Introducing axum Web Framework

Tomáš Jašek

April 16, 2024

Tomáš Jašek

- Rust backend engineer at Sonalake
- LinkedIn, Github

Goal

Show that Rust is ready for web server development

Agenda

- web 101
- what is a web framework
- notable web framework features
- Rust web frameworks overview
- **look at examples in axum**

Web 101

Static Site

- browsing to a URL causes the browser to connect to a web server and send a request
- web server translates the URL to a path on local filesystem
- web server responds with file contents

General Web Services

- the program sending the request doesn’t have to be a web browser
- web server can respond with different payload each time
- response doesn’t have to be HTML

What is a web framework?

- web framework is a type of crate offering some functionality related to web
- today, we'll discuss server-side only
- web frameworks allow us to focus on business logic rather than server implementation details

Notable Web Framework features

- Network I/O
- HTTP request deserialization
- HTTP response serialization
- middleware
- routing
- observability
- error handling
- testing

What about Rust?

- actix-web: stabilized, actively maintained
- pavex: in early stages
- axum: in development
- warp: in development

…and many more on crates.io: Web Programming/HTTP Server

Why look at axum?

- beginner-friendly
- bare-bones
- modular
- less constraints on handlers
- comprehensive documentation
- tokio & tower ecosystem

First look at axum: Hello World

```rust
#[tokio::main]
async fn main() {
    let app = Router::new().route("/", get(hello_world));

    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000")
        .await
        .unwrap();

    axum::serve(listener, app).await.unwrap();
}

async fn hello_world() -> &'static str {
    "Hello, World!"
}
```

Re-state that sending a request to the server causes the `hello_world` function to be called.

Routing

Problem: I have a web service that serves multiple endpoints. I'd like to invoke different functions depending on the URL.

```rust
use axum::{Router, routing::get};

// our router
let app = Router::new()
    .route("/", get(root))
    .route("/foo", get(get_foo).post(post_foo));

// handlers
async fn root() {}
async fn get_foo() {}
async fn post_foo() {}
```

axum::Router

Request parameters

Problem: In a single route handler, the caller has passed a request parameter. How can I access it?

- examples: query parameters, path parameters, request body

```
let app = Router::new()
    .route("/path/:user_id", get(get_user_by_id))
    .route("/search", post(search))
    .route("/form", post(process_form));

async fn get_user_by_id(Path(x): Path<u32>) {}
async fn search(Query(params): Query<HashMap<String, String>>) {}
async fn process_form(Form(params): Form<HashMap<String, String>>) {}
```

axum::extract

Dealing with Rejection

But what if the request parameter fails to parse?

axum offers a fallible implementation. Instead of using `Path<T>`, we can use `Result<Path<T>, Path<T>::Rejection>`.

```rust
let app = Router::new()
    .route("/path/:user_id", get(get_user_by_id));

async fn get_user_by_id(
    result: Result<Path<u32>, Path<u32>::Rejection>,
) {}
```

Response body

Problem: I want to set a response body.

```
let app = Router::new().route("/", get(hello_world));


async fn hello_world() -> &'static str {
    "Hello, World!"
}
```

Setting status code

Problem: I want to return a 404 error code from my handler.

```
async fn hello_world() -> (StatusCode, &'static str) {
    (
        StatusCode::NOT_FOUND,
        "Hello, World!"
    )
}
```

Attaching response headers

Problem: I want to set a header in a reponse from my handler.

```rust
async fn hello_world() -> (HeaderMap, &'static str) {
    let mut headers = HeaderMap::new();
    headers.insert(HeaderName::CONTENT_TYPE, "application/json");
    (
        headers,
        "{ \"hello\": \"world\" }"
    )
}
```

Note on two different response types

Problem: What if my handler's behavior branches?

```rust
let app = Router::new()
    .route("/path/:user_id", get(get_user_by_id));

async fn get_user_by_id(
    user_id: Result<Path<u32>, Path<u32>::Rejection>,
) -> Response {
    match result {
        Ok(user_id) => format!("User with id={user_id}").into_response(),
        Err(err) => (
            StatusCode::NOT_FOUND,
            format!("user not found. reason: {err}"),
        ).into_response(),
    }
}
```

axum::response::IntoResponse

Moving some logic out of a handler

Problem: My handler is too complex!

```rust
async fn get_user_by_id(
    user_id: UserId,
) -> String {
    format!("User with id={}", user_id.0)
}
```

```rust
struct UserId(pub u32);

#[axum::async_trait]
impl<S> FromRequestParts<S> for UserId {
    type Rejection = Response;

    async fn from_request_parts(
        parts: &mut Parts,
        state: &S,
    ) -> Result<Self, Self::Rejection> {
        Path::<u32>::from_request_parts(parts, state)
            .await
            .map(|path| Self(path.0))
            .map_err(|err|
                (StatusCode::NOT_FOUND,
                    format!("user not found. reason: {err}"),
                ).into_response())
    }
}
```

axum::extract::FromRequestParts

Global State

Problem: I need to access database in my route handler

```rust
let app = Router::new()
    .route("/", get(index))
    .with_state(PgPool::new(...));

// #[axum::debug_handler]
async fn index(State(pg_pool): State<PgPool>) {
    // use pg_pool
}
```

- axum::extract::State
- other examples: redis connection, configuration file, CA root of trust, …
- compile-time check that Router has the desired type of thing attached
  - *we can add debug_handler to prettify error messages*
- alternative: `Extension` (no compile-time checks)

Middleware

Problem: I have a single reusable piece of functionality which applies to multiple endpoints. How can I reuse it?

Typical example: authentication via axum-login

```
let app = Router::new()
    .route(
        "/dashboard",
        get(user_dashboard),
    )
    .route_layer(login_required!(Backend, login_url = "/login"));

async fn user_dashboard(auth_session: AuthSession) -> Response {
    match auth_session.user {
        Some(user) => DashboardTemplate {
            username: &user.username,
        }
        .into_response(),

        None => StatusCode::UNAUTHORIZED.into_response(),
    }
}
```

Observability

Problem: I want to instrument the HTTP requests & responses using tracing

```rust
tracing_subscriber::fmt::init();

let mut layer = ServiceBuilder::new()
    .layer(
        TraceLayer::new_for_http()
            .make_span_with(|request: &Request<Full<Bytes>>| {
                tracing::info_span!("HTTP request")
            })
    );

let app = Router::new()
    .route("/", get(hello_world))
    .layer(layer);

async fn hello_world() -> &'static str {
    "Hello, World!"
}
```

Error Handling

Problem: Route handling isn't always successful

```rust
pub struct MyError;

async fn hello_world() -> Result<&'static str, MyError> {
    // we can now use `?` in the handler!
    Err(MyError)
}

impl IntoResponse for MyError {
    fn into_response(self) -> Response {
        (StatusCode::BAD_REQUEST, "Bad Request").into_response()
    }
}
```

Testing

Problem: I want to send a request to the web server without binding it to a port.

```rust
fn create_router() -> Router { todo!() }

#[tokio::test]
async fn test() {
    let app = create_router();

    let response = app
        .oneshot(
            Request::builder()
                .uri("/")
                .body(Body::empty())
                .unwrap()
        )
        .await
        .unwrap();

    assert_eq!(response.status(), StatusCode::OK);
}
```

More resources

- Are We Web Yet?
- axum docs
- axum examples
- axum login