Asynchronous code. Promises

idea-case-backend

8.2.2023

Haaga-Helia

Asynchronous backend

- *Most JavaScript runtime environments are single-threaded, including Node.js*
    - *Google Chrome V8 JavaScript engine*
- *Node.js is handling one request at a time, in a pipe or a loop*
- *What happens if handling one request needs services from outside of that Node single thread?*
    - *E.g. file read from operating system’s file system*
    - *Or database operation (same computer or remote, it’s still a different thread and unknown execution time*
    - *Or using other backends and calling e.g. further REST API or GraphQL backends*
- *You must make those requests asynchronous, non-blocking.*
    - *That outside 1. action started, and 2. callback/result mechanism set up, but not keeping that single thread there waiting for the fulfilment of the operation, but your 3. handler functions (success and failure) defined that will be called, when result later ready.*

Promise

- *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise*
- *Promise is an object that is for sharing that asynchronized operation between the caller parts and remote parts.*
- *You ask e.g. a database operation to be started now, receive a Promise object immediately as return value and to that Promise object *) you attach your handlers by immediately calling the*
  - *.then(your success handler function defined here) and*
  - *.catch(your failure handler function defined here)*
- **) actually often we call .then and .catch attached to the function call, that returns that object, but it is finally the same. So first that function call returns the object, and then .then and .catch are called.*
- *Actually technically .then is called, and it further returns the Promise object and .catch is called again.*
- *This chaining of the function call makes the code cleaner*

LET'S LOOK at an example from the Node.js backend, possibly another from frontend AJAX if time

Promise-based code example from Node end-point

```
category.get("/", function(req, res) {
  knex.select().from("Category")
    .then((data) => {
      successHandler(res, data, "category.get/all: listed ok from DB");
    })
    .catch((error) => {
      databaseErrorHandler(res, error, "category.get/all: ");
    });
});
```

// What to see in this code? First, when is each part of the code executed? First think, then see the answer on the next slide

Haaga-Helia

Code executed in three different moments in time

```
category.get("/", function(req, res) {
  knex.select().from("Category")
    .then((data) => {
      successHandler(res, data, "category.get/all: listed ok from DB");
    })
    .catch((error) => {
      databaseErrorHandler(res, error, "category.get/all: ");
    });
});
```

// Bolded underline parts are _executed_ at the server setup/startup! Rest of the code is _definition_ of the request handler, executed when and if a matching request will later arrive. Maybe days later.

Code execution phase two – Request arrived

```
category.get("/", function(req, res) {
  knex.select().from("Category")
    .then((data) => {
      successHandler(res, data, "category.get/all: listed ok from DB");
    })
    .catch((error) => {
      databaseErrorHandler(res, error, "category.get/all: ");
    });
});
```

// Bolded underline parts are _executed_ if and when a matching request has arrived. Rest of the code is _definition_ of the database operation result handlers, executed when database operation ready. Maybe milliseconds or seconds later. (Possibly microseconds if using in-memory database).

Code execution phase three – DB operation ready

```
category.get("/", function(req, res) {
  knex.select().from("Category")
    .then((data) => {
      successHandler(res, data, "category.get/all: listed ok from DB");
    })
    .catch((error) => {
      databaseErrorHandler(res, error, "category.get/all: ");
    });
});
```

Now only executing code. No more definitions.

What to remember in Promise-based coding?

```
category.get("/", function(req, res) {
  knex.select().from("Category")
    .then((data) => {
      // whatever you want to do after 'succesful' DB operation must be only here.
    })
    .catch((error) => {
      // whatever you want to do after 'crashed' DB operation must be only here.
    });
    // No code here!
});
// No code here!
```

1. ***Notice the handler functions, e.g. (data)…(error)… Write your code only inside the handler functions.***
2. ***E.g. write anything dependent on successful DB operation only inside the green code block.***