VS Code Extension

The Roost GPT VS code extension allows you to generate tests for your code using RoostGPT with just a click, straight from your VS Code workspace.

Download:

https://marketplace.visualstudio.com/items?itemName=RoostGPT.roostgpt

Installation:

To use the RoostGPT VS Code extension, you must have VS Code ready and installed in your system, as well as any dependencies that are required to run your code, as RoostGPT will often run its generated test code to improve it.

You can download and install VS Code for your Operating system here.

After VS Code is Successfully installed in your system, you can go ahead and download the Roost GPT VS Code extension from the VS Code marketplace, just simply search for Roost GPT in the extension store. Alternatively, you can download and install the VS code extension from here.

Once the extension is installed, you are ready to generate tests for your code.

Configuration:

Once the extension has been successfully installed in your system, you can then proceed with configuring the extension to start generating tests. This involves providing information that is required for test generation.
To configure the extension to use it, simply open the extension settings for Roost GPT, you can search for it in the extension store, or you can find it in the list of your installed extensions.

You can then set up the required values according to your workspace and needs. Following are the required fields which will be required for you to set no matter what:

Required Fields

- Roost Token: you can get your roost token from my profile page in app.roost.ai. If you don't have a Roost token, you can sign up for a free trial and try out RoostGPT for free, using your organization email from here.
- Roost Domain: Enter the Domain in which the provided roost token is active, the default value is app.roost.ai.
- Timeout: Set the timeout for test generation (in hours), default value is 1.
- Language: Select the language your workspace/source code is written in, currently supports Java, Python, Go, NodeJS, C# and C++.
- Board Type: Select the type of scrum/kanban board, Required for functional tests. set as none other test types, supports none, jira and Azure boards. default value is none.
- Iterations: Set the number of iterations for test generation, If you provide an iteration value greater than 0 then it will run the generated test cases and pass the error that occurred(if any) to the ai model then update the test case inside the same file and run again till the number of times of iteration and stop if it ran successfully in between. The default value is 1.
- Telemetry: Set as False if you do not want to send telemetry data to roost. The default value is true.
- Generative AI model: Select which model to use for test code generation, which supports OpenAI, Google Vertex, Azure Open AI, Claude AI, DeepSeek, AWS Bedrock, and DBRX Hosted Models including OpenSource Hosted Models (LLAMA2 and starchat).
- ProvideInput: Set as true if you want to provide your own input before test generation, you will be prompted for input before the test generation begins. The default value is false.
- MaxDepth: This is used to specify how deep into the workspace the extension will traverse to scan for files to generate tests for. default value is traverse to all subdirectories.
- EnvFile: This can be used to provide the path to an env file, which will provide RoostGPT with env file with user environment variables which will be taken into account in the test generation progress. You can keep this field empty.
- AI model Details:
  - If the Generative AI model is Open AI:
    - OpenAI API Key: Provide your OpenAI API key if you plan on using an OpenAI as the generative AI model to generate your test cases.
    - OpenAI API model: Provide the AI model the provided API key has access to, supports gpt-4o, gpt-4, gpt-3.5-turbo.
  - If the Generative AI model is Google Vertex:
    - Vertex Bearer Token: Provide your vertex Bearer Token if you plan to use Google Vertex as the generative AI model to generate your test cases.
    - Vertex Project ID: Provide the ID of your Google vertex project.
    - Vertex Region: Enter the region where your vertex region is present
    - Vertex Model: Select the Vertex model to be used for code generation; supports text-bison, code-bison, and codechat-bison. Default value is text-bison.
  - If Generative AI model is Claude AI:

    - AI model: Select the AI model to be used from the dropdown menu.
    - API Key: Provide your Claude AI API key.
  - If the Generative AI model is Azure Open AI:
    - API Key: Provide the API Key for your Azure Open AI model.
    - API Endpoint: Provide the API Endpoint where your Azure Open AI model is hosted.
    - Deployment Name: Enter the Deployment Name for your Azure Open AI API model.
  - If the Generative AI model is Open Source:
    - Open Source Model Endpoint: provide the endpoint for the open source model if you plan on using one of the roost provided open source models, you need to provide it in the format 'http://MODEL_IP:5000/generate' where MODEL_IP is the IP address for the instance where you have the model's container running.
    - Open Source AI model: Select the AI model to be used for test generation, supports meta-llama/Llama-2-13b-chat, and HuggingFaceH4/starchat-beta. The default value is meta-llama/Llama-2-13b-chat.

Advanced Fields (optional):

- Optional fields are available to use the LLM effectively like
  - advanced.UseAssistant can be set to True for gpt turbo models,
  - AiTemperature can be set to values other than default (0.2) to control the test quality,
- VerifyTest can be set to a boolean value True. Default value is False,
- UseDocker can be set to True to generate the tests inside a docker. This will require Docker Engine server to be accessible locally.

Test Generation:

Once your extension configuration is complete, you can then start using the VS Code extension to generate tests for your workspace. To generate tests, simply right-click on a file in your Explorer menu and select the type of test you want to generate from the context menu that shows up. Note that each test type has some requirements to start test generation.
The test types currently supported are as follows:

- Unit Tests.
- API Tests.
- Functional Tests.
- Integration.

Test Requirements

Following are the Required Fields, and other instructions for test generation according to each supported test type:

- Unit Tests: To generate Unit tests, just simply select the directory your file is present in, right-click on a file and select unit test generation, and then select the test framework you want to use from the popup. It will generate unit tests for all the files present in that file's parent directory. No extra fields other than the above-mentioned required fields are needed for unit test generation. Make sure that the language set in the extension settings matches the language your source code is in.
  - If you want the tests to be generated for only a few specified functions and not for the entire codebase, then provide the function names for which you want the tests to be generated in the FunctionsToTest input box in the advanced section of the extension settings in a comma separated fashion (e.g. Func1, Func2,...). This will ensure that tests are generated only for the specified functions.
  - For Java unit tests, please make sure to trigger the test execution from a valid module that contains a valid pom.xml, or generate tests for the entire project if pom.xml is present in the workspace folder.
  - For React Unit tests, you need to use GPT-4 turbo model and set use assistant as true in the advanced section of the vs-code extension settings.
- API Tests: To generate API tests, you need to right-click on Postman collection json or swagger API spec file and then select the Generate API tests. If you choose any file other than your API spec file, the test generation will fail. Then you need to select the test framework to be used for test generation (artillery, postman, or rest-assured) from the provided popup. No extra fields other than the above-mentioned required fields are needed for API Tests. For API tests, you can also filter out which HTTP verbs(such as post, get, etc.) will be tested by changing the HttpFilters setting from the advanced section. Please note that if you select postman as the test framework you will need newman cli installed in your system in order to run generated tests from the RoostGPT extension, you can install newman cli by using the command: npm install -g newman.
  - If you want to generate API tests for some specific HTTP verbs (get, post, put, patch, delete, etc.) then you can select which specific verbs need to be tested in the advanced section of the extension settings under the HTTP filters attribute.
  - If you want to generate API tests for some specific API endpoints matching a given regex pattern, you can set the regex pattern in the advanced tab of the extension settings under the HTTP endpoints for testing attribute.
  - For karate and rest-assured tests, make sure that the Test generation is triggered from within a valid java/maven repo, i.e. put the api spec file in the java repo and put start the test generation from there.
- Integration Tests: When generating Integration tests, you need to right-click on your Postman collection json or swagger API spec file and then select the Integration Tests option. If you choose any file other than your API spec file, the test generation will fail, after selecting the option, then need to select the test framework to be used for test generation (artillery, postman, or rest-assured), then you need to select the type of your gherkin template you can either select file and browse to your gherkin template file or you can choose URL and provide the URL to your Gherkin template. No extra fields other than the above-mentioned required fields are needed for Integration Tests. Please note that if

you select Postman as the test framework you will need newman cli installed in your system to run generated tests from the RoostGPT extension, you can install newman cli by using the command: npm install -g newman. For karate and rest-assured tests, make sure that the Test generation is triggered from within a valid java/maven repo, i.e. put the api spec file in the java repo and put start the test generation from there.

- Functional tests: To generate functional tests, you need to select your board type to be either JIRA or Azure and Then the Below details are also required:
  - If the Board Type is Jira:
    - Jira Email.
    - Jira Token.
    - Jira Hostname.
  - If the Board Type is Azure:
    - Azure Org.
    - Azure Token.
    - Azure Project.

Improve and Analyze Generated Tests

After the test generation process is complete, a side panel will open, showing you all the generated test files, you can select the file you want to view by using the dropdown provided, if you want you can also edit the files in the panel itself and save your changes using the provided save button.
If you want to run the generated tests, you can do so from the provided run button in the side panel, doing so will run the selected test file. NOTE that you will need to have all the dependencies required for running the tests installed in your local system for test generation to take place. and for artillery tests, after you click the run button, you will be prompted to enter the target URL for the tests, if you want to provide a target URL, please provide so in the input box, and then you will be asked if you want to upload a .env file to provide environment variables, if yes then you can upload the env file for the same.
If you are not satisfied with the generated tests and want some improvements or changes in the test, then at the bottom of the side panel, you will find a feedback prompt, enter the feedback prompt that you want to give to the AI model and then click the improve button, this will trigger the test improvement.

Revision #18
Created 10 July 2023 06:01:54 by Divyesh
Updated 8 September 2025 16:35:16 by Harish