# Assignment 2

## Instructions:

- Code will be executed using NodeJS.
- Grading is based on implementing the required functionality and coding style, specifically: clearly organized code, appropriate variable naming, code readability, coding conventions demo'd in class, etc.
- Javascript syntax rules:
    - Variables must be declared using let/const, not var
    - Functions must be declared using arrow function syntax, not function() syntax
    - When checking equality, use strict equality (triple equals ===), **not** double equals (==)
    - Do **NOT** use higher order array functions: forEach, map, reduce, filter, closest, etc.
- Unless otherwise noted, **do not** implement any **server side** or **client side form field** validation.

## Submission Checklist:

1. Create NodeJS project

- ❏ Create a folder called **A2FirstName**. Replace **FirstName** with your name, example: **A2David**
- ❏ Inside the folder, create a new NodeJS project.
- ❏ Within your project's **package.json** file, update the **author** field with your name, and the **description** with a description of your project.
- ❏ Within the project, create a Javascript file called **server.js**. Put solution code in the **server.js** file.

2. When you are ready to submit:

- ❏ Create a zip file containing your project folder.
- ❏ Rename your zip file **A2FirstName**.zip. Replace FirstName with your name, example: **A2David.**zip.
    - ❏ Ensure you use a zip file. Rar and 7zip files are not accepted.

## Academic Integrity

- You are responsible for familiarizing yourself with the college's Academic Integrity Policy.
- This is an individual assessment
- Situations which often cause academic integrity issues:
    - Reposting any part of the assessment to online forums or homework help websites
    - Contract plagiarism: Purchasing a solution, or completing a solution for financial compensation
    - Sharing or receiving source code, references, or assistance from others

## Problem Description:

Using NodeJS and Express, create a server side web application for a parking lot company.

### *Webpages*

The application has two client-facing HTML pages:

- Pay for parking: customers can purchase parking at a selected parking lot
- Admin login: if a user has a valid username and password, then that user can view the total amount of money collected by the parking lot.

### *Data Store*

The application also contains a **non-persistent data store (global array)** that tracks the payments made by users to the parking lot.

- Payment must be represented as a Javascript object literal.
- Every payment contains properties for:
  - the a user's vehicle license plate
  - the total amount paid for parking

### **Pay for Parking Page**

This page is displayed when the user visits the app's default endpoint (/)

The page must show:

- Header section that contains the webpage's navigation menu
- Main body section that displays a form for the user to purchase parking time

The form contains fields to accept data for

- The selected parking lot. Provide a minimum of 3 parking lots. Each parking lot must have a different hourly rate.
- The number of hours to park
- The user's vehicle license plate

When the form data is submitted to the server, **calculate** the total cost of parking, **save** the relevant data to the data store, and **send a *string response*** to the client. The string response must contain the user's receipt.

The receipt must display:

- The number of requested hours
- The hourly rate of the selected parking lot
- Subtotal, tax (13%), and the final amount for the user to pay

Note: The maximum number of hours that can be requested is 8. If the user requests parking for more than 8 hours, send a *string* containing an error message back to the client.

### **Admin Login**

This page is displayed when the user visits the /admin endpoint.

The page must show:

- Header section that contains the webpage's navigation menu (this menu should be identical to the menu presented on the Pay for Parking Page)
- Main body section that displays a form for the user to enter a username and password

When the form data is submitted to the server, check that the user entered a valid **administrator username** and **password**.

- If the credentials are valid, **calculate** the total amount of fees collected by the parking lot, and **send** the results to the client **as a string**. The calculation for the total fees must be **programmatically calculated** based on the payment items in the application's data store (ie: loop through the array and calculate the total fees)
- If the credentials are invalid, **send** an error message to the client as a **string**.

You may assume the only valid admin credentials are:

- Username: admin
- Password: 0000

## HTML and CSS Styling

**For the Pay for Parking and Admin page**

- HTML must be structured using *semantic* elements.
- You are responsible for choosing the appropriate elements for both structure and content
- Styling must be implemented using **external CSS stylesheets**. You should **customize** the colors and fonts. Ensure your final styling is reasonably pretty and easily readable.

**String data sent by the server**

- Any string data sent by the server (error messages, receipts, etc) must be styled (HINT: Use inline styling)

You are **not allowed** to use 3rd party libraries or frameworks (Bootstrap, Handlebars, etc)

**- END OF ASSESSMENT -**