Google Summer of Code 2024

API Dash

Code Generators

Project Proposal Report

Apoorv Dwivedi | apoorvdwi

Contents:

1. About
2. University Info
3. Motivation and Past Experience
4. Project Proposal Information

1. About

Full Name - Apoorv Dwivedi

Contact info -

Email - apoorvd14@gmail.com

Mobile - +917678449601

Discord handle - apoorvdwi

Home page - https://apoorvdwivedi.in

GitHub profile link - apoorvdwi (Apoorv Dwivedi) · GitHub

Socials -

LinkedIn - Apoorv Dwivedi - Skyflow | LinkedIn

Twitter - Apoorv Dwivedi (He/Him) (@Apoorvdwi) / X

Time zone - IST

Link to a resume - ApoorvResume.pdf

2. University Info

University Name - Guru Gobind Singh Indraprastha University

Institute Name - Maharaja Agrasen Institute of Technology

Degree - Bachelor of Technology in Computer Science

Batch - 2019-2023

Graduation Date - July 2023

CGPA - 9.201

3. Motivation and Past Experience

Have you worked on or contributed to a FOSS project before? Can you attach repo links or relevant PRs?

API Dash is the first FOSS project that I've contributed to. I discovered API Dash while browsing the projects for GSOC 2024 and though I've never worked with Flutter and Dart before I was pretty intrigued by the project and listed problem statements for GSOC. I installed Flutter and set it up locally and made my first contribution as a troubleshooting section in the CONTRIBUTING.md. After that, I started adding code generators and it was pretty fun since I got to work with languages and libraries I did not previously work with and it was super fun building those. I will continue to add more code generators before the program officially starts and after it ends as well.

List of PRs I've raised till now at API Dash - https://github.com/foss42/apidash/pulls?q=is%3Apr+author%3Aapoorvdwi

What is the one project/achievement that you are most proud of? Why?

After I and my team won the Smart India Hackathon, we were given a chance to appear for a shortlisting test for the UNESCO India Africa International Hackathon. I was able to clear that and then new teams were formed which included 3 African participants and 3 Indian participants. I and my team were assigned a problem statement related to education and career counselling. I was the only person in my team who had experience in development and building full-stack projects. We were assigned mentors and I took most of the project development and the rest of the team members contributed alongside. We were able to deliver an impressive solution to judges and outperformed all other teams. Eventually, we won and it was such an incredible experience leading the team, building most of the product, adding and refining it alongside mentorship and judging rounds and then finally winning an international hackathon and receiving the prize from the Vice President of India on an

international stage with more than 30+ ambassadors and ministers present there.
Here are some glimpses of it
https://www.linkedin.com/feed/update/urn:li:activity:7002248258341212160/

What kind of problems or challenges motivate you the most to solve them?
I'm most motivated to tackle problems or challenges that involve creating innovative solutions or using already existing technology in a creative way to solve things. I like challenges that require me to think creatively, problem-solve, and utilize my technical skills to develop a sustainable and efficient solution. For example, at work, I often do tasks that involve optimizing performance and it gets me super intrigued as I find code pieces and insights on how adding 1 statement or removing 1 statement can really make the difference. Additionally, I'm passionate about solving real-world problems that have a positive impact on people's lives, whether it's developing software to streamline processes, enhance user experiences, or address societal issues. These challenges drive me to continuously learn, grow, and make a difference through my work as a software engineer.

Will you be working on GSoC full-time? In case not, what will you be studying or working on while working on the project?
I work as a remote software engineer at Skyflow. I will be working on GSoC part-time on weekdays and work on weekends if required. During the initial contribution that I made, I had a fairly good idea of the codebase and a clear idea of how to implement the things that are needed. I can assure you that my work and GSoC won't hamper or collide with each other. I've worked on multiple internships alongside my college and they have never impacted each other.

Do you mind regularly syncing up with the project mentors?
I don't mind syncing up with the project mentors regularly. I believe that would help me to understand certain things fast or if I need to make some changes in my approach. I am okay with any sync-up structure throughout the program.

What interests you the most about API Dash?

I have used Postman a lot and it is a really interesting tool. When I saw API Dash I saw that this is a really good project on similar lines and there is a lot of scope for adding new features. Though I've never worked or learned Flutter and Dart before this I found the project interesting to contribute to, and to top that the mentors and maintainers are supportive and helpful which convinced me to contribute to this. After discovering API Dash I didn't look at any other orgs and projects and solely contributed to this. Even if I'm not selected for GSoC in API Dash I will continue to contribute and if I'm selected then it is like a cherry on the cake.

Can you mention some areas where the project can be improved?

While the project has a lot of features that can be added and I'm sure they are already on the roadmap, I will focus on areas that can be improved in the product to have better monitoring, productivity, and analytics.

- Currently, I don't think we collect metrics or errors that may have been experienced by the users but go unreported. I believe to build good software we need to work on that.
- Another thing is, setting up CI pipelines or workflows that can automate certain tasks like running test cases when PR is created, having a code coverage analysis, optimizing images and assets if there are any, automated generic code reviews to validate coding style, best practices, commit messages, etc.
- While adding code generators, setting up a test file is a bit repetitive and exhausting at least for the template part. I think that can be automated and I've developed a solution and guidelines for my use which we can use in general after some refinement.

I believe addressing the above things alongside feature development will make sure that API Dash becomes a mature and contribution-friendly open-source software.

3. Project Proposal Information

Proposal Title - Code Generators

Project Length - 12 weeks

Abstract - I will focus on adding extensive support for code generators for popular languages and their corresponding libraries. This will include adding appropriate test cases and automating some parts of the process wherever necessary to reduce manual dependence.

Description -

I have added 2 code generators and validated one which also included working from scratch almost till now in API Dash where 2 have already merged and 1 is up for review while writing this proposal. My workflow for building new code generators involves the following steps -

1. I start by checking the language and library documentation for which code generator needs to be added.
2. I use various LLMs like GPT, Claude, etc to generate code snippets where I can see how necessary features are implemented in the library and what the syntax is like. Necessary features involve setting the URL, the HTTP method, setting query params, and headers, adding different types of body content like raw, JSON, form data, adding files to form data, etc.
3. Once I'm clear with the template code, I run the code with all necessary features on the code sandbox or somewhere else to check if there are any wrong syntaxes, warnings, etc.
4. Once the code part is ready, I start creating the corresponding templates and write the code for the code generator.
5. After the code generator is done, I've written a custom script that generates the test file for the code generator automatically. Below is the code of that script

```dart
import 'package:apidash/codegen/codegen.dart';
import 'package:apidash/consts.dart';
import '../request_models.dart';
import 'dart:io';

void main() {
  String testFileContent = generateTestFileContent(allRequestModels);
File('test/codegen/curl_rust_codegen_test.dart').writeAsStringSync(testFileCo
ntent);
}

String generateTestFileContent(List<Map<String, dynamic>> requestModels) {
  StringBuffer testFileBuffer = StringBuffer();
  final codegen = Codegen();

  // Write the imports and main function start
  testFileBuffer.writeln("import 'package:apidash/codegen/codegen.dart';");
  testFileBuffer.writeln("import 'package:apidash/consts.dart';");
  testFileBuffer.writeln("import 'package:test/test.dart';");
```

```
  testFileBuffer.writeln("import '../request_models.dart';\n");
  testFileBuffer.writeln("void main() {");
  testFileBuffer.writeln("  final codeGen = Codegen();\n");

  // Group by request method
  var groupedByMethod = groupByRequestMethod(requestModels);

  // Write the test groups
  groupedByMethod.forEach((method, models) {
    testFileBuffer.writeln("  group('${method.toUpperCase()} Request', ()
{");

    models.forEach((model) {
      var expectedCode = codegen.getCode(
          CodegenLanguage.rustCurl, model['requestModel'], "https");
      testFileBuffer.writeln("    test('${model['id']!.toUpperCase()}', ()
{");
      testFileBuffer.writeln("      const expectedCode = r\"\"\"");
      testFileBuffer.writeln("$expectedCode\"\"\";");
      testFileBuffer.writeln("      expect(");
      testFileBuffer.writeln("          codeGen.getCode(");
      testFileBuffer.writeln(
          "              CodegenLanguage.rustCurl, ${model['name']},
\"https\"),");
      testFileBuffer.writeln("          expectedCode);");
      testFileBuffer.writeln("    });");
    });

    testFileBuffer.writeln("  });\n");
  });

  // Close the main function
  testFileBuffer.writeln("}");

  return testFileBuffer.toString();
}

Map<String, List<Map<String, dynamic>>> groupByRequestMethod(
    List<Map<String, dynamic>> requestModels) {
  Map<String, List<Map<String, dynamic>>> grouped = {};

  for (var model in requestModels) {
    var method = (model['requestModel']).method.toString().split('.').last;
    grouped.putIfAbsent(method, () => []).add(model);
  }
  return grouped;
}
```

6. Once the test file is generated, I start to run the codes for all 27 test cases in the code sandbox or any other environment for final testing.
7. Once all of the above is done and sorted, I run the dev build for API Dash and check the formatting and indentation in the generated code. If there are any changes required, I fix them and re-generate the test file for updated code.
8. Finally, I run the flutter test for complete testing and once it is passed I raise the PR.

This workflow helps me add code generators in the most efficient way possible while maintaining the required code quality. We can even use the script above for everyone who wants to contribute to add code generators and it will make the process super efficient.

Below is the list of languages and corresponding libraries I am planning to add

1. JavaScript
   - SuperAgent - https://www.npmjs.com/package/superagent
   - Needle - https://www.npmjs.com/package/needle
2. Python
   - Urllib3 - https://pypi.org/project/urllib3/
3. Swift
   - SwiftHTTP - https://github.com/daltoniam/SwiftHTTP
   - Moya - https://github.com/Moya/Moya
4. Golang
   - Resty - https://github.com/go-resty/resty
   - Req - https://github.com/imroc/req
5. PHP
   - HTTPlug - https://github.com/php-http/httplug
6. C#
   - HttpClient - https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient
   - Restsharp - https://restsharp.dev/
7. Ruby
   - Faraday - https://github.com/lostisland/faraday
   - Httparty - https://github.com/jnunemaker/httparty

8. Rust
    - Hyper - https://hyper.rs/
9. Kotlin
    - Fuel - https://github.com/kittinunf/fuel
10. OCaml
    - Ocaml-http - https://github.com/mirage/ocaml-cohttp

Weekly Timeline -

Note: I have kept the project length to 12 weeks with the below-selected languages and corresponding libraries. I believe these cover all the popular languages and libraries that are not supported yet. We can always swap some of these below with others if mentors feel like some important ones are not included and they need to be implemented.

Week 1 (May 27 - June 2):

- Start working on the code generator for superagent (JS) and urllib3 (Python)
- Get the PR raised for both of them and add relevant tests.
- Discuss with mentors alongside these code generators on “automated test file generation workflow”

Week 2 (June 3 - June 9):

- Start working on the code generator for needle (JS) and SwiftHTTP (Swift)
- Get the PR raised for both of them and add relevant tests.

Week 3 (June 10 - June 16):

- Start working on the code generator for resty (Go)
- Get the PR raised and add relevant tests.

Week 4 (June 17 - June 23):

- Start working on the code generator for req (Go)
- Get the PR raised and add relevant tests.

Week 5 (June 24 - June 30):

- Start working on the code generator for HTTPlug (PHP)
- Get the PR raised and add relevant tests.

Week 6 (July 1 - July 7):

- Start working on the code generator for Moya (Swift)
- Get the PR raised and add relevant tests.

—| MID EVALUATION BREAK |—

Week 7 (July 12 - July 19):

- Start working on the code generator for C# (HttpClient) (C#) and restsharp (C#)
- Get the PR raised for both of them and add relevant tests.

Week 8 (July 19 - July 26):

- Start working on the code generator for faraday (Ruby) and httparty (Ruby)
- Get the PR raised for both of them and add relevant tests.

Week 9 (July 26 - August 2):

- Validate Java async-http-client code generator and add relevant tests
- Get the PR raised and add relevant tests.

Week 10 (August 2 - August 9):

- Start working on the code generator for hyper (Rust)
- Get the PR raised and add relevant tests.

Week 11 (August 9 - August 16):

- Start working on the code generator for Fuel (Kotlin) and ocaml-cohttp (OCaml)
- Get the PR raised for both of them and add relevant tests.

Week 12 (August 16 - August 23):

- Final inspection of all the code generators I added and fix any issues in any of them
- Finalise the “automated test file generation workflow” (subject to approval)