CS25320 Coursework 2019: Canvas Game with Persistent Data

Radoslaw Wawrzyk

raw32@aber.ac.uk

radekwawrzyk@gmail.com

Description

I decided to create typical **Snake game** with scoreboard. I put big attention to the code quality. I'm aware that the snake game concept is not revealing, so that's why I focused on the clean & modular code. Also I used **Node.js** for the **Back-End**, I thought that it would be more interesting than **PHP & PostgreSQL**, or even **SQL**, because I used **NoSQL - MongoDB**. I belive that I don't have to explain snake mechanisms.

The whole **Front-End JavaScript** code is modular and based on the **ES6+** features - Async/Await, let/const, arrow functions, try/catch. Basically I tried to make game with the **OOP** standards. Moreover I used **Webpack & SCSS** . I splited all styles/scripts into smaller modules. By this approach code is clean and modular.

Also there is a form that connects with the **Node.js REST API** by **AJAX Http request** with **try/catch** error handling method. In addition player can change speed of the snake and type his name. If player doesn't set his nickname, his name will be **"Unknown"** in th database.

What is worth to mention, after each lost game I send **POST** http request to the **Node.js REST API**, and save the game result in the **MongoDB**. Below the canvas snake element there is a **Scoreboard** - it updates itself after each page visit or reload or. It happens thanks to AJAX request after page initial render.

I decided to write Back-End in the **Node.js**, cause I know the **JavaScript** pretty well, so choice was simply for me. I made **MVC REST API** application with **MongoDB** databse. The database is host by Atlas provider in the cloud, so you don't have to install it on your localhost! The whole **Front-End JavaScript** code is modular and based on the **ES6+** features such a like on the **Front-End** part.

Back-End (Node.js)

Node.js is required to run this application correctly. Also, you need **nodemon** npm package to run `npm run/yarn dev` script. I tried to write back-end with **JavaScript ES6/ES7** newest features like **Async/Await, arrow functions** etc. There is one model for database (player model), one controller for player, and 2 **REST API end-points** (POST & GET). Application is modular, because is based on **MVC design pattern**, therefore the development/maintance of application is simple and logic.

I created Back-end in the **Node.js**, because I feel very comfortable in the **JavaScript**, so that's why chose it. For data I used **MongoDB** - cloud client by Atlas provider. Basically I made simply **MVC REST API**.

HTTP methods End-point stable

HTTP Method	URL path	Description
POST	/api/players	Add score the database
GET	/api/players	Retrieves all scores from database

Technology stack:

- JavaScript/Node.js/Express.js
- MongoDB database (cloud storage) - Atlas
- ES6+
- REST API with MVC architecture
- Webpack module bundler

Install

```
$ cd back-end
$ npm/yarn install
```

Run

```
$ npm run/yarn dev (requires nodemon package)
$ npm run/yarn start
```

Build

```
$ npm run/yarn build
```

Front-End

Node.js is required to run this application. Basically Website is based on Webpack boilerplate **App** has only one route **('/')**. I decided to write **OPP** code, by **ES6+** I could use classes & splited code into smaller modules.

The whole JavaScript code is splited into 4 classes/modules -**Game, Snake, Ranking** and the **Apple** class. Each class has **constructor** and the bunch of few **methods**. For main game module I used `Canvas` - Game, Snake, Apple files, however, for scoreboard I created the `Ranking` class, which basically render code based on response from **HTTP** request from **REST API**.

Technology stack:

- HTML5/CSS3/SCSS

- JavaScript (ES6+, OOP, Async/Await/, AJAX)
- Canvas with OOP
- Ajax & Axios.js promise based HTTP client for the browser and node.js
- Webpack module bundler