Requirements

Aaron Power

20-10-2015

Contents

List of Figures

Chapter 1

Requirements

1.1 Requirements Analysis

The user(developer) should be able to pass in a file written in TBD, and optionally data stored in a key-value pairing. This data will then be passed to a compiler, which will return a html file based on the file, and data to be passed back as the HTTP response.

Figure 1.1: Layout file

```
doctype html
html
    head
    title= title
    meta(charset="utf-8")
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
    meta(http-equiv="X-UA-Compatible" content="IE=edge,chrome=1")
    link(rel='stylesheet', href='/stylesheets/normalize.css')
    link(rel='stylesheet', href='/stylesheets/skeleton.css')
    link(rel='stylesheet', href='/stylesheets/style.css')
    script(type='text/javascript', src='/javascripts/jquery.min.js' defer)
    script(type='text/javascript', src='/javascripts/script.js' defer)
    body
        block content
```

Figure 1.2: Index file

```
extends layout

block content
    h1 Hello World

    p Lorem Ipsum...
```

Figure 1.3: Rendered file

```
<!DOCTYPE html>
<html>
    <head>
        <title>Express</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
        <link rel="stylesheet" href="/stylesheets/normalize.css">
        <link rel="stylesheet" href="/stylesheets/skeleton.css">
        <link rel="stylesheet" href="/stylesheets/style.css">
        <script type="text/javascript" src="/javascripts/jquery.min.js" defer>
        </script>
        <script type="text/javascript" src="/javascripts/script.js" defer>
        </script>
    </head>
    <body>
        <h1>Hello World</h1>
        <p>Lorem ipsum...</p>
    </body>
</html>
```

1.1.1 Technical Requirements

All the other technologies would be programmed in Rust, and directly interfaced within a Rust program.

Rust

A new programming language from Mozilla, released in May 15th, 2015.[1] Rust is a systems level programming language. Meaning that Rust is run directly on top of the existing OS(Operating System), as opposed to languages like Java, or Ruby which is run on top of a VM(Virtual Machine). Rust is designed to *combines low-level control over performance with high-level convenience and safety guarantees*[1].

Figure 1.4: Fizz buzz in rust.

```
fn main() {
    for i in 1..101 {
        match (i%3, i%5) {
            (0, 0) => println!("FizzBuzz"),
            (0, _) => println!("Fizz"),
            (_, 0) => println!("Buzz"),
            (_, _) => println!("{}", i),
        }
    }
}
```

Iron

A high level web framework designed for making web servers in Rust.[2] Iron is designed to enable other users to create plugins for it. The end-goal would be to integrate the transcompiler with Iron, so a user could easily integrate within the system, and start using the templating language with ease.

Figure 1.5: Hello world server using Iron.

```
extern crate iron;

use iron::prelude::*;
use iron::status;

fn main() {
    fn hello_world(_: &mut Request) -> IronResult<Response> {
        Ok(Response::with((status::Ok, "Hello World!")))
    }

    Iron::new(hello_world).http("localhost:3000").unwrap();
    println!("On 3000");
}
```

1.2 System Model

The system is mainly two parts, the actual web server written in rust with Iron, and the transcompiler, which will be used as middleware with Iron. Middleware being defined as software that runs between handling the requests, making it easier for the user, or providing new functionality, like adding a templating language.

As shown in TBD. After a request has been handled by the user, but before the response has been sent, the transcompiler will parse the template file, and any file the template requires. Building an AST(Abstract Syntax Tree), and creating a HTML file from the AST. As this is a transcompiler, the program doesn't need to worry about code optimisation, or typical code generation that a typical compiler would. The main work of the program,would be to do lexical analysis, Syntax analysis, and Semantic analysis, and generate the HTML source code if the source is correct, and provide effective, and clear errors if the source code is incorrect.

1.3 Feasibility

There are a lot of risks with making a transcompiler. The most important first step is to have an iron-clad definition of the language it is transcompiling, as changes in the fundamental syntax can cause large sections of code to be rewritten, and could even require in the how the parsing of the syntax is done.

Of course there is always the risk that when the project finishes that it won't have all features specified in the requirements document due to time constraints, but this can be handled with effective time management, and being able to do effective cost analysis such as how viable a feature may be, and how much value does it provide over how much time it will take.

Bibliography

[1] The Rust Core Team. Announcing rust 1.0. http://blog.rust-lang.org/2015/05/15/Rust-1.0.html, May 2015.

[2] Johnathan Reem. Iron. https://github.com/iron/iron.