**Building a HTTP Client with Reqwest | Rust**

[[](https://medium.com/@carlosmarcano2704?source=post_page-----c049cbe4bc2b--------------------------------)](https://medium.com/@carlosmarcano2704?source=post_page-----c049cbe4bc2b--------------------------------)

[Carlos Armando Marcano Vargas](https://medium.com/@carlosmarcano2704?source=post_page-----c049cbe4bc2b--------------------------------)

·

Follow

8 min read

·

Oct 24

12



Photo by [Thula Na](https://unsplash.com/@thula25?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

In this article, we are going to build a basic HTTP client with Reqwest to test REST APIs

This program will have just basic functionalities, like showing the body and the status code from GET, and POST requests.

Also, we will add a feature that allows the program to read the URL from a text file. And to read the URL and the HTTP method we want to execute from a TOML file.

**Requirements**

* Rust installed
* A REST API to test the HTTP client.

**cargo.toml**

[dependencies]  
tokio = { version = "1.15", features = ["full"] }  
reqwest = { version = "0.11.22", features = ["json"] }

[**main.rs**](http://main.rs/)

//main.rs  
use reqwest::Error;  
  
async fn get\_request() -> Result<(), Error> {  
 let response = reqwest::get("https://www.fruityvice.com/api/fruit/apple").await?;  
 println!("Status: {}", response.status());  
  
 let body = response.text().await?;  
 println!("Body:\n{}", body);  
  
 Ok(())  
}  
  
#[tokio::main]  
async fn main() -> Result<(), Error> {  
 get\_request().await?;  
 Ok(())  
}

In the above, we create the get\_resquest() function to make a GET request and show in the console the status code of the response and its body.

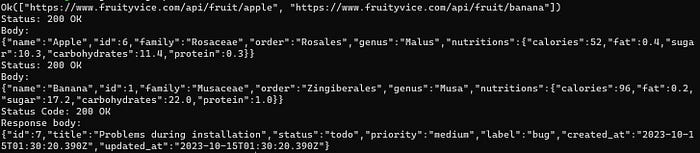
Then we call the get\_request() function in the main() function to make the request when the program runs.

**POST requests**

//main.rs  
async fn post\_request() -> Result<(), Error> {  
 let url = "http://localhost:4000/tasks";  
 let json\_data = r#"{"title":"Problems during installation","status":"todo","priority":"medium","label":"bug"}"#;  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .post(url)  
 .header("Content-Type", "application/json")  
 .body(json\_data.to\_owned())  
 .send()  
 .await?;  
   
 println!("Status Code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
  
}  
  
#[tokio::main]  
async fn main() -> Result<(), Error> {  
 ...  
  
 post\_request().await?;  
 Ok(())  
}

Here we create the post\_request() function, to make a POST request to an API. Inside the post\_request() function we defined the URL and JSON data we want to add to the server. When this function is called, it will print in the console the status code and the body of the response.

Then we call the post\_request() function in the main() function and run the cargo run command in the console.



**PUT Request**

//main.rs  
async fn put\_request() -> Result<(), Error> {  
 let url = "http://localhost:4000/tasks/7";  
 let json\_data = r#"{"title":"Problems during installation","status":"todo","priority":"low","label":"bug"}"#;  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .put(url)  
 .header("Content-Type", "application/json")  
 .body(json\_data.to\_owned())  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
}

Here we create the put\_request() function, to make a PUT request to an API. Inside the put\_request() function we defined the URL and JSON data we want to update to the server. When this function is called, it will print in the console the status code and the body of the response.

Then, we add the put\_request() function to the main() function and run the program.

Make sure to delete the post\_request() function from the main() function, so the program does not call it when it runs.



**DELETE Request**

//main.rs  
async fn delete\_request() -> Result<(), Error> {  
 let url = "http://localhost:4000/tasks/5";  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .delete(url)  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
}

Here we create the delete\_request() function, to make a DELETE request to an API. Inside the delete\_request() function we defined the URL with the parameter of the row we want to delete. When this function is called, it will print in the console the status code and the body of the response.

Then we call the delete\_request() function in the main() function and run the cargo run command in the console.

//main.rs  
#[tokio::main]  
async fn main() -> Result<(), Error> {  
 let file\_path = "./urls.txt";  
 let url\_vector = read\_file\_lines\_to\_vec(&file\_path.to\_string());  
   
 println!("{:?}", url\_vector);  
   
 delete\_request().await?;  
 Ok(())  
}



**Complete code**

//main.rs  
use reqwest::Error;  
mod helpers;  
use helpers::{read\_file\_lines\_to\_vec};  
  
async fn get\_request() -> Result<(), Error> {  
  
 let file\_path = "./urls.txt";  
 let url\_vector = read\_file\_lines\_to\_vec(&file\_path.to\_string());  
 match &url\_vector {  
 // If the operation was successful, make requests to urls in the file.  
 Ok(file\_contents) => {  
 for url in file\_contents {  
 let response = reqwest::get(url).await?;  
 println!("Status code: {}", response.status());  
  
 let body = response.text().await?;  
 println!("Response body:\n{}", body);  
 }  
 }  
  
 // If the operation failed, print the error message to the console.  
 Err(error) => {  
 println!("Error reading file: {}", error);  
 }  
 }  
 Ok(())  
   
}  
  
async fn post\_request() -> Result<(), Error> {  
 let url = "http://localhost:4000/tasks";  
 let json\_data = r#"{"title":"Problems during installation","status":"todo","priority":"medium","label":"bug"}"#;  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .post(url)  
 .header("Content-Type", "application/json")  
 .body(json\_data.to\_owned())  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
  
}  
  
async fn put\_request() -> Result<(), Error> {  
 let url = "http://localhost:4000/tasks/7";  
 let json\_data = r#"{"title":"Problems during installation","status":"todo","priority":"low","label":"bug"}"#;  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .put(url)  
 .header("Content-Type", "application/json")  
 .body(json\_data.to\_owned())  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
}  
  
async fn delete\_request() -> Result<(), Error> {  
 let url = "http://localhost:4000/tasks/5";  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .delete(url)  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
}  
  
#[tokio::main]  
async fn main() -> Result<(), Error> {  
 let file\_path = "./urls.txt";  
 let url\_vector = read\_file\_lines\_to\_vec(&file\_path.to\_string());  
   
 println!("{:?}", url\_vector);  
   
 delete\_request().await?;  
 Ok(())  
}

Using just a text file will make this program difficult to use. The .txt file for just making a GET request to multiple URLs will be fine. But what if we want to use the other HTTP methods?

So, we are going to add another feature, to read a config file where we write the URLs we want to make the requests and the HTTP methods we want to use.

**Adding Config file**

As a config file, we are going to use a TOML file. To parse this file we have to add a TOML parser and Serde to the project’s dependencies.

**cargo.toml**

...  
serde = {version = "1.0", features = ["derive"]}  
toml = "0.8.2"

Now, we create a config.toml file in the project's root directory.

**config.toml**

[config]  
url = "http://localhost:4000/tasks/8"  
method = "DELETE"

[**main.rs**](http://main.rs/)

Here we have to create the struct of the data we want to deserialize from the TOML file.

use serde::Deserialize;  
use std::fs;  
use serde\_json::Value;  
  
#[derive(Deserialize)]  
struct Data {  
 config: Config,  
}  
#[derive(Deserialize)]  
struct Config {  
 url: String,  
 method: String,  
}

I going to comment on the code that opens the .txt file in the main() function. And write the code that opens the .toml file.

//main.rs  
#[tokio::main]  
async fn main()-> Result<(), Error> {  
 //let file\_path = "./urls.txt";  
 // let url\_vector = read\_file\_lines\_to\_vec(&file\_path.to\_string());  
   
 ...   
 let filename = "config.toml";  
   
 let contents = match fs::read\_to\_string(filename) {  
   
 Ok(c) => c,  
   
 Err(error) => {  
 (&error).to\_string()   
 }  
 };

Now, we parse the content of the TOML file.

let data: Data = match toml::from\_str(&contents) {  
 Ok(d) => d,  
   
 Err(error) => {  
   
 eprintln!("Unable to load data because `{}`", error);  
   
 std::process::exit(1);  
 }  
 };

For POST requests and PUT requests, we need to send JSON data.

Something I figured out was to use a JSON parser and extract the body from a JSON file.

For this, we have to add serde\_json as a dependency.

serde\_json = "1.0"

In the main function, we write the code to open and parse the JSON file.

#[tokio::main]  
async fn main()-> Result<(), Error> {  
 ...  
  
 let body = {  
 let file\_content = fs::read\_to\_string("./body.json").expect("Error reading file");  
 serde\_json::from\_str::<Value>(&file\_content).expect("Error serializing to JSON")  
 };  
  
 Ok(())  
  
}

Finally, we add the control flow.

//main.rs  
#[tokio::main]  
async fn main()-> Result<(), Error> {  
 ...  
  
  
 if data.config.method == "DELETE" {  
 delete\_request(data.config.url).await;  
 } else if data.config.method == "POST" {  
 post\_request(data.config.url, body.to\_string()).await;  
  
 } else if data.config.method == "PUT" {  
 put\_request(data.config.url, body.to\_string()).await;  
 } else {  
   
 get\_request(data.config.url).await;  
 }  
  
   
 Ok(())  
  
   
}

Now, we create the http\_method.rs file and move all the functions related to the HTTP requests to this new file.

Then, create a function with the control flow in the main.rs file.

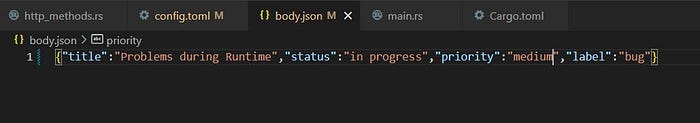
//main.rs  
async fn method\_control(http\_method: &str, url: String, body: String  
) -> Result<(), reqwest::Error> {  
  
 match http\_method {  
 "POST" => post\_request(url, body).await,  
 "PUT" => put\_request(url, body).await,  
 "DELETE" => delete\_request(url).await,  
 \_ => get\_request(url).await,  
  
 }  
}

Now, let’s try our HTTP client.

Write the URL and the HTTP method in the config.toml file.



Define the body you want to send to the API.



Then run the cargo run command in the console.



Complete the http\_method.rs file.

use reqwest::Error;  
  
pub async fn get\_request(url: String) -> Result<(), Error> {  
  
  
 let response = reqwest::get(url).await?;  
 println!("Status code: {}", response.status());  
  
 let body = response.text().await?;  
 println!("Response body:\n{}", body);  
 Ok(())  
   
}  
  
pub async fn post\_request(url: String, json\_data: String) -> Result<(), Error> {  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .post(url)  
 .header("Content-Type", "application/json")  
 .body(json\_data.to\_owned())  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
  
}  
  
pub async fn put\_request(url: String, json\_data: String) -> Result<(), Error> {  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .put(url)  
 .header("Content-Type", "application/json")  
 .body(json\_data.to\_owned())  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
}  
  
pub async fn delete\_request(url: String) -> Result<(), Error> {  
  
 let client = reqwest::Client::new();  
  
 let response = client  
 .delete(url)  
 .send()  
 .await?;  
   
 println!("Status code: {}", response.status());  
  
 let response\_body = response.text().await?;  
  
 println!("Response body: \n{}", response\_body);  
  
 Ok(())  
}

Complete main.rs file.

use serde::Deserialize;  
use std::fs;  
use serde\_json::Value;  
  
mod http\_methods;  
use http\_methods::{get\_request, post\_request, put\_request, delete\_request};  
  
  
#[derive(Deserialize)]  
struct Data {  
 config: Config,  
}  
#[derive(Deserialize)]  
struct Config {  
 url: String,  
 method: String,  
}  
  
#[tokio::main]  
async fn main()-> Result<(), Error> {  
   
 let filename = "config.toml";  
   
 let contents = match fs::read\_to\_string(filename) {   
 Ok(c) => c,   
 Err(error) => {  
 (&error).to\_string()   
 }  
 };  
  
 let data: Data = match toml::from\_str(&contents) {   
 Ok(d) => d,  
 Err(error) => {   
 eprintln!("Unable to load data because `{}`", error);  
 std::process::exit(1);  
 }  
 };  
  
 let body = {  
 let file\_content = fs::read\_to\_string("./body.json").expect("Error reading file");  
 serde\_json::from\_str::<Value>(&file\_content).expect("Error serializing to JSON")  
 };  
  
 let result = method\_flow(&data.config.method, data.config.url, body.to\_string()).await;  
  
 match result {  
 Ok(contents) => contents,  
 Err(e) => println!("Error during the request: {}", e),  
 }  
 Ok(())  
   
}  
  
async fn method\_control(http\_method: &str, url: String, body: String) -> Result<(), reqwest::Error> {  
  
 match http\_method {  
 "POST" => post\_request(url, body).await,  
 "PUT" => put\_request(url, body).await,  
 "DELETE" => delete\_request(url).await,  
 \_ => get\_request(url).await,  
 }  
}

**Conclusion**

In this article, we learn how to use the Reqwest crate to make HTTP requests. And also we built a HTTP client to take its input from a text file and a configuration file.

I built this program just for learning purposes and learn how to use the Reqwest crate. But I want it to build something different than just a program that makes GET requests.

The source code is [here](https://github.com/carlosm27/reqwest-demo).

Thank you for taking the time to read this article.

If you have any recommendations about other packages, architectures, how to improve my code, my English, or anything; please leave a comment or contact me through [**Twitter**](https://twitter.com/Carlos_marcv), or [**LinkedIn**](https://www.linkedin.com/in/carlos-marcano-a2135a134/).

**Resources**

[Reqwest documentation](https://docs.rs/reqwest/latest/reqwest/)

[Making HTTP Requests in Rust With Reqwest](https://blog.logrocket.com/making-http-requests-rust-reqwest/)

[Making HTTP requests in Rust with Reqwest](https://www.makeuseof.com/rust-reqwest-http-requests/)

[How to Work With TOML Files in Rust](https://www.makeuseof.com/working-with-toml-files-in-rust/)

[Rust Load a TOML File](https://codingpackets.com/blog/rust-load-a-toml-file/)

*Originally published at*[*https://carlosmv.hashnode.dev*](https://carlosmv.hashnode.dev/building-a-http-client-with-reqwest-rust)*.*