**Alpha 27 JS - JavaScript Backend**

Full Stack Project Survival Guide

1. Rule one - do not think about the frontend for now.
2. Rule two - read rule one again.
3. Rule three - focus on the must requirements

1. The Database

Each backend must have a useful database. The way we store data determines the way we used said data. Carefully designed database will save you a lot of work later.

**Very important note**: You will not get it right from the first attempt. Database design requires experience, which you do not have, do not be hard on yourself.

1. Gather with your team and read the requirements.
2. Try to identify the data that you will need to store. Example requirement:

*A user can create posts with title, text, date. Other users can comment on posts.*

This humble sentence reveals much:

* 1. We need a table posts - with columns: at least an id(int), title(varchar), text(varchar), date(some-date-type)
  2. We need a table comments - with columns: at least an id(int) and text(varchar)
  3. We need a table users - no useful information here, but somewhere else there might be more on users.
  4. There are **relationships**:
     1. *A user can create posts...* - one-to-many relationship, so the posts table needs a column userId, which is a FK to the users PK
     2. *Other users can comment on posts...* - 'other' users - does that mean that the author **cannot** comment. We better ask the client. What we know is that a **comment belongs to a user and to a post**. So we need additional columns inside table comments
        + userId - FK to the comment author
        + postId - FK to the id of the post that we are commenting on.

1. Open the workbench tool and try to create all the tables and relationships. Once ready and satisfied with the initial version, read [our guide](https://learn.telerikacademy.com/mod/page/view.php?id=11207&forceview=1) on exporting databases as sql script.
2. Share the schema with your teammate so you can start work.
3. You might add some test data by using the workbench interface.

2. The Express Server

1. Don't forget to install all the required packages (express, body-parser, mariadb, cors...)
2. Write some code:
   1. Connect to the database (the createPool configuration);
   2. Write a simple query (SELECT \* FROM posts);
   3. Write a service that consumes the query and returns the data;
   4. Write a request handler that consumes the service;
   5. Test everything from postman.
3. You might find it useful to save requests to postman. Open the **Collections** tab and create requests to quickly test your backend. Examples (GET /posts, POST /posts, POST /posts/{id}/comments, etc..)

3. Authentication

1. For several days, you will not know how to authenticate users. However, authentication is important, because the user that sends the request (POST /posts, for example) is also the POST author, usually. We need that id to set it to the userId column inside the posts database.
2. To **mimic** authentication, you can:
   1. Write a middleware that attaches mock userId to the request object. Request handlers can use that id
   2. Or, send a mock userId as a header. Request handlers can get that id from the request object
   3. Or, send a mock userId as a body param.
   4. Or, send a mock userId as a query param.
   5. Or, just hardcode a false id somewhere in your app
3. A downside is that all post, comments, etc will belong to the user with the false id.
4. Once we know how to authenticate, we will get the actual id of the logged in user.

4. Frontend.

1. Your endpoints are the public application programming interface of your server.
2. The frontend guys will figure out how to use it.
3. Read rule one again.

5. Tips:

1. Test SQL queries in workbench first. Also, write them there, because there is syntax highlighting. Once a query is finished, copy/paste it to the app code.
2. Do not think if your code follows about best practices. First, just make it work. Once you understand how it works, now it is time to apply principles and practices.
3. While you develop your application, your database **WILL** change. Communicate all changes as soon as possible with your teammate. Quickly reexport the new schema structure, so that your teammate does not write queries against old db schema.
4. Commit as often as possible, and, preferably, **no more than 3 files** (excluding the initial commit, which will be more than 3 files).