"Open-source Gemini Example Apps" proposal for Google Summer of Code 2025

| Categories | Details |
|---|---|
| Difficulty | Medium |
| Size | 350hrs |
| Skills | Python, JavaScript/TypeScript, Gemini SDKs, technical writing |



# 1. Abstract

# 1.1 Overview

The Gemini Cookbook is a set of sample applications and tutorials illustrating different functionalities of the Gemini APIs. The intent of this proposal is to modernize current tutorials and documentation to assist with the new unified Gemini SDKs for JavaScript/TypeScript and Python. This involves bringing examples from Python into JS/TS, creating new end-to-end tutorials, and modernizing examples for other open source libraries utilizing outdated versions of the Gemini APIs. By giving users new, current learning tools, this project hopes to facilitate adoption of the Gemini APIs.

Here are the areas where `Gemini Cookbook` can be improved:

- Currently the Cookbook is only available in Python consisting of examples and tutorials that demonstrate the capabilities of the Gemini APIs.
- The new unified Gemini SDKs have additional features and improvements that are not reflected in the current examples, which can be expanded to showcase a wider range of Gemini capabilities.

- With the new unified SDKs, the Python SDK being released just a couple of months ago, and the JS/TS SDK being in preview, there are many open-source libraries that use outdated versions of the Gemini APIs. These libraries need to be updated to use the latest SDKs and examples.

## 1.2 Goals

This project's main goal is to improve Gemini SDKs' usability and accessibility by giving developers access to current, thorough, and useful resources. The following crucial areas will be the project's main focus:

1. **Migrate Existing Python Examples to TypeScript**
   Convert the current Python-based examples in the Gemini Cookbook to idiomatic TypeScript using the latest Gemini SDK. This will ensure cross-language support and make sure the resources are accessible to a wider range of developers.

2. **Create End-to-End Tutorials for Real-World Use Cases**
   Design and implement comprehensive, practical tutorials that showcase the capabilities of the Gemini APIs. These tutorials will include projects such as intelligent chatbots, document summarization tools, and image generation from prompts, covering both basic and advanced use cases. New ideas for tutorials can be submitted by the community, and can be sourced from the existing cookbook issues.

3. **Modernize Open-Source Libraries Using Gemini APIs**
   Identify and contribute to popular open-source projects that use outdated versions of the Gemini APIs. Update the examples and documentation to leverage the features and improvements introduced in the latest unified Gemini SDKs.

4. **Improve Developer Guidance & Documentation**
   Provide clear, concise, and well-structured documentation for all migrated and newly developed examples. Emphasize both code implementation and conceptual understanding of Gemini functionalities and best practices.

## 1.3 Benefits

The Gemini community and the larger developer ecosystem will gain a number of advantages from the project's successful completion:

- **Increased Adoption**: With the latest and varied learning materials, developers are more likely to use the Gemini SDKs within their own projects, leading to increased usage and community involvement.
- **Improved Learning Resources**: The new and updated tutorials will function as essential references for developers of all skill levels, helping them to quickly understand and make the most of the Gemini APIs.
- **Community Contributions**: By modernizing open-source libraries and encouraging community involvement, this project will foster a cooperative setting where developers can share their experiences and improvements.
- **Showing Versatility**: A wide range of sample use cases will highlight the flexibility of the Gemini SDKs, attracting more users and encouraging experimentation across various use cases.

# 2. Approach

## 2.1 Environment & SDK Setup

Setup a development environment for JS/TS for the JS SDK. This includes installing required dependencies, configuring tools such as `eslint`, `prettier`, and `tsc` for linting, formatting, and type validation, among others, as well as establishing workflows for automated testing and deployment.

Some tools we can establish are:

- **ESLint** + **Prettier**: To check and format code. This ensures a uniform coding style throughout the project. Another option for simplification might be utilizing biome.js, a comprehensive tool for linting, formatting, and type checking.
- **Pre-commit**: To execute linters and tests prior to code commits and on certain git hooks. This guarantees that the code adheres to the established coding standards and passes all tests before being committed to the repository.
- **Jest**: Used for executing unit tests. Makes sure that the code is functioning as intended and that any modifications or additions do not introduce new bugs. This is particularly important for the JS/TS SDK, as it is still in preview and may have bugs or issues that need to be addressed.
- **GitHub Actions**: To automate the testing and deployment workflow. This will assist in making certain that the code remains in a deployable condition and identifies problems promptly.

## 2.2 Cookbook Migration

The cookbook is majorly divided into 3 parts:

- **Examples**: These are illustrations that showcase how to use the Gemini APIs. Typically, they are compact and centered around a particular feature or function, which can be either `pure` (i.e. directly utilizing the SDK to implement a paper) or `impure` (i.e. employing the SDK alongside additional libraries or frameworks).
- **Quickstart**: This section provides a brief introduction to getting started with the Gemini SDKs, including installation instructions and basic usage examples and capabilities of Gemini models and APIs. Examples: `Grounding`, `Spatial Awareness` etc. Quite a few of them still use the legacy SDKs and need to be updated to use the new unified SDKs. Examples: `Asynchronous Requests`, `Audio`, `Video` etc.
- **gemini-2**: This section contains quickstart guides and examples for the Gemini-2.0 Flash models and unified SDKs. The python edition of the cookbook requires additional examples to be included in the `gemini-2` section. The JS/TS edition of the cookbook must be developed from the ground up.

For instance, here is an example migration of a section in the `Get_started.ipynb` notebook to the JS/TS version of the cookbook.

- Setup the environment and install the required dependencies:
  - Install the `@google/genai` package using npm:

    ```
    $ npm install @google/genai
    ```

  - Add your environment variables to a `.env` file:

    ```
    GEMINI_API_KEY=your_api_key_here
    ```

Alternatively, you can set the environment variables in your notebook or script:

```
%env GEMINI_API_KEY=your_api_key_here
```

- Create a new instance of the `GoogleGenAI` class, passing in your API key as an environment variable:

```
import {GoogleGenAI} from '@google/genai';
const GEMINI_API_KEY = process.env.GEMINI_API_KEY;

const ai = new GoogleGenAI({apiKey: GEMINI_API_KEY});
```

- Generate content using the `generateContent` method of the `GoogleGenAI` instance:

```
const response = await ai.models.generateContent({
  model: 'gemini-2.0-flash-001',  // Choose the model you want to use
  contents: 'What\'s the largest planet in our solar system?',
});
console.log(response.text);
```

- The complete code snippet would look like this:

```
import {GoogleGenAI} from '@google/genai';
const GEMINI_API_KEY = process.env.GEMINI_API_KEY;

const ai = new GoogleGenAI({apiKey: GEMINI_API_KEY});

async function main() {
  const response = await ai.models.generateContent({
    model: 'gemini-2.0-flash-001',
    contents: 'What\'s the largest planet in our solar system?',
  });
  console.log(response.text);
}

main();
```

### 2.2.1 Quickstart/Gemini-2 Migration

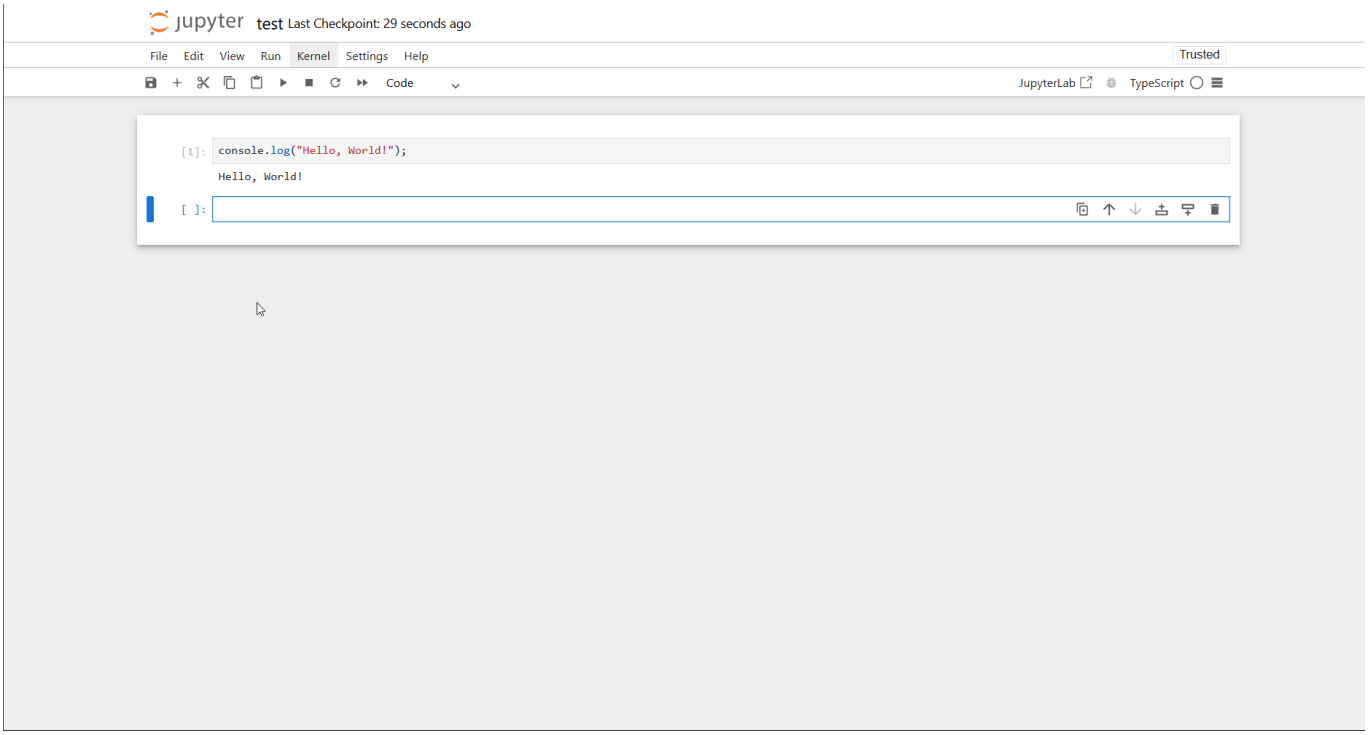
The python version of the cookbook needs a few more examples to be included in the `gemini-2` section. The examples that need to be ported over are listed here. The JS/TS edition of the cookbook must be developed from the ground up.

As many of these guides make use of notebooks, we can employ the Jupyter Book together with the `IJavaScript Kernel` to develop the JS/TS edition of the cookbook. This will enable us to maintain the same

content and structure as the Python version of the cookbook, while also permitting the use of the JS/TS SDK. A comparable `ITypeScript Kernel (tslab)` is also available that can be regarded for the same.

Ideally, typescript should be preferred over javascript for the cookbook as it provides better type safety and developer experience. However, if the need arises, we can also use javascript for the same.

```
$ pip install jupyter
$ npm install -g tslab
$ tslab install
$ jupyter kernelspec list
Available kernels:
  javascript    C:\Users\Triyan
Mukherjee\AppData\Roaming\jupyter\kernels\javascript
  jslab         C:\Users\Triyan Mukherjee\AppData\Roaming\jupyter\kernels\jslab
  tslab         C:\Users\Triyan Mukherjee\AppData\Roaming\jupyter\kernels\tslab
  python3       C:\Users\Triyan
Mukherjee\AppData\Local\Programs\Python\Python310\share\jupyter\kernels\python3
```




### 2.2.2 Examples Migration

Examples include specific application/usage based code samples that demonstrate the capabilities of the Gemini APIs, showcasing real-world use cases and best practices. The complete list of all available examples can be found here. Few of these examples in the python version of the cookbook need to be updated to use the latest SDKs, and additional examples should be created for the JS/TS version to ensure comprehensive coverage of the Gemini capabilities added since the latest release of the `2.0` flash models.

Quite a few examples are also (impure) utilizing various libraries, frameworks or third party integrations such as `langchain`, `chromadb`, `Arduino/ESP32` etc. These examples would include scouting for JS/TS equivalent libraries or ports to implement the same functionality.

For example, the `langchain` library is a popular library for building applications with LLMs and has a JS/TS equivalent. The `chromadb` library is a vector database that is used for storing and querying embeddings, and has a JS/TS equivalent as well. The `Arduino/ESP32` examples can be implemented using the Johnny-Five library which is a JavaScript Robotics and IoT platform.

## 2.3 New Tutorials

The latest unified SDKs introduce numerous new features and functionalities that can be highlighted through fresh tutorials. These tutorials may focus on practical applications and can feature projects like smart chatbots, document summarizers, and image generation from prompts. The tutorials can include a diverse array of Gemini features, such as text/image generation, code execution and multimodal interactions.

With several AI models now available, numerous developers and companies are seeking methods to incorporate these models into their applications and everyday operations. This is where the latest tutorials can assist. They may serve as independent, complete applications ready to be deployed, or developers can fork them as starting point for their own applications. The tutorials must be thoroughly documented and straightforward to follow, providing clear explanations of the code and the fundamental Gemini concepts.

A potential example of this tutorial could be a `Gemini Chatbot` that utilizes the Gemini APIs to create responses for user inquiries, preserving context and delivering pertinent information. This may be executed with the subsequent technology stack:

- **Next.js**: To create the chatbot's frontend. We can utilize well-known libraries like `shadcn/ui`/`assistant-ui` for creating the UI elements of the chatbot. Ensuring a consistent user interface and experience across the application.
- **FastAPI**: To create the backend API for the chatbot. This will handle requests from the frontend and communicate with the Gemini APIs to generate responses. FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
- **Redis + Celery**: To cache the responses obtained from the Gemini APIs. This will aid in minimizing the latency and enhancing the performance of the chatbot. Celery can manage background tasks and schedule requests to the Gemini APIs, meaning it prevents overwhelming the APIs with too many requests and ensures that requests are processed promptly.
- **PostgreSQL + pgvector**: Used for saving user queries and sessions, the `pgvector` extension allows for efficient storage and retrieval of vector embeddings, enhancing the chatbot's ability to maintain context and improve response accuracy.

## 2.4 Library Updates

Identify open-source libraries that use outdated versions or the legacy SDKs of the Gemini APIs and update their examples and or create an issue request to introduce the new unified SDKs. This will help in ensuring that the libraries are up-to-date and can be used with the latest features and improvements introduced in the Gemini SDKs.

A quick search from the deprecated legacy python sdk network dependents shows about `~750` packages that are using the legacy SDK. A few such popular packages are:

- Pathway
- OpenInterpreter
- GeeMap

- Podcastfy
- LLM Reasoners
- Devin Cursor Rules

> **Note**
> At the time of writing this proposal, the above packages are using the legacy SDK. Legacy SDK is one of the listed reqiured dependencies for these packages, i.e `requirements.txt, pyproject.toml` etc. However, this may change in the future as the packages are updated to use the new unified SDKs. The above list is not exhaustive and is only meant to give an idea of the packages that can be updated. The final list of packages will be decided in consultation with the mentors and the community.

# 3. Timeline

The project is expected to be completed in `350` hours over the course of `12` weeks. The timeline is divided into `5` phases, each with its own set of tasks and milestones. On average, I would be devoting 30 hours per week to this project, with roughly 2-3 hours on weekdays and 5-6 hours on weekends.

## 3.1 Community Bonding Period (May 8 - June 1)

- Familiarize myself with the Gemini SDKs, the existing examples and tutorials in the Cookbook, and the open-source libraries that use outdated versions of the Gemini APIs.
- Set up the development environment for both the Python and JavaScript/TypeScript SDKs.

## 3.2 Cookbook Migration (June 2 - July 14)

- Migrate or add equivalent tutorials for the guides still using the legacy SDKs to use the new unified SDKs. This includes updating the examples to use the latest features and improvements introduced in the Gemini SDKs.
- Create the JS/TS version of the cookbook from scratch using the Jupyter Book and IJavaScript/ITypeScript kernel. This includes creating the necessary examples and tutorials to cover a wide range of Gemini capabilities.
- Update the existing documentation to reflect the changes made during the migration process, ensuring clarity and accessibility for developers.
- Port example applications showcasing third-party integrations such as `langchain`, `chromadb`, `Arduino/ESP32` etc. to libraries natively available in JS/TS.

## 3.3 New Tutorials (July 15 - August 15)

- Design and implement new end-to-end tutorials demonstrating various Gemini use cases, such as building a chatbot, creating a text summarization tool, or generating images from text prompts.
- Scoping of ideas for new tutorials can be done in consultation with the community and mentors and can be based on the existing cookbook issues.
- Ensure that all new tutorials are well-documented, with clear explanations of the code and the underlying Gemini concepts, and can be bootstrapped easily by the community.

## 3.4 Library Updates (August 16 - August 25)

- Identify open-source libraries that use outdated versions of the Gemini APIs and update their examples to use the latest SDKs.
- Based on scale of the libraries, this may include creating issues for the libraries to update their examples to use the latest SDKs or creating PRs to update the examples and documentation.
- Consult with the mentors and the community to determine the best approach for updating the libraries and ensuring compatibility with the new SDKs.

### 3.5 Final Review and Submission (August 26 - September 1)

Conduct a final review of all the work done during the project, ensuring that all examples, tutorials, and documentation are up-to-date and well-structured. Prepare a final report summarizing the work done, the challenges faced, and the lessons learned during the project. Submit the final report and all the work done during the project to the mentors and the community.

## 4. About Me

I am `Triyan Mukherjee` and currently a third year undergraduate student at the `Manipal Institute of Technology` in India, pursuing a degree in Computer and Communication Engineering. I have a strong background in Python and have worked with multiple web frameworks. I have experience with documentation/technical writing and making guides, and have developed my own libraries and packages in the past and published them to PyPI. I have extensive experience working with both Python and Typescript, creating robust and scalable applications.

- discord.py Masterclass: A comprehensive tutorial for building Discord bots using the discord.py library. This project includes a series of lessons and examples that cover everything from basic bot creation to advanced features like slash commands, audio playbacks, pagination etc.
- Pokelance: PokeLance is a modern, async-ready PokéAPI wrapper built with Python, designed for speed, efficiency, and ease of use. With a pythonic API, automatic caching, and full mypy support.
- WhatsApp Web 2.0: A responsive and modern real-time WhatsApp clone made with Typescript, Next.js, Pusher, Prisma and integrating zegocloud for voice and video calls. This project implements all the features of WhatsApp Web, including sending and receiving messages, creating groups, and making voice and video calls. It also includes a responsive design that works on both desktop and mobile devices.

### 4.1 Contact Information


- Email: triyanmukherjee@gmail.com
- LinkedIn: triyanmukherjee
- GitHub: FallenDeity
- Timezone: GMT+5:30 (IST)