Actix-Web

actix-web is a framework for writting web applications and REST APIs.

Installation

```
# To install:
cargo add actix-web

# Recommended for validation
cargo add validator actix-web-validator
```

Simple example

```
#[get("/hello")
async fn hello() -> Response {
  HttpResponse::Ok().body("Hello!")
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(move || {
        App::new()
            .app_data(here_you_can_add_dependencies)
            .service(hello)
    })
    .bind(("0.0.0.0", 8080))?
    .run()
    .await
}
```

JSON request bodies

To parse a JSON body from a request, you define a struct, which represents the body, and then use the `web::Json` extractor.

```
#[derive(Deserialize)]
struct HelloRequest {
  name: String,
}

#[post("/hello")
async fn hello(payload: web::Json<HelloRequest>) -> Response {
  let name = payload.name;
  HttpResponse::Ok().body(format!("Hello, {name}!"))
}
```

You can also use the actix-web-validator package to create a validation. The request must then match your rules:

```
#[derive(Deserialize, Validate)]
struct HelloRequest {
  #[validate(length(min = 1, max  = 32)]
  name: String,
}

#[post("/hello")
async fn hello(payload: actix_web_validator::Json<HelloRequest>) -> Response {
  let name = payload.name;
  HttpResponse::Ok().body(format!("Hello, {name}!"))
}
```

Scopes

You can group multiple services into a scope, to give them the same prefix.

```
let v1 = web::scope("/api/v1")
  .service(list_entities)
  .service(create_entity)
  .service(delete_entity);
```

```
App::new()
  .service(v1)
```

Custom extractors

Extractors are used in the parameter lists of your service functions. You can write own to **extract information from a request** (e.g. get a user by authorization header). To write such a custom header, you need to create a struct (which will be used in the parameter list and contain the extracted information) and then implmenet the FromRequest trait:

```
struct MyExtractor {
  ...
}

impl FromRequest for MyExtractor {
    type Error = actix_web::Error;
    type Future = future_utils::LocalBoxFuture<'static, Result<Self, Self::Error>>;

    fn from_request(req: &HttpRequest, payload: &mut Payload) -> Self::Future {
        let req = req.clone();

        Box::pin(async move {
            // Do extraction...

            Ok(MyExtractor { ... })
        })
    }
}
```

To access app_data, you can use the request:

```
let pool = match req.app_data::<web::Data<Pool<Postgres>>>() {
    Some(pool) => pool,
    None => panic!(),
};
```

Revision #4
Created 2026-03-07 20:55:22 UTC by Matthias
Updated 2026-03-07 21:28:29 UTC by Matthias