11. Cookie w/ Session and Redis

#express #express-session #redis #cookie #backend

How does it work

- Redis is an in-memory key-value data store.
- When `req.session.userId = user.id` is executed, { userId : 1 } is sent to Redis, and a key is defined, which could look something like this:

  sessxasdljhsaflieghegin en which maps to { userId: 1 }
- Then **express-session will set a cookie on the browser**: q986hfqkfbjqwl8763487355839
- When user makes a request this cookie q986hfqkfbjqwl8763487355839 is sent to server
- Server decrypts the cookie using the secret defined in session configuration in index.ts, to obtain the redis key

  q986hfqkfbjqwl8763487355839 ---decrypt--> sessxasdljhsaflieghegin en
- Server makes a request to redis and looks up for the value matching the key sessxasdljhsaflieghegin en

  sessxasdljhsaflieghegin en maps to { userId: 1 }

Install packages

```
yarn add express-session redis connect-redis ioredis
yarn add -D @types/express-session @types/redis @types/connect-redis
@types/ioredis
```

- We will use express-session to keep the user logged in. This stores data (cookie) about user on the server.
- We will store this data in redis, which is a very lightweight and fast in-memory database
- github.com/expressjs/session lists other ways this data can be stored (postgreSQL, MongoDB, etc...)
- We install ioredis library and import Redis from there instead of "redis", since "redis" is crap, does not have Promises built into it etc. (at the time of this tutorial)

Setup session with redis and update apollo-server context

We start by updating index.ts amd setting up redis store and using apollo-server to make the session available in the resolvers via context, by passing the {req, res} provided by express into the context:

- session is accessed via req as req.session

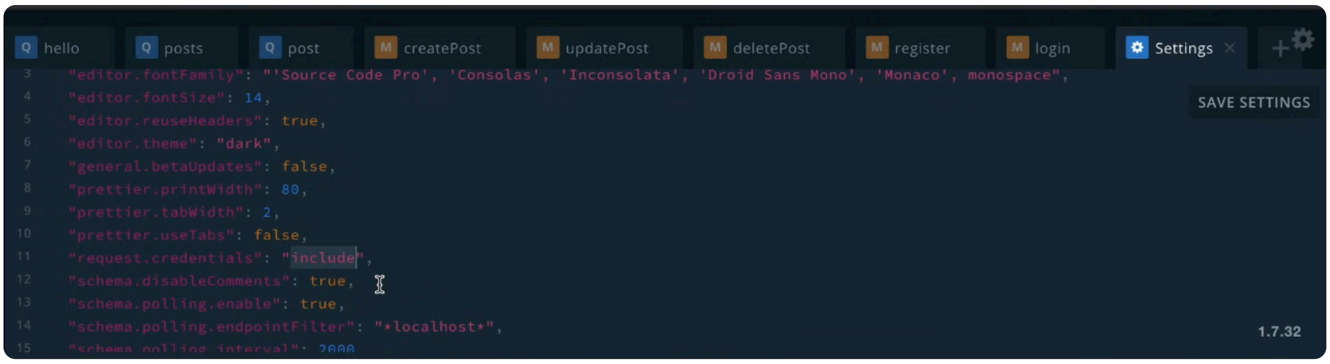
- `req.session.userId = user.id` logs in the user (by sending cookie to browser). It is implemented in register and login queries
- An extended type is defined in types.ts (5) to add session.userId to req
- https://expressjs.com/en/api.html#req
- https://expressjs.com/en/api.html#res
- https://stackoverflow.com/questions/4696283/what-are-res-and-req-parameters-in-express-functions

index.ts

```
import "reflect-metadata";
import { MikroORM } from "@mikro-orm/core";
import { COOKIE_NAME, __prod__ } from "./constants";
import microConfig from "./mikro-orm.config";
import express from "express";
import { ApolloServer } from "apollo-server-express";
import { buildSchema } from "type-graphql";
import { HelloResolver } from "./resolvers/hello";
import { PostResolver } from "./resolvers/post";
import { UserResolver } from "./resolvers/user";
import connectRedis from "connect-redis";
import session from "express-session";
import Redis from "ioredis";
import { MyContext } from "./types";

const main = async () => {
  const orm = await MikroORM.init(microConfig);
  await orm.getMigrator().up(); // run migration

  const app = express();

  const RedisStore = connectRedis(session);
  const redis = new Redis();

  // Initialize session storage before Apollo since it will be used from inside
Apollo.
  app.use(
    session({
      name: COOKIE_NAME,
      store: new RedisStore({
        client: redis,
        disableTTL: true, // keep session alive forever
        disableTouch: true, // disable TTL reset at every touch
      }),
      cookie: {
        maxAge: 1000 * 60 * 60 * 24 * 365 * 10, // 10 years
        httpOnly: true, // prevent accessing the cookie in the JS code in the frontend
        sameSite: "lax", // csrf
        secure: __prod__, // cookie only works in https
      },
      saveUninitialized: false,
      secret: "asdfasdfasdf", // used to sign cookie - should actually be hidden in an env variable
      resave: false,
    })
  );

  const apolloServer = new ApolloServer({
    schema: await buildSchema({
      resolvers: [HelloResolver, PostResolver, UserResolver],
      validate: false,
    }),
    context: ({ req, res }: MyContext) => ({ em: orm.em, req, res, redis }), // context is shared with all resolvers
  });

  apolloServer.applyMiddleware({
    app,
  });

  app.listen(4000, () => {
    console.log("server started on localhost:4000");
  });
};

main().catch((err) => {
  console.log(err);
});
```

Change GraphQL config

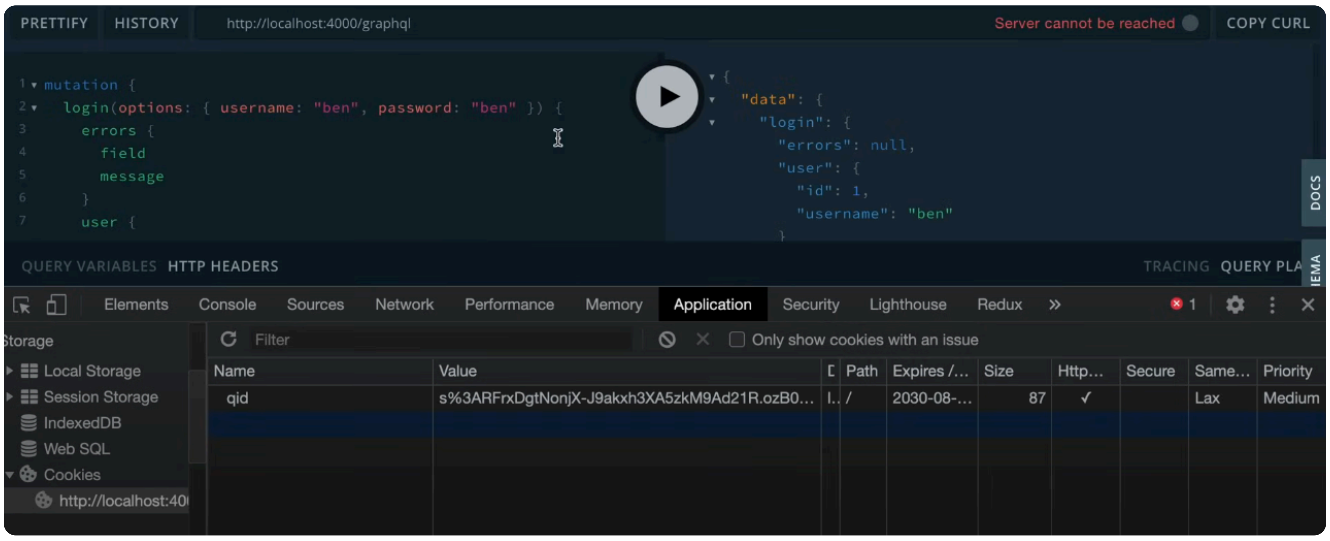
- In GraphQL, change the request.credentials setting from “ommit” to “include”



```
  "editor.fontFamily": "'Source Code Pro', 'Consolas', 'Inconsolata', 'Droid Sans Mono', 'Monaco', monospace",
  "editor.fontSize": 14,
  "editor.reuseHeaders": true,
  "editor.theme": "dark",
  "general.betaUpdates": false,
  "prettier.printWidth": 80,
  "prettier.tabWidth": 2,
  "prettier.useTabs": false,
  "request.credentials": "include",
  "schema.disableComments": true,
  "schema.polling.enable": true,
  "schema.polling.endpointFilter": "*localhost*",
  "schema.polling.interval": 2000
```

Result

And now when you register or log in you should see the cookie named “qid” placed in the browser:



```
mutation {
  login(options: { username: "ben", password: "ben" }) {
    errors {
      field
      message
    }
    user {
```

```
{
  "data": {
    "login": {
      "errors": null,
      "user": {
        "id": 1,
        "username": "ben"
      }
```

Name	Value	D	Path	Expires /...	Size	Http...	Secure	Same...	Priority
qid	s%3ARFrxDgtNonjX-J9akxh3XA5zkM9Ad21R.ozB0...	l...	/	2030-08-...	87	✓		Lax	Medium