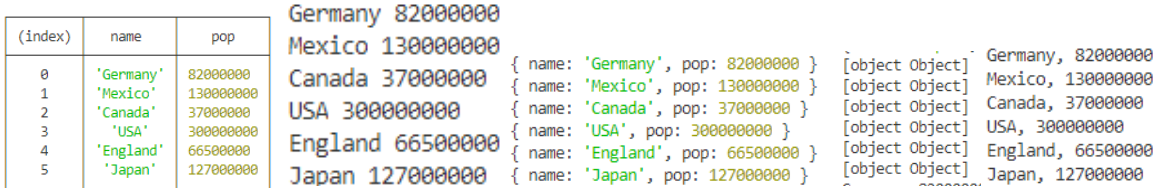
INFO3139 Lab 7

Rev 1.0

General JavaScript Topic #6 – Console.log formats

We have all used a console.log to help in debugging, but there are some templates you may not be aware of. If we rework our spread example from last class to use objects instead of strings we can see how to log the data different ways:

```
const getOriginalCountries = () => {
  let originalCountries = [
    { name: "Canada", pop: 37000000 },
    { name: "USA", pop: 300000000 },
  ];
  let endCountries = [
    { name: "England", pop: 66500000 },
    { name: "Japan", pop: 127000000 },
  ];
  originalCountries.push(...endCountries); // appends on the end using spread
  return originalCountries;
};
const getAllCountries = () => {
  let original = getOriginalCountries();
  let beginCountries = [
    { name: "Germany", pop: 82000000 },
    { name: "Mexico", pop: 130000000 },
  ];
  original.unshift(...beginCountries); // appends to beginning using spread
  return original;
};
// nice way to dump an array on the console
console.table(getAllCountries());
// old school string
getAllCountries().forEach((country) =>
  console.log("%s %i", country.name, country.pop)
);
// old school template object
getAllCountries().forEach((country) => console.info("%o", country));
// es6+ template object
getAllCountries().forEach((country) => console.info(`${country}`));
// es6+ template string
getAllCountries().forEach((country) =>
  console.log(`${country.name}, ${country.pop}`)
);
```



(index)	name	pop
0	'Germany'	82000000
1	'Mexico'	130000000
2	'Canada'	37000000
3	'USA'	300000000
4	'England'	66500000
5	'Japan'	127000000

Now onto today's lab and case material….

Case Study# 1 Work – cont'd

We are **not** going to be using a countries collection in the case study (last lab was just to get some practice in creating custom documents). We will instead be creating a new collection called **alerts** based **on both** the Government of Canada's travel alerts JSON from a few classes ago and the ISO JSON we utilized last class

- Travel Alerts - http://data.international.gc.ca/travel-voyage/index-alpha-eng.json
- ISO - https://raw.githubusercontent.com/elauersen/info3139/master/isocountries.json

1. Update the .env file in the case study project **and add** this constant:

```
ALERTCOLLECTION=alerts
```

2. Update **config.js** accordingly.

3. Copy the week4\class1**db_routines.js** to the case study project

4. Update **project1_setup.js**, functionality for this method should now:
    a. Delete any existing documents from an **alerts** collection in the database remember to use the .env/config.js variable
    b. Obtain the country **ISO JSON** from GitHub and place it in an array variable (249 countries)
    c. Obtain the **ALERT JSON** from the GOC site (230 alerts)
    d. With each country, look up the corresponding JSON in the alerts (if there is one), you'll need to use the **.data** property to do this e.g., alertjsonvariable.data[….Some of the data **is tricky** to get at. We already know that properties with hyphens need the ["prop-erty"] syntax. We also can get at a particular object with something like variable.data["CA"] would get the object for Canada. You will need to take both techniques into account to obtain the data correctly
    e. With each country**,** create a new **alert object** that has the **following 6 properties** and add the object to an array:
        i. country – data to come from country's **alpha-2** property
        ii. name – data to come from country's **name** property
        iii. text – data to come from **advisory-text** from the alerts.json for the country whose code is equal to the value in the country property
        iv. date – data to come from the alert.json's date-published object's **date** property for the country whose code is equal to the value in the country property
        v. region – to come from the **region** property in the country data

vi. subregion – to come from the **sub-region** property in the country data

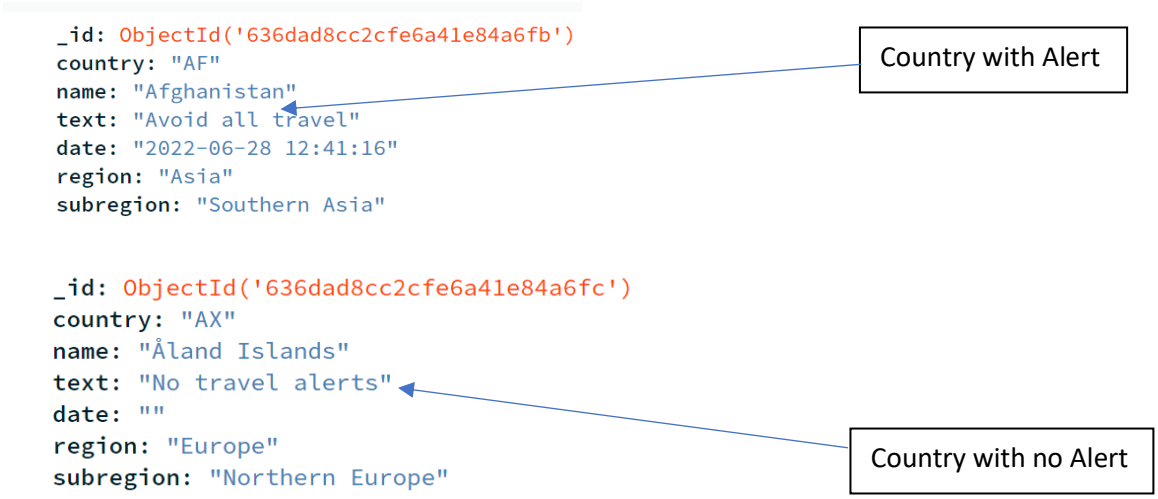
f. Add the resulting alert array contents to the alerts collection on Atlas

5. You will run into an issue because you will have **249** countries in the ISO JSON, and only **230** alerts in the ALERT JSON. Where there is **no alert**, create a custom document with the following 6 properties:
   i. country – data to come from country's **alpha-2** property
   ii. name – data to come from country's **name** property
   iii. text – a hardcoded value of "**No travel alerts**"
   iv. date – an empty string
   v. region – to come from the **region** property in the country data
   vi. subregion – to come from the **sub-region** property in the country data

6. Console log templated strings after each of the main steps e.g.:

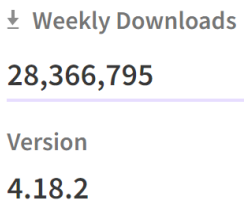
```
PS E:\winter2023\info3139\programming\nodeexercises\project1> node project1_setup
establishing new connection to Atlas
Deleted  249 existing documents from the alerts collection.
Retrieved Alert JSON from remote web site.
Retrieved Country JSON from GitHub.
added 249 documents to the alerts collection
```

7. Confirm the 2 types of documents in the new alerts collection



Introduction to Express

Today, we're going to have a quick look at a node web framework called **express**. **Express** is an **unopinionated** web framework (configure it the way you want to). It is an open-source framework developed and maintained by the Node.js foundation. Express download stats at the time of this this writing were (Note we're going to use a newer framework called Fastify instead of Express for the first case study, but we will be using Express in the second case study):



8. Issue the following command from the **nodeexercises** root to install **express**:

```
PS E:\winter2023\info3139\programming\nodeexercises> npm i express

added 57 packages, and audited 191 packages in 8s

23 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "got": "^12.5.2",
    "mongodb": "^4.11.0",
    "yargs": "^17.6.0"
```

9. Before using the new packages, we need to configure some dotenv variables. Create a subfolder called **week4\class2** and then create a **.env** file in it and add the following contents:

```
PORT=5000
DBURL=mongodb+srv:// the rest of connection string goes
DB=info3139db
COLLECTION=users
```

10. Add the following week4\class2**\config.js** file:

```
import { config } from "dotenv";
config();
export const atlas = process.env.DBURL;
export const db = process.env.DB;
export const collection = process.env.COLLECTION;
export const port = process.env.PORT;
```

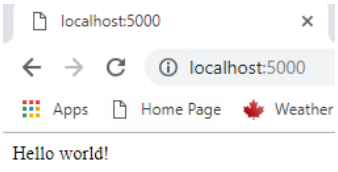
11. Then create a file called week4\class2**app.js** with the following contents:

```
import { port } from "./config.js";
import express from "express";
const app = express();

app.get("/", (req, res) => {
  res.send("\n\nHello world!\n\n");
});

app.listen(port, () => {
  console.log(`listening on port ${port}`);
});
```

12. Run the app.js (here I'm using the newish **--watch** switch for hot swappable code) in the terminal window and you should see:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class2> node --watch app
(node:10012) ExperimentalWarning: Watch mode is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
listening on port 5000
```

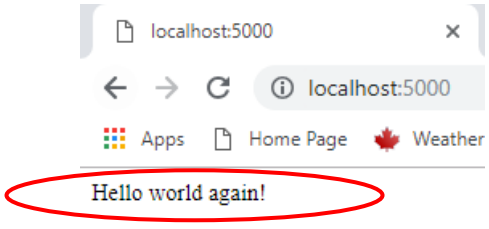
13. Then open a browser and point the browser to **localhost:5000**



Doing this confirms:

- Our web server is running
- It located our custom environment file .env
- It was able to serve on the custom port we defined in the .env file

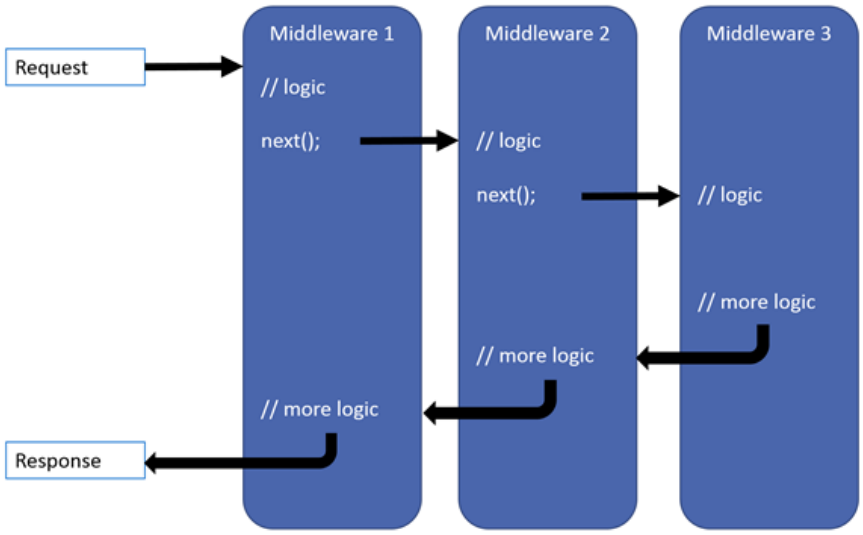
14. Test out the **new experimental watch switch**, Make a change to **app.js**, like maybe say **Hello world again!,** then refresh the browser and you should see:



Also notice in the terminal window, what's going on:

```
Restarting 'app'
listening on port 5000
```

Middleware



When doing any reading of express functionality, you will probably come across the term **"Middleware**". Before going any further, we better look a little closer at this term. Google defines middleware as:

"*a subset of chained functions called by the **ExpressJS** routing layer before the user-**defined** handler is invoked. **Middleware** functions have full access to the request and response objects and can modify either of them*"

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- Ends the request-response cycle.
- Call the next middleware function in the stack.

An Express application can use the following types of middleware:

1. **Application middleware** - Bind application-level middleware to an instance of the app object by using the app.use() and app.METHOD() functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.

2. **Error-handling middleware** - functions in the same way as other middleware functions, except with **four** arguments instead of three, specifically with the signature (err, req, res, next))

3. **Router-level middleware** - works in the same way as application-level middleware, except it is bound to an instance of **express.Router()**

4. **Builtin middleware** – There are 3 built-in middleware functions in Express:
    - express.static. This function is based on serve-static, and is responsible for serving static assets such as HTML files, images, and so on.
    - express.json. This function parses incoming payloads with JSON
    - express.urlencoded. This function parses incoming requests with URL encoded payloads (e.g. form data)

An example of each type of middleware:

15. We already did a piece of **application middleware** in our initial example (app.get…) but here is another example. Add the following code to the current app.js file **before** the current app.get (note this code executes with **every server request**):

```
app.use((req, res, next) => {
  console.log('Time:', new Date() + 3600000 * -5.0); // GMT-->EST
  next();
});
```

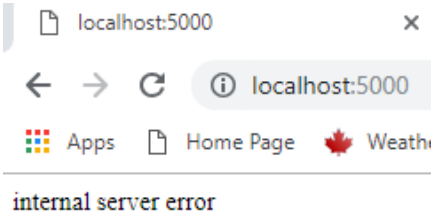
16. Save the app.js to restart the server, refresh the browser a few times and look in the terminal tab and you should see something like this:

```
Restarting 'app'
listening on port 5000
Time: Fri Nov 11 2022 09:25:40 GMT-0500 (Eastern Standard Time)-18000000
Time: Fri Nov 11 2022 09:25:41 GMT-0500 (Eastern Standard Time)-18000000
```

17. Next, do an example of **error middleware**. Comment out the current app.get routine and add this code right before the app.listen:

```
app.get('/', (req, res, next) => {
  next(new Error('Something went wrong :-('));
});

app.use((err, req, res, next) => {
  // Do logging and user-friendly error message display
  console.error(err);
  res.status(500).send('internal server error');
});
```

18. Save the file and refresh a browser window at localhost:5000 and you should see the following in the browser and terminal window:



```
Restarting 'app'
listening on port 5000
Time: Fri Nov 11 2022 09:28:23 GMT-0500 (Eastern Standard Time)-18000000
Error: Something went wrong :-(
```

We need to place error routines at the end of the app.js, and remember **it uses 4 parameters** not the standard 3 that the other middleware examples use

Next, we will look at a **routing middleware** example, routes are typically stored in a series of application specific files, so we will use that technique here:

19. Create a new JavaScript file called week4\class2**routes.js** with the following code (defines 2 routes):

```
import { Router } from "express";
const router = Router();

// define a default route
router.get("/", (req, res) => {
  res
    .status(200)
    .send({ msg: `this would be a response from the default route` });
});

// define a get route with a name parameter
router.get("/:name", (req, res) => {
  let name = req.params.name;
  res
    .status(200)
    .send({ msg: `this would be a response using the ${name} parameter` });
});

export default router;
```

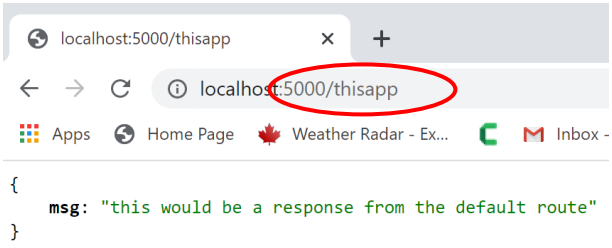
20. Modify the existing **app.js** file and add the following import at the top

```
import router from "./routes.js";
```

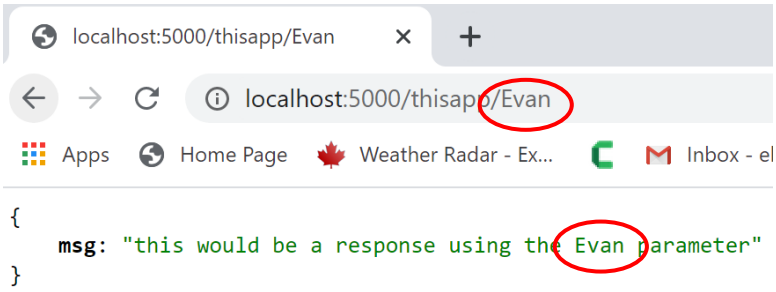
21. Remove the error app.get code and still in the app.js file add the following line of code in its place:

```
app.use("/thisapp", router);
```

22. Then point a browser to **locahost:5000/thisapp** and you should see:



23. Then test the parameter route with a url of **locahost:5000/thisapp/Yourname:**



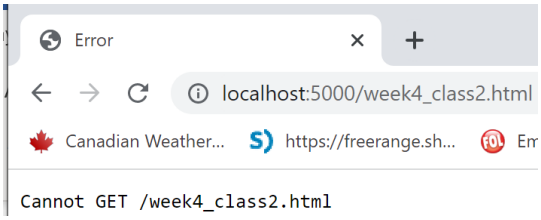
Finally let us look at an example of the **builtin middleware** using **express.static**. As mentioned earlier we use this to serve static files like images, css, html…

24. Create a new folder called week4\class2**public**

25. Add a new html file called **week4class2.html** to the new folder with the following:

```
<html>
    <body>We're looking at a Week 4 Class 2 Html Page in Express</body>
</html>
```

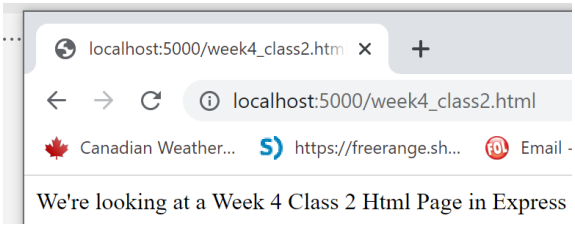
26. Try browsing to the url **localhost:5000/week4_class2.html** and you should see the following error:



27. Now to enable this static file we install the static middleware withthe following code in app.js file right after the routing entry:

```
app.use(express.static('public'));
```

28. Then browse to **localhost:5000/week4class2.html**



Lab 7 – 2.5%

Part A - (2%)

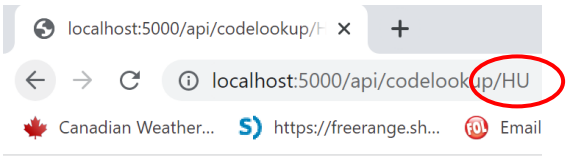
Re-work the lookup part of lab6.js by passing a country code as a request parameter in the URL and then display the corresponding ISO country name for that code in a web page. If you recall, we ran lab 6 like this:

```
PS C:\Evan\Winter2021\INFO3139\programming\jsexercises\week4\class1> node lab6 --code CA
establishing new connection to Atlas
deleted 249 documents from the countries collection
The code CA belongs to the country of Canada
there are 249 documents currently in the countries collection
PS C:\Evan\Winter2021\INFO3139\programming\jsexercises\week4\class1> node lab6 --code XX
establishing new connection to Atlas
deleted 249 documents from the countries collection
The code XX is not a known country alpha-2 code
there are 249 documents currently in the countries collection
PS C:\Evan\Winter2021\INFO3139\programming\jsexercises\week4\class1> node lab6 --code AD
establishing new connection to Atlas
deleted 249 documents from the countries collection
The code AD belongs to the country of Andorra
there are 249 documents currently in the countries collection
```

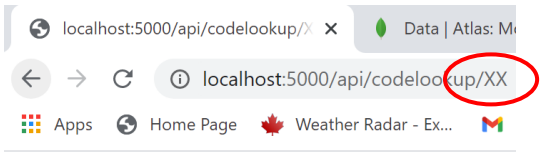
For this lab, setup a new nodeexercises\week4\class2**\lab7** folder. Create all the needed support files in this new folder (.env, config.js, db_routines.js, routes.js and app.js).

Set up a single route (**api/codelookup/…**) to be called from a web page to mimic the lookup part of lab 6, display “The code xx belongs to the country of yyyyy” line in the browser.

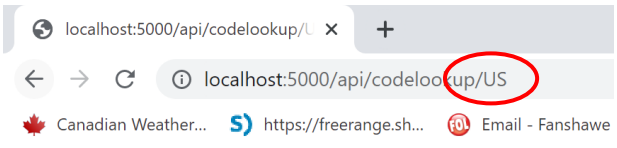
Submit **3 screen shots** in a single Word document along with the **source code for routes.js**, for the codes **HU**, **XX** and **US**, make sure the code in the URL is visible in the screen shot:



The code HU belongs to the country of Hungary



The code XX is not a known country alpha-2 code



The code US belongs to the country of United States of America

Part B - (.5%) - Lab 7 Theory Quiz on FOL - 10 questions

Review Questions

1. What is the discrepancy between the 2 sets of JSON to be used in the case study?
2. How are we going to rectify this discrepancy?
3. What is express?
4. What does it mean if a framework is unopinionated?
5. How does the experimental --watch switch help a developer?
6. What new constant was added to the .env file for processing web requests?
7. T/F - **Middleware** functions have full access to the request and response objects?
8. What is middleware?
9. What are the 4 types of middleware?
10. What 2 **function** names are used in application middleware?
11. Where do you place error handling middleware?
12. What is routing middleware bound to?
13. What is the only built-in middleware function in express?
14. What is a typical application of using the only built-in function in express?
15. Why did we add 360000*-5.0 to the Date in our application middleware example?
16. What is different in calling error processing middleware than the other 3 types?