

Automated Dart Test Generator

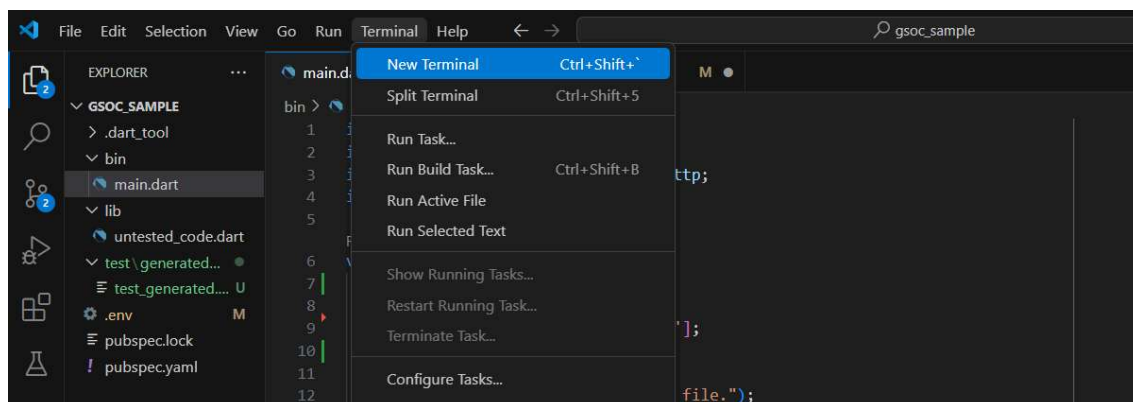
```
GSOC_...
├─ .dart_tool/           # Dart tool config (auto-generated)
├─ bin/
│   └─ main.dart         # Main entry point, sends code to Gemini API
├─ lib/
│   └─ untested_code.dart # Source code assumed to be untested
├─ test\
│   └─ generated_tests\
│       └─ test_generated.txt # Auto-generated test cases from Gemini
├─ .env                  # Contains your GEMINI_API_KEY
├─ pubspec.lock          # Dart dependency lock file
└─ pubspec.yaml          # Project metadata and dependencies
```

The `bin/main.dart` file works like a controller. It reads your Dart source code from `lib/untested_code.dart`, sends it to Google's Gemini AI using an API key stored securely in a `.env` file, and saves the generated test files in the `test/generated_tests/` folder.

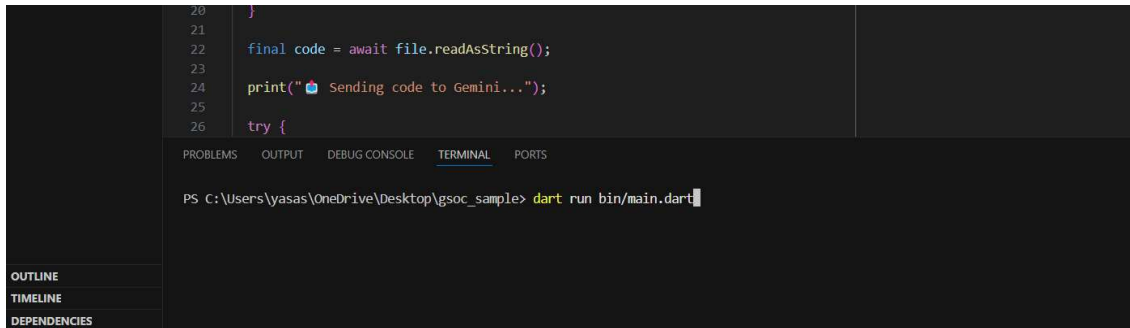
The `pubspec.yaml` file handles all the dependencies, like the `http` package for API requests and `dotenv` for managing environment variables. The `.dart_tool` folder is automatically created by Dart and is used internally — you can ignore it.

The assumed untested Dart code is first **read and converted into a string**, which is then sent to the **Gemini API** for analysis. Gemini processes the code and, if valid, returns a set of **auto-generated test cases** formatted using the `test` package. These tests are saved into a **text file** for review and execution.

STEP 1: Open New terminal on VsCode



STEP 2: To execute the Dart project and generate test cases, run the following command in your terminal



The screenshot shows an IDE with a Dart file on the left and a terminal window on the right. The Dart code includes a `readAsString()` call and a `print` statement. The terminal shows the command `dart run bin/main.dart` being executed.

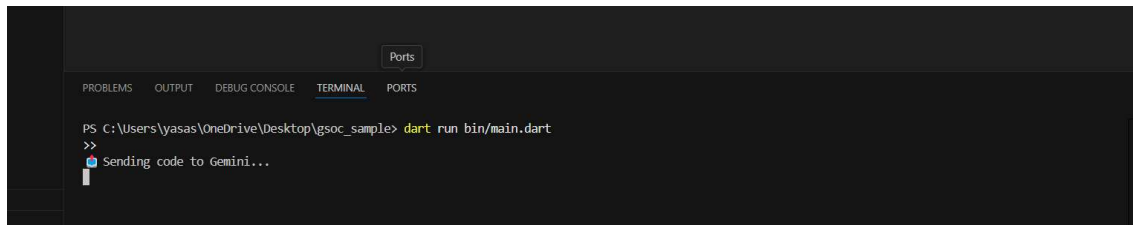
```
20 }
21
22 final code = await file.readAsString();
23
24 print("📄 Sending code to Gemini...");
25
26 try {
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

PS C:\Users\yajas\OneDrive\Desktop\gsoc_sample> dart run bin/main.dart

OUTLINE
TIMELINE
DEPENDENCIES

STEP 3: On execution, your Dart code is sent to Gemini AI, and in return, you get clean, testable code along with automatically generated test cases from Gemini.

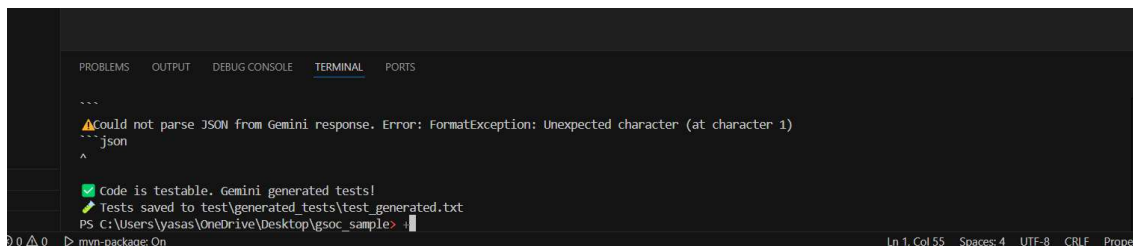


The screenshot shows the terminal window with the command `dart run bin/main.dart` and its output, which includes the message "Sending code to Gemini...".

Ports

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

PS C:\Users\yajas\OneDrive\Desktop\gsoc_sample> dart run bin/main.dart
>>
📄 Sending code to Gemini...



The screenshot shows the terminal window with an error message about parsing JSON from the Gemini response, followed by a success message indicating that the code is testable and tests have been saved to a file.

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

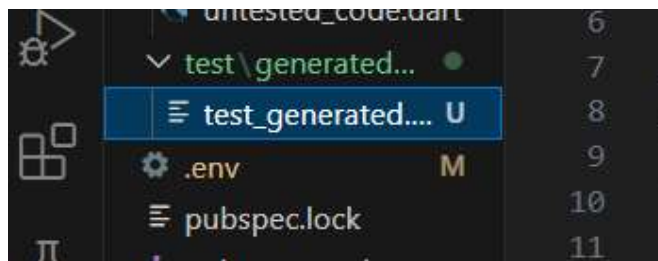
...
⚠️ Could not parse JSON from Gemini response. Error: FormatException: Unexpected character (at character 1)
...json
^

✅ Code is testable. Gemini generated tests!
📄 Tests saved to test\generated_tests\test_generated.txt

PS C:\Users\yajas\OneDrive\Desktop\gsoc_sample>

0 0 0 mvn-package: On Ln 1, Col 55 Spaces: 4 UTF-8 CRLF Propert

STEP 4: Generated Test cases are stored in `test_generated.txt` file



STEP 5: The generated file contains all the relevant test cases created by Gemini

```
test > generated_tests > test_generated.txt
1  | ``json
2  | {
3  |   "status": "success",
4  |   "generated_tests": ""
5  | import 'package:test/test.dart';
6  |
7  | // A simple sample Dart function to test
8  | int add(int a, int b) {
9  |   return a + b;
10 | }
11 |
12 | void main() {
13 |   group('add', () {
14 |     test('adds two positive integers', () {
15 |       expect(add(2, 3), equals(5));
16 |     });
17 |
18 |     test('adds two negative integers', () {
19 |       expect(add(-2, -3), equals(-5));
20 |     });
21 |
22 |     test('adds a positive and a negative integer', () {
23 |       expect(add(2, -3), equals(-1));
24 |     });
25 |
26 |     test('adds zero to an integer', () {
27 |       expect(add(0, 5), equals(5));
28 |       expect(add(5, 0), equals(5));
29 |     });
30 |
31 |
32 |   });
33 | }
34 | ""
35 | }
36 | ``
```