Google Summer of Code

About

Full Name	Tanish Khare
Email	Tanish Khare
Discord Handle	@bakaotaku
Homepage/Website	https://bakaotaku.dev/
Blog	https://blog.bakaotaku.dev/
Github	Tanish2002
Twitter	https://twitter.com/baka_otaku2002
Time Zone	Indian Standard Time (IST)
Resume	Tanish_Khare_RESUME.pdf

University Info

University	SRM University
Program	B.Tech - CSE
Year	4
Expected Graduation	2024 (July)

Motivation & Past Experience

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

Indeed, I have made contributions to several Free and Open Source Software (FOSS) projects. My contributions have mainly been on projects that I use regularly and wanted particular features or where I ran into technical problems. I have contributed in the following ways (aside from contributions to API Dash):

- Nixvim: A nix framework to configure neovim using the nix programming language.
    - plugins/catppuccin: add colorscheme + test
    - add: added statix, deadnix, nixpkgs_fmt to null-ls sources
    - feat: register mapping with no actions in which-key when enabled
    - perform some statix linting and fixes
    - plugins/toggleterm: init + test
    - change type of pylsp_mypy.overrides
- Hyprland: A dynamic tiling Wayland compositor based on wlroots
    - Add files for nix build
- Nixpkgs: Nix Packages collection & NixOS
    - Imagemagick: add curl dependency
- Robbb: The main bot for the r/unixporn discord server!
    - Add reason to ban reply message
    - Modify nix Files
    - send a different DM when ban reason contains "ice"
    - reply when mentioned

2. What is your project/achievement you are most proud of? Why?

In my career, I've had a blast working on all kinds of projects. One of my favorites was working on an app called SafeHer during the SMART India Hackathon with my five awesome teammates. My responsibilities included working on the backend development, in addition to contributing to frontend development by assisting my teammates. We won the institutional level, but we didn't quite make it to the regional level. Still, SafeHer is something I'm super proud of.

SafeHer is a mobile application focussed on the security and wellness of female students on college campuses. It's got all sorts of features to keep them safe and connected.
Here's the lowdown on the tech we used:

- React Native (Expo): The app's foundation
- Zustand: The state management library
- Python (FastAPI): The backend server
- Websockets: Real-time communication for chatrooms
- PostgreSQL: The database
- PineCone: For the chatbot

Features:

- **Incident Reporting:** Female students can report safety incidents, such as harassment or emergencies, using the SOS button on the app. When an incident is reported, the app collects incident details and the user's real-time location.
- **Emergency Contacts:** Users can add up to five trusted friends or teachers as emergency contacts within the app. When an incident is reported, selected emergency contacts receive instant notifications.
- **Real-Time Location Sharing:** The app shares the user's real-time location with emergency contacts, ensuring a swift response in case of an emergency.
- **Community Chat:** For each reported incident, the app creates an incident-specific chat room. Students can join the chat to provide support, share information, and offer assistance to those affected.
- **High-Risk Area Awareness:** The app utilizes incident data to identify and highlight campus areas with security concerns. A map feature shows high-risk areas, raising awareness among users.
- **FAQ Chatbot:** The app includes a chatbot that answers frequently asked questions related to safety. Users can quickly access information and guidance on various safety topics.

See More: SafeHer

3. What problems or challenges motivate you the most to solve them?

Solving problems that involve automating repetitive or time-consuming tasks gets my blood pumping. As a "lazy developer" (in the best way possible), I'm always on the lookout for ways to streamline my workflow and make my life easier. I believe that automation is a game-changer when it comes to efficiency

and productivity, and I'm always excited to explore new ways to put it into practice.

One of the automation projects I've successfully implemented is managing my home multimedia setup. I've set up a home server that hosts a Jellyfin media library. The system is fully automated to gather releases from preconfigured sources, ensuring that whenever I search for a movie or series on my mobile phone, it gets downloaded and made ready for viewing. The interface is user-friendly, making it accessible even for my parents.

I carry the same ambitions for API Dash and believe some areas can be automated which I will discuss further in the proposals section.

4. Will you be working on GSoC full-time? If not, what will you be studying or working on while working on the project?

Sure, I am committed to working on API Dash until the end of the GSOC timeline and beyond! But I wanted to let you know that I'm currently looking into internships/jobs since I'm in my final year. If I find one, I'll be able to work on the projects on the weekends otherwise I'm available all week.

Overall I plan to become a regular contributor who isn't here just for GSoC.

5. Do you mind regularly syncing up with the project mentors?

Certainly not! I strongly believe regular sync-ups are essential to ensure that I am working in the correct direction.

6. What interests you the most about API Dash?

I got to know about API Dash through Google Summer of Code (GSOC), and I've grown to like it after using it for a while. Coming from a backend-focused background, an API testing tool is essential for me.

I've tried various tools like Postman, Insomnia, and Hoppscotch, each with its pros and cons. What I appreciate about API Dash is its simple and intuitive UI.

Other options like Postman have a cluttered UI that's hard to look at, even with the dark theme that's still grayish and just not great.

Another thing I love about API Dash is its built-in support for rendering different types of data. It recognizes various MIME types like PDF and audio, allowing users to view them directly within the client. Meanwhile, other clients typically just spew out confusing binary gibberish.

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

End-User Improvements

- Missing technologies: Websockets, Socket.IO, RPC
- Keyboard accessibility for opening panes, running requests, etc.
- Variables for setting up different environments

Developer/Maintainer Improvements

- CI Pipelines for testing.
- Pre-commit hooks.
- Improved documentation on a dedicated website (e.g. docs.apidash.dev)

Project Proposal #1

Title: Realtime API Testing Support - WebSocket, SSE, MQTT

Abstract

Testing protocols such as WebSocket, MQTT (Message Queuing Telemetry Transport), and SSE (Server-Sent Events) are essential for ensuring reliability, scalability, and security in real-time communication systems. These protocols are used in various domains for multiple purposes, including web applications and IoT (Internet of Things) devices. Various API clients, such as Postman and Insomnia, provide usable support for Real-time API Testing. The goal of our project is to design the architecture of the core library and understand the specifications necessary to implement support for testing, visualization, and possible integration code generation of Real-time APIs in API Dash.

Right now, we can only support REST-style requests, like GET, POST, DELETE, etc. We're using the "http" package to make these requests. For managing state in memory, we're using Riverpod, and for managing state on the local disk, we're using Hive. We're also planning to set up a similar configuration for WebSockets, Server-Sent Events (SSE), and Message Queuing Telemetry Transport (MQTT) using libraries like "**dart:io**," "**mqtt_client**," and "**sse**." The primary goal of this project is to make sure that the request models for these protocols are compatible with Riverpod and Hive.

Detailed Description

Currently, we're utilizing the **http** package for making REST-type requests. We've also created a `RequestModel` class which represents a REST request.

The main parts of the REST Request Model are these:

Variable	Description
HTTPVerb method	• HTTP method of the request • Can be GET, POST, DELETE, etc
url	URL of the API endpoint
name	Name of the request shown in side pannel

description	Description of the request.
requestHeaders	List of Headers with name-value pairs
requestParams	List of Query Parameters with name-value pairs
isHeaderEnabledList	List of booleans representing enabled Headers
isParamEnabledList	List of booleans representing enabled Params
requestBodyContentType	• Type of request • Can be plain-text • Can be json
requestBody	Actual request body contents
requestFormDataList	List of Form Items with name, value and type.
responseStatus	• Response of the sent request shown in UI. • Can be 200, 400, 500, etc

I plan to create a similar request model separate from the original one in a new file `websocket_request_model.dart`
An example request model for a WebSocket type request would look like this:

Variable	**Description**
url	URL of the API endpoint
name	Name of the request shown in side pannel
description	Description of the request.
requestParams	List of Query Parameters with name-value pairs
isParamEnabledList	List of booleans representing

	enabled Parameters
messages	• List of messages that were sent • Will be a custom model • This model will have 2 properties. • String **message_content** representing the actual message contents • Enum sender for determining who sent the message, server or user.
responseStatus	• This will be simple enum representing the state of request. It can be Connected, Disconnected or Failed. • By default this will be disconnected.

Headers are absent here since they aren't possible in Websockets.

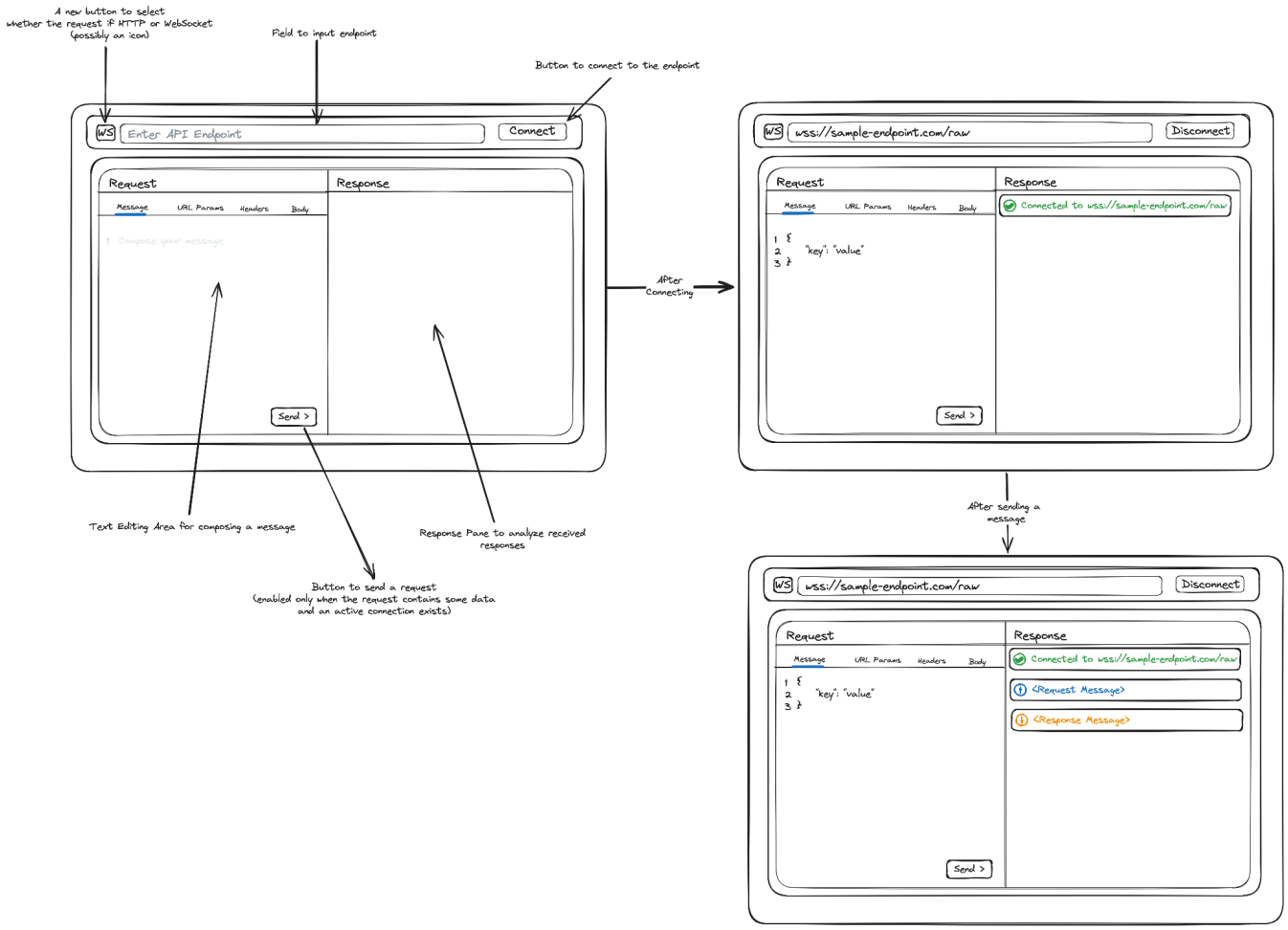
Similar request models need to be created for SSE and MQTT.

I will then also create the necessary services. Like the current **http_service** using packages like "**dart:io**," "**mqtt_client**," and "**sse**."

Finally, I want to document everything from time to time so it's easier for new developers to learn how to implement a new protocol for new API types in the future such as GRPC (which isn't present at this time).

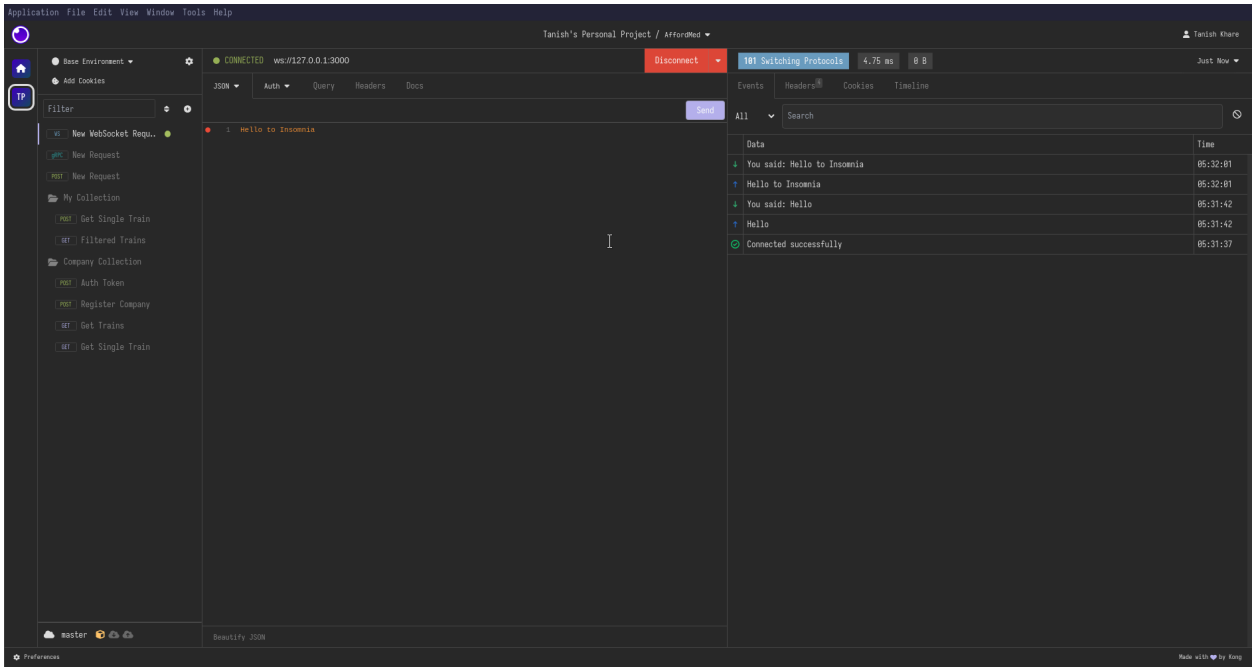
I have a clear idea for backend implementation with all the working bits.

For the Frontend I'll be using Material UI with our current theme and following the diagram shared by Ankit.



The actual implementation for WebSocket wouldn't have Headers when connected since that [isn't possible](#) in Websockets.

In Contrast, this is what Insomnia looks like:



For the request Body, it supports JSON and plain text which should be easy to implement due to already present editor widgets.

It also has a Search Box in the responses panel, This can be handy for users trying to filter out a large amount of messages. This can be implemented if needed.

MQTT and SSE have similar external workings as they share an endpoint and a request body. The responses are streamed from the server. However, they have different internal workings. Therefore, the UI implementation for each protocol would not differ much.

I would also look into the possibility of implementing support for Socket.io which is built on top of WebSockets and has neat features like auto reconnection, and polling when upgrading to WebSocket fails.

In conclusion, this project will make a significant contribution to the comprehensive API ecosystem by delivering novel protocols to end users of API Dash. Additionally, it will serve as an invaluable resource for developers seeking to incorporate additional protocols into API Dash or simply gain an understanding of the underlying mechanisms of these protocols.

Timeline

Weeks	Dates	Description
Week 1	May 27 - June 2	• Planning the project with the mentors • Discussing project deliverables with mentors. • Discuss the request models for each of the protocols. • Discuss the new constants that need to be created in the `consts.dart` file
Week 2	June 3 - June 9	• Start working on the WebSocket implementation • Create a WebSocket Manager class to handle the connections, messages, and failures. • Write tests for this service.
Week 3	June 10 - June 16	• Create a Request Model for WebSocket Request. • Integrate it with Hive and Riverpod • Write tests for the Request Model
Week 4	June 17 - June 23	• Start working on SSE Implementation • Create an SSE Manager class that will handle connections, messages, and failures • Write tests for this service
Week 5	June 24 - June 30	• Create a Request Model for WebSocket Request. • Integrate it with Hive and Riverpod • Write tests for the Request Model
Week 6	July 1 - July 7	• Start working on MQTT • Create a MQTT Manager Class to handle connections, messages, and failures. • Write tests for this service
MID	**TERM**	**EVALUATION**
Week 7	July 12 - July 19	• Start working on the front end. • Create a Protocol Dropdown Button. Issue #307

		• Test the button for all different protocols
Week 8	July 19 - July 26	• Start creating common widgets. • Write Widgets that can be reused. • Since all 3 protocols would have a similar request body we can create a common widget.
Week 9	July 26 - Aug 2	• Keep creating common widgets • A response widget would be created to show the list of messages with an indicator for the sender.
Week 10	Aug 3 - Aug 9	• Modify the Side Panel to show the type of request using Icons. • Write tests for all created widgets.
Week 11	Aug 10 - Aug 16	• Integrate all widgets for all different protocols in screens. • Test out the integrated final version • Fix any bugs UI or backend-related.
Week 12	Aug 17 - Aug 23	• Mark the PR ready for review. • Apply any changes as required/asked. • Possibly write some foundation for CodeGen at least for websockets. • Maintenance and Plan for future developments.