“No-Code” API Generator

Marcus Gallegos & Prinn Prinyanut

4/29/2021

Abstract

In this project, we attempt to create a no/low-code solution for engineers to create backend web servers. With this product, development teams can collaborate, build and maintain RESTful APIs using only a GUI interface. During the development of this project, numerous startups came to the mainstream such as Autocode which promises “[to] combine a web IDE, always-on hosting and a standard library of APIs to make web programming faster and easier than ever before.” While no-code startups still have hope, the research we have conducted has shown that these products may not be worth the cost of development.

Method

While building this product, we utilized many open source technologies. The main one is the OpenAPI specification aka “Swagger.” Using this well defined schema for defining RESTful APIs is at the heart of our solution.

For a user to store their API in our database, we use a JSON representation of the Open API schema. This specification requires an operation ID for every route which acts as a primary key within our database.

Using this primary key, we store the users input for generating code. The operations we let them select are all the CRUD methods for interacting with the Firestore database.

Generating Code

OpenAPI generator scaffolds the routes for us using an MVC architecture. Within a service or controller, route functions will be named according to the route’s operation id. With this, we can easily parse and replace functions generated by OpenAPI using Regular Expressions. Here is an example.

The Open API generator gives us the following code:

```
const getPetById = ({ petId }) =>
  new Promise(async (resolve, reject) => {
    try {
      resolve(
        Service.successResponse({
          petId,
        })
      )
    } catch (e) {
      reject(
        Service.rejectResponse(e.message || 'Invalid input', e.status || 405)
      )
    }
  })
```

Using the regular expression **`/const getPetById (.*\s)*?\)\;/`** , we find this function in the appropriate file and replace it the code to retrieve a document from the database and return it to the client.

```
const getPetById = ({ petId }) =>
  new Promise(async (resolve, reject) => {
    try {
      db.doc('pets/' + petId)
        .get()
        .then((doc) => {
          resolve(Service.successResponse({ success: true, data: doc.data() }))
        })
    } catch (e) {
      reject(
        Service.rejectResponse(e.message || 'Invalid input', e.status || 405)
      )
    }
  })
```

Database Modularity

In our implementation of this application, we built it with the concept of database modularity. No matter which database you use, all will need to complete CRUD applications. So long as they have an intuitive Node JS library or can be accessed via HTTP methods, our solution can easily be extended to work with any database.

Features & Extension

In the original plan of this project, we intended to integrate with multiple APIs, other than databases. For example, Stripe for processing payments, Sendgrid for automating email marketing. Our application is built on the premise that these APIs would be integrated and so is easy to extend. You simply need to supply a dictionary, or switch statement, of operations and the templated code with a list of parameters to generate such functionality.

Another natural fit to extend this project is to add the ability to do Html templating. For example, our solution only provides the ability to return JSON data. With Express, we could easily add a templating engine so that webpages can be rendered instead of just data.

Conclusion

We have spent a year building this application and are happy with what we have achieved. However, building this project has forced us to think about software engineering and have a personal analysis of the tools that help us build software.

While we believe that the future of software is certain to be dominated by web technologies, we're less convinced that we will be building software *on* the web. If we were to start this project again, we would look to build something integrated to the IDE and probably a command line interface rather than a web application. This would allow users to integrate our application with their current workflow and extend the generated code if needed.

Finally, on the concept of "no-code," we believe there is a niche of tech savvy business users who need to be able to automate tasks. No-code solutions are perfect for this use case. However, building no-code generators is extremely complex and is prone to bugs and security vulnerability. The opportunity cost is better spent building intuitive libraries to interact with complex APIs, or spent building tools integrated with IDEs to make writing code easier.

Our original aspiration was to make this a live application in production and to form a trendy startup company. However, our application still has many bugs and security obstacles to making it suitable for production use. We feel our time would be better spent contributing to open source libraries and other projects. We hope our venture will help other software engineers learn more about what tools are helpful and which ones are no more than novel.