Backend code demo project

idea-case-backend (Not model for all parts, but contains several good ideas)

5.2.2023

There are max 12 files you need to look into

- **README.md**
- **.gitignore**
- **.env** (You need to create this based on given info. It’s gitignored file, not in repo)
- **package.json**
- **src/index.js**
- **src/routes/index.js**
- **src/routes/category.js**
- **src/db/index.js**
- **src/responseHandlers/index.js**
- **src/utils/logger.js**
- **logs/backendLog.log & logs/errorLog.log**

README.md

- *How to setup/install/clone/run the project.*
- *Possibly introduce the libraries and give links to information about each library*

.gitignore

- *file that lists what files and folders will not be tracked nor added to the git repository*
- *e.g. we do not want to put the **.env** file anywhere in public place, it might contain secrets, like ip addresses, usernames, passwords…*
    - *Thus you will also not get it with git clone.*
        - *Thus you will manually add that file to your project*
- *if you forget to enter the file / folder in the gitignore the non-ignored file will be in the version history, and could be exposing secrets in remote repo, like github!*

.env file

- *The* ***.env file (1.)*** *that the dotenv library (see in the package.json (2.), added there with* ***npm install dotenv****) after you have run* **dotenv.config({}); (3.)** *reads from file system and offers as 'environment variables' to other modules with this kind of code* ***process.env.VARIABLE_NAME_HERE (4.)***
- *Here we can keep the settings we want to be able to change without changing the tested code*
  - *e.g. IP address of the database server*
  - *database schema, database connection pool optimization settings*
  - *ports*
  - *or usernames and passwords*
- *per developer file, thus could also contain developer's preference, like ip to own version of database*
  - *Some organizations divide .env files into .env.local (not to repo) and .env (could then go to repo)*

package.json

- *Created when the Node project was created, e.g. with the **npm init** command*
- *Locate e.g. the '**npm start**' script and see how it will run '**node src/index.js**' / '**nodemon src/index.js**' or similar.*
- *See also the dependencies and development dependencies*
  - *New libraries can be added there like this: **npm install dot_env** / **npm i dot_env***
  - *When you have cloned a project, the dependencies mentioned in package.json can and must be installed with the **npm install** command. Also if some other developer has added a new module.*
- *package.json file you want to keep in git, as it's important file that makes a Node/npm project to be a project and also e.g. keeps track of the libraries*

- *(…whereas node_modules folder and package-lock.json file can always be removed if needed for fixing some probs. **npm install** will download and generate those again)*

src/index.js

- *This is the starting point of the Node backend.*
- *A file that setups up and configures the Node project.*
- *Here also the middleware functions are attached to the request processing loop*
  - *E.g. express-router is added here.*
  - *As well as the express.json for from HTTP request body json text to JavaScript object porting*
- *Things here are run only once, setting up things at the server startup*
- *Try to see where the all the API routing starts!*

src/routes/api/index.js

- *Routing continues here*
- *Just routing the endpoint calls to specific routing files, based on* ***part*** *of the URI pattern.*
- *index.js is a shared file, add your own entries to common files fast and deliver them to others also fast to avoid merge trouble.*

src/routes/api/category.js

- *An example how the category business object related REST API endpoints are gathered in one file*
  - *Other options could be role based bundling of the endpoints*
- *Now there are about nine endpoints, and file length ~260 lines*
  - *Thus maybe could be divided into category.js and categoryMultiple.js (or something like that),*
    - *first one with endpoints adding, updating, finding, deleting **single** categories.*
    - *the other one having all the endpoints related to **multiple** category objects.*
- *Here only look at the five endpoints marked with >>> for Example / for Exam*
  - *Why? Because they follow the new idea of responseHandler and logging*
  - *Why aren't all endpoints in this project with the new model? That would take time, and the five already demonstrate the new model well*
  - *Possible old examples in this evolving project are also interesting for comparison. They are remarkably longer and have unnecessarily repeating code lines.*

src/db/index.js

- *File that (1.) defines a database configuration object*
    - *(Though the database settings are also **originally** from the **.env** file)*
- *…and (2.) exports a database connection object (we call it this time ‘knex’) based on that configuration object*
- *Other code modules can then execute database operations with that knex object*

src/responseHandlers/index.js

- *File that unifies the handling of the response, after database operation has been completed, either successfully or unsuccessfully*
- *Kind of a utility library, reusable functions. Elsewhere we save a lot of code writing*
- *Writes also to the log using the logger.*
- *Finally gives the caller (e.g. our Frontend, or some test system, like Postman client) the HTTP response too.*
  - *In case of total success from business point of view: "200" and the possible data*
  - *In case of error that can be somehow pointed at the request: "400" and "Request error"*
  - *In case or error that can be somehow pointed at the backend server or database: "500" and "Server error"*
- *That's safer. The rest of the details developers can find from backend logs*

src/utils/logger.js

- *File first defines configuration for the winston logging library*
  - *E.g. we can define what is the least serious message level we write to each log file / console*
  - *Logging levels are: 0 error, 1 warning, 2 info, 3 debug, 4 verbose, 5 silly.*
    - *If you set level to silly, you get all of the above*
    - *If you set level to info, you only get error, warning and info*
  - *We also define a message format with nice timestamp at start*
- *Then the file creates and exports a logger object based on that configuration*

- *It would be possible to make this logger a lot more serious if you have time. Like put structured metadata and categories. Or separate logging for development time and production time… But already as it is it serves quite well on e.g. Softala or other software project courses.*

logs/backendLog.log & logs/errorLog.log

- *Files that keep history of the logged messages*
- *We don’t put these to git repository as it will be generated when needed. That means, when the application is running. (gitignored)*
- *From time to time we might empty the log files*

- *The errorLog only logs errors, so it is maybe the first place for the developer to look into (often errors are easy to understand and fix).*
- *When more complicated case arrives the backendLog also tells the events leading to the error.*
- *Notice that in some systems and some cases the order of the events might be different from log order. In JavaScript environments likely in order though. As they are single-threaded.*