Day1 Express and creating a starter project

Main exercise for today is to create a start project which we will use for the rest of the semester for projects that requires the following features

- Uses Node and the Express.js Server
- Uses typescript *the simplest possible way*
- Must include a "public folder" to hold plain HTML, JavaScript, CSS, images etc.
- Uses nodemon (or something similar) for easy development
- Allow developers to debug, both production and test code
- Handles secure values (PORT, DB-credentials, Token keys etc) in a "developer friendly way".
- Allow us to start several instances of the server, as for example a specific instance for testing
- Provide an efficient way to handle error scenarios
- Provide a ready to use setup for unit and integration testing
- Support for Cors
- Provide the project with a src folder with the following **folders, interfaces, facades, models, routes**
- Must use a modern way to switch debug (what you used to do with console.log) statements on/of
- Must include log support for runtime logging.
- Must include a simple LOGIN system (we will use HTTP Basic Authentication to simplify matters)

Later we will add Mongo-DB and GraphQL support

Push this project to Github and use it for all future protects given this semester (we will add new features to this start code as we go on)

This is my version from where we left in the class: https://github.com/fullstack-cphbusiness/startcode.git

---

Using the start code.

https://www.youtube.com/watch?v=seHQso7fHAY

Lidt hjælp til at komme i gang med opgaven.
Første del er til REST-delen, den sidste (fra omkring 12.00) en kort opsummering på TypeScript og generics ud fra koden givet næste side.

Fork a copy of the start code (implemented in the previous step) and do the following.

1) In the interfaces folder add a file **IFriend.ts** and paste this code into the file:

```
export interface IFriend {
  id: string
  firstName: string
  lastName: string
  email: string
  password: string
}
```

2)

In the facades folder add a file **DummyDB-Facade.ts** and add this code:

We will discuss this in the class before you start

```
import { IFriend } from '../interfaces/IFriend';

function singleValuePromise<T>(val: T | null): Promise<T | null> {
  return new Promise<T | null>((resolve, reject) => {
    setTimeout(() => resolve(val), 0);
  })
}
function arrayValuePromise<T>(val: Array<T>): Promise<Array<T>> {
  return new Promise<Array<T>>((resolve, reject) => {
    setTimeout(() => resolve(val), 0);
  })
}

export default class FriendsFacade {
  friends: Array<IFriend> = [
    { id: "id1", firstName: "Peter", lastName: "Pan", email: "pp@b.dk", password: "secret" },
    { id: "id2", firstName: "Donald", lastName: "Duck", email: "dd@b.dk", password: "secret" },
  ]
  async addFriend(friend: IFriend): Promise<IFriend> {
    throw new Error("Not Yet Implemented")
  }
  async deleteFriend(friendEmail: string): Promise<IFriend> {
    throw new Error("Not Yet Implemented But return element deleted or null")
  }
  async getAllFriends(): Promise<Array<IFriend>> {
    const f: Array<IFriend> = this.friends;
    return arrayValuePromise<IFriend>(this.friends);
  }
  async getFrind(friendEmail: string): Promise<IFriend | null> {
    let friend: IFriend | null
    friend = this.friends.find(f => f.email === friendEmail) || null;
    return singleValuePromise<IFriend>(friend);
  }
}
```

3) Implement, inspired by Moshe's video, a full REST-API in the project using the DummyFacade given above.

Day2 Express and Middleware

The main task for today is to add the following pins to our start code, using middleware

- Provide an efficient way to handle error scenarios
- Support for CORS
- Include support for runtime logging.
- Include a simple LOGIN system (we will use HTTP Basic Authentication to simplify matters)

It's essential to understand the concept of middleware when using Express. In the following, you will first write your own middleware, and after that use existing middlewares to do more detailed logging and add authentication and authorization to the app.
Startcode as we did in the class https://github.com/fullstack-cphbusiness/startcode.git day2startcode

First middleware example - A Simple Application-logger for the app

In this exercise, you will create a simple logging middleware which you can use as the application logger for your start code. Obviously, for something as important as logging, (better) existing packages exist, but before any of those, you should write our own in order to learn the concept middleware.

a) Create a simple middleware, which for ALL incoming API-request will log (console.log in this first version) the following details:

- *Time*,
- The *method* (GET, PUT, POST or DELETE),
- the URL
- The remote ip

b) Move your logger-middleware into the folder middlewares, in a file `simpleLogger.ts`, and export it from here. Import and use it in `app.ts`.

A more realistic logger middleware, using existing logger packages

a) 🟥 (see b for an easier way) Add a new file logger.ts to the middlewares folder. Add the required code using the Winston package so that all logging-info goes into a file `/logs/combined.log` and errors go into a file: `/logs/error.log`

b) 🟩 Here is my suggestion for a middleware function that logs as described above:

- Middleware
- How to use it

Handling CORS with middleware

a) Handle CORS manually, as explained in the video for today (go to 3.25)
First just do as he explains, but after that move the middleware into your own file myCors.ts in the middleware folder.
Then import and use the middleware THE RIGHT PLACE, so that only your REST endpoints will set the the CORS-headers.

b) Leave your own my-cors.ts file for future reference, but change your code to use the cors-package.
Once you have installed this package this is as simple as importing it, and use it, that is two lines of code.

Info: Obviously when deployed, and for a real-life app, you should fine tune the cors-headers to your specific needs, but the two lines referred to above will do for now.

Error Handling

Provide the project with an error handling strategy, that will generate json-responses for errors related to the API, and use the default error handler for other errors.

We will do this partly in the class, and we will uses this file to reprepresent API-errors:

https://github.com/fullstack-cphbusiness/code-snippets/blob/main/apiError.ts

Basic Authentication with middleware

Write your own middleware to handle basic authentication and authorization for the project.
Use this package, https://www.npmjs.com/package/basic-auth to handle all details related to Basic HTTP Auth.

Unless you really like the challenge ;-) this is actually meant as a code-along tutorial given in this video https://www.youtube.com/watch?v=U-4MM728c-0

This video-tutorial also explains what basic http authentication is, how to use it, its pros and CONS, and visualizes how to write our own middleware that uses basic-http-authentication.
At the end of the video, you will find a few additional security hints, relevant for all, but definitely for those of you following the security course.

Install these packages before you start:

```
npm install basic-auth
npm install tsscmp
npm install @types/basic-auth --save-dev
npm install @types/tsscmp --save-dev
```

Create a file basic-auth.ts in the middleware folder and this content to the file

```
import auth from 'basic-auth'
import compare from 'tsscmp'

import { Request, Response } from "express"
import facade from "../facades/DummyDB-Facade"

const authMiddleware = async function (req: Request, res: Response, next: Function) {

}

export default authMiddleware;
```

The final middleware file (but you should code it yourself)

***Important:** For a real life application you would not write your own security middleware, but use an existing (trusted) one. Passport would be a realistic candidate, but for this exercise, middleware, and writing middleware, was the main focus.*

Day3 Express, MongoDB with TypeScript

Before you start, you should have completed the exercise "Getting started with MongoDB with Typescript" which ensures you have a ready to use database on Atlas, and a fundamental understanding on how to do basic CRUD operations on a MongoDB database.

Your task is to update your startcode with a ready-to-go example with MongoDB and a facade + REST endpoint using your friend database. Preferably you should also update your authentication middleware to get users from a database.

1) Ensure you have a startcode version, with support for MongoDB

You can do this in two ways. Both requires you to use the **day3 branch** of this project:
https://github.com/fullstack-cphbusiness/startcode/tree/day3

a) Updating your own code

If you want to update your own code (**definitely the preferred way**) you can either clone the project given above,(remember to use the day3 branch) or just copy the relevant files from github as explained in the video below.
Follow the steps in this video, and update your code, **step-by-step**

https://www.youtube.com/watch?v=7YlO58NPr-U

b) Use my code from the day3 branch as your "own" future startcode.

Clone the project given above, and switch to the day3 branch. Now watch **ALL the steps in the video** and **MAKE SURE you understand what happens**. If you use this for the exam, you are expected to explain about how it's designed, and the purpose of all the code.

Quick shortcuts to the individual parts of the video given above

Intro (2 min.)	https://www.youtube.com/watch?v=7YlO58NPr-U
Dependencies	https://www.youtube.com/watch?v=7YlO58NPr-U&t=131s
Database Connector	https://www.youtube.com/watch?v=7YlO58NPr-U&t=163s
Update bin/www.ts	https://www.youtube.com/watch?v=7YlO58NPr-U&t=297s
Update IFriend	https://www.youtube.com/watch?v=7YlO58NPr-U&t=465s
Test data for development	https://www.youtube.com/watch?v=7YlO58NPr-U&t=504s
Delete DummyDB-Facade.ts	https://www.youtube.com/watch?v=7YlO58NPr-U&t=693s
Add friendFacade.ts	https://www.youtube.com/watch?v=7YlO58NPr-U&t=718s
Update the Auth-middleware	https://www.youtube.com/watch?v=7YlO58NPr-U&t=1085s
Add/update the logger.ts middleware	https://www.youtube.com/watch?v=7YlO58NPr-U&t=1312s
Delete the original friendroutes	https://www.youtube.com/watch?v=7YlO58NPr-U&t=1411s
Add one of the two friendRoutes	https://www.youtube.com/watch?v=7YlO58NPr-U&t=1411s
The easy one	https://www.youtube.com/watch?v=7YlO58NPr-U&t=1439s
The one with authentication	https://www.youtube.com/watch?v=7YlO58NPr-U&t=1573s

2) Complete the TODO-steps included in the code samples from branch3

Complete the facade and Router for the code that operates on the friend-collection.
Test POST, PUT and DELETE manually via Postman. Next week we add automated tests, both for the facade and for the endpoints