CWEB280 -wk7 – LO4 -Manage DB tables

# API Server and Single Page Client UI

For the remaining time in this course, we will focus on using an Application Program Interface (API) server that only accepts and returns data in JSON format. We then create a new Vuejs project that will be a single page application that runs on a browser on a client computer

RESTful API Server

* There is growing diversity in mobile technologies and a growing demand for cross platform interactions,
* Having an API (Application Program Interface) server, as the main gateway to the data store, allows developers to create multiple user interfaces like xbox, ps5, ios, android, pc and websites to communicate with a central API server.
* API servers usually do not server up webpages. Rather API servers usually accept JSON formatted data in the body of a request and return JSON as the response
* REST (REpresentational State Transfer) is a way to access the API server using HTTP request methods:
  + POST (usually for Create)
  + GET (usually for Read)
  + PUT (usually for Update)
  + DELETE(usually for Delete)
* REST technology generally uses less bandwidth (JSON formatted data) than other technologies like SOAP (uses XML formatted data)

Learn More about RESTful API: <https://searchapparchitecture.techtarget.com/definition/RESTful-API>

# TypeORM Project

We will discuss ORMs (Object Relational Mapper) further in Learning outcome 5. For now we will create a project that uses the TypeORM framework as a continuing example project for the course.

Learn more about TypeORM: <https://typeorm.io/>

## Creating a TypeOrm project

Watch the provided video.

## TypeORM Project Folder Structure

lo4serverapi

├── src -- place of your TypeScript code

│ ├── controller -- place where your controllers (actions) are stored

│ │ └── UserController.ts --sample controller (actions:all,one,save,and remove)

│ ├── entity -- place where your entities (database models) are stored

│ │ └── User.ts -- sample entity (defines table columns and types)

│ ├── migration -- place where your migrations are stored

│ └── index.ts -- start point of the app (like app.js from lo1serveronly)

│ └── routes.ts -- acts like a traffic cop – directing a request to an action

├── .gitignore -- standard gitignore file

├── database.sqlite -- sqlite database file – filename specified in ormconfig.json

├── ormconfig.json -- ORM and database connection configuration

├── package.json -- node module dependencies

├── README.md -- simple readme file

└── tsconfig.json -- TypeScript compiler options

The example project will add two users to the database every time the app is started/ restarted. We wil change that later but for now we will let it stand.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>

## TypeScript

The project uses TypeScript which a strongly typed version of JavaScript developed by Microsoft.

* Typescript does not run natively in a Browser or in Nodejs
* Typescript must be transpiled into JavaScript that then runs in Nodejs or a browser
* Transpiling is the process of converting source code in one language to source in another language
* Transpiling is also called “source to source compiling”
* Because TypeScript is transpiled, developers can use “decorations” in the code to add metadata that TypeORM can read and use when creating the database tables
* “Decorations” or “Annotations” inject additional code (not visible to the developer) when the source code is transpiled or interpreted

## Routing in the TypeORM project

**Review a snippet of code from *\src\routes.ts:*** The objects in the array specify which http request method triggers which action

import {UserController} from "./controller/UserController";  
  
export const ***Routes*** = [{  
 method: "get", // HTTP Request method used by the client  
 route: "/users", // the path in the address bar of the client  
 controller: UserController, // the class that contains the "actions" (aka "methods")  
 action: "all" // the action to call in the controller  
},

.

.

.

];

**Review a snippet of code from *\src\controller\UserController.ts:*** The action (aka method) called ‘all’ gets executed when a GET request is made to the ‘http://localhost:3000/users’ url.

import {getRepository} from "typeorm";  
import {NextFunction, Request, Response} from "express";  
import {User} from "../entity/User";  
  
export class UserController {  
  
 private userRepository = getRepository(User);  
  
 async all(request: Request, response: Response, next: NextFunction) {  
 return this.userRepository.find();  
 }

**Review a snippet of code from *\src\index.ts:*** The code loops through the routes array and create express routes ad handlers

// register express routes from defined application routes  
***Routes***.forEach(route => {  
 (app as any)[route.method](route.route, (req: Request, res: Response, next: Function) => {  
 const result = (new (route.controller as any))[route.action](req, res, next);  
 if (result instanceof ***Promise***) {  
 result.then(result => result !== null && result !== undefined ? res.send(result) : undefined);  
  
 } else if (result !== null && result !== undefined) {  
 res.json(result);  
 }  
 });  
});

# Making RESTful calls to the API server Using Postman

First ensure you have installed PostMan: <https://www.postman.com/downloads/>

Seconds change the port in the \src\index.ts file: Add the code in violet

// start express server\  
const port = process.env.PORT || 3004;  
app.listen(port);  
  
// insert new users for test  
await connection.manager.save(connection.manager.create(User, {  
 firstName: "Timber",  
 lastName: "Saw",  
 age: 27  
}));  
await connection.manager.save(connection.manager.create(User, {  
 firstName: "Phantom",  
 lastName: "Assassin",  
 age: 24  
}));  
  
console.log(`Express server has started on port ${port}. Open http://localhost:${port}/users to see results`);

We will be running multiple web apps simultaneously and each must have a different port number

## To run the project from command prompt:

npm start

## To run the project in WebStorm

1. Open the cweb2021/lo4serverapi folder in WebStorm as an existing project
2. Once the project is open click “Add Configurations…” near top right
3. In the Run/Debug Configurations window click the plus sign next to the top left
4. Select “npm” from the list of available configurations (left window area)
5. Then on the right window area find the Command (drop down) and select “start”
6. In the “Environment” text box add the following: PORT=3004
7. Click the ok button
8. You should see “start” where “Add Configurations…” used to be.
9. Then click the green play button immediately to the right of “start”

## Open Postman

Watch provided video

# Manage Database Tables with ‘Database Navigator Plugin for Webstorm’

## Install the Database Navigator Plugin in Webstorm

Watch provided video

## Create a new table

First, in the DB Browser, right click server api db and select Auto-Commit

In the server api db console:

Create the student table

DROP TABLE IF EXISTS Student;  
CREATE TABLE Student(  
 id *INTEGER* PRIMARY KEY AUTOINCREMENT,  
 lastName *NVARCHAR*(50) NOT *NULL*,  
 firstName *NVARCHAR*(50) NOT *NULL*,  
 userName *VARCHAR*(50) NOT *NULL*,  
 address *VARCHAR*(100)  
);

Alter the table

ALTER TABLE Student  
 ADD phone *VARCHAR*(15);

Add a seed for the id column

INSERT INTO sqlite\_sequence (name,seq) VALUES('Student',100000);

Insert a new student:

insert into "Student" (  
 address,  
 "firstName",  
 "lastName",  
 "userName")  
values (  
 '1 Happy Place',  
 'Jane',  
 'Doe',  
 'doee1234')  
;

Double Click the Student table and examine the rows in the table.

Create another table with a foreign key

DROP TABLE IF EXISTS CSTStudent;  
CREATE TABLE CSTStudent(  
 studentID *INTEGER* PRIMARY KEY,  
 account *VARCHAR*(6),  
 classGroup CHAR(1),  
 yearInCourse *INTEGER*,  
 CONSTRAINT fk\_student  
 foreign key (studentID)  
 references Student (id)  
);

Exercise

Insert a CST Student record

Delete a record from Student and CSTStudent