Express Framework Lab Assignments

1. Problem Statement

Make a hello world Express program that will display "Hello, world!" at the root URL:/

Ans:-

```
const express = require('express');
const event = new Date();

// express app
const app = express();

// routes
app.get('/', function (req, res) {
    res.send('<h1>Hello, World!</h1>');
})

app.get('/time', function (req, res) {
    res.send(`<h4>${event.toISOString()}</h4>`);
})

app.listen(process.env.port || 8080);
```

> Hello World

2. Problem Statement

Tell the year you were born

Make an express program that will display the year you were born when you report your age in a query parameter. When you type in http://localhost:3000/year?age=32 into the address bar of your browser, for example, it will display You were born in 1984. Reference: https://scotch.io/tutorials/use-expressjs-to-get-url-and-post-parameters

Ans:-

```
const express = require('express');

const app = express();

app.get('/year', (req, res) => {

    res.set({ 'Content-Type': 'text/plain; charset=utf-8' });

    const age = req.query.age;
    let currentYear = new Date().getFullYear();
    let year = currentYear - age;
    res.send(`You were born in ${year}.`);
});

app.listen(3000, () => console.log('Application started on port 3000...'));
```

```
> node "c:\Users\ASUS\Desktop\npm\Express\DOB.js"
Application started on port 3000...
You were born in 1989.
```

3. Problem Statement

Create an Express.js app that outputs "Hello World!" when somebody goes tb /home. The port number will be provided to you by expressworks as the first argument of the application, ie. process.argv[2].

Ans:-

```
const express = require('express')
const app = express()

app.get('/home', function(req, res) {
  res.end('Hello World !')
})

app.listen(process.argv[2])
```

```
> Hello World !
```

4. Problem Statement

Make a Node.js program named express-hello-world.js that outputs "Hello World" to browsers making a GET request to the root (/) url. Also, to browsers that make a GET request to the /time url, send the current date and time in ISO format: 2015-12-31T23:59:59.999Z. Finally, use an environment variable named PORT for the port number if one is provided. If one is not provided use 8080.

i.e. The command below should start a server on the port 1337.

PORT=1337 node express-hello-world.js

and the command below should start a server on the port 8080.

node express-hello-world.js

Ans:-

```
const express = require('express');
const event = new Date();

// express app
const app = express();

// routes
app.get('/', function (req, res) {
    res.send('<h1>Hello World!</h1>');
})
```

```
app.get('/time', function (req, res) {
    res.send(`<h4>${event.toISOString()}</h4>`);
})

app.listen(process.env.port || 8080);
```

> Hello World!
2021-06-10T08:42:03.637Z

5.Problem Statement :Forms

This exercise will teach you how to process the traditional (non-AJAX) web form. Write a route (/form) that processes HTML form input (<form> <input name="str"/></form>) and responds with the value of str backwards.
To handle a POST request, use the post() method which is used the same way as get():
app.post('/path', function(req, res)(...))
Express.js uses middleware to provide extra functionality to your web server.
Simply put, a middleware is a function invoked by Express.js before your own request handler.
Middleware provide a large variety of functionality such as logging, serving static files, and error handling.
A middleware is added by calling use() on the application and passing the middleware as a parameter.
To parse x-www-form-urlencoded request bodies, Express.js can use urlencoded() middleware from the body-parser module.
const bodyparser require('body-parser") app.use(bodyparser.urlencoded((extended: false)))

HINTS

Here is how we can print characters backwards (just one way to do it):
req.body.str.split('').reverse().join('') Extended set to true (qs) or false (querystring) determines the parser module.
Read more about Connect middleware here:
https://github.com/senchalabs/connect#middleware
The documentation of the body-parser module can be found here:
https://github.com/expressjs/body-parser

Ans:-

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

app.use(bodyParser.urlencoded({extended: false}))

app.post('/forms', function(req, res) {
  res.send(req.body.str.split('').reverse().join(''))
})

app.listen(process.argv[2])
```

6. Create a Express server that uses static files. Place an index.html, an image and a css file in a directory named public. Use the index.html to display the photo and some caption text in red or another color using css.
Ans:-

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Express server</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h4>Coding is like an Art.</h4>
    <img src="codechef.jpg" alt="Codechef">
</body>
</html>
```

style.css

```css
body{
    background-color: cadetblue;
}

h4{
    font-family: 'Courier New', Courier, monospace;
    color: whitesmoke;
}
```

caption.js

```javascript
const express = require('express');
const app = express();

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

app.listen(3000, () => {
  console.log('listening at port 3000');
});
```

7. <u>Shopping List</u>

We will be building a simple application where we will store a shopping list. You should use an array to store your items in the shopping list.

Bhushan: Our application should have the following routes:

1. GET /items - this should render a list of shopping items.
2. POST /items - this route should accept form data and add it to list the shopping.
3. GET /items/:id - this route should display a single item's name and price
4. PATCH /items/id, this route should modify a ngle item's name and/or price
5. DELETE /items/:id - this route should allow you to delete a specific item from the array.

Ans:-

```
const express = require('express');
const bodyParser = require('body-parser');

// Express app
const app = express();

// Initializing array
var items = [{ id: 1, product: 'Bat' }, { id: 2, product: 'Ball' }, { id: 3, product: 'Laptop
' }, { id: 4, product: 'Macbook' }];

// Middleware for parsing bodies from URL
app.use(bodyParser.urlencoded({ extended: false }));

// GET/items
app.get('/items', (req, res) => {
  if (items) {
    res.send(items);
  } else {
    res.send('List is empty. Please add items in the list');
  }
});

// POST/items
app.post('/items', (req, res) => {
  var item = req.body;
  items.push(item);
  res.send('Item added successfully in the list');
});

// GET/items/:id
app.get('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id);
  const item = items.find(item => item.id === itemId);
  if (item) {
    res.send(item);
  } else {
    res.send('Item is not available in the list');
  }
});

// PATCH/items/:id
app.patch('/items/:id', (req, res) => {
  var item = items.findIndex(item => item.id == req.params.id);
  if (items[item]) {
    items[item].product = req.body.product;
    res.send('Item is updated in the list');
    // console.log(items[item]);
  }
  else {
    res.send('Item is not available in the list');
  }
});

// DELETE/items/:id
app.delete('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id);
  var item = items.find(item => item.id == itemId);
  if (item) {
    items.splice(items.indexOf(item), 1);
    res.json(items);
  }
  else {
    res.send('Item is not available in the list');
  }
});

// To bind and listen the connections on the specified host and port
app.listen(3000, console.log("Listening on port 3000..."));
```

```
> http://localhost:3000/items -
[{"id":1,"product":"Bat"},{"id":2,"product":"Ball"},{"id":3,"product":"Laptop"},{"id":4,"
product":"Macbook"}]
```

> http://localhost:3000/items/:id - Item is not available in the list

8. Pug Template Engine

Create an Express.js app with a home page rendered by the Pug template engine.
The home page should respond to /home.
The view should show the current date using 'new Date().toDateString()'.
We use 'toDateString0 to simply return the date in a human-readable format without the time.

HINTS
The Pug template file index.pug must look like this:
h1 Hello World
p Today is #(date).

You can use our index.pug (recommended). The path to index.pug will be provided in process.argv[3]. Of course, you are welcome to use your own Pug file. Just make sure it's exactly the same as ours.
This is how you can specify the path to the template files in the folder templates:
app.set("views', path.join(_dirname, "templates')) The dirname is the absolute path of this file and path.join is used to produce cross
platform path (Win vs. Linux/Mac).
To tell Express.js app what template engine to use, apply this line to the Express.js configuration:
app.set('view engine', 'pug')
Instead of Hello World's res.end(), the res.render() function accepts a template
name and data (called locals):
res.render("index", (date: new Date().toDateString()))
We use topateString() to simply return the date in a human-readable format without the time.

NOTE
When creating your projects from scratch, install the pug dependency with npm. If you run $ npm install on this package (expressworks), you should have pug installed. Again, the port to use is passed by (appname) to the application as process.argv[2]. If you receive Error: Cannot find module 'pug', it is because Express is looking for Pug relative to its path. You can fix this by running npm install pug.
Ans:-

index.pug

```
h1 Hello World
p Today is #{date}.
```

Pug.js

```
const express = require('express')
const app = express()
const pug = require('pug')
const path = require('path')

app.set('views', path.join(__dirname, 'templates'))
app.set('view engine', 'pug')
app.get('/home', function(req, res){
    res.render('index', {date: new Date().toDateString()})
})

app.listen(process.argv[3])
```

9. JSON

Write a server that, when it receives a GET, reads a file, parses it to JSON, and responds with that content to the user. The server should respond to any GET that matches the books resource path. As always, the port is passed in process.argv[2]. The file to read is passed in process.argv[3]. Respond with:
res.json (object)
Everything should match the /books resource path. For reading the file, use the fs module, e.g.
fs.readFile(filename, callback)

<u>HINTS</u>

While the parsing can be done with JSON.parse: object-150N.parse(string)
No need to install the fs module. It's part of the core and the Node.js platform.
Ans:-

books.json

```
[
    {
        "title": "Express.js Guide",
        "tags": [
            "node.js",
            "express.js"
        ],
        "url": "http://expressjsguide.com"
    },
    {
        "title": "Rapid Prototyping with JS",
        "tags": [
            "backbone.js",
            "node.js",
            "mongodb"
        ],
        "url": "http://rpjs.co"
    },
    {
        "title": "JavaScript: The Good Parts",
        "tags": [
            "javascript"
        ]
    }
]
```

JSON1.js

```
const { errorMonitor } = require('events')
const express = require('express')
const app = express()
const fs = require('fs')

app.get('/books', function(req, res){
  const filename = process.argv[3]
  fs.readFile(filename, function(err, data) {
    if (err) return res.sendStatus(500)
    try {
      books = JSON.parse(data)
    } catch (err) {
      res.sendStatus(500)
    }
    res.json(books)
  })
})

app.listen(process.argv[2])
```