Fast and safe web services with axum

About me

- Tim Eggert
- Working at a Startup called drant as a Staff Engineer & Security Officer
- Recently moved to Baldenhain
- **Looking for a cozy place to talk about tech frequently!**

Goal

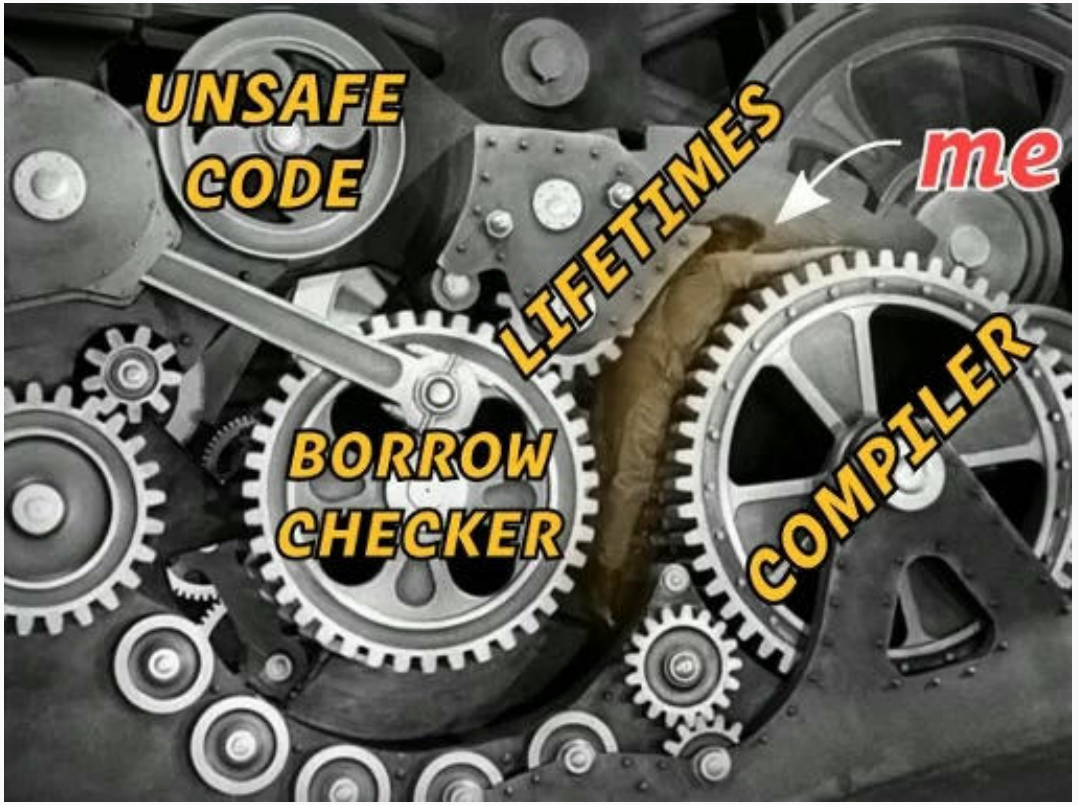
- Broad overview about Rust and it's current usage/applications
- Detailed example to illustrate some features
- Encourage audience to try it out
- Provide resources for continued research

My way to Rust

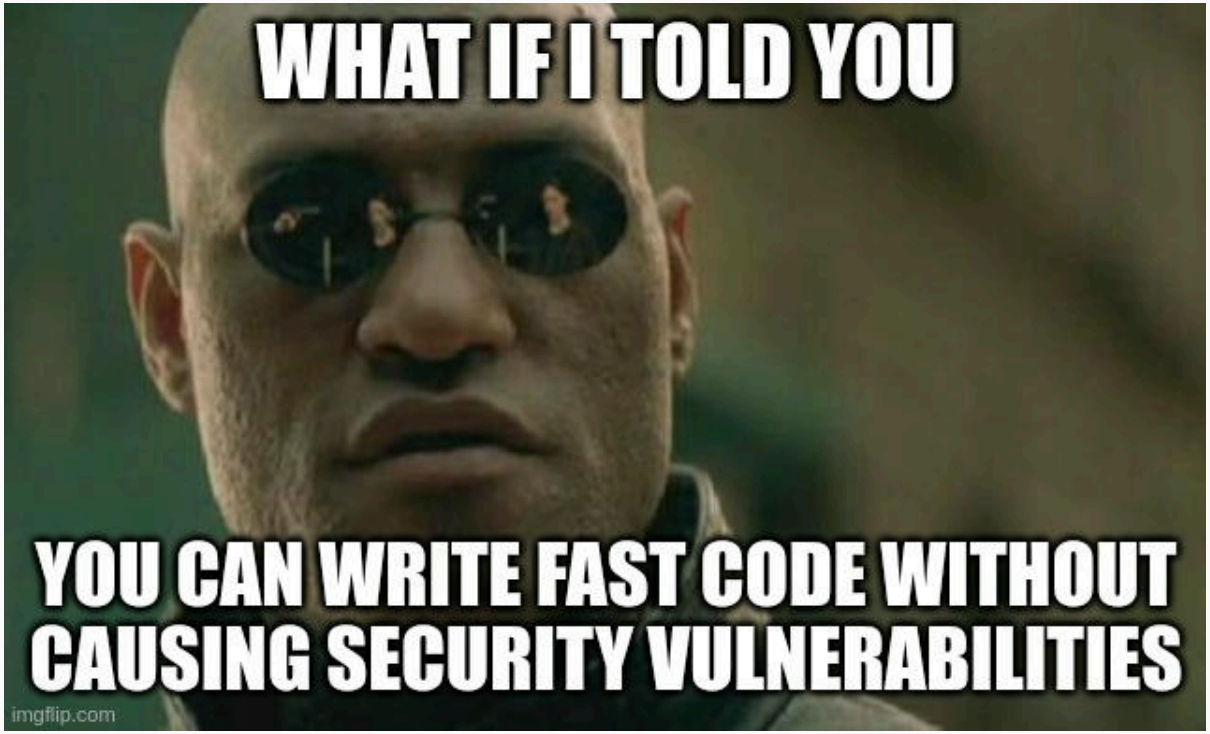
- Started programming with dynamic languages (PHP, later Python)
- Sometimes Java, but too verbose…
- Some C++ 11 with type inference

My way to Rust

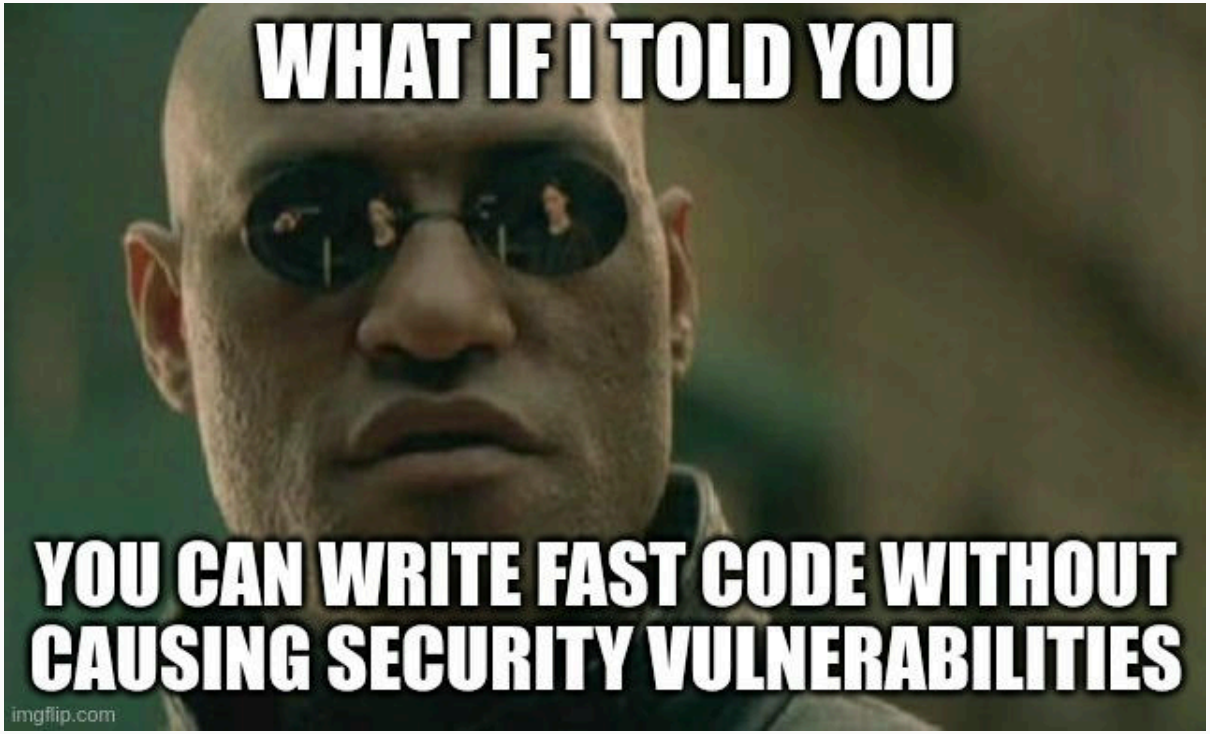
- Started programming with dynamic languages (PHP, later Python)
- Sometimes Java, but too verbose…
- Some C++ 11 with type inference
- Started my Rust journey in 2018 (freetime only)
  - Steep learning curve
  - Lot's of headaches with the Memory model



Rust Intro: Purpose



Rust Intro: Purpose



"A language empowering everyone to build reliable and efficient software."

— https://www.rust-lang.org

Rust Intro: Disclaimer

Disclaimer!!!
I am a fan of Rust, there might be a bias towards it,
which I cannot justify rationally...

Rust Intro: Typical language features

- Compiled, statically typed language
- Generics
- Async
- Macros
- Multi-Threading
- Zero Cost Abstractions
- Minimal Runtime
- Inline Assembly Code

Rust Intro: Novel / exotic language features

- Ownership with Borrow Checker
- Memory Safety without Garbage Collection
- Pattern Matching + Advanced Enums
- Traits + Trait Objects (Structs, but no Classes; Composition over Inheritance)
- Option (no nulls pointers!) and Result Types
- Immutable by Default

Rust Intro: Novel / exotic language features

- Ownership with Borrow Checker
- Memory Safety without Garbage Collection
- Pattern Matching + Advanced Enums
- Traits + Trait Objects (Structs, but no Classes; Composition over Inheritance)
- Option (no nulls pointers!) and Result Types
- Immutable by Default
- Ecosystem at hand with `cargo`: Build, test, release, format, lint, manage dependencies (so-called crates)
- Many different build targets
  - Embedded devices like ESP32
  - Major CPU architectures: x86, ARM,
  - Multiple platforms: Linux, Mac, Windows, Android, Web Assembly, ...

Rust Intro: Novel / exotic language features

- Ownership with Borrow Checker
- Memory Safety without Garbage Collection
- Pattern Matching + Advanced Enums
- Traits + Trait Objects (Structs, but no Classes; Composition over Inheritance)
- Option (no nulls pointers!) and Result Types
- Immutable by Default
- Ecosystem at hand with `cargo`: Build, test, release, format, lint, manage dependencies (so-called crates)
- Many different build targets
  - Embedded devices like ESP32
  - Major CPU architectures: x86, ARM,
  - Multiple platforms: Linux, Mac, Windows, Android, Web Assembly, ...
- Widely adopted own documentation standard
- Enterprise features for stability and maintainability (Editions)


INFORMATION YOU GIVE
TOO MUCH IT IS
makeameme.org

Rust Intro: Origins

- Brainchild of Graydon Hoare started in 2006
- Sponsored (2009) + officially adopted (2010) by Mozilla
- Decision at Mozilla: Build new Browser Engine (Servo) from scratch based on Rust
- Rust 1.0 was released (2015)
- Adotion by bigger companies in 2020 (Amazon, Google, Microsoft)
  - Rust Foundation was created in 2021
  - Big Corps hired core engineers (Ex-Mozillians) and also long standing community members

Rust Intro: Usage / Applications

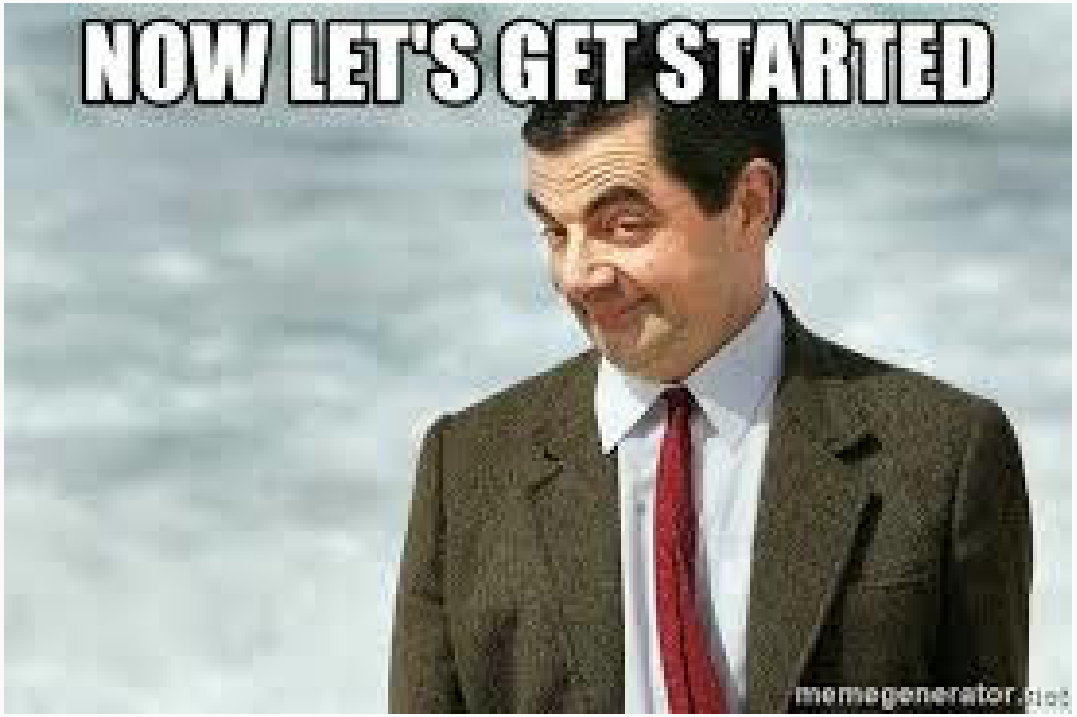
- Android
- AWS Lambda (via Firecracker)
- Discord, Dropbox, Cloudflare backend systems
- Mozilla Servo (browser engine)

Rust Intro: Usage / Applications

- Android
- AWS Lambda (via Firecracker)
- Discord, Dropbox, Cloudflare backend systems
- Mozilla Servo (browser engine)
- Databases (Meilisearch, Qdrant, ...)
- CLI Tools, Editors, Terminal Emulators, Shells, Language tooling (ruff, uv, ...)
- Deno (nodejs runtime competitor)
- Operating Systems (Redox OS, Linux Kernel)
- Cryptocurrency projects

... and many many more

Demo Time



Demo Time: Setup / Installation

- Install Rust toolchain via rustup.rs:

```shell
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

- Initiate a new project

```shell
cargo init hello_world
cd hello_world
cargo run
```

Demo Time: Hello World

Rust

```rust
fn main() {
  println!("Hello World!");
}
```

Rust Web Backend Frameworks

- Famous choices: Actix Web, Rocket, Warp, Axum, Rouille, Tide, …
- Framework Comparison
- For benchmark results, see the Tech Empower Web Framework Benchmarks

Initialize project

```Shell
cargo init todos
cd todos
```

Install dependencies

```Shell
cargo add axum
cargo add tokio --features full
cargo add serde --features derive
cargo add uuid --features v4 --features serde
```

ToDo Example: Structure

- Simple static axum server
- Static JSON response
- Dynamic JSON response
- Create Todos
- Delete Todos

ToDo Example: Static axum server

Static Axum Server Rust

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

#[tokio::main]
async fn main() {
    let app = Router::new().route("/todos", get(list_todos));
    println!("Server running on http://0.0.0.0:3000");
    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: Static axum server

Static Axum Server Rust

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

#[tokio::main]
async fn main() {
    let app = Router::new().route("/todos", get(list_todos));
    println!("Server running on http://0.0.0.0:3000");
    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: Static axum server

Static Axum Server Rust

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

#[tokio::main]
async fn main() {
    let app = Router::new().route("/todos", get(list_todos));
    println!("Server running on http://0.0.0.0:3000");
    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: Static axum server

Static Axum Server Rust

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

#[tokio::main]
async fn main() {
    let app = Router::new().route("/todos", get(list_todos));
    println!("Server running on http://0.0.0.0:3000");
    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: static list_todos function

handler function to list Todos Rust

```rust
async fn list_todos() -> impl IntoResponse {
    // Just return a static string for now
    return "Todos";
}
```

ToDo Example: Running the static server

```
cargo run --example 01_static
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/examples/01_static`
Server running on http://0.0.0.0:3000
```

Shell

ToDo Example: Static JSON response

Add a TodoItem type / struct Rust

```rust
use serde::{Deserialize, Serialize};
use uuid::Uuid;

#[derive(Serialize, Deserialize, Clone)]
struct TodoItem {
    id: Uuid,
    title: String,
    completed: bool,
}
```

ToDo Example: Static JSON response

Return static TodoItem as JSON Rust

```rust
use axum::Json;

async fn list_todos() -> Json<Vec<TodoItem>> {
    let todo = TodoItem {
        id: Uuid::new_v4(),
        title: "First Todo".into(),
        completed: false,
    };
    return Json(vec![todo]);
}
```

ToDo Example: Dynamic JSON response

Add shared state Rust

```rust
use tokio::sync::RwLock;

#[derive(Default)]
struct AppState {
    todos: RwLock<Vec<TodoItem>>,
}
```

ToDo Example: Dynamic JSON response

Add shared state Rust

```rust
use std::sync::Arc;

#[tokio::main]
async fn main() {
    let state = Arc::new(AppState::default());
    let app = Router::new()
        .route("/todos", get(list_todos))
        .with_state(state);
    ...
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: Dynamic JSON response

Use AppState in list handler Rust

```rust
use axum::extract::State;

async fn list_todos(State(state): State<Arc<AppState>>) -> Json<Vec<TodoItem>> {
    let todos = state.todos.read().await.clone();
    return Json(todos);
}
```

ToDo Example: Create Todo feature

Add `POST /todos` Rust

```rust
use std::sync::Arc;

#[tokio::main]
async fn main() {
    let state = Arc::new(AppState::default());
    let app = Router::new()
        .route("/todos", get(list_todos).post(create_todo))
        .with_state(state);
    ...
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: Create Todo feature

Add a request type for TodoItem Rust

```rust
#[derive(Serialize, Deserialize, Clone)]
struct TodoItemCreateRequest {
    title: String,
    completed: bool,
}
```

ToDo Example: Create Todo feature

Add `create_todo` handler function — Rust

```rust
async fn create_todo(
    State(state): State<Arc<AppState>>,
    Json(payload): Json<TodoItemCreateRequest>,
) -> Json<TodoItem> {
    let mut todos = state.todos.write().await;
    let todo = TodoItem {
        id: Uuid::new_v4(),
        title: payload.title,
        completed: payload.completed,
    };
    todos.push(todo.clone());
    return Json(todo);
}
```

ToDo Example: Delete Todo feature

Add `DELETE /todo/{id}` Rust

```rust
use axum::routing::delete;

#[tokio::main]
async fn main() {
    let state = Arc::new(AppState::default());
    let app = Router::new()
        .route("/todos", get(list_todos).post(create_todo))
        .route("/todos/{id}", delete(delete_todo))
        .with_state(state);
    ...
    axum::serve(listener, app).await.unwrap();
}
```

ToDo Example: Delete Todo feature

Add `DELETE /todo/{id}` — Rust

```rust
use axum::{extract::Path, http::StatusCode};

async fn delete_todo(
    Path(id): Path<Uuid>,
    State(state): State<Arc<AppState>>
) -> StatusCode {
    let mut todos = state.todos.write().await;
    if let Some(pos) = todos.iter().position(|todo| todo.id == id) {
        todos.remove(pos);
        return StatusCode::NO_CONTENT;
    } else {
        return StatusCode::NOT_FOUND;
    }
}
```

ToDo Example

Building a release version

```
$> cargo build --release
```

Shell

```
$> ls -lah ./target/release/todos
-rwxr-xr-x@ 1 tim  staff   1.8M Jan 11 00:33 ./target/release/todos*
```

Questions now or later while socializing :)

Additional Notes

- Weekly newsletter: “This week in Rust”
- Books / Reads:
  - “The Rust Programming Language”
  - rustbyexample.io
  - “Programming Rust: Fast, Safe Systems Development”
  - “Rust for Rustaceans”
  - “Zero To Production In Rust”
- Youtube channels:
  - Crust of Rust (Jon Gjengset)
  - Ryan Levick (old but nice)
  - Chris Biscardi

Fun fact: Those slides were created with typst (a Latex successor written in Rust)

Thank you for listening and a big Thanks to our organizers!