Kapitel 1

RESTinio

RESTinio ist eine header-only Bibliothek, mit dem man REST - APIs schnell und einfach entwickeln kann und ebenso um Requests asynchron verarbeiten zu können.

Um RESTinio verwenden zu können, muss man neben RESTinio auch die Bibliotheken asio, http_parser und fmt inkludieren, da RESTinio diese verwendet.

1.1 Express

Mithilfe dieser Bibliothek kann man Server wie node.js Express Server entwickeln.

1.1.1 main

Um einen Express Server zu starten, muss die main Funktion wie folgt aussehen:

```
1 int main() {
2     using router_t = restinio::router::express_router_t<>;
3     try {
4         using traits_t =
5             restinio::traits_t<
6                 asio_timer_manager_t,
7                 single_threaded_ostream_logger_t,
8                 router_t >;
9 
10        restinio::run(
11            restinio::on_this_thread<traits_t>()
12            .port(8080)
13            .address("localhost")
14            .request_handler(server_handler()));
15    } catch( const std::exception & ex ) {
16        cerr << "Error: " << ex.what() << endl;
17        return 1;
18    }
19 }
```

In main muss die Funktion **restinio::run** aufgerufen werden, welche die Funktion **restinio::on_this_thread** erwartet. Diese Funktion erwartet zumindest 3 Parameter: den Port (8080), die Adresse (localhost) und den Request Handler, welcher alle eingehenden Request verarbeitet (server_handler()).

Bei Express Server benötigt man zusätzlich noch Traits, das wichtigste Trait wäre dabei **express_router_t**, mit diesem gibt man an, dass es sich um einen Express Router handelt.

1.1.2 server_handler()

Damit der Server weiß, welche Request man wie verarbeiten muss, definiert man einen Request Handler, dieser könnte wie folgt aussehen:

```
1 auto server_handler() {
2     auto router = std::make_unique< router_t >();
3
4     //GET request for route /
5     router->http_get("/", [](auto req, auto) {
6         return req->create_response()
7             .set_body("Hello World")
8             .done();
9     });
10
11    //will be called, if route doesn't match exisitng ones
12    router->non_matched_request_handler(
13        []( auto req ){
14            return
15                req->create_response(status_not_found())
16                    .append_header_date_field()
17                    .connection_close()
18                    .done();
19        });
20    return router;
21 }
```

In dieser Funktion muss man eine Variable **router** definieren, mit dieser kann man weitere Handler definieren. Diese Handler bearbeiten den jeweiligen Request und schicken einen entsprechenden Response zurück.

In diesem Beispiel wurde nur ein Handler für einen GET Request auf / definiert. Wenn dieser aufgerufen wird, dann wird ein Response mithilfe der Funktion **create_response()** erstellt, der Body dieses Response wird auf "Hello World"gesetzt und anschließend wird dieser an den Client gesendet.

Falls aber ein Request auf eine undefinierte Route erstellt wird, werden diese in der Funktion **non_matched_request_handler** bearbeitet. In dem Beispiel wird einfach nur der Status-Code "404 Not Found"gesetzt und die Verbindung geschlossen.

1.1.3 Header und Query Parameter

Bei Requests gibt es 2 Arten von Parametern: Header und Query Parameter.

Header Parameter werden im Request Header mitgeschickt. Mit den Methoden get_field(string headername) und has_field(string headername) kann man auf seinen Wert zugreifen und man kann überprüfen, ob dieser existiert.

Query Parameter werden in der URL mitgeschickt. Diese werden mit einem Doppelpunkt nach einem / gekennzeichnet und man kann auf sie mit dem Funktionsparameter params["parametername"] zugreifen.

Im folgenden Beispiel werden beide Arten von Parametern verwendet. Wenn der User über einen Authorization Header verfügt, sendet der Server einen Request mit seinem Wert und mit dem Parameter Wert zurück, wenn nicht, sendet der Server den Status-Code "401 Unauthorizedßurück:

```
1 router->http_get("/single/:name", [](auto req, auto params ){
2     if (req->header().has_field("Authorization")) {
3         string auth{req->header().get_field("Authorization")};
4 
5         return init_resp(req->create_response())
6                .set_body(auth + "," + params["name"])
7                .done();
8     } else {
9          return req->create_response(status_unauthorized()).done();
10    }
11 });
```

1.2 Senden von Dateien

1.2.1 Client zu Server

1.2.2 Server zu Client

1.3 TLS