Back-end architecture picture explained v. 2024-02-13 by JV

Front-end (1. from the development-time Node server OR 2. deployed build folder version)

- Front-end consists of the **Web Browser** program, its JavaScript runtime environment including the xhr object or fetch for sending and receiving AJAX requests/responses.
- … and the needed HTML, CSS and **JavaScript** (React, Redux, Axios, Material UI components defined as JS) that the browser gets from development-time Node server OR just from even static web server in the production version.

(Internet between Front-end and Back-end)

- At the moment our requests go unprotected / clear text with http.
- Later we should configure https

Back-end

1. Back-end consists of the Node.js program, its V8 JavaScript runtime environment and e.g. request handling and response writing modules.

   It runs in one thread and switches between handling possible several request going on. E.g. first request needs something from database. While waiting for the database operation to respond the Node.js might give another request code turn to start. And returns to the first request when "then" signaled and second thread has stopped at some await/promise returning function call.

2. Express is a wrapper around Node.js. So we write Express code ("we use Express) and Express uses internally Node.js. We use Express to:
   a. configure the Node.js server with the app.use() calls
   b. configure REST API service routing with the app.use() and routes.use() calls
   c. start the server with the app.listen() call
   d. define the end points with app.get() (And post, delete, put)

3. Knex is a query builder. We use Knex:
   a. to define and initiate database operations
   b. to catch database return value and go to either success or failure branch of the code´

4. Knex uses the 'mysql' or 'mariadb' module (The mariadb compatible database driver module)
   a. to take connection to the database
   b. to translate any SQL to the mysql/mariadb dialect
   c. to catch any mysql/mariadb success code

Back-end activities

Server setup and configuration:

a. CORS configuration
b. Logger creation/configuration
c. Configuring e.g. middleware for parsing JSON data <-> JS objects
d. DB access creation
e. Routing configuration
f. Port configuration
g. Server start

Routing configuration:

a. .ENV file route prefix "/api"
b. /src/index.ts to start the definitions
c. /src/routes/index.ts to list all subroutes and the files where more routing defined
d. /src/routes/subject.ts all subject related routes defined

Validation Middleware

Authentication and Authorization Middleware

Response handling Helper functions

Logging

SSH Tunneling, aca. "Secure/SSH port forwarding".

- Connection secured with the SSH protocol
- Tunnel creation is possible as database server has the port :22 open and sshd server process listening/serving at it
- Tunnel dragged **from** the remote port 3306
- To our local computer's port 3308 (in some examples 3307 is used here, even 3306)
- Thus for our back-end server it feels like it's using localhost:3308 (when actually forwarded to remote server port 3306)
- Note that the **tunnel will disappear** when the process not running anymore! Need to setup it each time.

Database server

- The DBMS MariaDB provides
    - DB user management and access management
    - Concurrency isolation
    - metadata
    - data
- We are at the moment using for the backend services a user account that has full access to the schema! (not to the whole database server, but to the schema = sandbox there). In that schema the current user is able to create tables, drop table, create, drop, and execute stored procedures and functions.
- When we would move the database away from IT services managed mariadb.haaga-helia.fi we could define less powerful database user to be used by the backend. With only those rights that the backend needs. And with authorization based on the provided logged in (real app) user.

Security needs to be implemented / means changes everywhere!

Security must be implemented in

- Front-end
- between Front and Back, HTTPS should be used
- Back-end (e.g. service level authorization based on the logged in user and his/her roles. And API key to allow to use the service to start with)
- between Back and Database (Though we already have the SSH tunnel there)
- Database