**Rust: The fastest rust web framework in 2024**

[[Randi Eka Setiawan](https://randiekas.medium.com/?source=post_page-----cf738c40343b--------------------------------)](https://randiekas.medium.com/?source=post_page-----cf738c40343b--------------------------------)

[Randi Eka Setiawan](https://randiekas.medium.com/?source=post_page-----cf738c40343b--------------------------------)

·

Follow

4 min read

·

3 days ago

103

3



As Rust gains popularity, choosing the right web framework is crucial.

We’ll compare the performance of **[Actix](https://actix.rs/" \t "_blank)**, [**Axum**](https://github.com/tokio-rs/axum), [**Rocket**](https://rocket.rs/), [**Tide**](https://github.com/http-rs/tide), [**Gotham**](https://gotham.rs/), [**Nickel**](https://nickel-org.github.io/), **[Ntex](https://ntex.rs/" \t "_blank)**, and [**Poem**](https://github.com/poem-web/poem) using the “Hello World” benchmark.

This simple test is just the beginning, and based on interest, we’ll delve into more complex scenarios like static file serving and JSON processing.

**Methodology:**

To ensure a fair and standardized comparison, we subjected each web framework to a basic “Hello World” scenario.

This involves creating a minimal web server that responds with a “Hello, World!” message to incoming HTTP requests.

The benchmarks were conducted on the same machine MacBook Pro 2018 with 32GB RAM and 6 Core Intel i9, and each framework was evaluated for its speed, resource usage, and ease of implementation.

The load tester is apache bench. we will execute the build release.

The code for each application is as follows:

**Actix**

[package]  
name = "actix"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
actix-web = "4"

use actix\_web::{get, App, HttpResponse, HttpServer, Responder};  
  
#[get("/")]  
async fn hello() -> impl Responder {  
 HttpResponse::Ok().body("Hello world!")  
}  
  
#[actix\_web::main]  
async fn main() -> std::io::Result<()> {  
 HttpServer::new(|| {  
 App::new()  
 .service(hello)  
 })  
 .bind(("127.0.0.1", 8080))?  
 .run()  
 .await  
}

**Axum**

[package]  
name = "axum-hello"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
axum = "0.7.3"  
tokio = { version = "1.0", features = ["full"] }

use axum::{response::Html, routing::get, Router};  
  
#[tokio::main]  
async fn main() {  
 // build our application with a route  
 let app = Router::new().route("/", get(handler));  
  
 // run it  
 let listener = tokio::net::TcpListener::bind("127.0.0.1:8080")  
 .await  
 .unwrap();  
 println!("listening on {}", listener.local\_addr().unwrap());  
 axum::serve(listener, app).await.unwrap();  
}  
  
async fn handler() -> Html<&'static str> {  
 Html("Hello world!")  
}

**Rocket**

[package]  
name = "rocket-hello"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
rocket = "0.5.0"

#[macro\_use] extern crate rocket;  
  
#[get("/")]  
fn hello() -> String {  
 format!("Hello world!")  
}  
  
#[launch]  
fn rocket() -> \_ {  
  
 let config = rocket::Config {  
 port: 8080,  
 log\_level: rocket::config::LogLevel::Off,  
 ..rocket::Config::debug\_default()  
 };  
  
 rocket::custom(&config)  
 .mount("/", routes![hello])  
  
}

**Tide**

[package]  
name = "tide-hello"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
tide = "0.16.0"  
async-std = { version = "1.8.0", features = ["attributes"] }

#[async\_std::main]  
async fn main() -> Result<(), std::io::Error> {  
   
 let mut app = tide::new();  
   
 app.at("/").get(|\_| async { Ok("Hello world!") });  
 app.listen("127.0.0.1:8080").await?;  
   
 Ok(())  
}

**Gotham**

[package]  
name = "gotham-hello"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
gotham = "0.7.2"

use gotham::state::State;  
  
pub fn say\_hello(state: State) -> (State, &'static str) {  
 (state, "Hello world!")  
}  
  
/// Start a server and call the `Handler` we've defined above for each `Request` we receive.  
pub fn main() {  
   
 gotham::start("127.0.0.1:8080", || Ok(say\_hello)).unwrap()  
  
}

**Ntex**

[package]  
name = "ntex-hello"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
ntex = { version= "0.7.16", features = ["tokio"] }

use ntex::web;  
  
#[web::get("/")]  
async fn index() -> impl web::Responder {  
 "Hello, World!"  
}  
  
#[ntex::main]  
async fn main() -> std::io::Result<()> {  
  
 web::HttpServer::new(||   
 web::App::new()  
 .service(index)  
 )  
 .bind(("127.0.0.1", 8080))?  
 .run()  
 .await  
}

**POEM**

[package]  
name = "poem-hello"  
version = "0.1.0"  
edition = "2021"  
  
[dependencies]  
poem = "1.3.59"  
tokio = { features = ["rt-multi-thread", "macros"] }

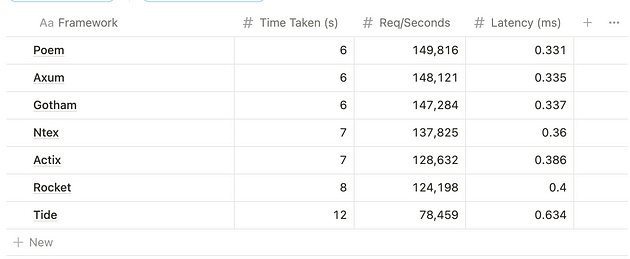
use poem::{  
 get, handler, listener::TcpListener, middleware::Tracing, EndpointExt, Route, Server,  
};  
  
#[handler]  
fn hello() -> String {  
 format!("Hello world!")  
}  
  
#[tokio::main]  
async fn main() -> Result<(), std::io::Error> {  
   
 let app = Route::new().at("/", get(hello)).with(Tracing);  
 Server::new(TcpListener::bind("0.0.0.0:8080"))  
 .name("hello-world")  
 .run(app)  
 .await  
   
}

**Results:**

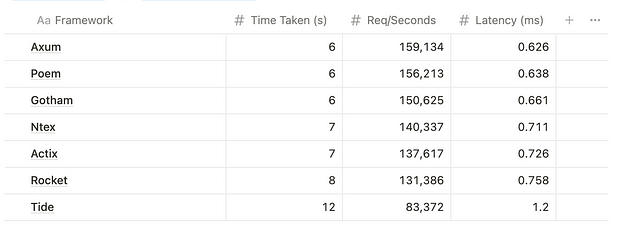
A total of 1.000.000 requests are executed for each test of 50, 100, and 150 connections.

The results in table form are as follows:

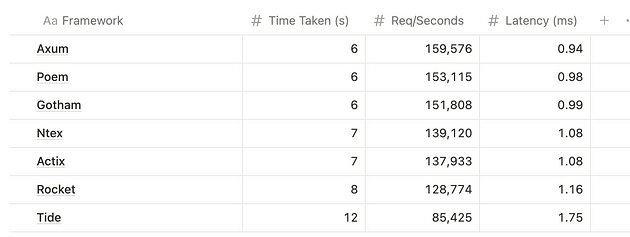
**With 50 Concurrent**



**With 100 Concurrent**



**With 150 Concurrent**



**Verdict**

Tide is the slowest of all (only can complete 1M request in 12 seconds, with average 159K Req / Seconds).

Axum is the fastest (can complete 1M Request in 6 Seconds).

The resource usage is almost the same for all the competitors.

**Winner: Axum**

This was only about the do-nothing ‘hello world’ server. The winning margins might not be as wide for more complex cases. I’ll get to them very soon.

source code you can download [**here**](https://github.com/randiekas/rust-web-framework-benchmark)! or

<https://github.com/randiekas/rust-web-framework-benchmark>

Thanks for reading this article!