Issue Generator

The problem:

When dealing with open-source libraries, occasional encounters with bugs are inevitable. Yet, the process of reporting these bugs poses a significant challenge.

To effectively communicate the issue and facilitate a resolution, one must generate a minimum reproducible example. Unfortunately, this task often becomes burdensome, involving the creation of a new repository, integration of the library into the project, meticulous recreation of the specific problem, and the completion of the GitHub issue documentation.

As one can imagine, this intricate procedure often leads individuals to abandon the idea of opening issues altogether, discouraged by the considerable effort involved.

Fortunately, we present a REAL solution to address this very real challenge.

Our Solution:

Our innovative solution involves the development of a software application leveraging OpenAI to streamline our processes.

This user-friendly application requires just four inputs to generate a minimal reproducible example efficiently.

Users will provide the library name, library version (even supporting older versions), a concise description of the encountered bug, and the desired repository name for the minimum reproducible example. Subsequently, our server will initiate the creation of a new Github Repository within the user's account. This repository will have the specified library and a swift example showcasing the problematic component.

Furthermore, the system will generate a sandbox environment with the provided code, so you could easily share it with someone else.

To enhance collaboration and issue tracking, a README file will be automatically generated, containing a pre-defined template for the impending GitHub issue. This comprehensive solution ensures a seamless and visually appealing process for bug reporting and resolution.

Software Created:

Presented below is an in-depth elucidation of our software, commencing with a comprehensive software diagram illustrated in Figure 1.

The application is meticulously segmented into two components: the front-end (UI) and the back-end. Leveraging next.js, the front-end embodies a seamless and engaging user interface, while the back-end, powered by node.js, ensures robust functionality and efficient data processing.

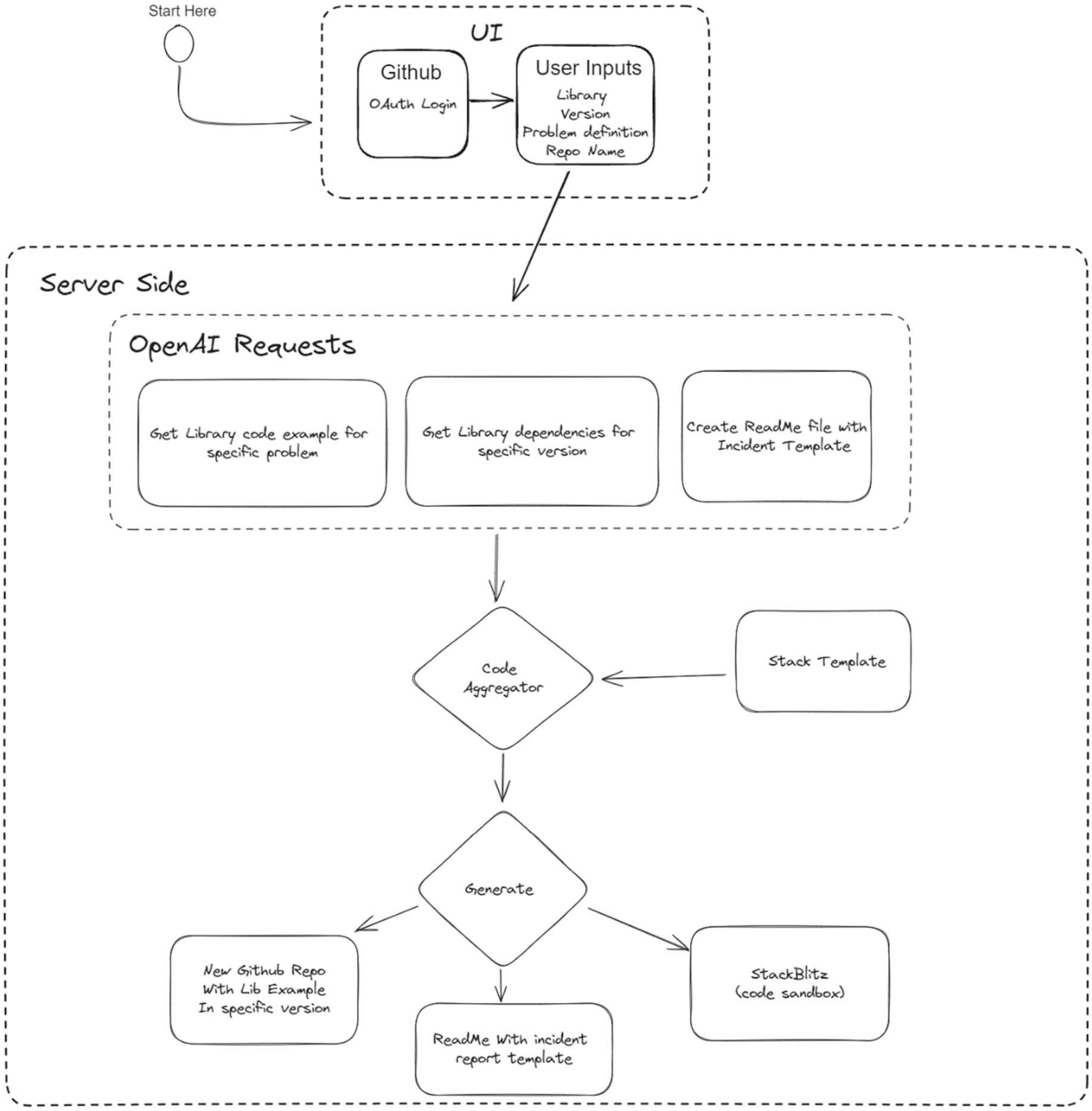
Start Here
UI
Github
OAuth Login
User Inputs
Library
Version
Problem definition
Repo Name
Server Side
OpenAI Requests
Get Library code example for specific problem
Get Library dependencies for specific version
Create ReadMe file with Incident Template
Code Aggregator
Stack Template
Generate
New Github Repo With Lib Example In specific version
ReadMe With incident report template
StackBlitz (code sandbox)

Figure 1 - Sofware Diagram

Client Side:

Login:

To access our application, we've streamlined the login process through OAuth via Github. This allows us to seamlessly connect and, in the future, create a dedicated repository for each user.

User Inputs:

- Library Name, enter the name of the library encountering the issue.
- Version Number, Specify the version causing the problem. If not provided, we'll default to the latest version.
- Problem Definition, provide a concise overview of the issue you're facing.
- Repo Name, Input the desired name for the new repository that will be generated.

Server Side:

On the server side, we employ a series of distinct prompts to interact with OpenAI.

Fine-tuning OpenAI:

To enhance the reliability of our model, we conducted fine-tuning using a variety of prompts and corresponding answers. An illustrative example of a fine-tuning prompt is as follows:

```
{"messages": [{"role": "user", "content": "Give me an example of the
library @mui/material@5.14.17 in tsx that exports a default component
with the name App, with a date-picker.       I don’t need instructions
on how to install @mui/material. Also give me a flat json just with all
the dependencies you’ve used on the example. The dependency name should
be a key and the version should be the value.       These peer
dependencies should respect the version of the library
@mui/material@5.14.17 that I specified. Do not include any text besides
the ```tsx``` and  ```json``` code blocks."},{"role": "assistant",
"content": "```tsx\nimport * as React from 'react';\nimport TextField
from '@mui/material/TextField';\nimport AdapterDateFns from
'@mui/lab/AdapterDateFns';\nimport LocalizationProvider from
'@mui/lab/LocalizationProvider';\nimport DatePicker from
'@mui/lab/DatePicker';\n\nconst App: React.FC = () => {\n  const
[selectedDate, setSelectedDate] = React.useState<Date | null>(new
Date());\n\n  return (\n    <LocalizationProvider
```

~~~~
dateAdapter={AdapterDateFns}>\n      <DatePicker\n        label=\"Date
picker\"\n        value={selectedDate}\n        onChange={(newValue) =>
{\n          setSelectedDate(newValue);\n        }}\n        renderInpu
t={(params) => <TextField {...params}
/>}\n      />\n    </LocalizationProvider>\n  );\n};\n\nexport default
App;\n```\n\n```json\n{\n  \"@emotion/react\":
\"^11.7.1\",\n  \"@emotion/styled\": \"^11.6.0\",\n  \"@mui/lab\":
\"^5.0.0-alpha.83\",\n  \"@mui/material\":
\"5.14.17\",\n  \"@types/react\": \"^17.0.0 || ^18.0.0\",\n  \"react\":
\"^17.0.0 || ^18.0.0\",\n  \"react-dom\": \"^17.0.0 ||
^18.0.0\"\n}\n```"}]}
~~~~

Table 1 - Fine-tuning example in jsonl

After the tuning is completed we used the following model:

```
model: "ft:gpt-3.5-turbo-1106:xgeeks::8K3oGqZe"
```

Table 2 - Trained model name

Open AI Prompts:

We created multiple prompts to extracting the following information:

- Retrieve a code example from the library tailored to a specific problem.
- Obtain the library dependencies associated with a specific version.
- Generate a ReadMe file complete with an Incident Template.

Aggregate information:

Upon acquiring the necessary data from OpenAI, we seamlessly integrate this information using our stack templates (currently using TypeScript).

With this aggregation, we generate a minimum project that have all library dependencies, accompanied by a demonstrative example of the identified issue.

Generated info:

During this stage, our software orchestrates a new GitHub repository housing the project created during the aggregate phase. Here it’s possible to find the project's core library, along with a user-friendly example encapsulating the user problem. Additionally, our software crafts a sandbox environment, allowing you to seamlessly test the code online or effortlessly share a link with collaborators for collaborative testing.

To enhance your experience, we've thoughtfully included a template for incident reporting in the readme file.

Future Work:

We currently have several ideas on how to enhance this project, but there are three primary issues that require additional attention.

Firstly, we are currently utilizing multiple stacks. Presently, we exclusively host TypeScript, but in the future, our AI could intelligently determine the library language and construct the stack, irrespective of the programming language used.

Secondly, addressing peer dependencies poses a significant challenge. Given that a single library can result in numerous dependencies, managing this effectively is not straightforward.

Lastly, refining the model through additional data could significantly enhance its performance. However, it's important to note that this process may be time-consuming.

Despite these challenges, we are committed to the continuous improvement of this project, since is something that we are going to use in the future.