Web server generation - Technical documentation

Domen Vake

June 18, 2021

1 Web servers

A web server is computer software and underlying hardware that accepts requests via HTTP, the network protocol created to distribute resources, or its secure variant HTTPS. A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP, and the server responds with the content of that resource or an error message. The server can also accept and store resources sent from the user agent if configured to do so.

Technologies such as REST and SOAP, which use HTTP as a basis for general computer-to-computer communication, have extended the application of web servers well beyond their original purpose of serving human-readable pages. The Representational state transfer (REST) architectural style emphasises the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems. REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating stateless, reliable web services. Any web service that obeys the REST constraints is informally described as RESTful. Such a web service must provide its Web resources in a textual representation and allow them to be read and modified with a stateless protocol and a predefined set of operations. This approach allows the greatest interoperability between clients and servers in a long-lived Internet-scale environment which crosses

organisational (trust) boundaries.
The constraints of the REST architectural style affect the following architectural properties:

- performance in component interactions, which can be the dominant factor in user-perceived performance and network efficiency;
- scalability allowing the support of large numbers of components and interactions among components.
- simplicity of a uniform interface;
- modifiability of components to meet changing needs (even while the application is running);
- visibility of communication between components by service agents;
- portability of components by moving program code with the data;
- reliability in the resistance to failure at the system level in the presence of failures within components, connectors, or data.

Web service APIs that adhere to the REST architectural constraints are called RESTful APIs.

2 Motivation

RESTful APIs are very common web servers, since they have an intuitive design and solve a highly common problem, that is storing and modifying resources for web pages, web applications, mobile applications and other similar applications.

Personally I often found myself programming RESTful APIs for all sorts of projects. When doing that I found, that they have a very modular architecture, but a lot of code is generics and repetitive boiler plate code. I decided that I want to design an application, that will auto generate fully working RESTful APIs for me.

3 Problem

The problem of generating fully functioning web servers, falls under the field of code generation. Code generation may refer to:

- Code generation (compiler), a mechanism to produce the executable form of computer programs, such as machine code, in some automatic manner
- Automatic programming (source code generation), the act of generating source code based on an ontological model such as a template

Where as my project refers more to the latter, also known as generative programming. Generative programming and the related term meta-programming are concepts whereby programs can be written "to manufacture software components in an automated way" just as automation has improved "production of traditional commodities such as garments, automobiles, chemicals, and electronics." Surce-code generation is the process of generating source code based on a description of the problem or an ontological model such as a template and is accomplished with a programming tool such as a template processor or an integrated development environment (IDE). These tools allow the generation of source code through any of various means. Modern programming languages are well supported by tools like Json4Swift (Swift) and Json2Kotlin (Kotlin).

The goal is to improve programmer productivity.

4 Project plan

The approach to project was to first split the development into phases. Each phase should complete a specific task, that the next phase depends on. The final plan was broken up into 7 phases, where first four of the phases are of theoretical nature and the final 3 phases are the actual realisation of the program.

1. Define result (output) of the program
2. Determine technologies needed for achieving desired result
3. Define the input to the program

4. Determine technologies needed for converting input to desired result
5. Implement a tool that generates the default version of the result
6. Implement input variations and modifications
7. Testing and deploying

5 Implementation

5.1 Define result (output) of the program

In this step I have created the user documentation for the final generated product. This step was required, to determine the goal of the project. I defined the functionality of the generated web server. I decided that the final code should be able to serve resources that have to oblige by the rules of RESTful design. It should be able to register and login users, and with that lock some resources behind a login-wall. The authentication should follow the JSON-web-token protocol and store the passwords hashed and salted. The generated application should be able to talk to a database, where it would store resources and should not loose the data in case of a shutdown (intentional or accidental) if the service.

5.2 Determine technologies needed for achieving desired result

For the technologies of the end product I have chosen NodeJS with express framework, due to the personal familiarity with the technologies. Also I have a lot of reference projects to help me with development of the product. I have encountered many errors so also debugging should be straight forward. The programm should also be generated in TypeScript, due to it making the code easier to maintain. Also Typescript enables easier learning of the generated code (if a user wanted to expand the functionality) due to hard typing of the language.

5.3 Define the input to the program

In this phase I expanded on the user documentation by defining the shape and variation of the possible inputs to the application. I have decided to

split the input into 2 different categories:

- Project meta data
- Project resource models

Each category should be a JSON file that contains the key-value pairs that correspond to a certain functionality within the tool. Mostly the they would determine the inclusion of modules (like authentication) or the way a certain module is implemented (default docker module or Traefik2 reverse proxy docker module). All the input specification is defined in user documentation.

5.4 Determine technologies needed for converting input to desired result

For implementation of the tool I have decided to use the language Rust. Rust is a multi-paradigm programming language designed for performance and safety, especially safe concurrency. Rust is syntactically similar to C++, but can guarantee memory safety by using a borrow checker to validate references. Rust achieves memory safety without garbage collection, and reference counting is optional. Rust's speed, safety, single binary output, and cross-platform support make it an ideal language for creating command line tools.

5.5 Implement a tool that generates the default version of the result

The development in this stage was split into three steps:

1. Read input files
2. Implement template files
3. Export default project template

in the first step a implemented a module, that is able to read the configuration files from the system and map them to models within the tool for reference. In the next step I programmed the whole result RESTful API in NodeJS that has the default functionality. I took the project and implemented a template in rust, that directly corresponds to the result project. And in

the final step I programmed module that takes a template and pareses it into strings, generates folder structure and exports the strings into files into correct folders.

5.6 Implement input variations and modifications

At that point I implemented multiple modules for the template. All the module variations were tied to an input. At that point I implemented default configuration inside the tool, so if the tool was used on insufficient input JSON it would automatically fill the blanks with default settings.

5.7 Testing and deploying

After the all implementation I tested the tool by building many web servers and cross referencing with a working one to find defects. Also I used a program, named Insomnia, to make HTTP requests, to the web servers to if they respond in a correct way.

Once the testing was completed I deployed the tool to the Snap-Store by using Snapcraft.io. I took advantage of the continuous integration, so whenever the master branch on project Github updated, the Snapcraft automatically roll out an update of my application. The tool is now available on the store for all Linux and Arch distributions, that support Snaps.