# Exercise: API Testing with C#

Exercises for the ["Back-End Test Automation"](https://softuni.bg/trainings/4729/back-end-test-automation-november-2024) course @ SoftUni

## GitHub API endpoints

GitHub Issues provides the standard RESTful **API endpoints**, which you can access with HTTP client from [**https://api.github.com**](https://api.github.com):

* **GET endpoints** – respond with **JSON** object as result.
  + **GET /repos/{user}/{repo}/issues** –returns the **issues** in given GitHub repo.
  + **GET /repos/{user}/{repo}/issues/{num}** –returns the specified **issue**.
  + **GET /repos/{user}/{repo}/issues/{num}/comments** –returns the **comments** for an issue.
  + **GET /repos/{user}/{repo}/issues/comments/{id}** –returns the specified **comment**.
* **POST / PATCH / DELETE endpoints** – all of them need **authentication**.
  + **POST /repos/{user}/{repo}/issues** –creates a new **issue**.
  + **PATCH /repos/{user}/{repo}/issues/{num}** –modifies the specified **issue**.
  + **POST /repos/{user}/{repo}/issues/{num}/comments** –creates a new **comment** for certain issue.
  + **PATCH /repos/{user}/{repo}/issues/comments/{id}** –modifies existing **comment**.
  + **DELETE /repos/{user}/{repo}/issues/comments/{id}** –deletes existing **comment**.

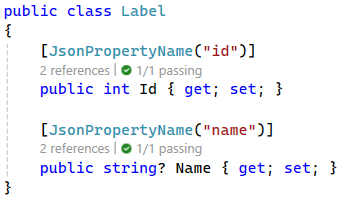
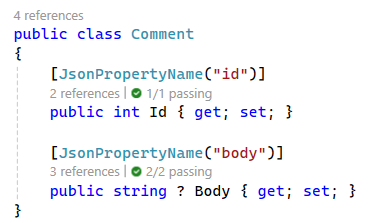
## GitHub API Project

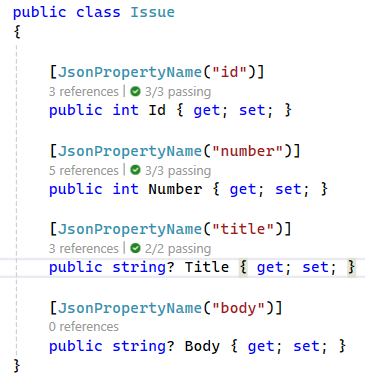
* **Projects:**
* The solution consists of two projects: **RestSharpServices** and **TestGitHubApi.**
* **RestSharpServices** is where you'll write **methods** that interact with the **GitHub API using RestSharp**.
* **TestGitHubApi** is where you'll write tests to verify the functionality of the **RestSharpServices** methods.
* **Models:**
  + Located in the Models folder of the RestSharpServices project.
  + These classes represent the data that the GitHub API will send and receive.
  + These classes are POCOs (Plain Old CLR Objects) and are already provided to you.

**Your main tasks will be to fill in the RestSharpServices.cs class with methods that use RestSharp to interact with GitHub's API and to create tests in the TestGitHubApi.cs file to ensure your code is working correctly.**

## Understanding Models

* Open the **Models folder** in the **RestSharpServices** project.
* Review the Issue.cs, Label.cs, and Comment.cs classes.
* Understand the properties of each class and how they map to the JSON structure returned by GitHub's API.

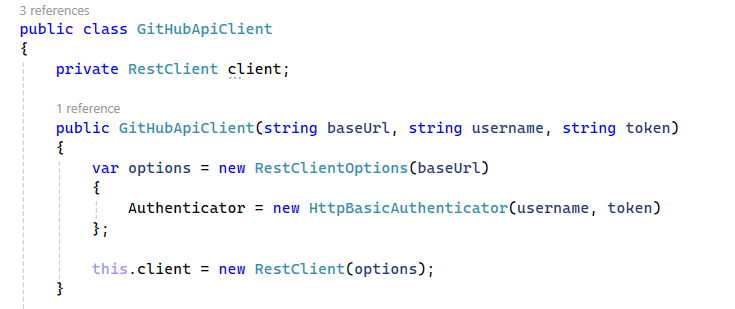
 



* If you want to know more about the additional properties of each one. Refer to [GitHub API Documentation](https://docs.github.com/en/rest/issues?apiVersion=2022-11-28)

## Setting Up the RestSharp Client

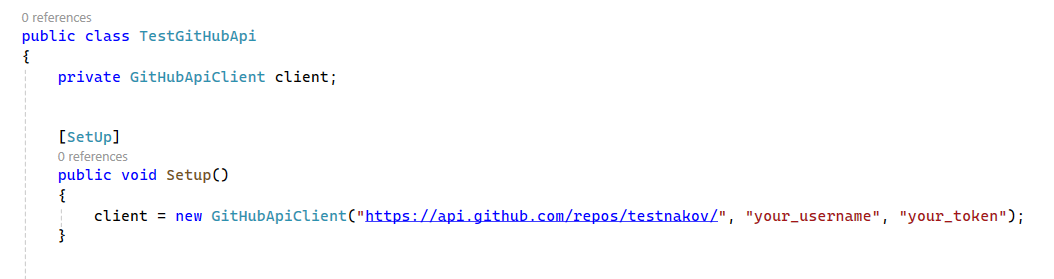
* Open the **RestSharpServices.cs** file in the **RestSharpServices** project.
* Create a constructor in the **GitHubApiClient** class that initializes the **RestClient** with the base URL for GitHub's API and authentication details.



* The **constructor** takes **three parameters:**
  + **baseUrl:** The root URL of the GitHub API. This is where all API requests will start from.
  + **username:** Your GitHub username.
  + **token:** Your GitHub personal access token. This is used for authentication and allows you to interact with the GitHub API.
* **RestClientOptions is a configuration object for the RestClient:**
  + **baseUrl** is assigned to the **RestClientOptions,** which means that all requests made by the **RestClient** will start with **this URL.**
  + **Authenticator** is set to a new instance of **HttpBasicAuthenticator,** which is a built-in authenticator in **RestSharp** that handles **HTTP Basic Authentication.** It takes the **username** and **token** and uses them to authenticate your **requests to the GitHub API.**
* **RestClient Initialization:**
  + Creates a new RestClient using the options we just configured. The RestClient is what actually sends HTTP requests and receives responses.

## TestGitHubApi project

* A dedicated place where **all the tests** related to the GitHub API functionality will reside.
* **References the RestSharpServices project** to gain access to the classes and methods you wish to test.
* Has a **TestGitHubApi class** where you will write your test cases.



* The Setup() method initializes the client field with a new instance of GitHubApiClient, configured with the base URL of the GitHub API and the repository you are targeting.
* **https://api.github.com/repos/testnakov/** is the base URL passed to the GitHubApiClient.

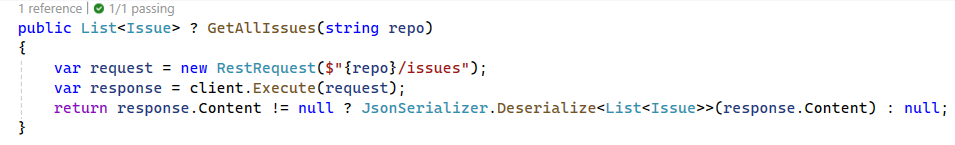
## Implementing API Methods and Tests

Your assignment is to implement a set of methods that interact with the GitHub API and then write tests to verify their functionality. Focus on ensuring that each method correctly performs the desired action, such as retrieving, creating, or deleting data. You have two options for approaching this task:

* **Incremental Approach:** Write one method, then immediately write the corresponding test to validate it. Repeat this process for each method. This approach helps you focus on one piece of functionality at a time and may make debugging easier.
* **Bulk Implementation Approach:** Write all the methods first, and once you're done, move on to write the tests for each method. This might suit you if you prefer to stay in the 'development zone' and focus on testing afterward.

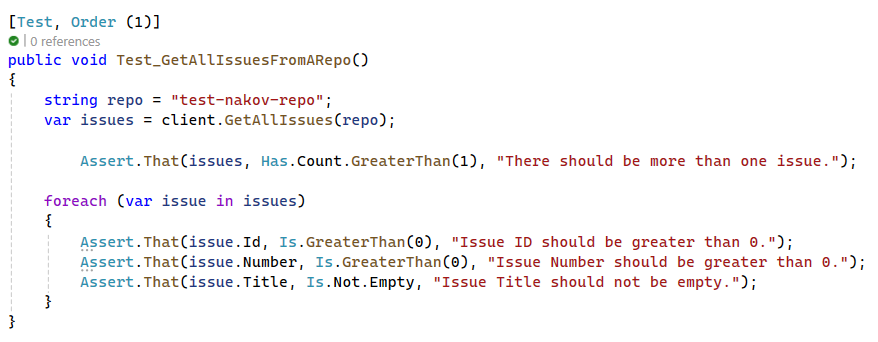
### Get All Issues Method

* **public List<Issue> GetAllIssues(string repo)**
* Retrieves a list of all issues for a specified repository. This method constructs a GET request to the GitHub issues endpoint and returns the issues if the response has content.



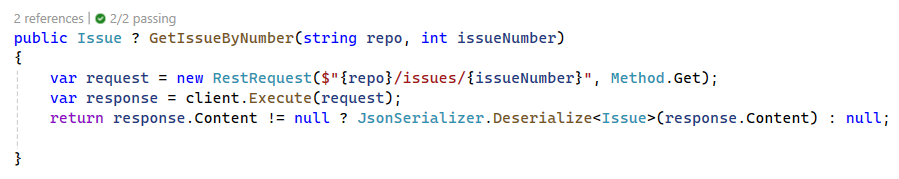
### Test Get All Issues from a Repo

* **public void Test\_GetAllIssuesFromARepo()**



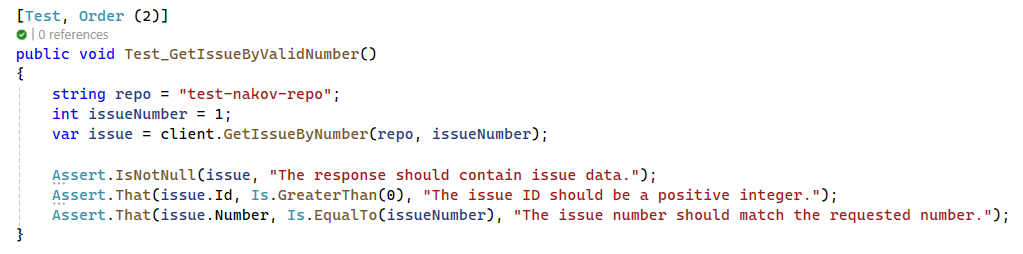
### Get Issue by Number

* **public Issue GetIssueByNumber(string repo, int issueNumber)**
* Fetches a single issue from a repository based on the issue number. It sends a GET request to the specific issue's endpoint and deserializes the response into an Issue object.



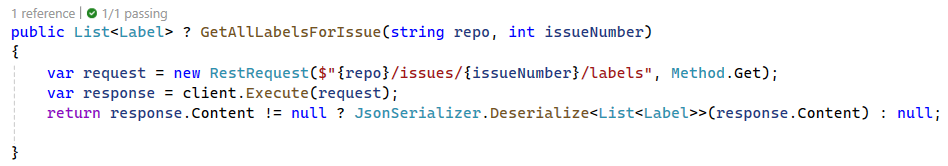
### Test Get Issue by Valid Number

* **public void Test\_GetIssueByValidNumber()**

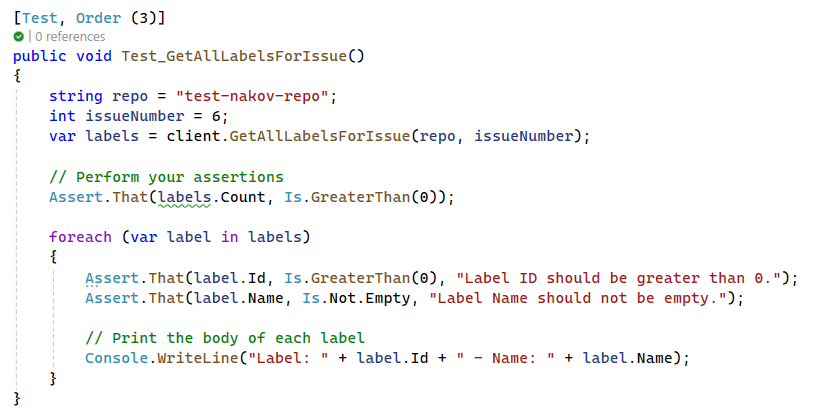


### Get All Labels for Issue

* **public List<Label> GetAllLabelsForIssue(string repo, int issueNumber)**
* Retrieves all labels for a particular issue within a repository. It sends a GET request to the issue's labels endpoint and deserializes the returned labels.

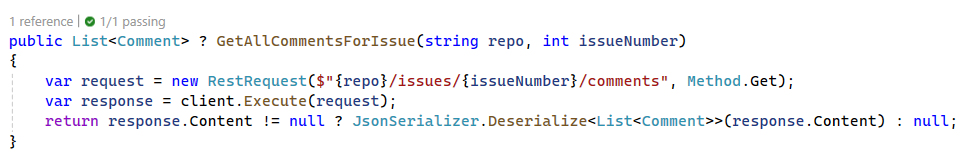


### Test Get All Labels for Issue



### Get All Comments for Issue

* **public List<Comment> GetAllCommentsForIssue(string repo, int issueNumber)**
* Gets all comments for a specified issue in a repository. The method sends a GET request to the issue's comments endpoint and returns a list of comments.

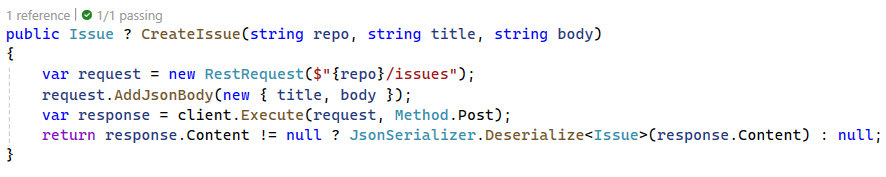


### Test Get All Comments for Issue



### Create Issue

* **public Issue CreateIssue(string repo, string title, string body)**
* Creates a new issue in the given repository with a specified title and body. This method sends a POST request with the issue details and returns the created issue.



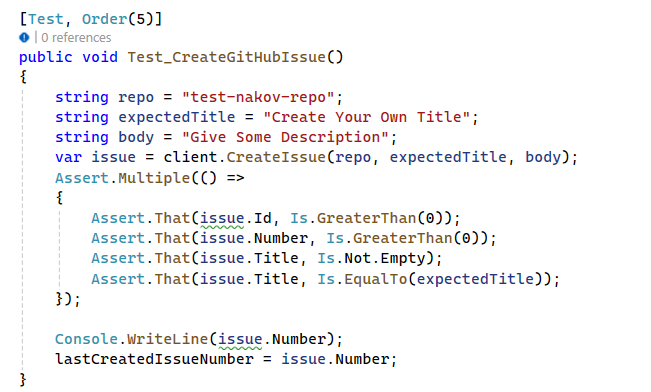
**Note: To keep track of the state across multiple test runs within the same execution context you can use static fields.**

* **lastCreatedIssueNumber**
  + This field stores the issue number of the most recently created issue in the GitHub repository. We use this number to refer to the issue in subsequent tests that might need to add comments to it, close it, or perform other operations.
* **lastCreatedCommentId**
  + Similarly, this field holds the ID of the most recently created comment. If we create a comment in one test, we can use this ID to reference that comment in later tests, such as when we want to edit or delete the comment.

A screenshot of a computer

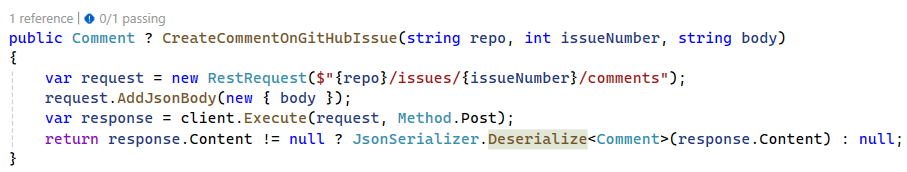
Description automatically generated

### Test Create Issue



### Create Comment on GitHub Issue

* **public Comment CreateCommentOnGitHubIssue(string repo, int issueNumber, string** body)
* Posts a new comment to a specific issue in the repository. This method sends a POST request with the comment's body and returns the created comment.



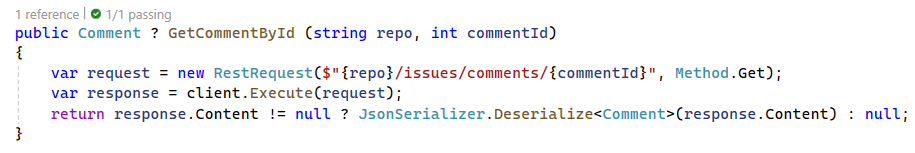
### Test Create Comment on GitHub Issue

A close-up of a computer screen

Description automatically generated

### Get Comment by Id

* **public Comment GetCommentById (string repo, int commentId)**
* Retrieves a specific comment by its ID from a repository's issue. It sends a GET request to the comment's endpoint and returns the comment if found.



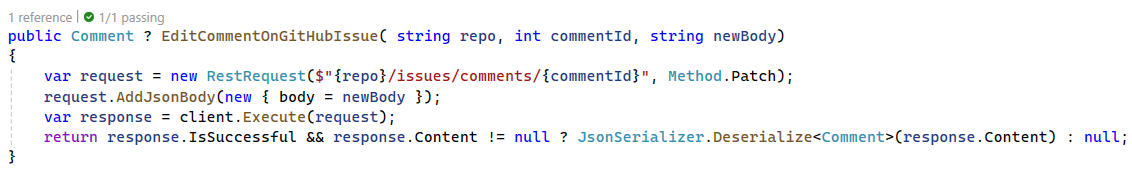
### Test Get Comment by Id

A computer code with text

Description automatically generated

### Edit Comment on Git Hub Issue

* **public Comment ? EditCommentOnGitHubIssue( string repo, int commentId, string newBody)**
* Updates the body of an existing comment by comment ID on a GitHub issue. This method sends a PATCH request with the new body content and returns the updated comment.

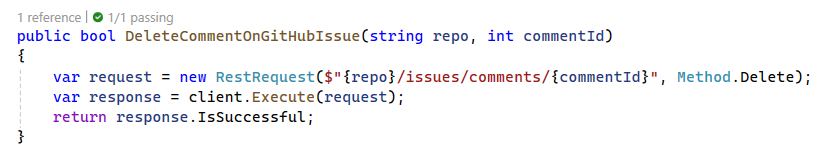


### Test Edit Comment on Git Hub Issue



### Delete Comment on Git Hub Issue

* **public bool DeleteCommentOnGitHubIssue(string repo, int commentId)**
* Executes a deletion of a specified comment from a GitHub issue. The method sends a DELETE request to the GitHub API and returns a boolean indicating whether the deletion was successful



### Test Delete Comment on Git Hub Issue

