
license MIT build passing pull-request open

Canduma rust GraphQL

A Rust authentication server with GraphQL API, Diesel, PostgreSQL session authentication and JWT

This repository contains a GraphQL server with JWT up and running quickly.

It uses actix-web, Juniper, Diesel and jsonwebtoken

Your own pull requests are welcome!

Benchmarks with insert into PostgreSQL

“shell script ▶ `./bombardier -c 125 -n 10000000 http://localhost:3000/graphql -k -f body -method=POST -H "Content-Type: application/json" -s Bombarding http://localhost:3000/graphql` with 10000000 request(s) using 125 connection(s)

10000000 / 10000000 [=====]
100.00% 28777/s 5m47s Done! Statistics Avg Stdev Max Reqs/sec 28788.66 2183.47 34605.95 Latency 4.32ms 543.07us 110.95ms HTTP codes: 1xx - 0, 2xx - 10000000, 3xx - 0, 4xx - 0, 5xx - 0 others - 0
Throughput: 20.75MB/s

```
1
2  ## Collection of major crates used in Canduma
3
4  - actix - [link](https://actix.rs/)
5  - actix-web - [link](https://docs.rs/actix-web/)
6  - diesel - [link](http://diesel.rs/)
7  - juniper - [link](https://graphql-rust.github.io/juniper/current/)
8  - chrono - [link](https://docs.rs/chrono/)
9  - serde_json - [link](https://docs.rs/serde_json/)
10 - argon2rs - [link](https://github.com/bryant/argon2rs)
11 - jsonwebtoken - [link](https://docs.rs/jsonwebtoken)
12 - anyhow - [link](https://github.com/dtolnay/anyhow)
13 - thiserror - [link](https://github.com/dtolnay/thiserror)
14 - shrinkwraps - [link](https://docs.rs/shrinkwraps/)
15
16 ## Required
17
18 - [Rustup](https://rustup.rs/)
19 - Stable Toolchain: `rustup default stable`
20 - Diesel cli with postgres `cargo install diesel_cli --no-default-features --features "postgres"`
```

```

21 - PostgreSQL database server or use our docker-compose.yml (require
    docker)
22
23 ## Getting Started
24
25 ```sh
26 git clone https://github.com/clifinger/canduma.git
27 cd canduma
28 docker-compose up
29 cp .env.example .env
30 diesel setup --database-url='postgres://postgres:canduma@localhost/
    canduma'
31 diesel migration run
32 cargo run

```

Test the GraphQL API with Insomnia

Register

The screenshot shows the Insomnia client interface. The top bar indicates a successful POST request to `http://localhost:3000/user/register` with a status of **200 OK**, a response time of **531 ms**, and a size of **28 B**. The left sidebar lists the request as **POST register**. The main panel shows the JSON body of the request: `{ "name": "my name", "email": "name3@domain.com", "password": "password" }`. The right sidebar shows the JSON preview of the response: `{ "email": "name3@domain.com" }`.

Login

The screenshot shows the Insomnia client interface for a login request. The top bar indicates a successful POST request to `http://localhost:3000/user/login` with a status of **200 OK**, a response time of **527 ms**, and a size of **0 B**. The left sidebar lists the request as **POST login**. The main panel shows the JSON body of the request: `{ "email": "name3@domain.com", "password": "password" }`. The right sidebar shows the **Cookie** tab with a single cookie named `auth` having a value of `GrS0rdCcmSAoV8azsp81dskgrBCzvtprgRzzbs1g1/wiNAc9sDQicM3p1YtPEtX/1wUgRtySUU=`. A **Manage Cookies** button is visible at the bottom right of the cookie list.

Test authentication with session in GraphQL by getting all users (for tests purpose)

The screenshot shows the Insomnia GraphQL interface. The left sidebar lists several API endpoints: POST register, POST login, GET me, POST GraphQL (selected), and GET logout. The main panel displays a GraphQL query named 'usersQuery' that fetches a list of users with fields: name, userUuid, email, and createdAt. The response is shown in the 'Preview' tab, indicating a 200 OK status with a response time of 4.71 ms and a size of 392 B. The JSON response contains an array of three user objects, each with their respective details.

```
1 query usersQuery {
2   users {
3     name
4     userUuid
5     email
6     createdAt
7   }
8 }
9
10 query tokenQuery {
11   token {
12     bearer
13   }
14 }
15
16 query decodeTokenQuery {
17   decode {
18     email
19     iss
20     iat
21     exp
22     sub
23   }
24 }
```

```
1 {
2   "data": {
3     "users": [
4       {
5         "name": "julien Lenne",
6         "userUuid": "be8f2b49-1216-4c80-86f6-dfed4a6f5119",
7         "email": "name@domain.com",
8         "createdAt": 1572877098.0
9       },
10      {
11        "name": "julien Lenne",
12        "userUuid": "2df40f17-e320-4954-94db-e6e2020a283d",
13        "email": "name2@domain.com",
14        "createdAt": 1572884245.0
15      },
16      {
17        "name": "my name",
18        "userUuid": "535756fc-1254-4e5e-9481-f3dd5bf2d3c2",
19        "email": "name3@domain.com",
20        "createdAt": 1572972128.0
21      }
22    ]
23  }
24 }
```

Logout

The screenshot shows the Insomnia GraphQL interface for a GET request to 'http://localhost:3000/user/logout'. The 'Header' tab is active, showing a 'New Header' section with 'NAME' and 'VALUE' columns. The 'Preview' tab shows the response headers: 'content-length' is 0, 'set-cookie' is 'auth=; HttpOnly; Path=/; Domain=localhost; Max-Age=0; Expires=Mon, 05 Nov 2018 09:02:57 GMT', and 'date' is 'Tue, 05 Nov 2019 09:02:57 GMT'. A 'Copy to Clipboard' button is visible at the bottom right.

NAME	VALUE
content-length	0
set-cookie	auth=; HttpOnly; Path=/; Domain=localhost; Max-Age=0; Expires=Mon, 05 Nov 2018 09:02:57 GMT
date	Tue, 05 Nov 2019 09:02:57 GMT

Raw code for Insomnia

```
1 ##### GraphQL Queries #####
2 query usersQuery {
3   users {
4     name
5     userUuid
6     email
7     createdAt
8   }
9 }
10
11 query tokenQuery {
```

```
12  token {
13    bearer
14  }
15 }
16
17 query decodeTokenQuery {
18   decode {
19     email
20     iss
21     iat
22     exp
23     sub
24   }
25 }
```

Test the GraphQL API with VScode REST Client

VScode plugin

See / open TEST.http file in vscode.

Build release

```
1 cargo build --release
2 cd target/release
3 ./canduma
```

Security

Important security considerations

We use session cookies for authentication.

Why not JWT authentication?

Stop Using JWT for sessions and why your solution doesn't work

The use of JWT remains secure only if you use adequate storage. This boilerplate is built for use in a micro-services architecture.

JWT can be use for representing claims to be transferred between two parties.

The private key should only be on this micro-service. public key can be used on all other parties to decode the token.

This boilerplate provides a complete example, so we included JWT also.

Generate RSA keys for JWT

In development mode you can keep the one in `/keys` folder.

“shell script // private key \$ openssl genrsa -out rs256-4096-private.rsa 4096

// public key \$ openssl rsa -in rs256-4096-private.rsa -pubout > rs256-4096-public.pem

```
1
2  ### Logging
3
4  Logging controlled by middleware::Logger [actix.rs](https://actix.rs/docs/errors/)
5
6  To enable debug logging set `RUST_LOG=debug` in `.env`
7
8  ### Testing
9
10 ##### Initialization
11
12 First run `yarn` or `npm install` to get all required packages
13
14 ##### npm run test
15
16 To run you can use `npm run test` or `yarn test`.
17
18 The testing system designed to automatically build `canduma` offline
19   and start in `tests/jest.beforeall.js`
20 We starting `canduma` in order to capture output from both rust and js
21   code using `testci` target
22
23 ##### npm run testci
24
25 ```bash
26 $ npm run testci
27
28 > canduma@ testci /home/olexiyb/b100pro/canduma
29 > cross-env RUST_LOG=debug DEBUG=canduma:* NODE_ENV=test jest
30
31 Determining test suites to run...
32 $ killall canduma
33 canduma: no process found
34
35 $ cargo build
36   Finished dev [unoptimized + debuginfo] target(s) in 0.07s
37 canduma:jest.beforeall.js build = { status: 0, signal: null, output:
38   [ null, null, null ], pid: 2447, stdout: null, stderr: null } +0ms
39
40 $ target/debug/canduma
41 [2020-04-02T18:17:19Z INFO actix_server::builder] Starting 24 workers
42 [2020-04-02T18:17:19Z INFO actix_server::builder] Starting server on
```

```

0.0.0.0:4000
40 Listening on 0.0.0.0:4000
41 started API
42
43   canduma:user.test.js /user/me body='Unauthorized' text="Unauthorized"
    +0ms
44
45 ...
46 [2020-04-02T18:17:22Z DEBUG canduma::user::handler] user_string={
    user_uuid:"f7cfa71e-096e-44d0-ae4f-7d16dd9e4baf","email":
    email1@nowhere.com","role":"bad_role"}
47   canduma:user.test.js /graphql body={ data: null, errors: [ { message:
    'Unauthorized', locations: [Array], path: [Array], extensions: [
    Object] } ] } +292ms
48 PASS   tests/user.test.js
49
50 ...

```

In example above you see output from jest tests as well as from rust code `debug!` (`"user_string={}"`, `user_string`);

CLion I also highly recommend to use CLion as a dev tool. It allows to run all tests or individual with single click and analyze logs

