };  
  };  
  responses: {  
    200: {
```

## Understanding Git Branches :

**1. Git Branches** - Git uses branches to manage different lines of development within a project.

**2. Purpose** - Branches help isolate features, bug fixes, and experiments, keeping the main codebase stable.

**3. Main Branches** - Master/Main: Production-ready code. & Develop: Ongoing development and integration.

**4. Working on a Branch** - Make changes, commit as needed on the branch.

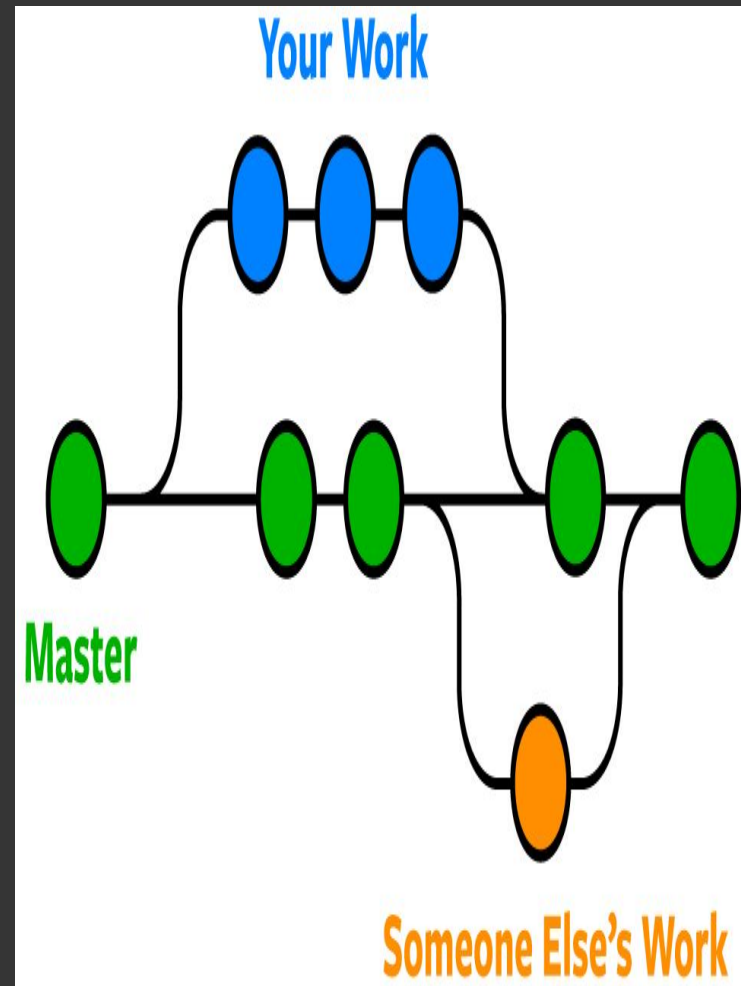
**5. Merging** - Merge changes from a branch back into the main branch.

### 6. Best Practices

- Keep branches short-lived.
- Merge frequently from the main branch.
- Use meaningful branch names.

### 7. Summary

- Git branches enable parallel development, collaboration, and version control.



## Creating a Git Branch with Octokit :

Octokit is a powerful JavaScript library for programmatically interacting with GitHub, enabling automation and integration.

- Authentication
  - Begin by securely authenticating with your GitHub account, typically using a personal access token. This ensures proper authorization.
- Select Repository
- Specify the target GitHub repository where the new branch will be created. This ensures precision in your actions.
- Create a Branch
- Utilize Octokit's robust API to programmatically create a new branch.

### Summary

- Recap the key steps:
- Securely authenticate using your access token.
- Precisely select the target repository.
- Leverage Octokit to automate the creation of a new Git branch.

```
1  const express = require('express');
2  const octokit = require('../github'); // Import Octokit
3  const router = express.Router();
4  router.post('/create/:owner/:repo/:branchName', async (req, res) => {
5      const {owner,repo,branchName} = req.params;
6      const commits = await octokit.repos.listCommits({
7          owner,
8          repo
9      });
10     const latestCommit = commits.data[0].sha;
11     const response = await octokit.git.createRef({
12         owner,
13         repo,
14         ref: `refs/heads/${branchName}`,
15         sha: latestCommit,
16     });
17     console.log(response);
18     res.json("Successful");
19 });
20 module.exports = router;
```

# Deleting a Git Branch with Octokit :

## Authentication

- Commence the process by securely authenticating your GitHub account, typically by employing a personal access token, ensuring proper authorization for actions.

## Select Repository

- Specify the GitHub repository in which you intend to delete a branch, maintaining precision in your operations.

## Delete a Branch

- Harness Octokit's robust API to programmatically delete a branch. Here's a professional JavaScript example:

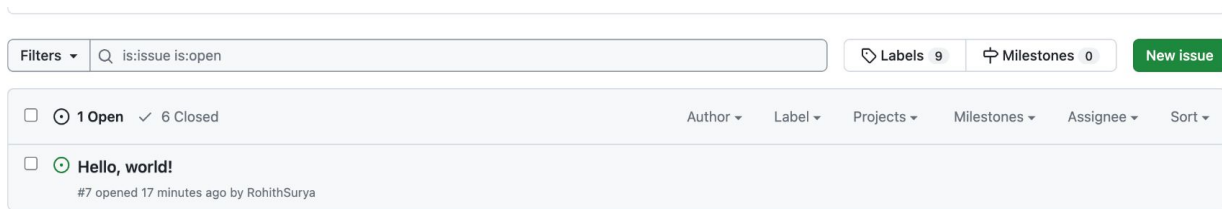
## Summary

- Authenticate securely using your access token.
- Precisely select the repository.
- Leverage Octokit for automated deletion of a Git branch.

```
1  const express = require('express');
2  const octokit = require('..github'); // Import Octokit
3  const router = express.Router();
4  router.delete('/delete/:owner/:repo/:branchName', async (req, res) => {    // Route to delete a branch
5      const { owner, repo, branchName } = req.params;
6      try {                                                                    // Get the SHA of the branch yo
7          const branchRef = `heads/${branchName}`;
8          const refResponse = await octokit.git.getRef({ owner, repo, ref: `heads/${branchName}` });
9          const branchSha = refResponse.data.object.sha;
10         const deleteResponse = await octokit.git.deleteRef({ owner, repo, ref: `heads/${branchName}` });
11         console.log(deleteResponse);
12         res.json("Branch has been successfully deleted.");
13     } catch (error) {
14         console.error("Error deleting branch:", error);
15         res.status(500).json("Error deleting branch.");
16     }
17 });
18 module.exports = router;
```

# Issues

1. Issues let you track your work on GitHub, where development happens.
2. Issues can be a bug fix, feature request or about inconsistent documentation
3. Issues can be assigned to a team member to work on them, and can have status of whether its open, closed.



💡 **ProTip!** `no:milestone` will show everything without a milestone.

# Create an issue

1. Although we can create an issue using GitHub user interface, its also possible to create issues using Github REST API.
2. Creating issues using API is useful in scenarios where a test case fail will trigger a issue creation.

```
router.post("/", async (req, res) => {
  const { body } = req;
  try {
    const response = await octokit.rest.issues.create(body);
    const id = response.data.number;
    res.json({ success: `Issue created: ${id}` });
  } catch (error) {
    res.json({ error: "Error occurred when creating issue" });
  }
});
```



# GET an issue

1. GET issue API fetches the issue details its status, assignee status and other information related to the issue.
2. It is necessary to keep in mind that an issue is related to a particular user and repository, so its required to send these details in the API.

```
router.get("/:owner/:repo/:id", async (req, res) => {
  const { params } = req;
  const { owner, repo, id } = params;
  try {
    const response = await octokit.rest.issues.get({
      owner: owner,
      repo: repo,
      issue_number: id,
    });
    res.json(response.data);
  } catch (error) {
    res.json({ error: `Error occurred while fetching issue ${id}` });
  }
});
```



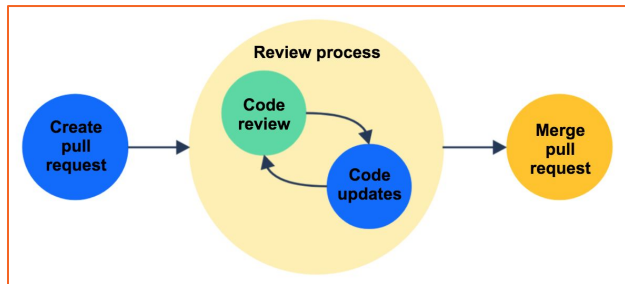
# What is a Pull Request?

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub.

Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

Pull Request= Code + Issue + Code Comments

## Lifecycle of a PR



## Components of a PR

**Title:** A brief descriptive title for the changes.

**Description:** An explanation of what the PR does, why it is needed and how to test it.

**Branches:** The source branch (the changes you want to merge) and the target branch (where you want to merge the changes)

## BENEFITS OF USING PULL REQUESTS

Encourages collaboration and peer review

Provides a structured way to suggest and discuss changes

Facilitates automatic and manual testing

Improves code quality and maintainability

Serves as a documentation for execution of the codebase.

# Create Pull Requests

1. We define a new route that listens for a POST request to create a pull request.
2. The route expects the owner and repo as parameters in the URL and expects additional data (title, head, base, body) to be provided in the request body.
3. Inside the route handler, we use the `octokit.pulls.create` method to create a pull request. We pass in the owner, repo, title, head (the source branch), base (the target branch), and an optional description (body) for the pull request.

```
// Route to create a pull request in a GitHub repository
router.post('/:owner/:repo', async (req, res) => {
  const { owner, repo } = req.params;
  const { title, head, base, body } = req.body; // Assuming these are provided in the request body

  try {
    // Create a pull request
    const pullRequest = await octokit.pulls.create({
      owner: owner,
      repo: repo,
      title: title,
      head: head, // The branch you want to merge from
      base: base, // The branch you want to merge into
      body: body, // Description or comments for the pull request
    });
    // Extract specific details
    const { number, title, user, state, created_at, html_url } = pullRequest.data;

    res.status(201).json({
      message: 'Pull request created successfully.',
      pullRequest: {
        number: number,
        title: title,
        user: user.login, // Extracting the login (username) of the user who created the pull request
      }
    });
  } catch (error) {
    // Handle error
  }
});
```

# List Pull Requests

1. We define a new route that listens for a GET request to list all pull requests for a GitHub repository.
2. The route expects the owner and repo as parameters in the URL.
3. Inside the route handler, we use the `octokit.pulls.list` method to retrieve a list of pull requests for the specified repository.

```
4 // Route to list all pull requests in a GitHub repository
5 router.get('/:owner/:repo', async (req, res) => {
6   const { owner, repo } = req.params;
7
8   try {
9     // List all pull requests for the repository
10    const pullRequests = await octokit.pulls.list({
11      owner: owner,
12      repo: repo,
13    });
14
15    res.status(200).json({
16      pullRequests: pullRequests.data,
17    });
18  } catch (error) {
19    res.status(500).json({ error: error.message });
20  }
21 }
```



## Conclusion

- We've explored the combination of GitHub API and Octokit.js for enhancing development and automating tasks.
- Octokit.js offers an efficient way to securely connect to the GitHub platform, benefiting from Node.js's capabilities for asynchronous API requests.
- From creating repositories to managing issues and pull requests, Octokit.js empowers developers to streamline their GitHub interactions.