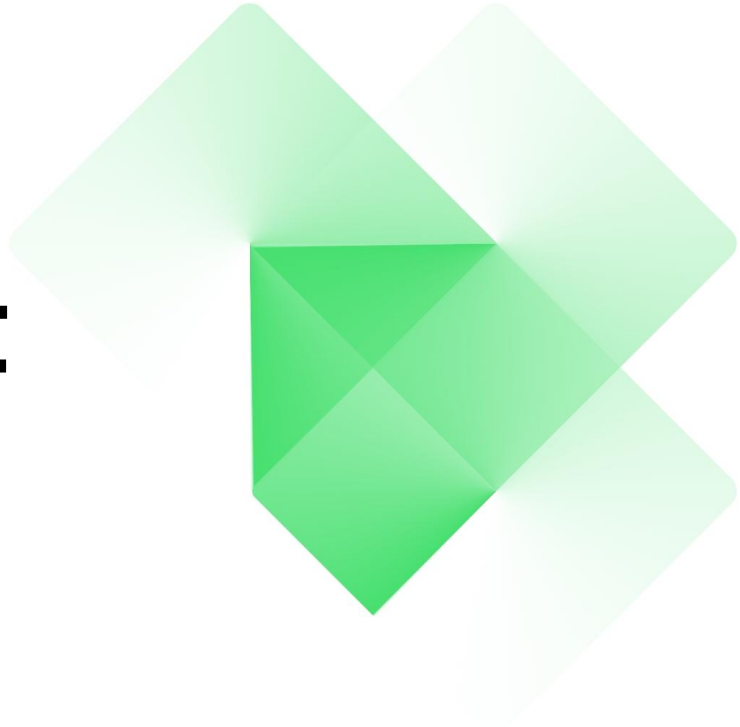
Marlon Baeten

We are web yet

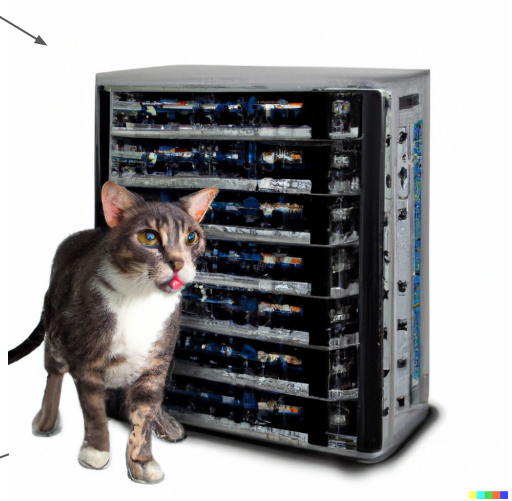


Contents

This is opinionated

What is web?

```
GET /images/cat.jpg HTTP/2
Host: tweedegolf.nl
user-agent: Mozilla/4.0
(compatible; MSIE 5.0)
accept: */*
```



```
HTTP/2 200
content-type: image/jpg
content-length: 44726
<cat.jpg>
```

What is web?

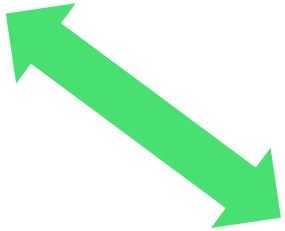
- Typical web-framework features:
  - **Parsing requests & constructing responses**
  - **Routing**
  - Authentication & access control
  - Storage & databases interaction

A bit of (my) history

- Started first company in 2010 together with fellow student Ruben
- We were poor, just like our clients → PHP hosting was the only option
- Let's try ALL the frameworks!

Magic vs. Control

Magic & speed & impossibilities



Control & boilerplate

Magic vs. Control

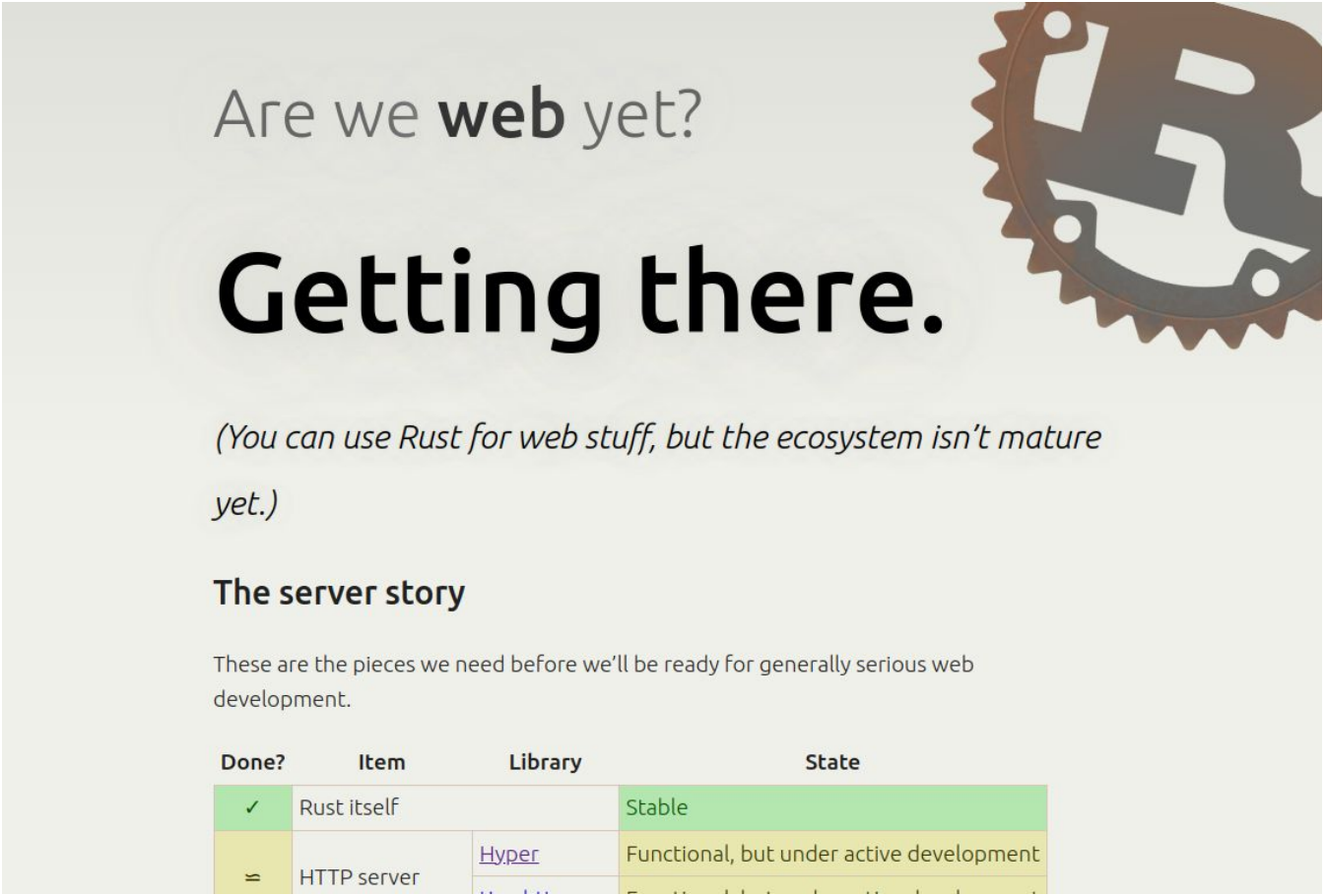
- Either:
  - You are in control
  - The framework is in control
- Search for “SQL on rails”
- Library vs. Framework
- Examples:
  - Express vs. NestJS
  - React vs. Angular
  - Symfony vs. Silex

```php
class PostsController extends AppController {
    public $helpers = array('Html', 'Form', 'Flash');
    public $components = array('Flash');

    public function index() {
        $this->set('posts', $this->Post->find('ALL'));
    }

    public function add() {
        if ($this->request->is('post')) {
            $this->Post->create();
            if ($this->Post->save($this->request->data)) {
                $this->Flash->success(__('Your post has been saved.'));
                return $this->redirect(array('action' => 'index'));
            }
            $this->Flash->error(__('Unable to add your post.'));
        }
    }
}
```

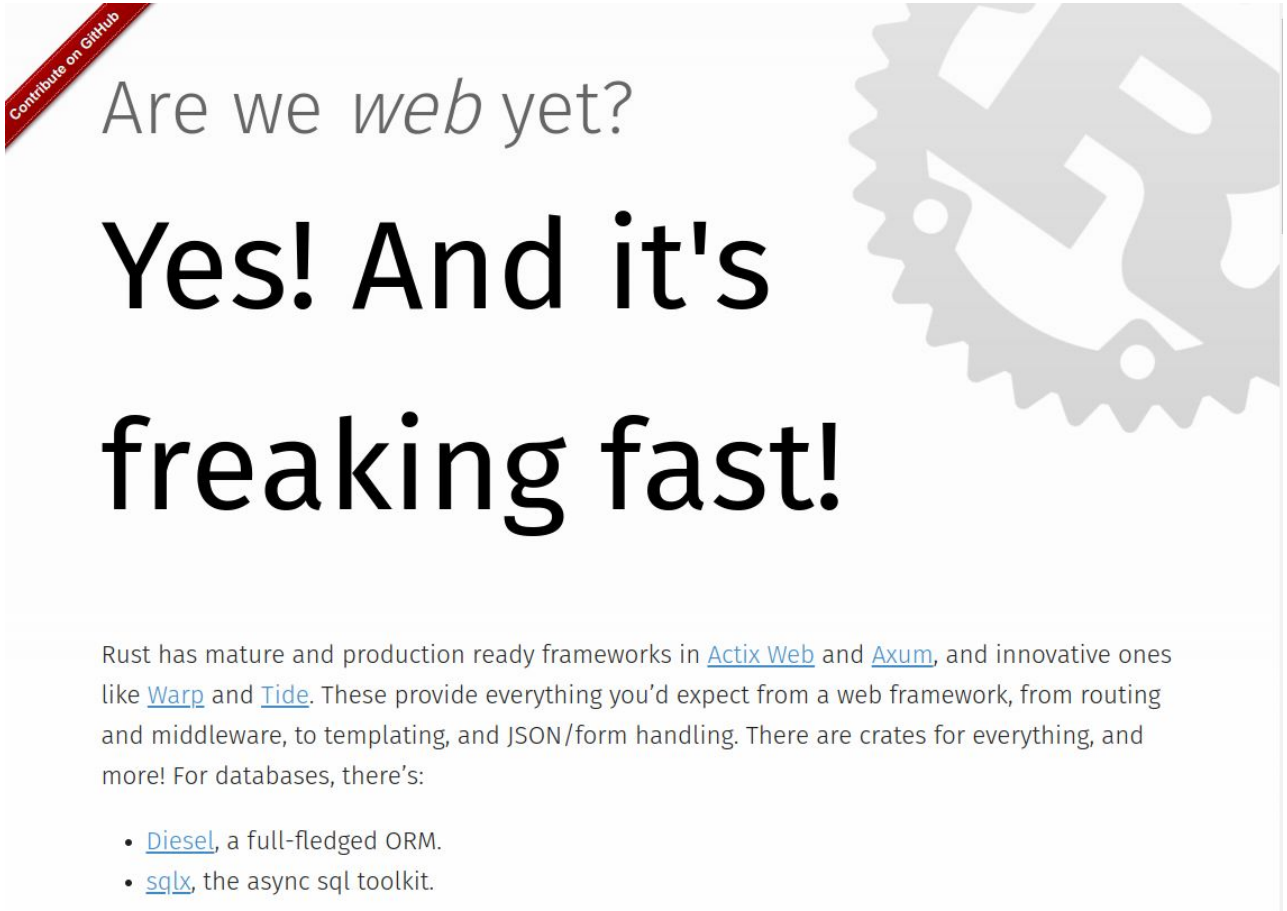
```php
class PostsController extends AppController {
    public $helpers = array('Html', 'Form', 'Flash');
    public $components = array('Flash');

    public function index() {
        $this->set('posts', $this->Post->find('all'));
    }

    public function add() {
        if ($this->request->is('post')) {
            $this->Post->create();
            if ($this->Post->save($this->request->data)) {
                $this->Flash->success(__('Your post has been saved.'));
                return $this->redirect(array('action' => 'index'));
            }
            $this->Flash->error(__('Unable to add your post.'));
        }
    }
}
```

```typescript
@UseGuards(AuthGuard('jwt'))
@Controller('blog')
export class BlogController {
  constructor(private readonly blogService: BlogService) {}

  @Get()
  async index(@Request() request): Promise<Pagination<BlogEntity>> {
    return await this.blogService.paginate({
      limit: request.query.hasOwnProperty('limit') ? request.query.limit : 10,
      page: request.query.hasOwnProperty('page') ? request.query.page : 0,
    });
  }

  @Post()
  async create(
    @Body(new ValidationPipe()) body: BlogModel,
  ): Promise<BlogEntity> {
    const exists = await this.blogService.findBySlug(body.slug);

    if (exists) {
      throw new UnprocessableEntityException();
    }

    return await this.blogService.create(body);
  }
}
```

https://www.arewewebyet.org/ - 2015



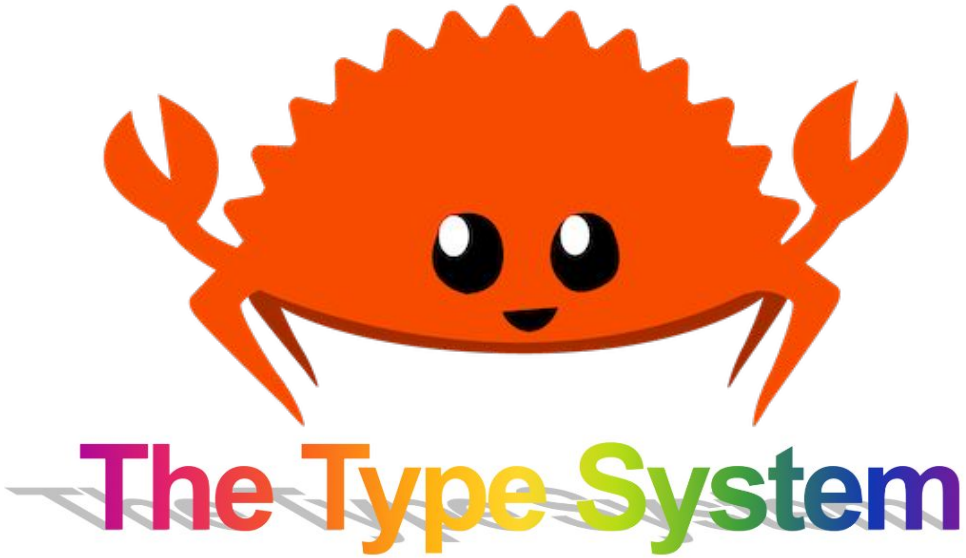
https://www.arewewebyet.org/ - 2023



Why Rust for web?

- The compiler helps you to write better code
- Fast
- Awesome crates like *serde*
- Enums and match statements provide clarity and structure to your code
- More errors during compilation & fewer at runtime

Why Rust for web?



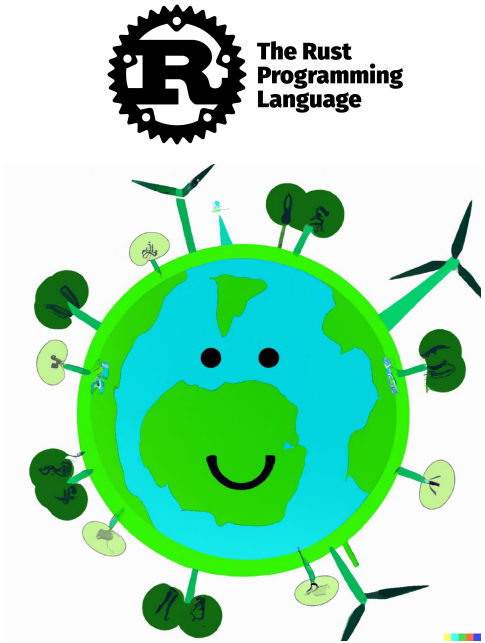
Rust helps you to make the *stringly typed* web more robust with its *strong type system*

```rust
pub async fn list_posts(repo: BlogRepo) -> AppResult<ListPostsTemplate> {
    Ok(ListPostsTemplate {
        posts: repo.posts().list().await?,
    })
}

pub async fn view_post(repo: BlogRepo, Path(id): Path<Uuid>) -> AppResult<ViewPostTemplate> {
    Ok(ViewPostTemplate {
        posts: repo.posts().get(&id).await?,
    })
}
```

```rust
pub async fn list_posts(repo: BlogRepo) -> AppResult<ListPostsTemplate> {
    Ok(ListPostsTemplate {
        posts: repo.posts().list().await?,
    })
}

pub async fn view_post(repo: BlogRepo, Path(id): Path<Uuid>) -> AppResult<ViewPostTemplate> {
    Ok(ViewPostTemplate {
        post: repo.posts().get(&id).await?,
    })
}
```

```
error[E0560]: struct `ViewPostTemplate` has no field named `posts`
  --> src/ssr/handlers.rs:21:9
   |
21 |         posts: repo.posts().get(&id).await?,
   |         ^^^^^ help: a field with a similar name exists: `post`

For more information about this error, try `rustc --explain E0560`.
```

Why is Rust for web?



	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

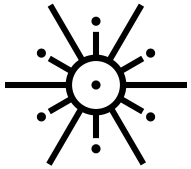
* Energy efficiency across programming languages: how do energy, time, and memory relate? In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2017).

What can I use?

ACTIX

Rocket

warp



Axum

* these are the 4 most used Rust web frameworks

Common concepts: FromRequest / IntoResponse

```rust
#[async_trait]
impl<S> FromRequestParts<S> for AuthenticatedUser
where
    AppState: FromRef<S>,
    S: Send + Sync,
{
    type Rejection = AuthenticationError;

    async fn from_request_parts(parts: &mut Parts, state: &S) -> Result<Self, Self::Rejection> {
        let state = AppState::from_ref(state);

        let jar = parts
            .extract_with_state::<SignedCookieJar, AppState>(&state)
            .await?;

        let session_cookie = jar.get(SESSION_COOKIE_NAME)?;

        let user: AuthenticatedUser = serde_json::from_str(session_cookie.value())?;

        Ok(user)
    }
}
```

What can I use?

warp

- Since 2018 - 8.2k stars
- Uses **Filters**
- Elegant - but also a bit esoteric

```rust
[tokio::main]
async fn main() {
    pretty_env_logger::init();

    // POST /employees/:rate  {"name":"Sean","rate":2}
    let promote = warp::post()
        .and(warp::path("employees"))
        .and(warp::path::param::<u32>())
        // Only accept bodies smaller than 16kb...
        .and(warp::body::content_length_limit(1024 * 16))
        .and(warp::body::json())
        .map(|rate, mut employee: Employee| {
            employee.rate = rate;
            warp::reply::json(&employee)
        });

    warp::serve(promote).run(([127, 0, 0, 1], 3030)).await
}
```

What can I use?

ACTIX

- Since 2017 - 17.8k stars
- Uses actor framework
- Stable & feature rich
- Nice documentation
- Has (had) problems integrating with tokio powered crates
- A bit more verbose than alternatives

```rust
use actix_web::{get, web, App, HttpServer, Responder};

#[get("/hello/{name}")]
async fn greet(name: web::Path<String>) -> impl Responder {
    format!("Hello {}!", name)
}

#[actix_web::main] // or #[tokio::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new().service(greet)
    })
    .bind(("127.0.0.1", 8080))?
    .run()
    .await
}
```

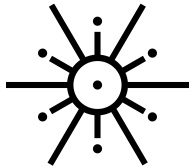
What can I use?

Rocket

- Since 2016 - 20.9k stars
- Easy to understand
- Great documentation
- Macro-heavy
- Used a lot @tweedegolf
- Bus-factor
- Almost waiting 5 years for the 0.5 version

```rust
#[macro_use] extern crate rocket;

#[get("/<name>/<age>")]
fn hello(name: &str, age: u8) -> String {
    format!("Hello, {} year old named {}!", age, name)
}

#[launch]
fn rocket() -> _ {
    rocket::build().mount("/hello", routes![hello])
}
```

What can I use?



Axum

- Since 2021 - 10.7k stars
- Part of the tokio ecosystem
- Uses tokio’s tower middleware
- Macro-free
- No fancy documentation, but great examples
- Clear and concise
- Simplicity: less glue code necessary!

```rust
#[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("127.0.0.1:3000")
        .await
        .unwrap();

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

async fn handler() -> Html<&'static str> {
    Html("<h1>Hello, World!</h1>")
}
```

```rust
async fn view_author(
    repo: AuthorRepository<PgPool>,
    user: LibraryUser,
    Path(id): Path<Id>,
    context: Context,
) -> Result<Index> {
    let author = repo.get_one(id).await?;

    Ok(Index {
        id,
        author,
        context,
    })
}
```

What can I use?

What can I use?

- You want great documentation and familiar concepts: **Rocket**
- You want stability and features: **Actix**
- You are a conceptual purist: **Warp**
- You want ergonomics and modularity without the magic: **Axum**

Database abstractions

- ORM's make it easy to write standard crud stuff:

```php
$dates = $booking->getDate()->getProduct()->getDates();
```

- SQL is the the powerful and flexible tool: it's already a DSL!
- **sqlx** is awesome!

```rust
sqlx::query!(
    "UPDATE posts SET title = $2, content = $3, category = $4 WHERE id = $1;",
    post.title,
    post.content,
    post.category,
    id
)
.execute(&self.pool)
.await?;
```

```rust
sqlx::query!(
    "UPDATE posts SET title = $2, content = $3, category = $4 WHERE id = $1;",
    post.title,
    post.content,
    post.category,
    id
)
.execute(&self.pool)
.await?;
```

```
error[E0308]: mismatched types
   --> src/repository.rs:117:13
    |
117 |             post.title,
    |             ^^^^
    |             |
    |             expected `Uuid`, found `String`
    |             expected due to the type of this binding

error[E0308]: mismatched types
   --> src/repository.rs:120:13
    |
120 |             id
    |             ^^
    |             |
    |             expected `&str`, found `&Uuid`
    |             expected due to the type of this binding
    |
    = note: expected reference `&str`
               found reference `&Uuid`
```

Future

- Hope for more Axum adoption
- Hope for Rocket 0.5
- Hope for even more Rust adoption

Questions?

Getting in touch

Contact Marlon or checkout Tweede golf on https://github.com/tweedegolf or https://tweedegolf.nl

Marlon Baeten
marlon@tweedegolf.nl